ITERATION 5

OUR CLIENT:

EPI-USE

groupelephant.com

Group 35

CANVAS

NGO PROJECT PORTFOLIO MANAGEMENT SYSTEM



Amahle Khoza

u22539702@tuks.co.za

+27 71 023 1872



Stephan Bredenhann

u23667436@tuks.co.za

+27 71 566 0997



Christopher Bailey

u24661092@tuks.co.za

+27 71 365 9963



Nicholas Bailey

u24671691@tuks.co.za

+27 71 610 6952



Alexander Steven

u22496786@tuks.co.za

+27 72 370 0225

Our Iteration 5 document focuses on the design and development of 83 use cases for EPI-USE and groupelephant.com. It includes our updated functional requirements and their descriptions, a technical use case diagram, technical ERD, technical use case narratives, technical context diagram, technical primitive level diagrams, screen designs, sequence diagrams, and state diagrams.

Table of Contents

Table of Tables 5
Table of Figures 11
1. Document Introduction 19
2. Design and Development of the Use Cases 20
2.1 Introduction 20
2.2 Use Case Outline 21
2.3 Screen Design 24
2.3.1 User Subsystem 25
2.3.2 Admin Subsystem 26
..... 29
2.3.3 Inventory Subsystem 30
2.3.4 Warehouse Subsystem 36
2.3.5 Vendor Subsystem 40
2.3.6 Project Management Subsystem 46
2.3.7 K9 Subsystem 52
2.3.8 Rhino Subsystem 68
2.3.9 Elephant Subsystem 74
2.3.10 Report Subsystem 77
2.3.11 Budget Subsystem 79
2.4 Technical Use Case Narratives 84
2.4.1 User Subsystem Technical Narratives 84
2.4.2 Admin Subsystem Technical Narratives 87
2.4.3 Inventory Subsystem Technical Narratives 117
2.4.4 Warehouse Subsystem Technical Narratives 167
2.4.5 Vendor Subsystem Technical Narratives 185
2.4.6 Project Management Subsystem Technical Narratives 225
2.4.7 K9 Subsystem Technical Narratives 255
2.4.8 Rhino Subsystem Technical Narratives 332
2.4.9 Elephant Subsystem Technical Narratives 358
2.4.10 Report Subsystem Technical Narratives 370

2.4.11 Budget Subsystem Technical Narratives 382
2.5 Technical Context Diagram 395
2.5.1 User Subsystem Context Diagram 395
2.5.2 Admin Subsystem Context Diagram 396
2.5.3 Inventory Subsystem Context Diagram 397
2.5.4 Warehouse Subsystem Context Diagram 398
2.5.5 Vendor Subsystem Context Diagram 399
2.5.6 Project Management Context Diagram 400
2.5.7 K9 Subsystem Context Diagram 401
2.5.8 Rhino Subsystem Context Diagram 402
2.5.9 Elephant Subsystem Context Diagram 403
2.5.10 Report Subsystem Context Diagram 404
2.5.11 Budget Subsystem Context Diagram 405
2.6 Technical Primitive Level Diagram 406
2.6.1 User Subsystem Primitive Diagrams 406
2.6.2 Admin Subsystem Primitive Diagrams 407
2.6.3 Inventory Subsystem Primitive Diagrams 415
2.6.4 Warehouse Subsystem Primitive Diagrams 425
2.6.5 Vendor Subsystem Primitive Diagrams 431
2.6.6 Project Management Subsystem Primitive Diagrams 442
2.6.7 K9 Subsystem Primitive Diagrams 453
2.6.8 Rhino Subsystem Primitive Diagrams 469
2.6.9 Elephant Subsystem Primitive Diagrams 474
2.6.10 Report Subsystem Primitive Diagrams 479
2.7 Sequence Diagram 489
2.7.1 User Subsystem Sequence Diagrams 489
2.7.2 Admin Subsystem Sequence Diagrams 490
2.7.3 Inventory Subsystem Sequence Diagrams 498
2.7.4 Warehouse Subsystem Sequence Diagrams 508
2.7.5 Vendor Subsystem Sequence Diagrams 514
2.7.6 Project Management Subsystem Sequence Diagrams 525
2.7.7 K9 Subsystem Sequence Diagrams 536

2.7.8 Rhino Subsystem Sequence Diagrams....552
2.7.9 Elephant Subsystem Sequence Diagrams....557
2.7.10 Report Subsystem Sequence Diagrams....562
2.7.11 Budget Subsystem Sequence Diagrams....566
2.8 Updated Requirements List....572
2.8.1 Introduction....572
2.8.2 Updated Functional Requirements Lists....572
2.8.3 Updated Functional Requirements Lists Descriptions....576
2.8.4 Functional Requirements List Detailed Changes Description....710
2.9 Technical Use Case Diagram....716
2.9.1 User Subsystem....716
2.9.2 Admin Subsystem....717
2.9.3 Inventory Subsystem....718
2.9.4 Warehouse Subsystem....719
2.9.5 Vendor Subsystem....720
2.9.6 Project Management....721
2.9.7 K9 Subsystem....723
2.9.10 Report Subsystem....726
2.9.11 Budget Subsystem....727
2.10 Technical Entity Relationship Diagram....728
2.11 State Diagrams....729
2.11.1 Projects Controller....729
2.11.2 Auth Controller....730
2.11.3 User Controller....731
2.11.4 Project Role Repository....732
2.11.5 Group Repository....733
3. Document Conclusion....734
4. Sign-off by Client....735
5. Sign-off by Team....736

Table of Tables

Table 1: Technical Narrative 1.9 View FAQs 86
Table 2: Technical Narrative 2.5 Add FAQ 90
Table 3: Technical Narrative 2.6 Update FAQ 94
Table 4: Technical Narrative 2.7 Remove FAQ 97
Table 5: Technical Narrative 2.8 Add FAQ Type 101
Table 6: Technical Narrative 2.9 Update FAQ Type 105
Table 7: Technical Narrative 2.10 Remove FAQ Type 109
Table 8: Technical Narrative 2.16 Update Role Permission 113
Table 9: Technical Narrative 2.17 View Role Permission 116
Table 10: Technical Narrative 3.1 Create Procurement 123
Table 11: Technical Narrative 3.2 View Procurement 127
Table 12: Technical Narrative 3.3 View Inventory 135
Table 13: Technical Narrative 3.4 Write-Off Inventory 140
Table 14: Technical Narrative 3.5 Allocate Inventory 145
Table 15: Technical Narrative 3.6 Create Inventory Item Type 149
Table 16: Technical Narrative 3.7 View Inventory Item Types 153
Table 17: Technical Narrative 3.8 Update Inventory Item Type 158
Table 18: Technical Narrative 3.9 Deactivate Inventory Item Type 162
Table 19: Technical Narrative 3.10 Reactivate Inventory Item Type 166
Table 20: Technical Narrative 5.1 Create Warehouse 169
Table 21: Technical Narrative 5.2 View Warehouse 171
Table 22: Technical Narrative 5.3 Update Warehouse 174
Table 23: Technical Narrative 5.4 Reactivate Warehouse 176
Table 24: Technical Narrative 5.5 Deactivate Warehouse 178
Table 25: Technical Narrative 5.6 Inventory Stock Take 184
Table 26: Technical Narrative 6.1 Create Vendor 188
Table 27: Technical Narrative 6.2 View Vendor 192
Table 28: Technical Narrative 6.3 View Vendors 195
Table 29: Technical Narrative 6.4 Edit Vendor 199
Table 30: Technical Narrative 6.5 Reactivate Vendor 203
Table 31: Technical Narrative 6.6 Deactivate Vendor 207
Table 32: Technical Narrative 6.7 Create Vendor Type 211
Table 33: Technical Narrative 6.8 Update Vendor Type 215
Table 34: Technical Narrative 6.9 View Vendor Types 218
Table 35: Technical Narrative 6.10 Reactivate Vendor Type 221
Table 36: Technical Narrative 6.11 Deactivate Vendor Type 224
Table 37: Technical Narrative 7.14 Capture Project Feedback 227
Table 38: Technical Narrative 7.15 View Project Feedback 230
Table 39: Technical Narrative 7.16 Create Impact Feedback Form 233

Table 40: Technical Narrative 7.17 View Impact Forms 236
Table 41: Technical Narrative 7.18 Update Impact Feedback Form 239
Table 42: Technical Narrative 7.19 Deactivate Impact Feedback Form 242
Table 43: Technical Narrative 7.20 Capture Impact Feedback 245
Table 44: Technical Narrative 7.21 View Impact Feedback 247
Table 45: Technical Narrative 7.22 Add Photo 250
Table 46: Technical Narrative 7.23 View Photo 252
Table 47: Technical Narrative 7.24 Delete Photo 254
Table 48: Technical Narrative 8.1 Add Dog 262
Table 49: Technical Narrative 8.2 View Dog 267
Table 50: Technical Narrative 8.3 View Dogs 271
Table 51: Technical Narrative 8.4 Update Dog 279
Table 52: Technical Narrative 8.5 Deactivate Dog 283
Table 53: Technical Narrative 8.6 Assign User to Dog 288
Table 54: Technical Narrative 8.7 View User-Dog Assignment 292
Table 55: Technical Narrative 8.8 Update User-Dog Assignment 297
Table 56: Technical Narrative 8.9 Reactivate User-Dog Assignment 302
Table 57: Technical Narrative 8.10 Deactivate User-Dog Assignment 306
Table 58: Technical Narrative 8.11 Add K9 Role 311
Table 59: Technical Narrative 8.12 Update K9 Role 316
Table 60: Technical Narrative 8.13 View K9 Roles 319
Table 61: Technical Narrative 8.14 Reactivate K9 Role 323
Table 62: Technical Narrative 8.15 Deactivate K9 Role 327
Table 63: Technical Narrative 8.16 View K9 Ancestry 331
Table 64: Technical Narrative 9.1 Add Rhino 337
Table 65: Technical Narrative 9.2 View Rhino 341
Table 66: Technical Narrative 9.3 View Rhinos 347
Table 67: Technical Narrative 9.4 Update Rhino 353
Table 68: Technical Narrative 9.5 Deactivate Rhino 357
Table 69: Technical Narrative 10.1 Add Elephant 360
Table 70: Technical Narrative 10.2 View Elephant 362
Table 71: Technical Narrative 10.3 View Elephants 364
Table 72: Technical Narrative 10.4 Update Elephant 367
Table 73: Technical Narrative 10.5 Deactivate Elephant 369
Table 74: Technical Narrative 12.1 Generate Custom Report 372
Table 75: Technical Narrative 12.2 View Custom Report 375
Table 76: Technical Narrative 12.3 Update Custom Report 378
Table 77: Technical Narrative 12.4 Remove Custom Report 381
Table 78: Technical Narrative 14.1 View Project Budget 384
Table 79: Technical Narrative 14.2 View Project Budgets 386
Table 80: Technical Narrative 14.3 Log An Expense 388

Table 81: Technical Narrative 14.4 Update An Expense 390
Table 82: Technical Narrative 14.5 View Expense 392
Table 83: Technical Narrative 14.6 Remove Expense 394
Table 84: 1.1 User Login – Description 576
Table 85: 1.2 User Logout Description 577
Table 86: 1.3 Reset Password – Description 579
Table 87: 1.4 Create User – Description 580
Table 88: 1.5 Update User – Description 581
Table 89: 1.6 View User – Description 582
Table 90: 1.7 Reactivate User - Description 583
Table 91: 1.8 Deactivate User - Description 584
Table 92: 1.9 View FAQ - Description 585
Table 93: 2.1 Create User Profile - Description 586
Table 94: 2.2 Update User Profile - Description 587
Table 95: 2.3 View User Profile - Description 588
Table 96: 2.4 Deactivate User Profile - Description 589
Table 97: 2.5 Add FAQ – Description 590
Table 98: 2.6 Update FAQ – Description 591
Table 99: 2.7 Remove FAQ – Description 592
Table 100: 2.8 Create Role – Description 596
Table 101: 2.9 View Role – Description 597
Table 102: 2.10 Update Role – Description 598
Table 103:2.11 Reactivate Role – Description 599
Table 104: 2.12 Deactivate Role – Description 600
Table 105: 2.13 Update Role Permission – Description 601
Table 106: 2.14 View Role Permission – Description 602
Table 107: 3.1 Receive Inventory Item – Description 603
Table 108: 3.2 View Procurement – Description 604
Table 109: 3.2 View Inventory – Description 605
Table 110: 3.3 Write-Off Inventory - Description 606
Table 111: 3.4 Allocate Inventory – Description 607
Table 112: 3.5 Create Inventory Item Type – Description 608
Table 113:3.7 View Inventory Item Types – Description 609
Table 114: 3.8 Update Inventory Item Type – Description 610
Table 115: 3.9 Deactivate Inventory Item Type - Description 611
Table 116: 3.10 Reactivate Inventory Item Type - Description 612
Table 117: 4.1 Create Group – Description 613
Table 118:4.2 View Group – Description 614
Table 119: 4.3 Update Group – Description 615
Table 120: 4.4 Deactivate Group - Description 616
Table 121: 4.5 Create Skill - Description 617

Table 122: 4.6 View Skills - Description 618
Table 123: View Skill - Description.....619
Table 124: 4.7 Update Skill - Description.....620
Table 125: 4.8 Deactivate Skill - Description 621
Table 126 :5.1 Create Warehouse - Description.....622
Table 127: 5.2 View Warehouse - Description 623
Table 128: 5.3 Update Warehouse - Description 624
Table 129 5.4: Reactivate Warehouse - Description 625
Table 130: 5.5 Deactivate Warehouse - Description 626
Table 131: Inventory Stock Take - Description 627
Table 132: 6.1 Create Vendor - Description 628
Table 133 6.2: View Vendor - Description 629
Table 134: 6.3 View Vendors - Description.....630
Table 135: 6.4 Edit Vendor - Description.....631
Table 136: 6.5 Reactivate Vendor - Description 632
Table 137: 6.6 Deactivate Vendor - Description.....633
Table 138: 6.7 Create Vendor Type - Description.....634
Table 139: 6.8 Update Vendor Type - Description 635
Table 140: 6.8 View Vendor Types - Description.....636
Table 141: 6.10 Reactivate Vendor Type - Description 637
Table 142: 6.11 Deactivate Vendor Type - Description 638
Table 143: 7.1 Create Project - Description.....639
Table 144: 7.2 View Project - Description.....640
Table 145: 7.3 Update Project - Description.....641
Table 146: 7.4 Deactivate Project - Description 642
Table 147: 7.5 Create Project Type - Description 643
Table 148: 7.6 View Project Type - Description.....644
Table 149: 7.7 Update Project Type - Description.....645
Table 150: 7.8 Deactivate Project Type - Description.....646
Table 151: 7.9 Add Indemnity Form - Description.....647
Table 152: 7.10 Update Indemnity Form - Description 648
Table 153: 7.11 View Indemnity Form - Description 649
Table 154: 7.12 Remove Indemnity Form - Description.....650
Table 155: 7.13 Capture Indemnity Form - Description 651
Table 156: 7.14 Capture Project Feedback - Description 652
Table 157: 7.15 View Project Feedback - Description 653
Table 158: 7.16 Create Impact Feedback Form - Description 654
Table 159: 7.17 View Impact Forms - Description.....655
Table 160: 7.18 Update Impact Form - Description 656
Table 161: 7.19 Deactivate Impact Form - Description 657
Table 162: 7.20 Capture Impact Form - Description 658

Table 163: 7.21 View Impact Feedback - Description.....659
Table 164: 7.22 Add Photo - Description.....660
Table 165: 7.23 View Photo - Description.....661
Table 166: 7.24 Delete Photo - Description.....662
Table 167: 8.1 Add Dog - Description.....663
Table 168: 8.2 View Dog - Description.....664
Table 169: 8.3 View All Dogs- Description.....665
Table 170: 8.4 Update Dog - Description.....666
Table 171: 8.5 Deactivate Dog - Description.....667
Table 172: 8.6 Assign User to Dog - Description.....668
Table 173: 8.7 View User-Dog Assignment – Description.....669
Table 174: 8.8 Update User-Dog Assignment - Description.....670
Table 175: 8.9 Reactivate User-Dog Assignment - Description.....671
Table 176: 8.10 Deactivate User-Dog Assignment - Description.....672
Table 177: 8.11 Add K9 Role - Description.....673
Table 178: 8.12 Update K9 Role - Description.....674
Table 179: 8.13 View K9 Role - Description.....675
Table 180: 8.14 Reactivate K9 Role - Description.....676
Table 181: 8.15 Deactivate K9 Role – Description.....677
Table 182: 8.16 View K9 Ancestry - Description.....678
Table 183: 9.1 Add Rhino - Description.....679
Table 184:9.2 View Rhino - Description.....680
Table 185: 9.3 View Rhinos - Description.....681
Table 186: 9.4 Update Rhino - Description.....682
Table 187: 9.5 9.5 Deactivate Rhino - Description.....683
Table 188: 10.1 Add Elephant - Description.....684
Table 189: 10.2 View Elephant - Description.....685
Table 190: 10.3 View Elephants - Description.....686
Table 191: 10.4 Update Elephant - Description.....687
Table 192: 10.5 Deactivate Elephant - Description.....688
Table 193: 11.1 Create Message – Description.....689
Table 194: 11.2 View Message - Description.....690
Table 195: 12.1 Generate Custom Report - Description.....691
Table 196: 12.2 View Custom Report - Description.....692
Table 197: 12.3 Update Custom Report - Description.....693
Table 198: 12.4 Remove Custom Report - Description.....694
Table 199: 12.5 View Bike Impact Report - Description.....695
Table 200: 12.6 View Budget Report – Description.....696
Table 201: 12.7 View Rhino Report - Description.....697
Table 202: 12.8 View Elephant Report - Description.....698
Table 203: 12.9 View K9 Report - Description.....699

Table 204: 12.10 View Machine Learning Dashboard - Description 700
Table 205: 13.1 Backup System - Description.....701
Table 206: 13.2 Restore System - Description.....702
Table 207: 13.3 View Audit Log - Description.....703
Table 208: 14.1 View Project Budget - Description 704
Table 209: 14.2 View Project Budgets - Description 705
Table 210: 14.3 Log An Expense - Description.....706
Table 211: 14.4 Update An Expense - Description 707
Table 212: 14.5 View An Expense - Description 708
Table 213: 16.6 Remove an Expense - Description.....709
Table 214: User Subsystem Modifications.....710
Table 215: Admin Subsystem Modifications 711
Table 216 Inventory Subsystem Modifications 711
Table 217 Group Subsystem Modifications.....711
Table 218: Warehouse Subsystem Modifications 712
Table 219 Vendor Subsystem Modifications 712
Table 220 Vendor Subsystem Modifications 713
Table 221 K9 Subsystem Modifications 713
Table 222 Rhino Subsystem Modifications.....714
Table 223 Elephant Subsystem Modifications.....714
Table 224 Message Subsystem Modifications.....714
Table 225: Report Subsystem Modifications.....714
Table 226: System Subsystem Modifications 715
Table 227 Budget Subsystem Modifications.....715

Table of Figures

Figure 1: 1.9 View FAQ Screen Design 25
Figure 2: 2.5 Add FAQ Screen Design 26
Figure 3: 2.6 Update FAQ Screen Design 26
Figure 4: 2.7 Remove FAQ Screen Design 27
Figure 5: 2.8 Add FAQ Type Screen Design 27
Figure 6: 2.9 Update FAQ Type Screen Design 28
Figure 7: 2.10 Remove FAQ Type Screen Design 28
Figure 8: Update Role Permission Screen Design 29
Figure 9: View Role Permission Screen Design 29
Figure 10: 3.1 Create Procurement Screen Design 30
Figure 11: 3.2 View Procurement Screen Design 30
Figure 12: 3.3 View Inventory Screen Design – Inventory Screen 31
Figure 13: 3.3 View Inventory Screen Design – Quantity Totals Screen 31
Figure 14: 3.4 Write-Off Inventory Screen Design 32
Figure 15: 3.5 Allocate Inventory Screen Design 32
Figure 16: 3.6 Create Inventory Item Type Screen Design 33
Figure 17: 3.7 View Inventory Item Types Screen Design 33
Figure 18: 3.8 Update Inventory Item Type Screen Design 34
Figure 19: 3.9 Deactivate Inventory Item Type Screen Design 34
Figure 20: 3.10 Reactivate Inventory Item Type Screen Design 35
Figure 21: 3.10 Reactivate Inventory Item Type Confirmation Screen Design 35
Figure 22: 5.1 Create Warehouse Screen Design 36
Figure 23: 5.2 View Warehouse Screen Design 36
Figure 24: 5.3 Update Warehouse Screen Design 37
Figure 25: 5.4 Reactivate Warehouse Screen Design 37
Figure 26: 5.5 Deactivate Warehouse Screen Design 38
Figure 27: 5.6 Inventory Stock Take Screen Design 1 38
Figure 28: 5.6 Inventory Stock Take Screen Design 2 39
Figure 29: 6.1 Create Vendor Screen Design 40
Figure 30: 6.2 View Vendor Screen Design 40
Figure 31: 6.3 View Vendors Screen Design 41
Figure 32: 6.4 Edit Vendor Screen Design 41
Figure 33: 6.5 Reactivate Vendor Screen Design 42
Figure 34: 6.6 Deactivate Vendor Screen Design 42
Figure 35: 6.7 Create Vendor Type Screen Design 43
Figure 36: 6.8 Update Vendor Type Screen Design 43
Figure 37: 6.9 View Vendor Types Screen Design 44
Figure 38: 6.10 Reactivate Vendor Type Screen Design 44
Figure 39: 6.11 Deactivate Vendor Type Screen Design 45

Figure 40: 7.14 Capture Project Feedback Screen Design 46
Figure 41: 7.15 View Project Feedback Screen Design 46
Figure 42: 7.16 Create Impact Feedback Form Screen Design 47
Figure 43: 7.17 View Impact Feedback Form Screen Design 47
Figure 44: 7.18 Update Impact Feedback Form Screen Design 48
Figure 45: 7.19 Deactivate Impact Feedback Form Screen Design 48
Figure 46: 7.20 Capture Impact Form Screen Design 49
Figure 47: 7.21 View Impact Form Screen Design 49
Figure 48: 7.22 Add Photo Screen Design 50
Figure 49: 7.23 View Photo Screen Design 50
Figure 50: 7.24 Delete Photo Screen Design 51
Figure 51: 8.1 Add Dog Screen Design 52
Figure 52: 8.1 Add Dog Screen Design Continued 52
Figure 53: 8.1 Add Dog Confirmation Screen Design 53
Figure 54: 8.2 View Dog Screen Design 53
Figure 55: 8.2 View Dog Screen Design Continued 54
Figure 56: 8.3 View Dogs Screen Design 54
Figure 57: 8.4 Update Dog Screen Design Continued 55
Figure 58: 8.4 Update Dog Screen Design 55
Figure 59: 8.4 Update Dog Confirmation Screen Design 56
Figure 60: 8.5 Deactivate Dog Screen Design 56
Figure 61: 8.6 Assign User to Dog Screen Design Continued 57
Figure 62: 8.6 Assign User to Dog Screen Design 57
Figure 63: 8.6 Assign User to Dog Confirmation Screen Design 58
Figure 64: 8.7 View User-Dog Assignment Screen Design 58
Figure 65: 8.7 View User-Dog Assignment Screen Design Continued 59
Figure 66: 8.8 Update User-Dog Assignment Screen Design 59
Figure 67: 8.8 Update User-Dog Assignment Confirmation Screen Design 60
Figure 68: 8.8 Update User-Dog Assignment Screen Design Continued 60
Figure 69: 8.9 Reactivate User-Dog Assignment Screen Design 61
Figure 70: 8.9 Reactivate User-Dog Assignment Confirmation Screen Design 61
Figure 71: 8.10 Deactivate User-Dog Assignment Screen Design 62
Figure 72: 8.10 Deactivate User-Dog Assignment Confirmation Screen Design 62
Figure 73: 8.11 Add K9 Role Confirmation Screen Design 63
Figure 74: 8.11 Add K9 Role Screen Design 63
Figure 75: 8.12 Update K9 Role Confirmation Screen Design 64
Figure 76: 8.12 Update K9 Role Screen Design 64
Figure 77: 8.13 View K9 Roles Screen Design 65
Figure 78: 8.14 Reactivate K9 Role Screen Design 65
Figure 79: 8.15 Deactivate K9 Role Screen Design 66
Figure 80: 8.16 View K9 Ancestry Screen Design 66

Figure 81: 8.16 View K9 Ancestry Screen Design Continued 67
Figure 82: 8.16 View K9 Ancestry Screen Design Continued 67
Figure 83: 9.1 Add Rhino Screen Design 68
Figure 84: 9.1 Add Rhino Confirmation Screen Design 68
Figure 85: 9.2 View Rhino Screen Design 69
Figure 86: View Rhino Map Screen Design 70
Figure 87: View Rhinos Heatmap Map Screen Design 70
Figure 88: View Rhino Screen Design 71
Figure 89: 9.4 Update Rhino Screen Design 72
Figure 90: 9.5 Deactivate Rhino Screen Design 73
Figure 91: 10.1 Add Elephant Screen Design 74
Figure 92: 10.2 View Elephant Screen Design 74
Figure 93: 5.3 View Elephants Screen Design 75
Figure 94: 5.4 Update Elephant Screen Design 75
Figure 95: 10.5 Deactivate Elephant Screen Design 76
Figure 96: 12.1 Generate Custom Report Screen Design 77
Figure 97:12.2 View Custom Report Screen Design 77
Figure 98: 12.3 Update Custom Report Screen Design 78
Figure 99: 12.4 Remove Custom Report Screen Design 78
Figure 100: 14.1 View Project Budget Screen Design 79
Figure 101: 14.2 View Project Budgets Screen Design 79
Figure 102: 14.2 View Project Budgets Screen Design Continued 80
Figure 103: 14.3 Log An Expense Screen Design 80
Figure 104: 14.4 Update An Expense Screen Design 81
Figure 105: 14.4 Update An Expense Screen Design (Continued) 81
Figure 106: 14.5 View An Expense Screen Design 82
Figure 107: 14.6 Remove An Expense Screen Design 82
Figure 108: 14.6 Remove An Expense Screen Design (Continued) 83
Figure 109: User Technical Context Diagram 395
Figure 110: Admin Technical Context Diagram 396
Figure 111: Inventory Technical Context Diagram 397
Figure 112: Warehouse Technical Context Diagram 398
Figure 113: Vendor Technical Context Diagram 399
Figure 114: Project Management Technical Context Diagram 400
Figure 115: K9 Technical Context Diagram 401
Figure 116: Rhino Technical Context Diagram 402
Figure 117: Elephant Technical Context Diagram 403
Figure 118: Report Technical Context Diagram 404
Figure 119: Budget Technical Context Diagram 405
Figure 120: Technical Primitive Level Diagram 1.9 View FAQ 406
Figure 121: Technical Primitive Level Diagram 2.5 Add FAQ 407

Figure 122: Technical Primitive Level Diagram 2.6 Update FAQ 408
Figure 123: Technical Primitive Level Diagram 2.7 Remove FAQ 409
Figure 124: Technical Primitive Level Diagram 2.8 Add FAQ Type 410
Figure 125: Technical Primitive Level Diagram 2.9 Update FAQ Type 411
Figure 126: Technical Primitive Level Diagram 2.10 Remove FAQ Type 412
Figure 127: Technical Primitive Level Diagram 2.16 Update Role Permission 413
Figure 128: Technical Primitive Level Diagram 2.17 View Role Permission 414
Figure 129: Technical Primitive Level Diagram 3.1 Create Procurement 415
Figure 130: Technical Primitive Level Diagram 3.2 View Procurement 416
Figure 131: Technical Primitive Level Diagram 3.3 View Inventory 417
Figure 132: Technical Primitive Level Diagram 3.4 Write-Off Inventory 418
Figure 133: Technical Primitive Level Diagram 3.5 Allocate Inventory 419
Figure 134: Technical Primitive Level Diagram 3.6 Create Inventory Item Type 420
Figure 135: Technical Primitive Level Diagram 3.7 View Inventory Item Types 421
Figure 136: Technical Primitive Level Diagram 3.8 Update Inventory Item Type 422
Figure 137: Technical Primitive Level Diagram 3.9 Deactivate Inventory Item Type 423
Figure 138: Technical Primitive Level Diagram 3.10 Reactivate Inventory Item Type 424
Figure 139: Technical Primitive Level Diagram 5.1 Create Warehouse 425
Figure 140: Technical Primitive Level Diagram 5.2 View Warehouse 426
Figure 141: Technical Primitive Level Diagram 5.3 Update Warehouse 427
Figure 142: Technical Primitive Level Diagram 5.4 Reactivate Warehouse 428
Figure 143: Technical Primitive Level Diagram 5.5 Deactivate Warehouse 429
Figure 144: Technical Primitive Level Diagram 5.6 Inventory Stock Take 430
Figure 145: Technical Primitive Level Diagram 6.1 Create Vendor 431
Figure 146: Technical Primitive Level Diagram 6.2 View Vendor 432
Figure 147: Technical Primitive Level Diagram 6.3 View Vendors 433
Figure 148: Technical Primitive Level Diagram 6.4 Edit Vendor 434
Figure 149: Technical Primitive Level Diagram 6.5 Reactivate Vendor 435
Figure 150: Technical Primitive Level Diagram 6.6 Deactivate Vendor 436
Figure 151: Technical Primitive Level Diagram 6.7 Create Vendor Type 437
Figure 152: Technical Primitive Level Diagram 6.8 Update Vendor Type 438
Figure 153: Technical Primitive Level Diagram 6.9 View Vendor Types 439
Figure 154: Technical Primitive Level Diagram 6.10 Reactivate Vendor Type 440
Figure 155: Technical Primitive Level Diagram 6.11 Deactivate Vendor Type 441
Figure 156: Technical Primitive Level Diagram 7.14 Capture Project Feedback 442
Figure 157: : Technical Primitive Level Diagram 7.15 View Project Feedback 443
Figure 158: Technical Primitive Level Diagram 7.16 Create Impact Feedback Form 444
Figure 159: Technical Primitive Level Diagram 7.17 View Impact Forms 445
Figure 160: Technical Primitive Level Diagram 7.18 Update Impact Feedback Form 446
Figure 161: Technical Primitive Level Diagram 7.19 Deactivate Impact Feedback Form 447
Figure 162: Technical Primitive Level Diagram 7.20 Capture Impact Feedback 448

Figure 163: Technical Primitive Level Diagram 7.21 View Impact Feedback 449
Figure 164: Technical Primitive Level Diagram 7.22 Add Photo.....450
Figure 165: Technical Primitive Level Diagram 7.23 View Photo 451
Figure 166: Technical Primitive Level Diagram 7.24 Delete Photo.....452
Figure 167: Technical Primitive Level Diagram 8.1 Add Dog 453
Figure 168: Technical Primitive Level Diagram 8.2 View Dog.....454
Figure 169: Technical Primitive Level Diagram 8.3 View Dogs 455
Figure 170: Technical Primitive Level Diagram 8.4 Update Dog.....456
Figure 171: Technical Primitive Level Diagram 8.5 Deactivate Dog 457
Figure 172: Technical Primitive Level Diagram 8.6 Assign User to Dog 458
Figure 173: Technical Primitive Level Diagram 8.7 View User-Dog Assignment.....459
Figure 174: Technical Primitive Level Diagram 8.8 Update User-Dog Assignment.....460
Figure 175: Technical Primitive Level Diagram 8.9 Reactivate User-Dog Assignment.....461
Figure 176: Technical Primitive Level Diagram 8.10 Deactivate User-Dog Assignment.....462
Figure 177: Technical Primitive Level Diagram 8.11 Add K9 Role 463
Figure 178: Technical Primitive Level Diagram 8.12 Update K9 Role.....464
Figure 179: Technical Primitive Level Diagram 8.13 View K9 Roles 465
Figure 180: Technical Primitive Level Diagram 8.14 Reactivate K9 Role.....466
Figure 181: Technical Primitive Level Diagram 8.15 Deactivate K9 Role 467
Figure 182: Technical Primitive Level Diagram 8.16 View K9 Ancestry.....468
Figure 183: Technical Primitive Level Diagram 9.1 Add Rhino 469
Figure 184: Technical Primitive Level Diagram 9.2 View Rhino.....470
Figure 185: Technical Primitive Level Diagram 9.3 View Rhinos 471
Figure 186: Technical Primitive Level Diagram 9.4 Update Rhino.....472
Figure 187: Technical Primitive Level Diagram 9.5 Deactivate Rhino 473
Figure 188: Technical Primitive Level Diagram 10.1 Add Elephant 474
Figure 189: Technical Primitive Level Diagram 10.2 View Elephant.....475
Figure 190: Technical Primitive Level Diagram 10.3 View Elephants 476
Figure 191: Technical Primitive Level Diagram 10.4 Update Elephant.....477
Figure 192: Technical Primitive Level Diagram 10.5 Deactivate Elephant 478
Figure 193: Technical Primitive Level Diagram 12.1 Generate Custom Report 479
Figure 194: Technical Primitive Level Diagram 12.2 View Custom Report.....480
Figure 195: Technical Primitive Level Diagram 12.3 Update Custom Report.....481
Figure 196: Technical Primitive Level Diagram 12.4 Remove Custom Report 482
Figure 197: Technical Primitive Level Diagram 14.1 View Project Budget 483
Figure 198: Technical Primitive Level Diagram 14.2 View Project Budgets 484
Figure 199: Technical Primitive Level Diagram 14.3 Log An Expense.....485
Figure 200: Technical Primitive Level Diagram 14.4 Update An Expense 486
Figure 201: Technical Primitive Level Diagram 14.5 View Expense.....487
Figure 202: Technical Primitive Level Diagram 14.6 Remove Expense 488
Figure 203: Sequence Diagram 1.9 View FAQ 489

Figure 204: Sequence Diagram 2.5 Add FAQ 490
Figure 205: Sequence Diagram 2.6 Update FAQ 491
Figure 206: Sequence Diagram 2.7 Remove FAQ 492
Figure 207: Sequence Diagram 2.8 Add FAQ Type 493
Figure 208: Sequence Diagram 2.9 Update FAQ Type 494
Figure 209: Sequence Diagram 2.10 Remove FAQ Type 495
Figure 210: Sequence Diagram 2.16 Update Role Permission 496
Figure 211: Sequence Diagram 2.17 View Role Permission 497
Figure 212: Sequence Diagram 3.1 Create Procurement 498
Figure 213: Sequence Diagram 3.2 View Procurement 499
Figure 214: Sequence Diagram 3.3 View Inventory 500
Figure 215: Sequence Diagram 3.4 Write-Off Inventory 501
Figure 216: Sequence Diagram 3.5 Allocate Inventory 502
Figure 217: Sequence Diagram 3.6 Create Inventory Item Type 503
Figure 218: Sequence Diagram 3.7 View Inventory Item Types 504
Figure 219: Sequence Diagram 3.8 Update Inventory Item Type 505
Figure 220: Sequence Diagram 3.9 Deactivate Inventory Item Type 506
Figure 221: Sequence Diagram 3.10 Reactivate Inventory Item Type 507
Figure 222: Sequence Diagram 5.1 Create Warehouse 508
Figure 223: Sequence Diagram 5.2 View Warehouse 509
Figure 224: Sequence Diagram 5.3 Update Warehouse 510
Figure 225: Sequence Diagram 5.4 Reactivate Warehouse 511
Figure 226: Sequence Diagram 5.5 Deactivate Warehouse 512
Figure 227: Sequence Diagram 5.6 Inventory Stock Take 513
Figure 228: Sequence Diagram 6.1 Create Vendor 514
Figure 229: Sequence Diagram 6.2 View Vendor 515
Figure 230: Sequence Diagram 6.3 View Vendors 516
Figure 231: Sequence Diagram 6.4 Edit Vendor 517
Figure 232: Sequence Diagram 6.5 Reactivate Vendor 518
Figure 233: Sequence Diagram 6.6 Deactivate Vendor 519
Figure 234: Sequence Diagram 6.7 Create Vendor Type 520
Figure 235: Sequence Diagram 6.8 Update Vendor Type 521
Figure 236: Sequence Diagram 6.9 View Vendor Types 522
Figure 237: Sequence Diagram 6.10 Reactivate Vendor Type 523
Figure 238: Sequence Diagram 6.11 Deactivate Vendor Type 524
Figure 239: Sequence Diagram 7.14 Create Project Feedback 525
Figure 240: Sequence Diagram 7.15 View Project Feedback 526
Figure 241: Sequence Diagram 7.16 Create Impact Feedback Form 527
Figure 242: Sequence Diagram 7.17 View Impact 528
Figure 243: Sequence Diagram 7.18 Update Impact Feedback Form 529
Figure 244: Sequence Diagram 7.19 Deactivate Impact Feedback Form 530

Figure 245: Sequence Diagram 7.20 Capture Impact Feedback Form 531
Figure 246: Sequence Diagram 7.21 View Impact Form.....532
Figure 247: Sequence Diagram 7.22 Add Photo 533
Figure 248: Sequence Diagram 7.23 View Photo 534
Figure 249: Sequence Diagram 7.24 Delete Photo.....535
Figure 250: Sequence Diagram 8.1 Add Dog 536
Figure 251: Sequence Diagram 8.2 View Dog.....537
Figure 252: Sequence Diagram 8.3 View Dogs 538
Figure 253: Sequence Diagram 8.4 Update Dog.....539
Figure 254: Sequence Diagram 8.5 Deactivate Dog 540
Figure 255: Sequence Diagram 8.6 Assign User to Dog.....541
Figure 256: Sequence Diagram 8.7 View User-Dog Assignment 542
Figure 257: Sequence Diagram 8.8 Update User-Dog Assignment 543
Figure 258: Sequence Diagram 8.9 Reactivate User-Dog Assignment 544
Figure 259: Sequence Diagram 8.10 Deactivate User-Dog Assignment.....545
Figure 260: Sequence Diagram 8.11 Add K9 Role.....546
Figure 261: Sequence Diagram 8.12 Update K9 Role 547
Figure 262: Sequence Diagram 8.13 View K9 Roles 548
Figure 263: Sequence Diagram 8.14 Reactivate K9 Role 549
Figure 264: Sequence Diagram 8.15 Deactivate K9 Role.....550
Figure 265: Sequence Diagram 8.16 View K9 Ancestry 551
Figure 266: Sequence Diagram 9.1 Add Rhino 552
Figure 267: Sequence Diagram 9.2 View Rhino.....553
Figure 268: Sequence Diagram 9.3 View Rhinos 554
Figure 269: Sequence Diagram 9.4 Update Rhino.....555
Figure 270: Sequence Diagram 9.5 Deactivate Rhino.....556
Figure 271: Sequence Diagram 10.1 Add Elephant 557
Figure 272: Sequence Diagram 10.2 View Elephant.....558
Figure 273: Sequence Diagram 10.3 View Elephant.....559
Figure 274: Sequence Diagram 10.4 View Elephants 560
Figure 275: Sequence Diagram 10.5 Deactivate Elephant 561
Figure 276: Sequence Diagram 12.1 Generate Custom Report.....562
Figure 277: Sequence Diagram 12.2 View Custom Report.....563
Figure 278: Sequence Diagram 12.3 Update Custom Report.....564
Figure 279: Sequence Diagram 12.4 Remove Custom Report.....565
Figure 280: Sequence Diagram 14.1 View Project Budget 566
Figure 281: Sequence Diagram 14.2 View Project Budgets.....567
Figure 282: Sequence Diagram 14.3 Log an Expense 568
Figure 283: Sequence Diagram 14.4 Update an Expense.....569
Figure 284: Sequence Diagram 14.5 View Expense.....570
Figure 285: Sequence Diagram 14.6 Remove Expense.....571

Figure 286: User Technical Use Case Diagram....716
Figure 287: Admin Technical Use Case Diagram....717
Figure 288: Inventory Technical Use Case Diagram....718
Figure 289: Warehouse Technical Use Case Diagram....719
Figure 290: Vendor Technical Use Case Diagram....720
Figure 291: Project Management Technical Use Case Diagram....721
Figure 292: Project Management Technical Use Case Diagram (Continued)....722
Figure 293: K9 Technical Use Case Diagram....723
Figure 294: Rhino Technical Use Case Diagram....724
Figure 295: Technical Use Case Diagram....725
Figure 296: Report Technical Use Case Diagram....726
Figure 297: Budget Technical Use Case Diagram....727
Figure 298: Project Controller State Diagram....729
Figure 299: Auth Controller State Diagram....730
Figure 300: User Controller State Diagram....731
Figure 301: Project Role Repository State Diagram....732
Figure 302: Group Repository State Diagram....733

1. Document Introduction

This document presents the fifth iteration of the Integrated ERP Project Portfolio Management System for EPI-USE and groupelephant.com.

Iteration 5 contains an updated list of requirements together with the summary description of the requirements, a use case diagram for the use cases which will be designed during Iteration 5, technical ERD for the data elements involved in Iteration 5. For the assigned use cases, this document has included the screen designs for the use cases, along with the technical use case narrative, technical primitive diagram, technical context diagram, state diagrams and the sequence diagrams.

2. Design and Development of the Use Cases

2.1 Introduction

This section of the document outlines the design and development of 81 use cases assigned to Iteration 5. It includes screen designs that follow efficient, intuitive, and visually appealing user interface aimed at optimizing user experience.

This section also includes technical use case narratives, technical context diagrams, technical primitive diagrams and sequence diagrams which help better understand how each use case works and help visualize the system's flow and interactions.

The present portion of the document also includes the requirements list, technical use case diagram, and the technical ERD which have been refined using client feedback from previous iterations.

2.2 Use Case Outline

As the system consists of 129 distinct use cases, at least 65% of the use cases need to be completed.

129 x 65% = 83.85

We have decided to complete 83 use cases and based on the outcomes of the dependency matrices; we decided to complete the following use cases:

User Subsystem (Subsystem 1):

1.9 View FAQ

Admin Subsystem (Subsystem 2):

2.5 Add FAQ

2.6 Update FAQ

2.7 Remove FAQ

2.8 Add FAQ Type

2.9 Update FAQ Type

2.10 Remove FAQ Type

2.16 Update Role Permission

2.17 View Role Permission

Inventory Subsystem (Subsystem 3):

3.1 Create Procurement

3.2 View Procurement

3.3 View Inventory

3.4 Write-off Inventory

3.5 Allocate inventory

3.6 Create Inventory Item Type

3.7 View Inventory Item Types

3.8 Update Inventory Item Type

3.9 Deactivate Inventory Item Type

3.10 Reactivate Inventory Item type

Warehouse Subsystem (Subsystem 5):

5.1 Create Warehouse

5.2 View Warehouse

5.3 Update Warehouse

5.4 Reactivate Warehouse

5.5 Deactivate Warehouse

5.6 Inventory Stock Take

Vendor Subsystem (Subsystem 6):

6.1 Create Vendor

6.2 View Vendor

6.3 View Vendors

6.4 Edit Vendor

6.5 Reactivate Vendor

6.6 Deactivate Vendor

6.7 Create Vendor Type

6.8 Update Vendor Type

6.9 View Vendor Types

6.10 Reactivate Vendor Type

6.11 Deactivate Vendor Type

Project Management Subsystem (Subsystem 7):

7.14 Capture Project Feedback

7.15 View Project Feedback

7.16 Create Impact Feedback Form

7.17 View Impact Feedback Form

7.18 Update Impact Feedback Form

7.19 Deactivate Impact Feedback Form

7.20 Capture Impact Feedback Form

7.21 View Impact Feedback Form

7.22 Add Photo

7.23 View Photo

7.24 Delete Photo

K9 Subsystem (Subsystem 8):

8.1 Add Dog

8.2 View Dog

8.3 View Dogs

8.4 Update Dog

8.5 Deactivate Dog

8.6 Assign User to Dog

8.7 View User-Dog Assignment

8.8 Update User-Dog Assignment

8.9 Reactivate User-Dog Assignment

8.10 Deactivate User-Dog Assignment

8.11 Add K9 Role

8.12 Update K9 Role

8.13 View K9 Roles

8.14 Reactivate K9 Role

8.15 Deactivate K9 Role

8.16 View K9 Ancestry

Rhino Subsystem (Subsystem 9):

9.1 Add Rhino

9.2 View Rhino

9.3 View Rhinos

9.4 Update Rhino

9.5 Deactivate Rhino

Elephant Subsystem (Subsystem 10):

10.1 Add Elephant

10.2 View Elephant

10.3 View Elephants

10.4 Update Elephant

10.5 Deactivate Elephant

Report Subsystem (Subsystem 12):

12.1 Generate Custom Report

12.2 View Custom Report

12.3 Update Custom Report

12.4 Remove Custom Report

Budget Subsystem (Subsystem 14):

14.1 View Project Budget

14.2 View Project Budgets

14.3 Log An Expense

14.4 Update An Expense

14.5 View Expense

14.6 Remove Expense

2.3 Screen Design

Each screen design has been created focusing on creating intuitive, efficient and visually appealing user interfaces. Attached figures of the screen design will be parted with its use case.

2.3.1 User Subsystem

2.3.1.1 View FAQ: Use Case 1.9

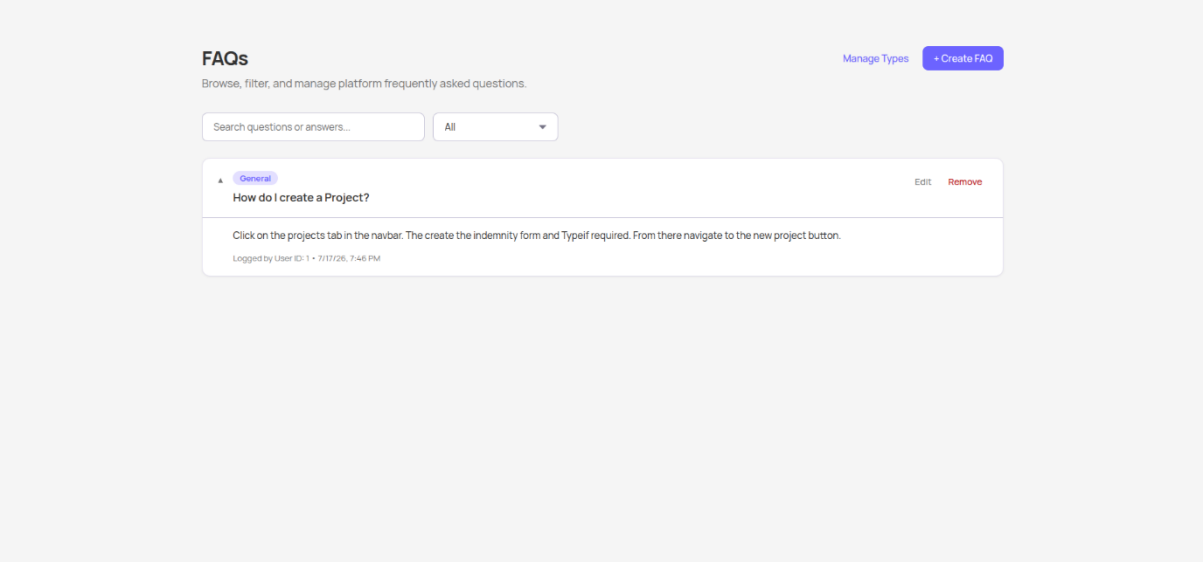


Figure 1: 1.9 View FAQ Screen Design

2.3.2 Admin Subsystem

2.3.2.1 Add FAQ: Use Case 2.5

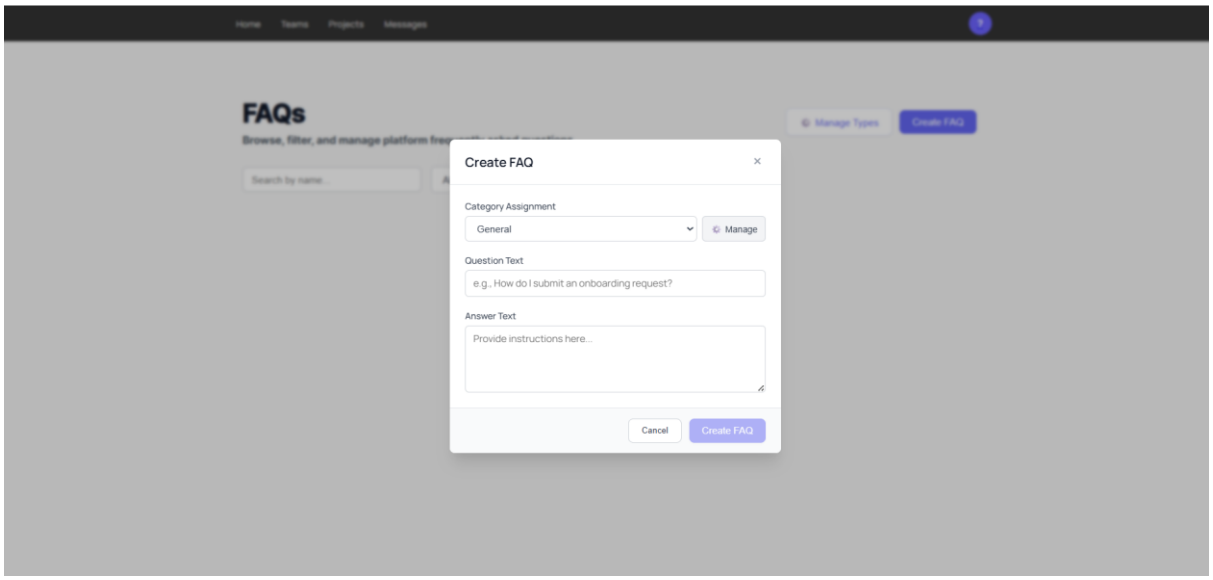


Figure 2: 2.5 Add FAQ Screen Design

2.3.2.2 Update FAQ: Use Case 2.6

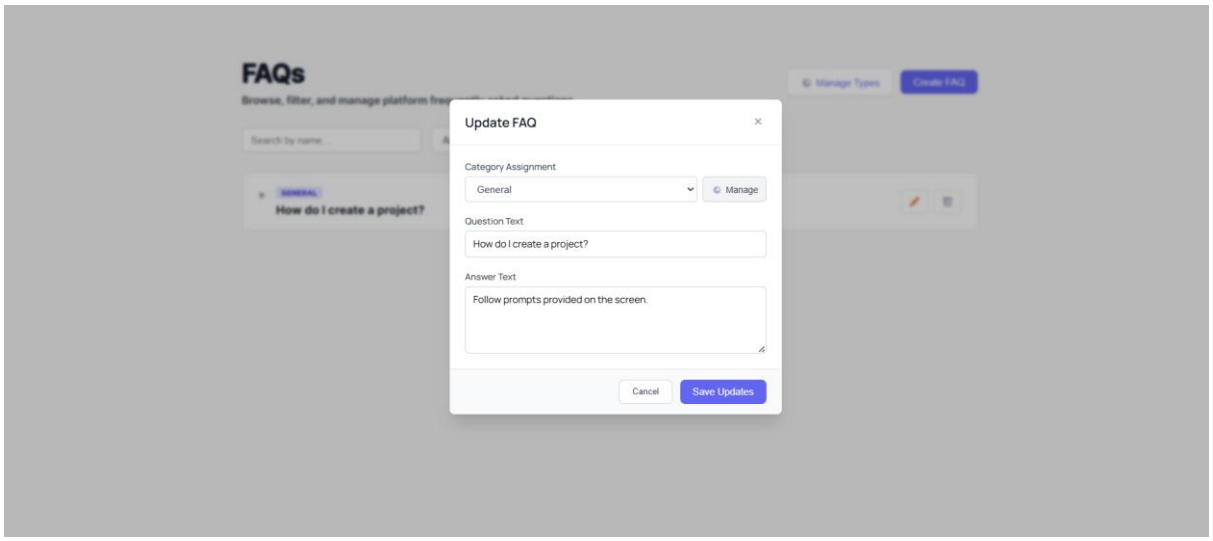


Figure 3: 2.6 Update FAQ Screen Design

2.3.2.3 Remove FAQ: Use Case 2.7

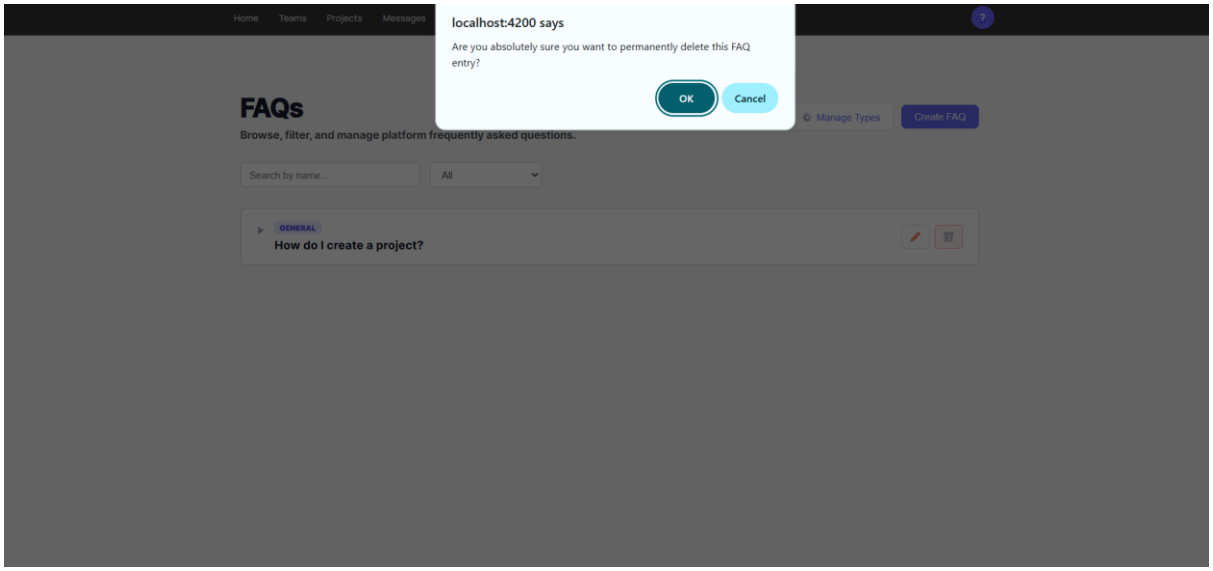


Figure 4: 2.7 Remove FAQ Screen Design

2.3.2.4 Add FAQ Type: Use Case 2.8

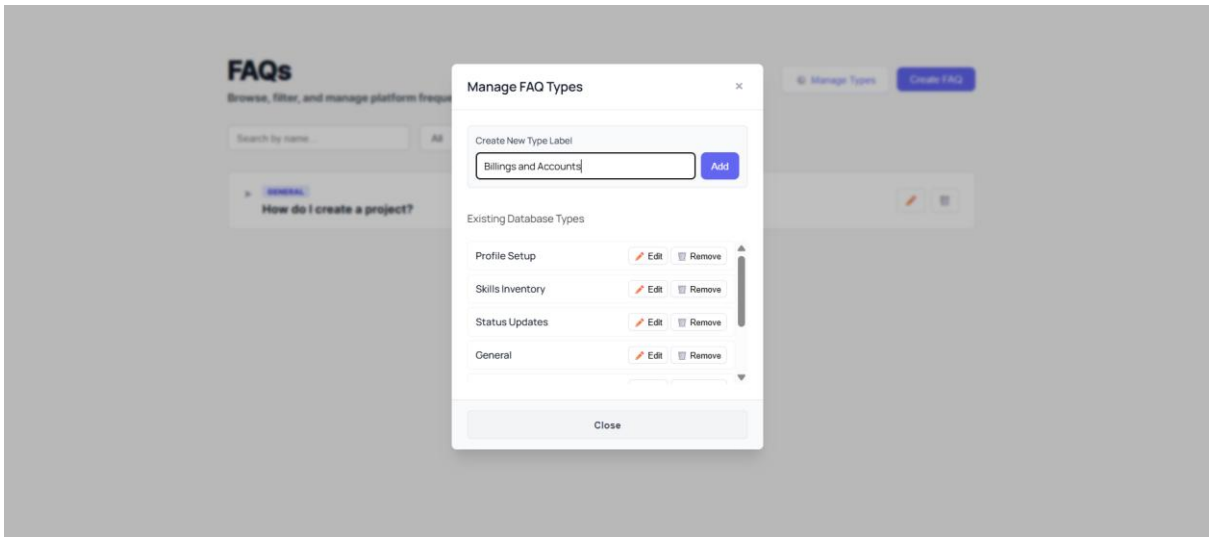


Figure 5: 2.8 Add FAQ Type Screen Design

2.3.2.5 Update FAQ Type: Use Case 2.9

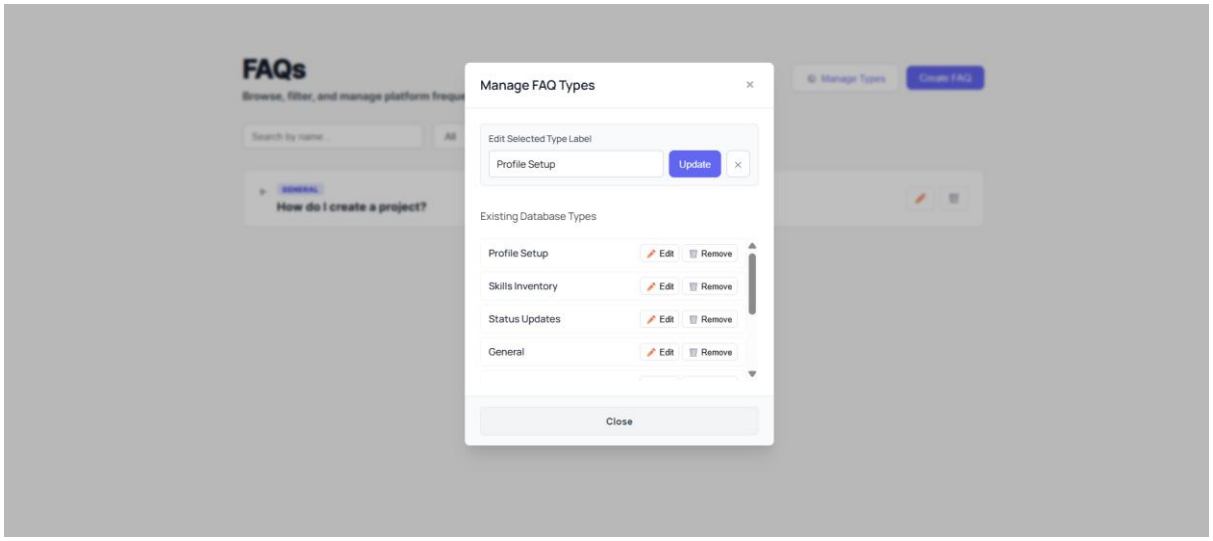


Figure 6: 2.9 Update FAQ Type Screen Design

2.3.2.6 Remove FAQ Type: Use Case 2.10

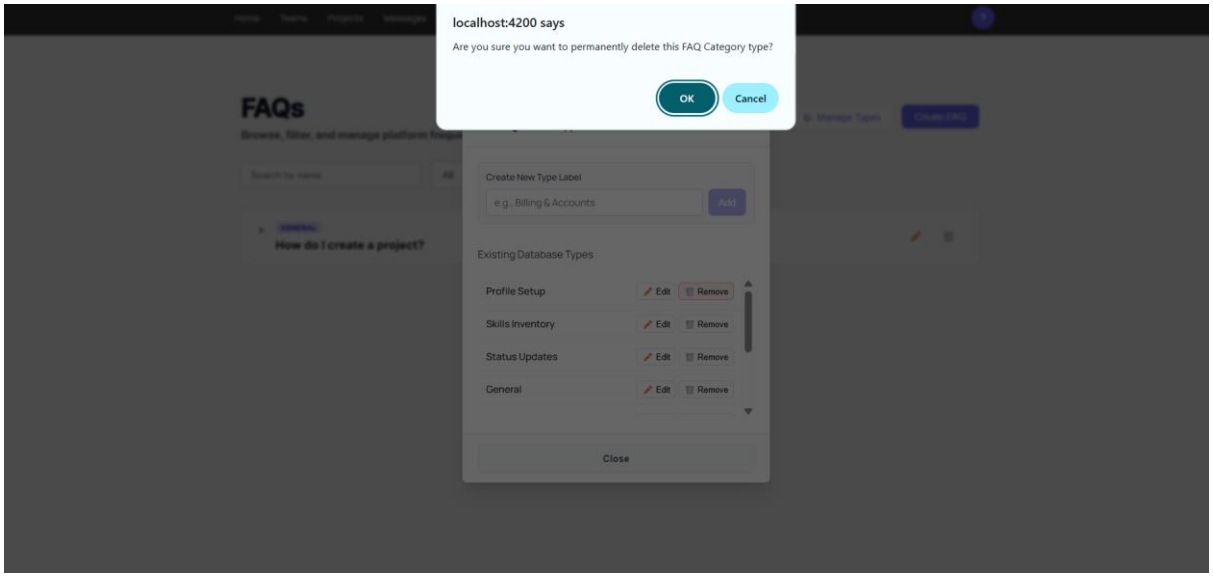


Figure 7: 2.10 Remove FAQ Type Screen Design

2.3.2.7 Update Role Permission: Use Case 2.16

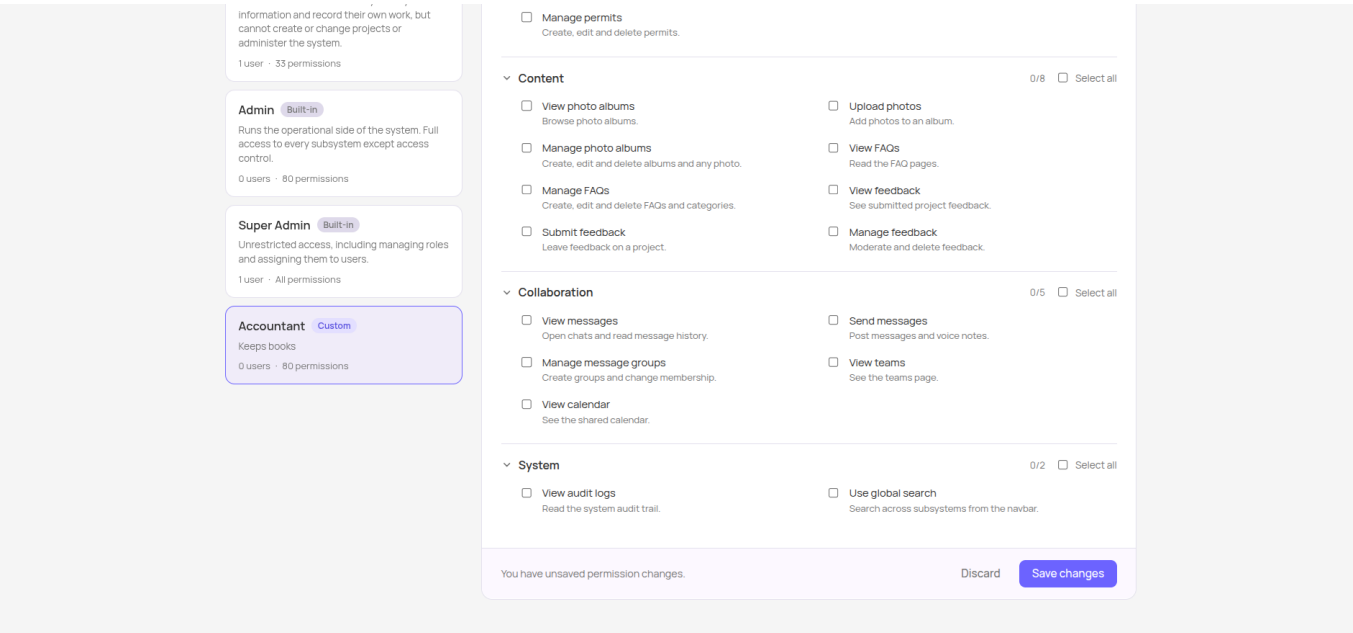


Figure 8: Update Role Permission Screen Design

2.3.2.8 View Role Permission: Use Case 2.17

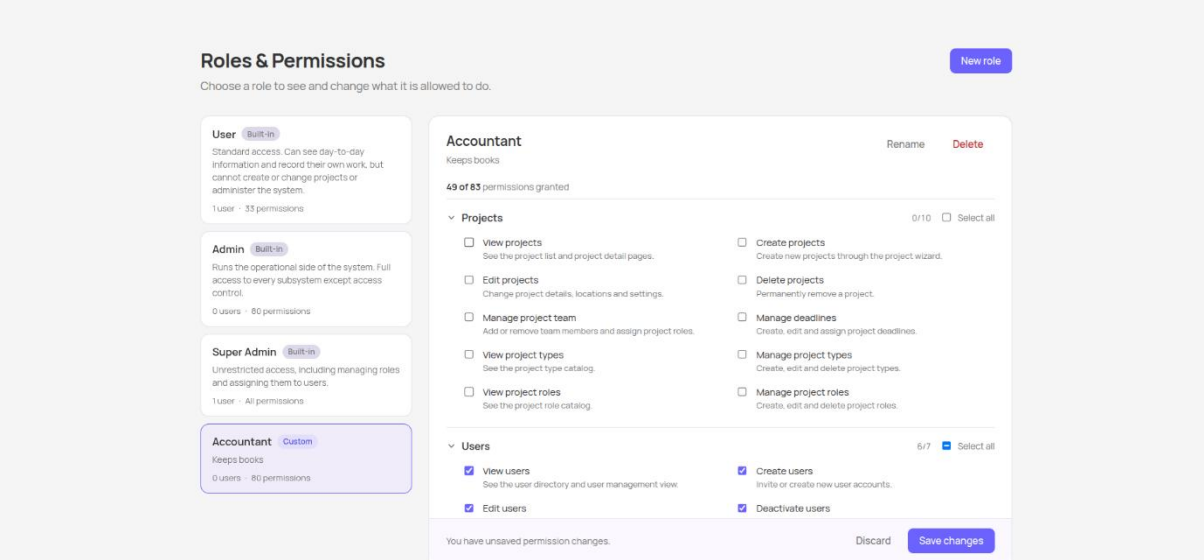


Figure 9: View Role Permission Screen Design

2.3.3 Inventory Subsystem

2.3.3.1 Create Procurement: Use Case 3.1

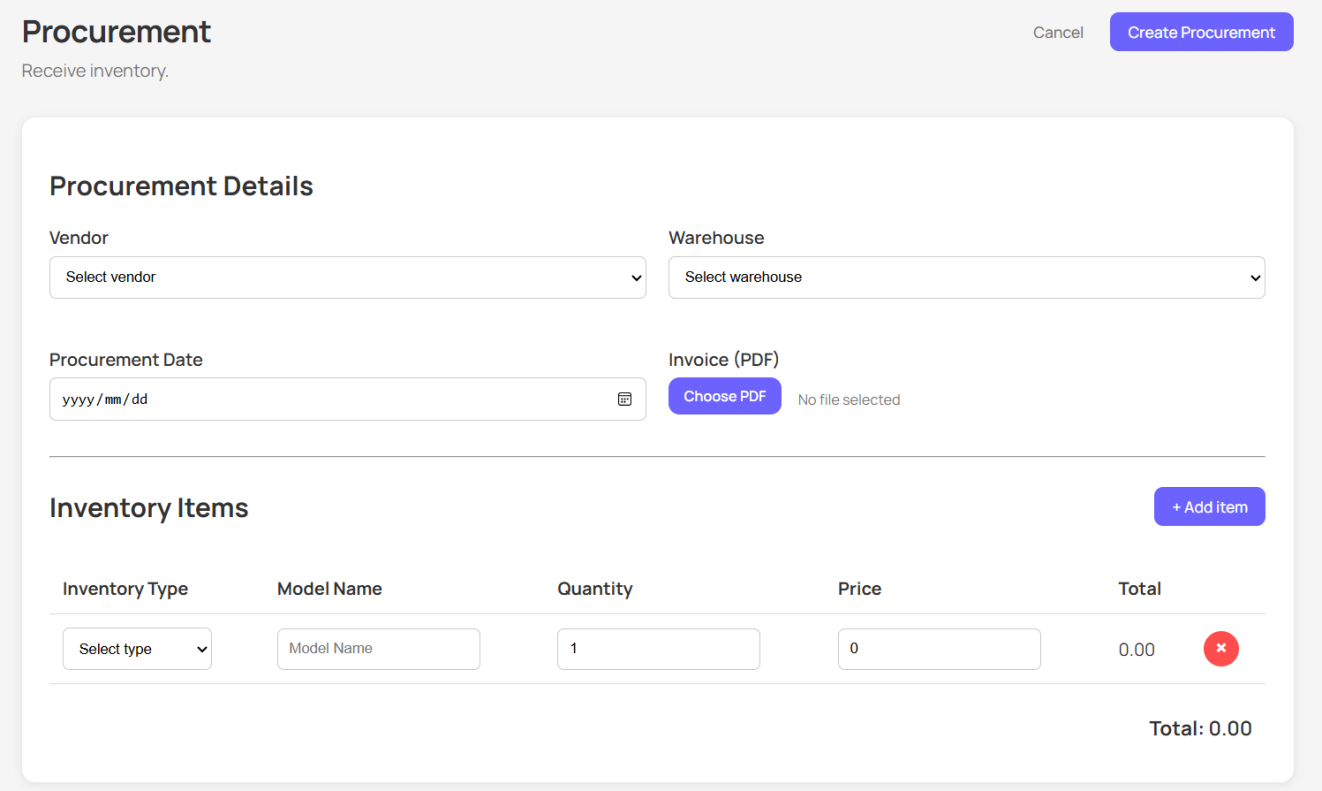


Figure 10: 3.1 Create Procurement Screen Design

2.3.3.2 View Procurement: Use Case 3.2

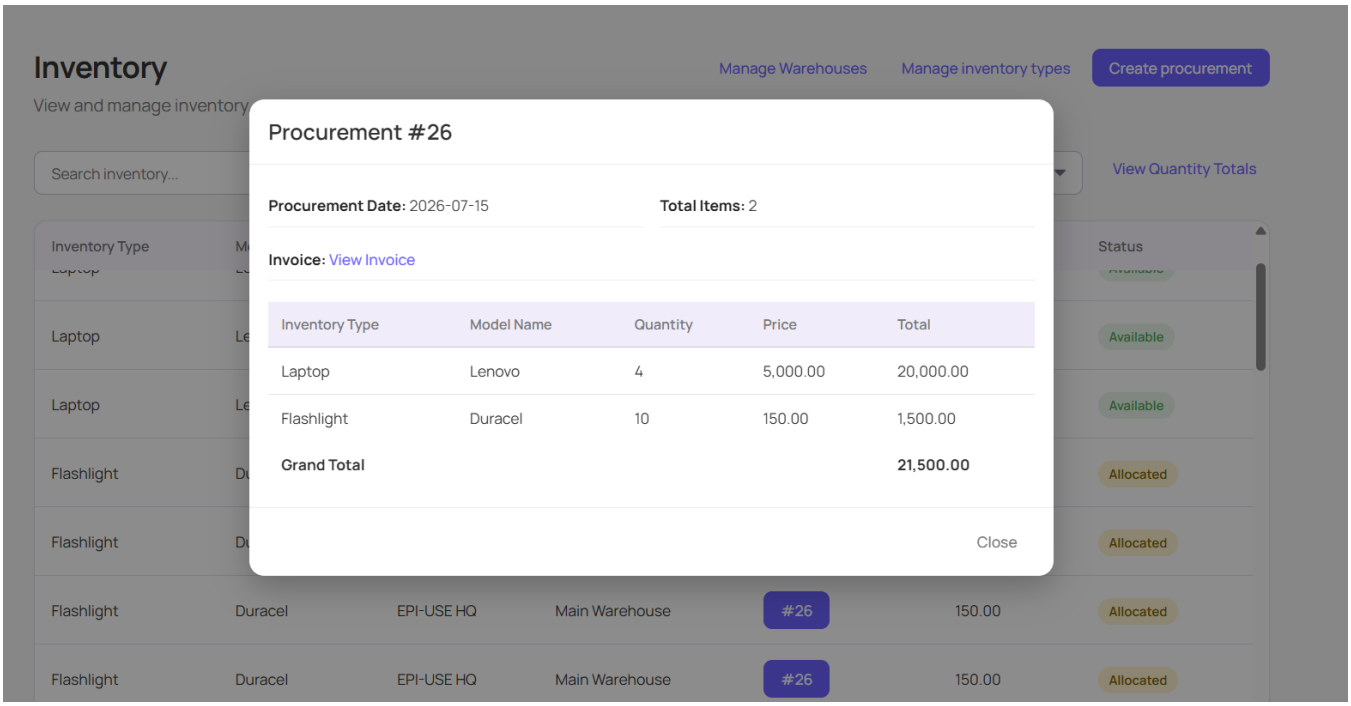


Figure 11: 3.2 View Procurement Screen Design

2.3.3.3 View Inventory: Use Case 3.3

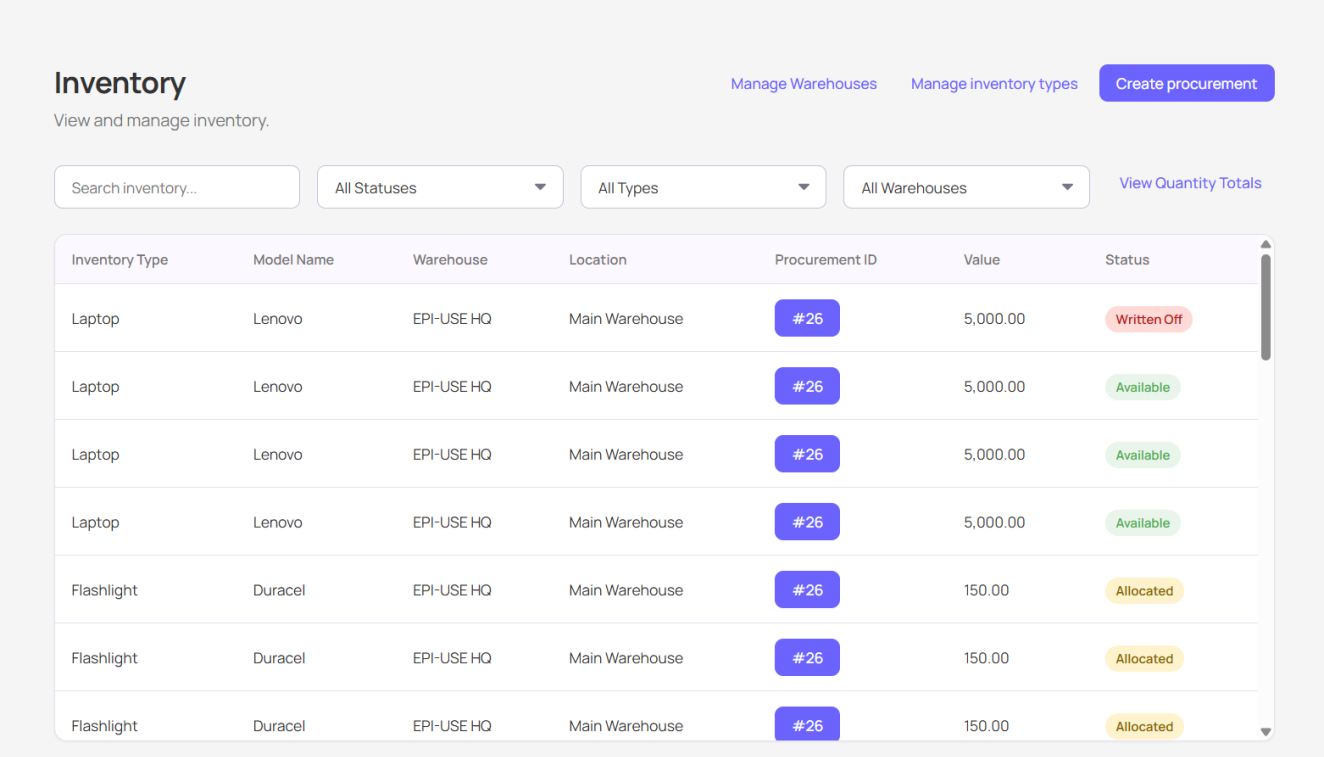


Figure 12: 3.3 View Inventory Screen Design – Inventory Screen

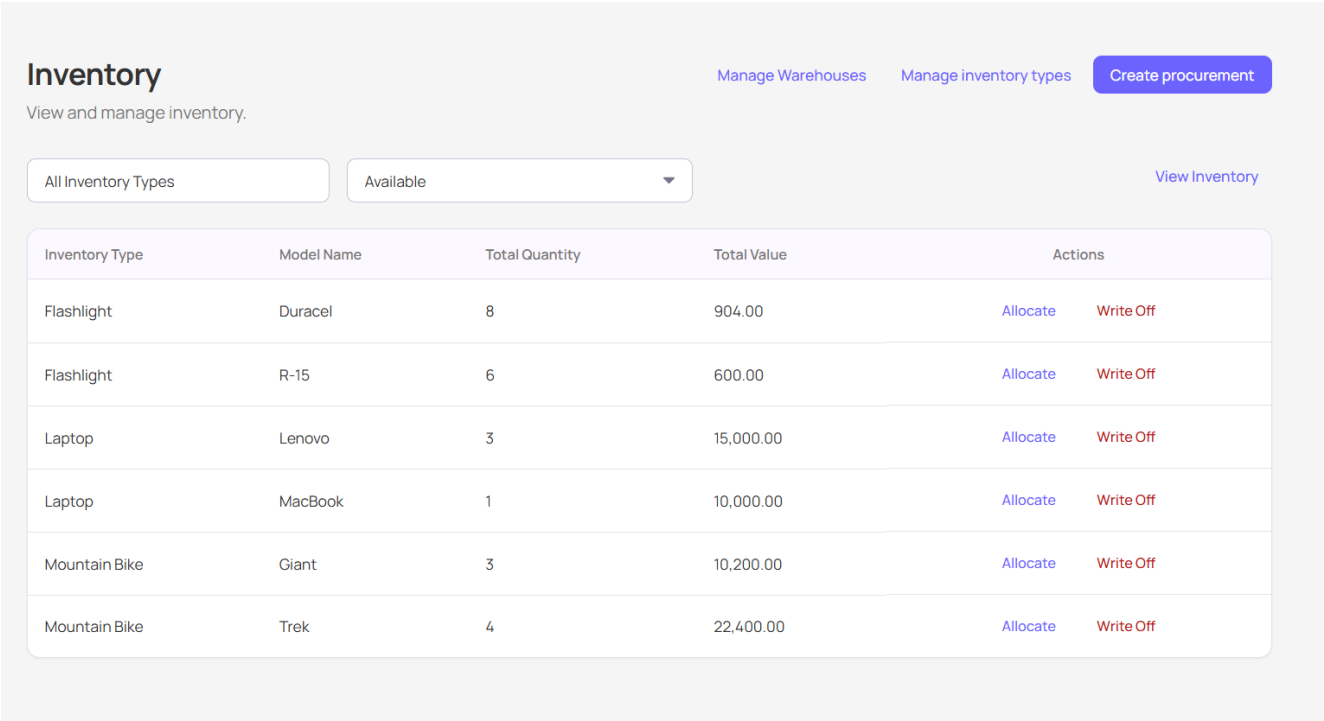


Figure 13: 3.3 View Inventory Screen Design – Quantity Totals Screen

2.3.3.4 Write-off Inventory: Use Case 3.4

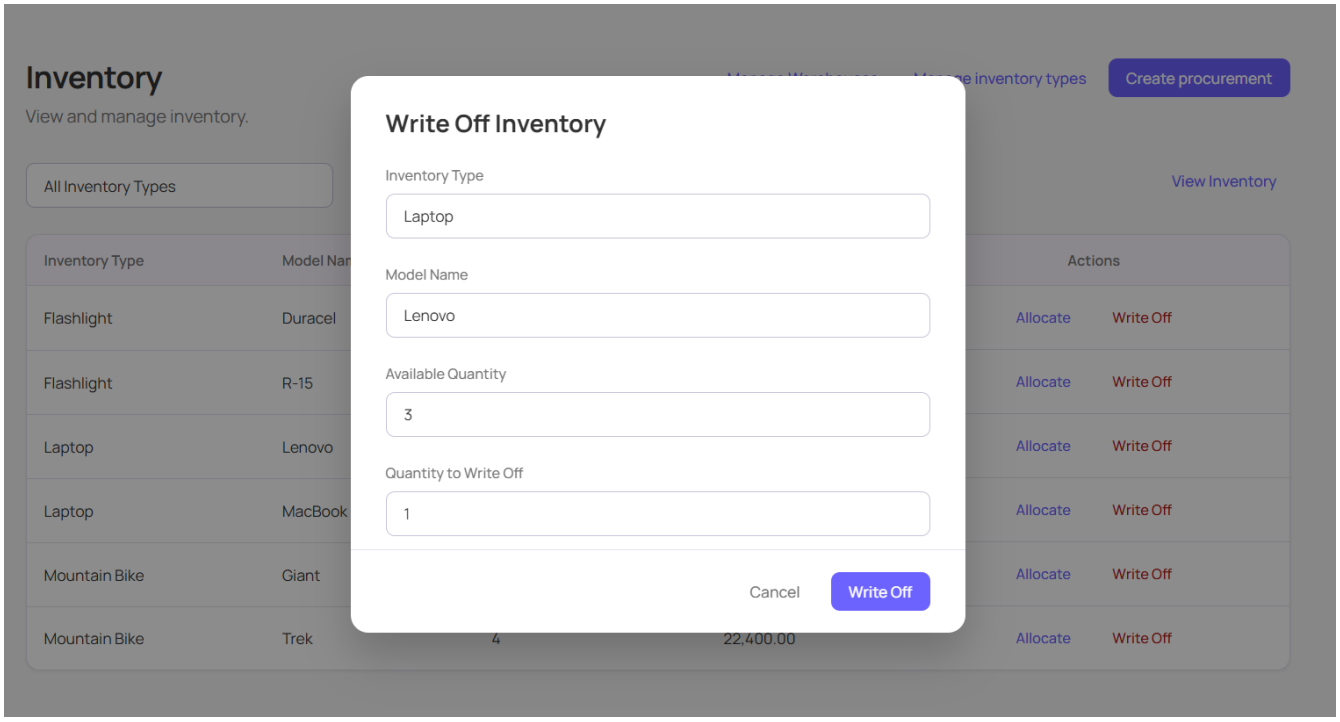


Figure 14: 3.4 Write-Off Inventory Screen Design

2.3.3.5 Allocate Inventory: Use Case 3.5

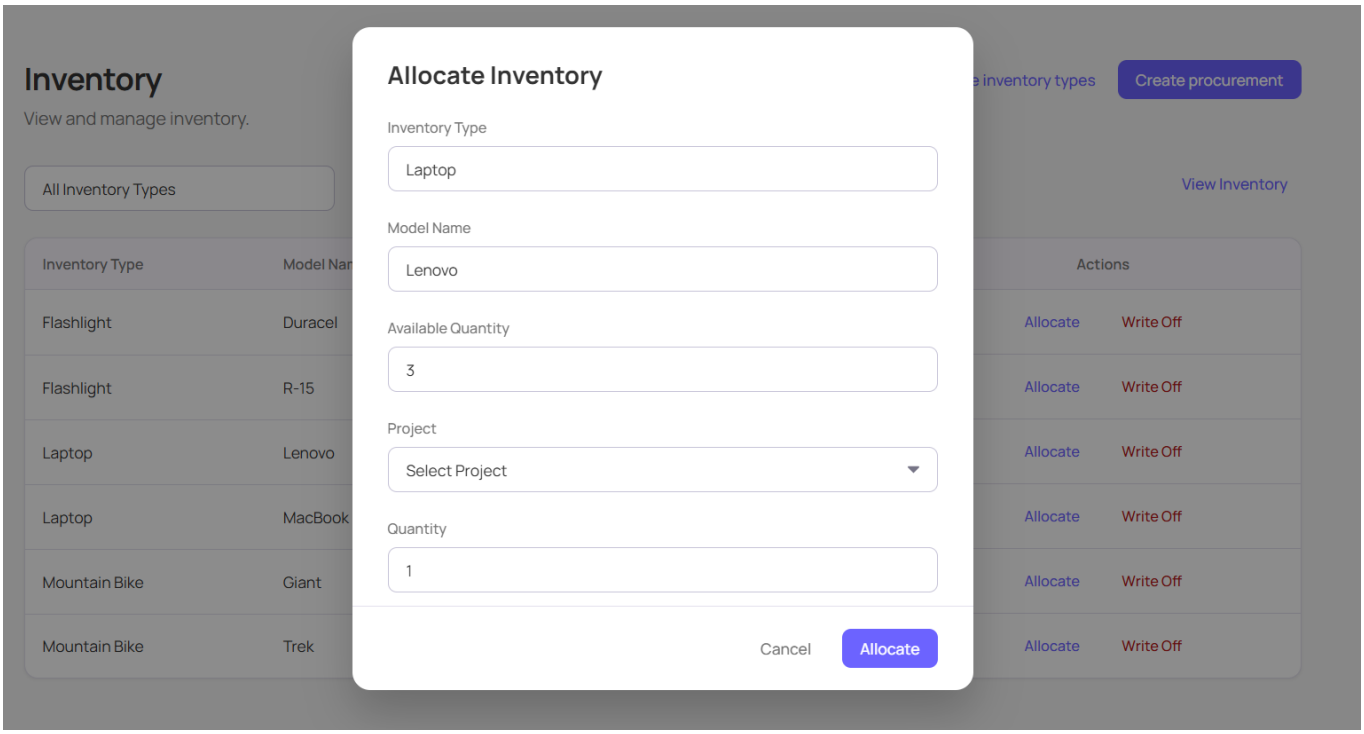


Figure 15: 3.5 Allocate Inventory Screen Design

2.3.3.6 Create Inventory Item Type: Use Case 3.6

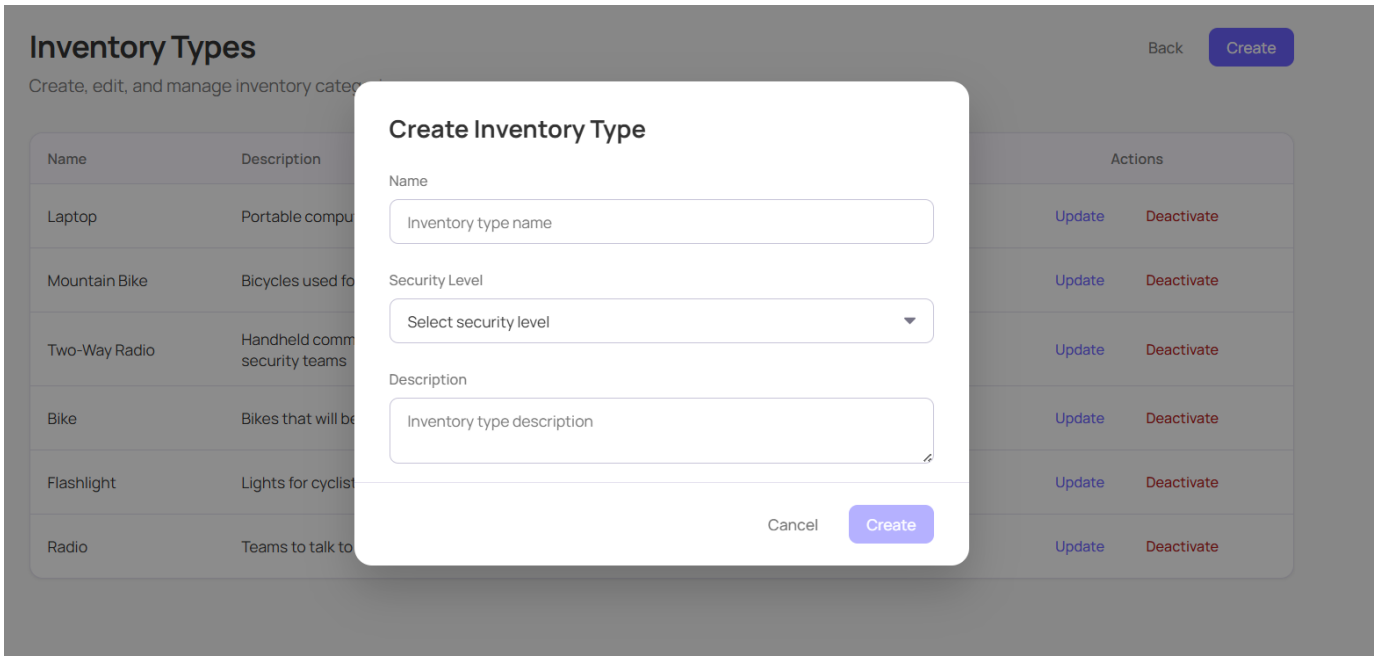


Figure 16: 3.6 Create Inventory Item Type Screen Design

2.3.3.7 View Inventory Item Type: Use Case 3.7

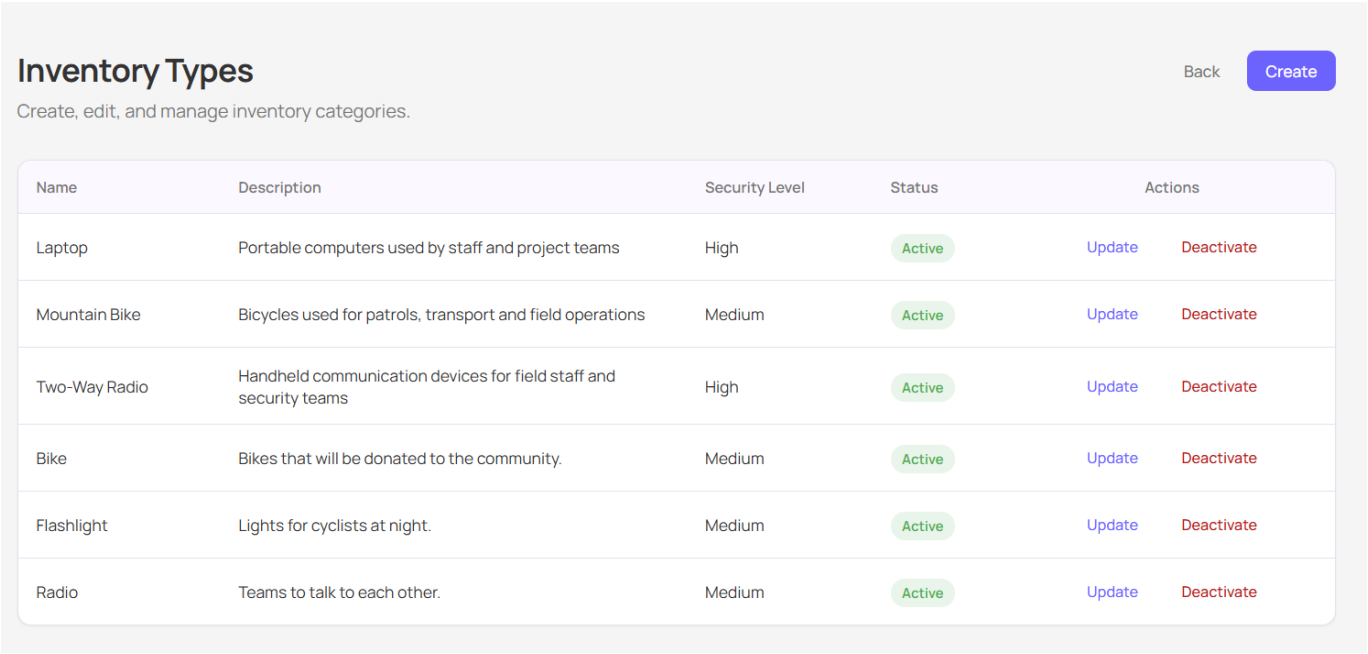


Figure 17: 3.7 View Inventory Item Types Screen Design

2.3.3.8 Update Inventory Item Type: Use Case 3.8

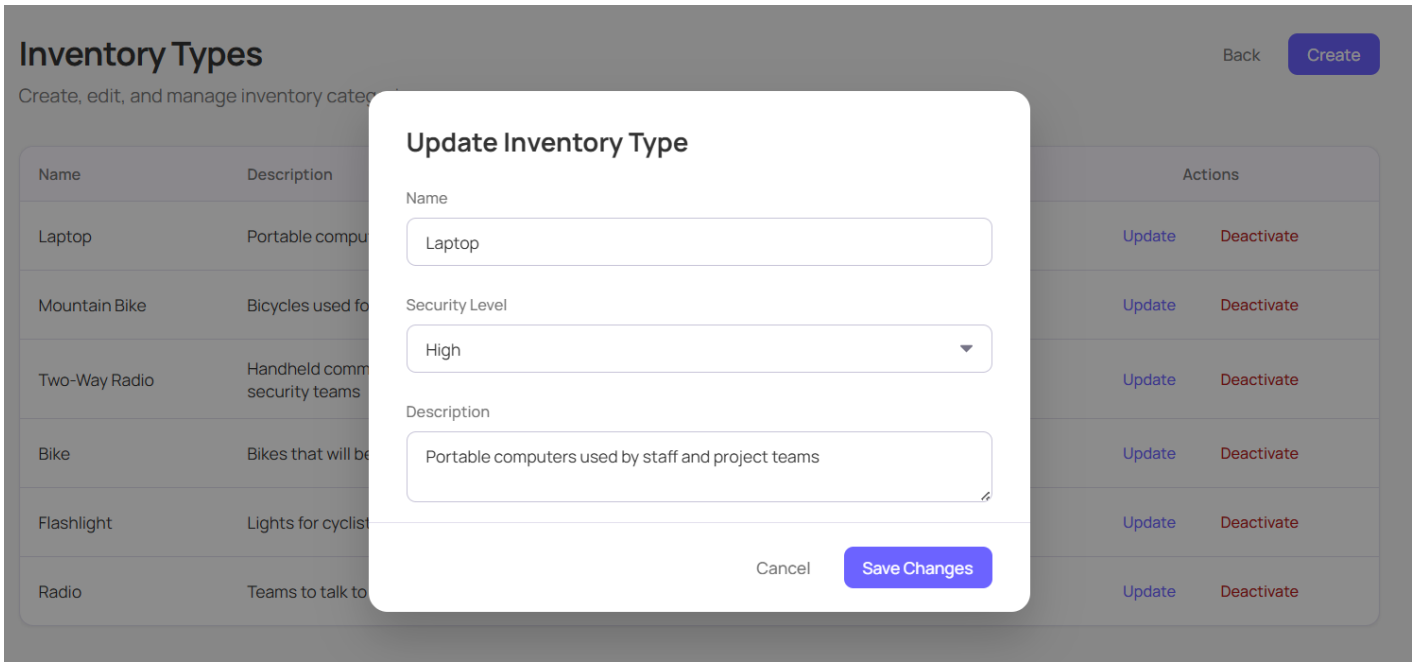


Figure 18: 3.8 Update Inventory Item Type Screen Design

2.3.3.9 Deactivate Inventory Item Type: Use Case 3.9

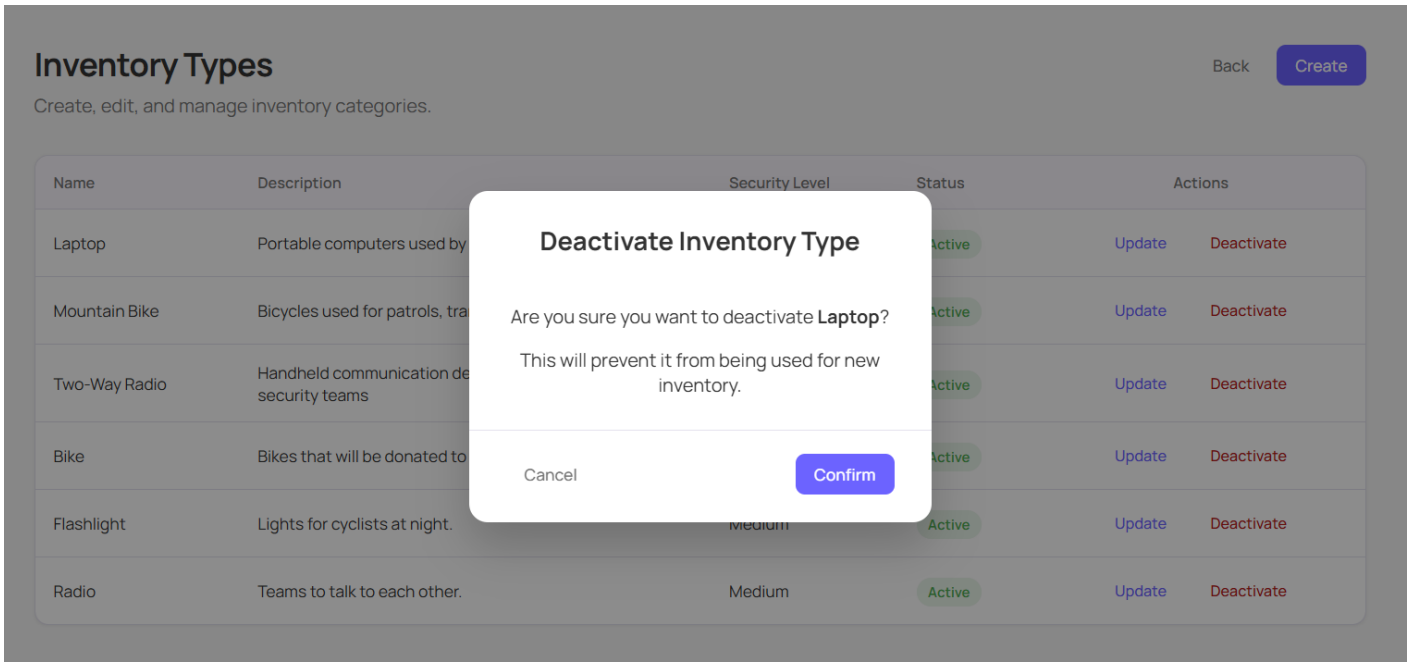


Figure 19: 3.9 Deactivate Inventory Item Type Screen Design

2.3.3.10 Reactivate Inventory Item Type: Use Case 3.10

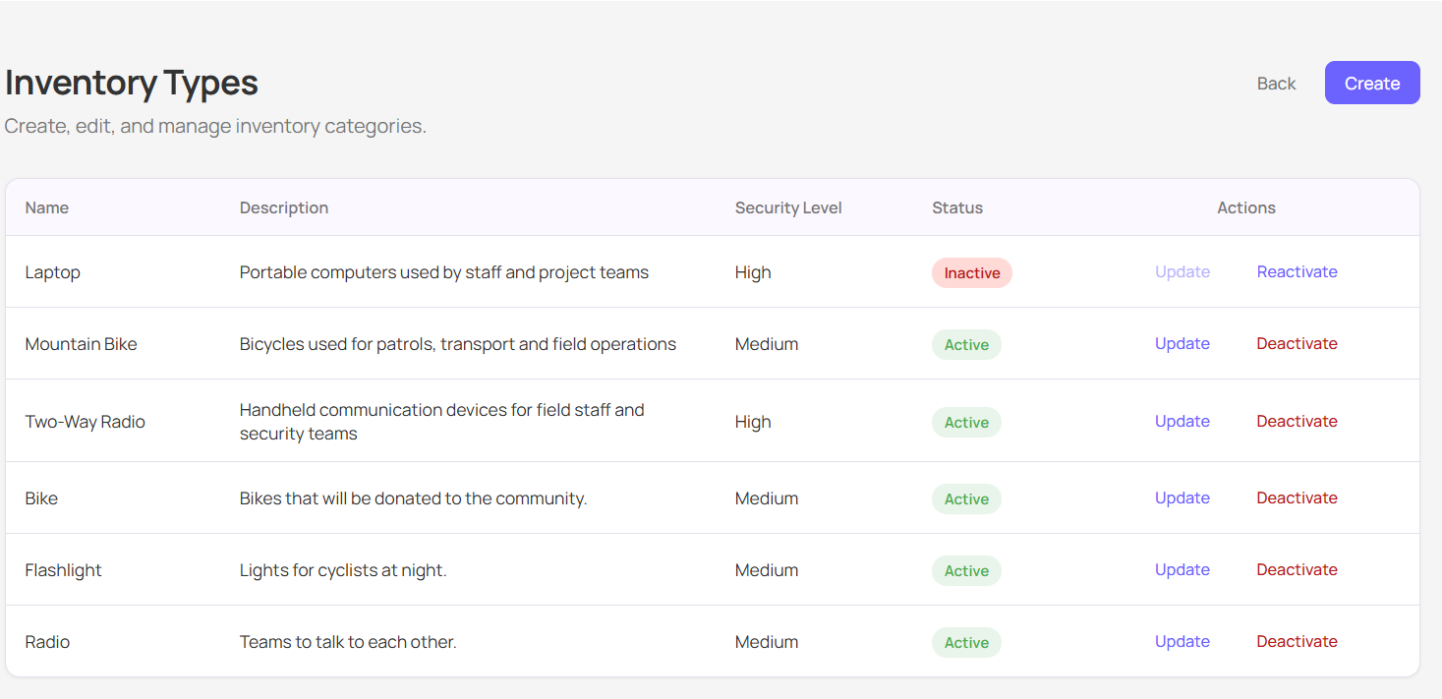


Figure 20: 3.10 Reactivate Inventory Item Type Screen Design

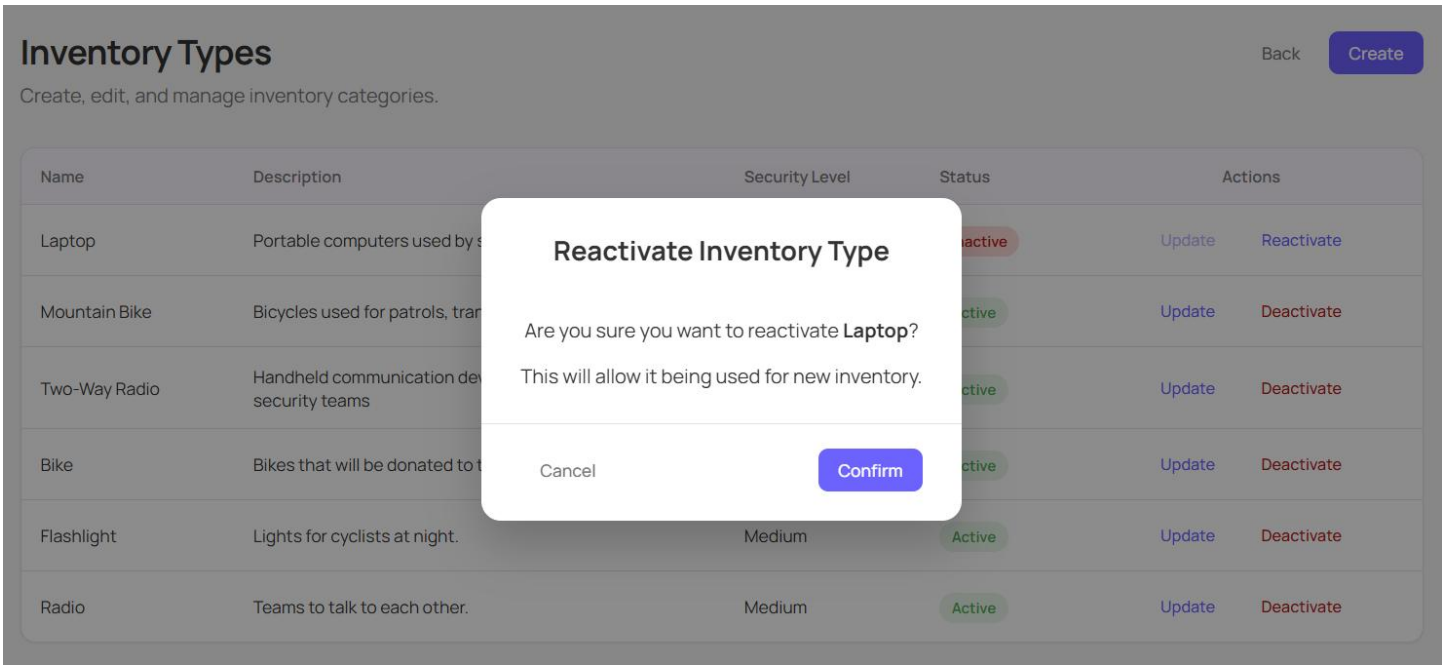


Figure 21: 3.10 Reactivate Inventory Item Type Confirmation Screen Design

2.3.4 Warehouse Subsystem

2.3.4.1 Create Warehouse: Use Case 5.1

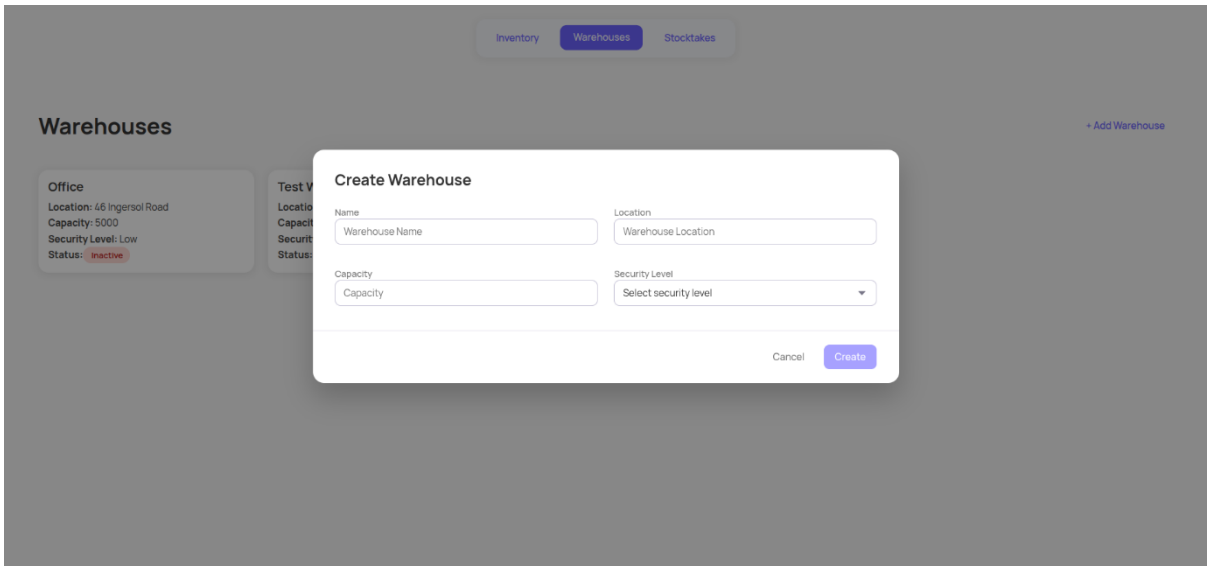


Figure 22: 5.1 Create Warehouse Screen Design

2.3.4.2 View Warehouse: Use Case 5.2

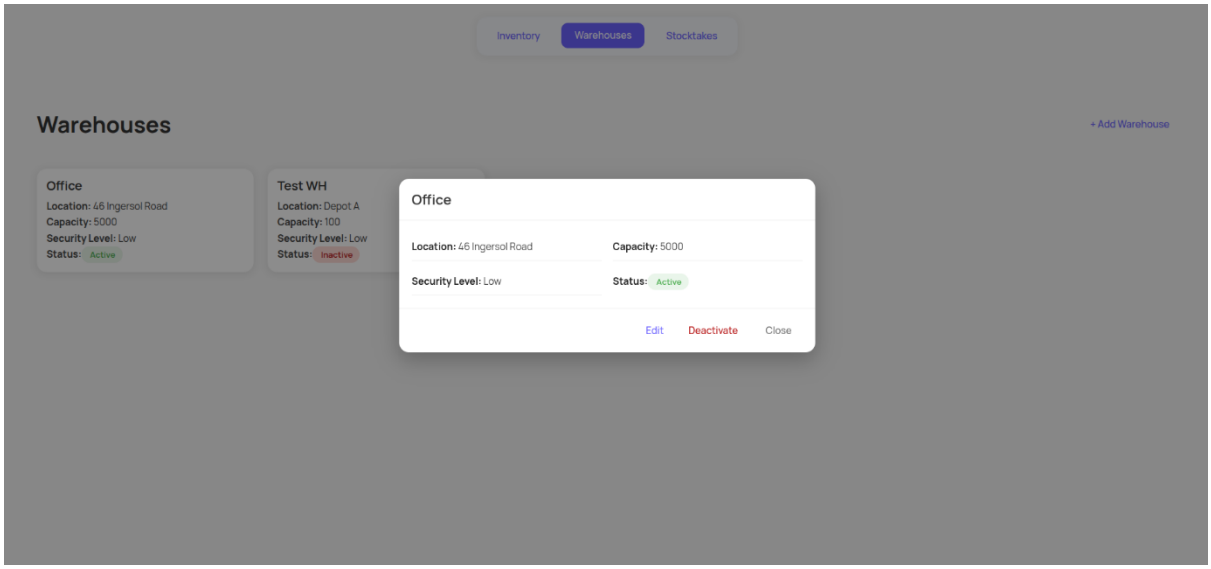


Figure 23: 5.2 View Warehouse Screen Design

2.3.4.3 Update Warehouse: Use Case 5.3

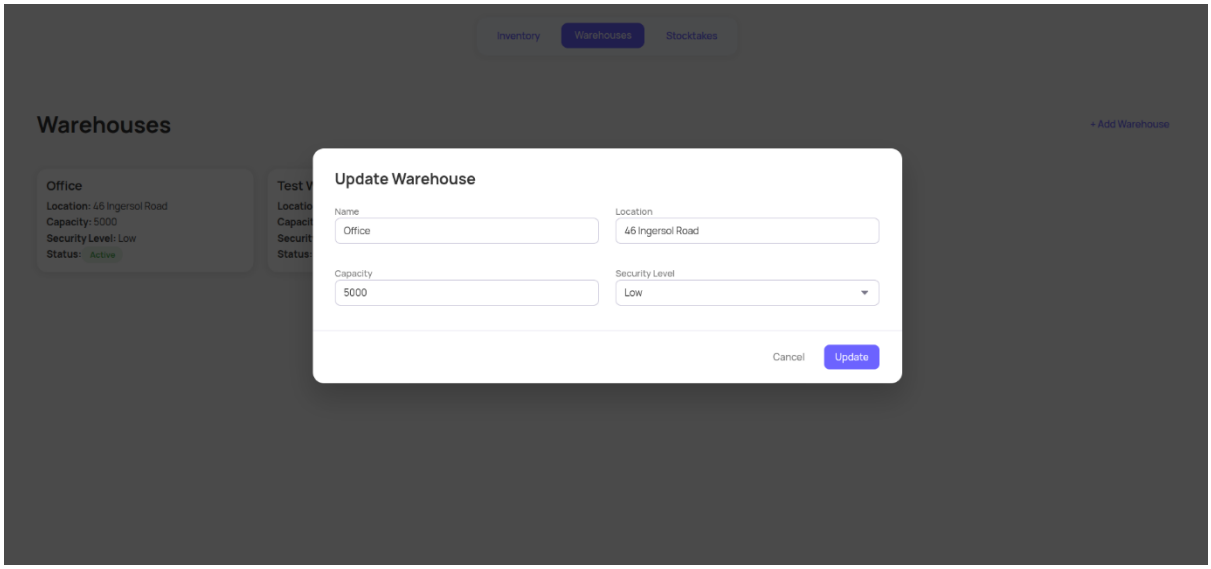


Figure 24: 5.3 Update Warehouse Screen Design

2.3.4.4 Reactivate Warehouse: Use Case 5.4

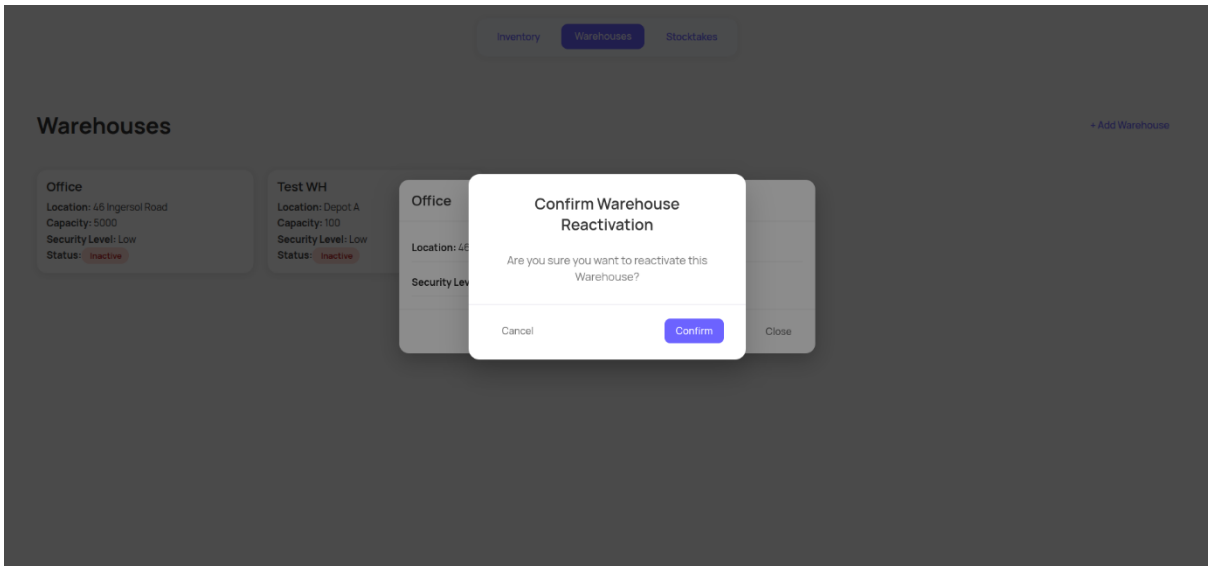


Figure 25: 5.4 Reactivate Warehouse Screen Design

2.3.4.5 Deactivate Warehouse: Use Case 5.5

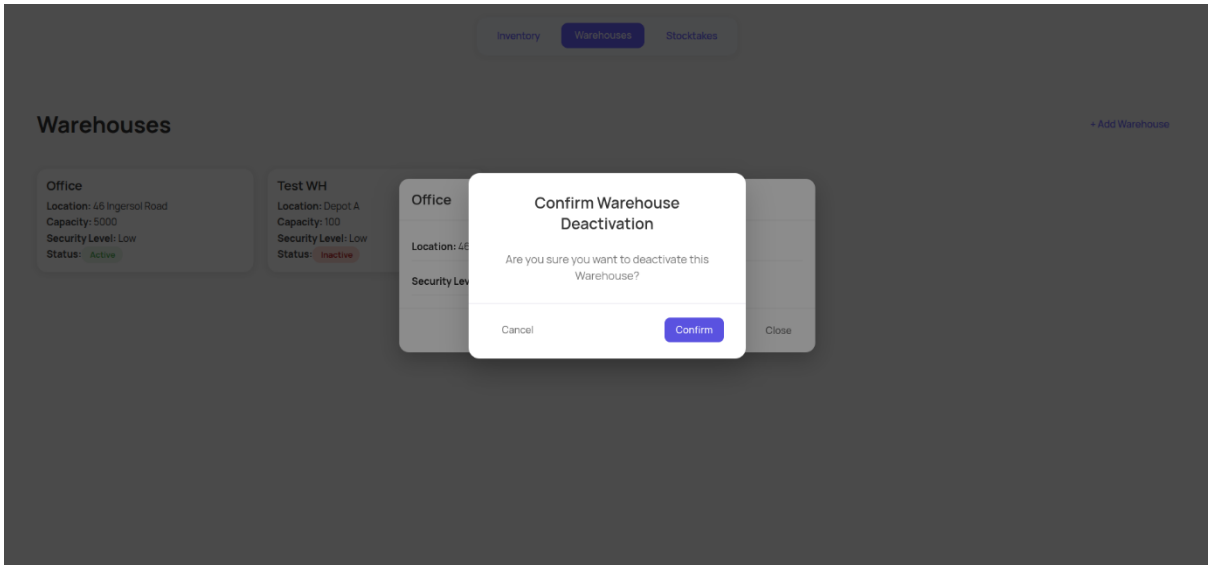


Figure 26: 5.5 Deactivate Warehouse Screen Design

2.3.4.6 Inventory Stock Take: Use Case 5.6

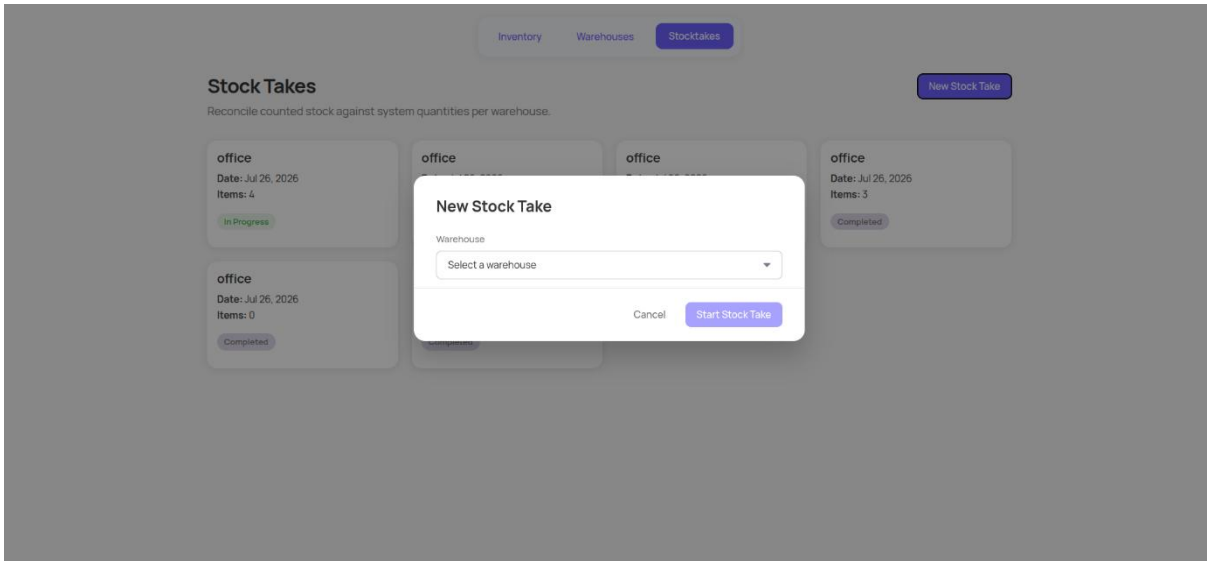


Figure 27: 5.6 Inventory Stock Take Screen Design 1

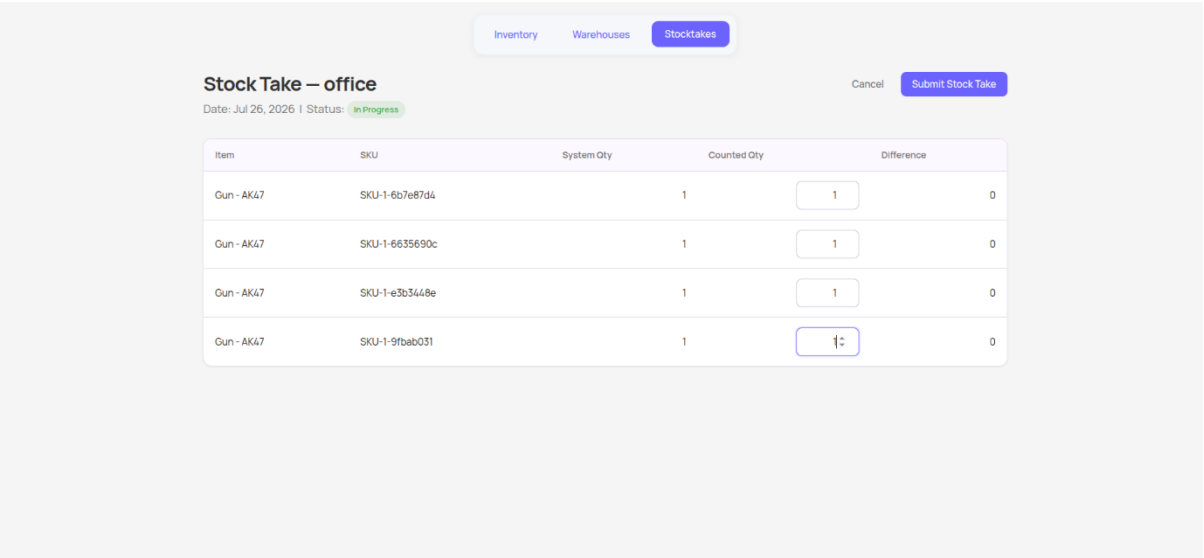


Figure 28: 5.6 Inventory Stock Take Screen Design 2

2.3.5 Vendor Subsystem

2.3.5.1 Create Vendor: Use Case 6.1

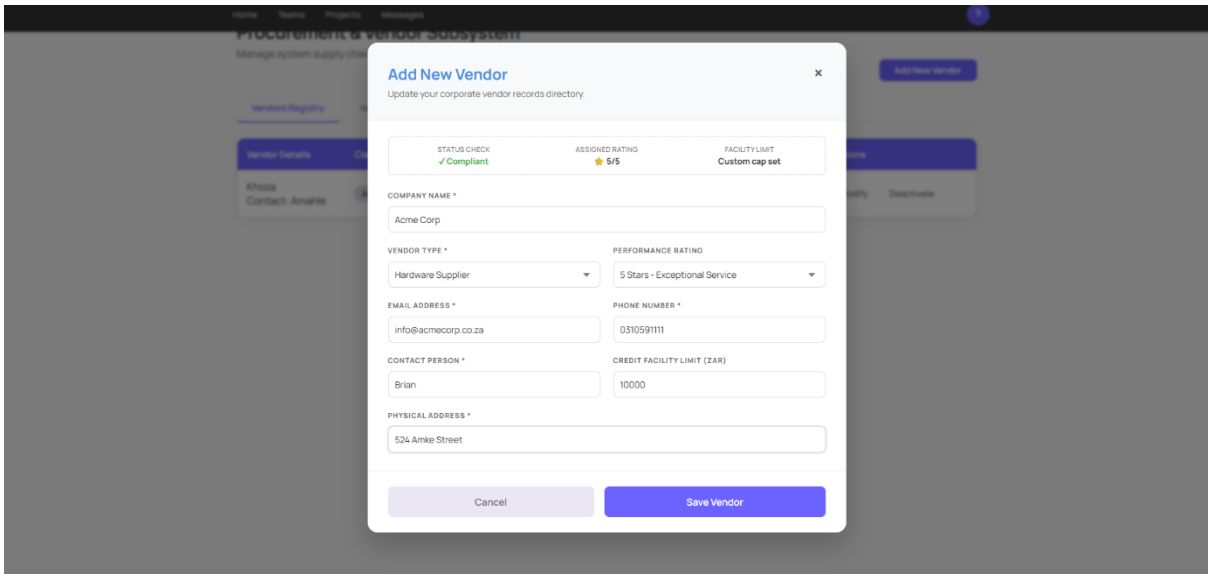


Figure 29: 6.1 Create Vendor Screen Design

2.3.5.2 View Vendor: Use Case 6.2

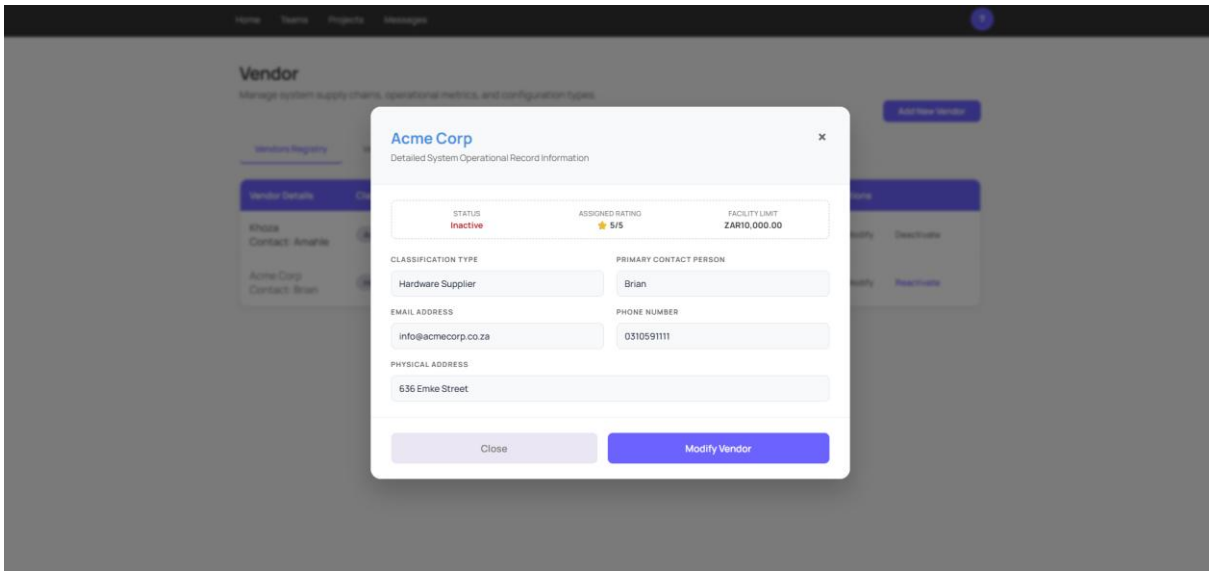


Figure 30: 6.2 View Vendor Screen Design

2.3.5.3 View Vendors: Use Case 6.3

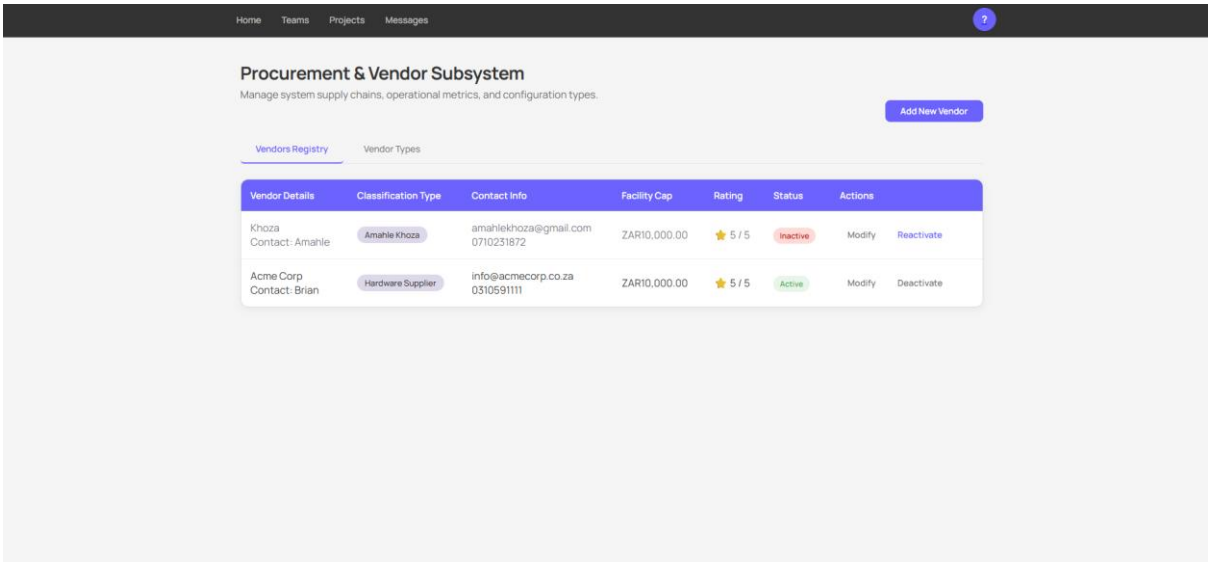


Figure 31: 6.3 View Vendors Screen Design

2.3.5.4 Edit Vendor: Use Case 6.4

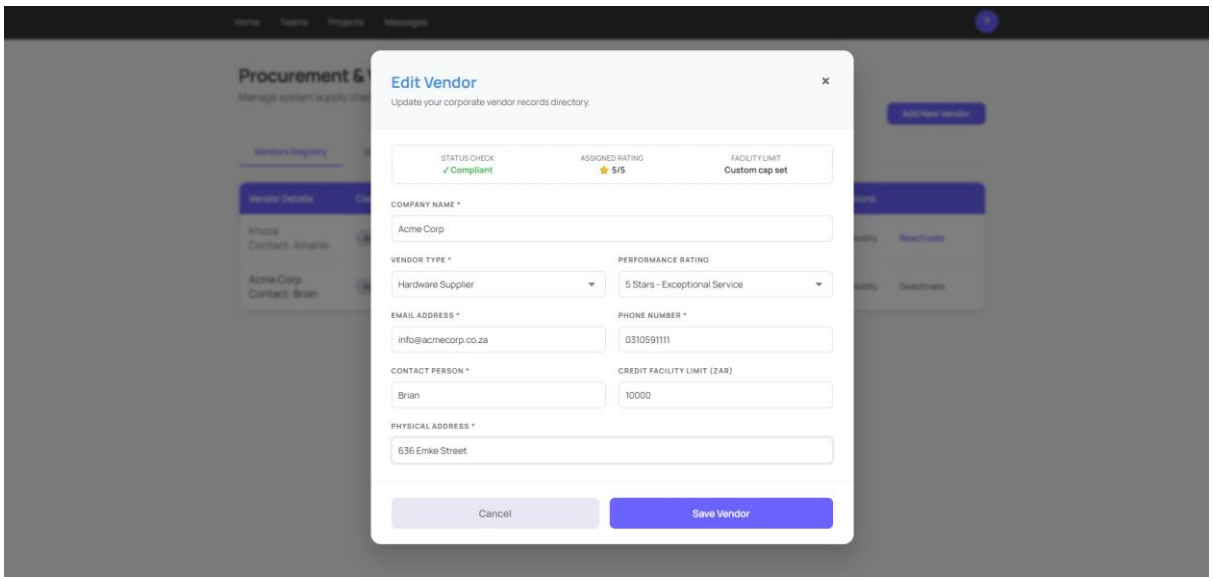


Figure 32: 6.4 Edit Vendor Screen Design

2.3.5.5 Reactivate Vendor: Use Case 6.5

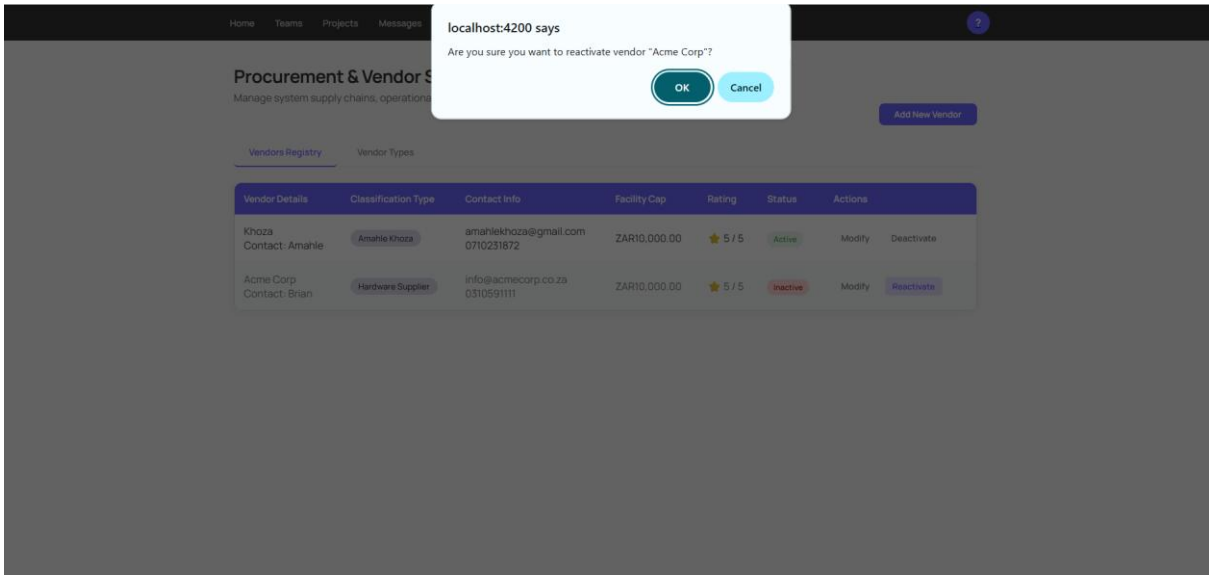


Figure 33: 6.5 Reactivate Vendor Screen Design

2.3.5.6 Deactivate Vendor: Use Case 6.6

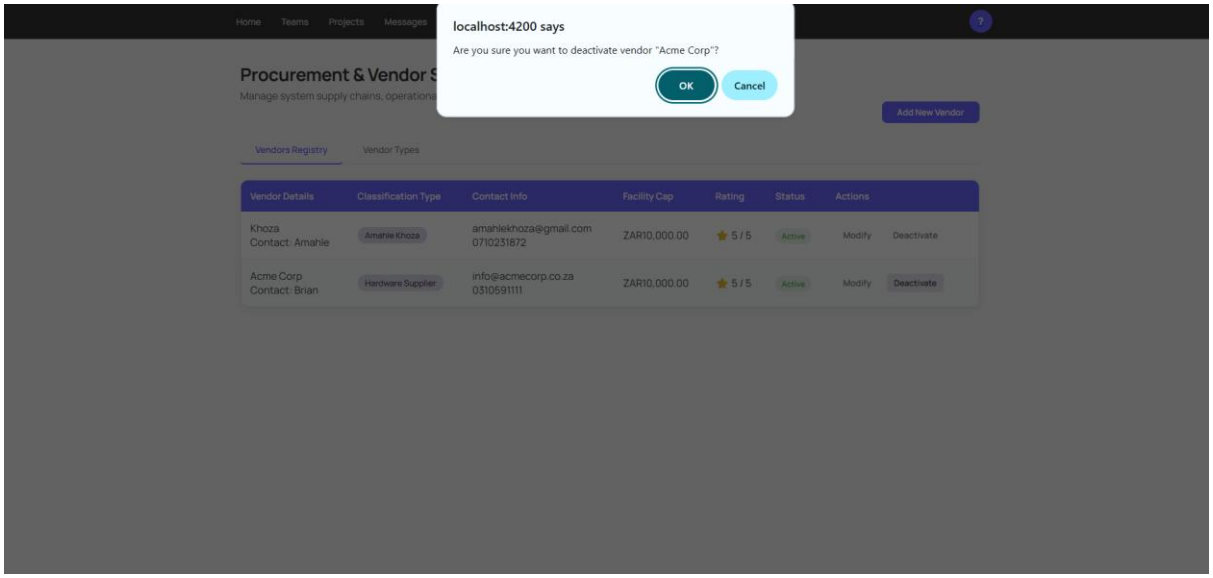


Figure 34: 6.6 Deactivate Vendor Screen Design

2.3.5.7 Create Vendor Type: Use Case 6.7

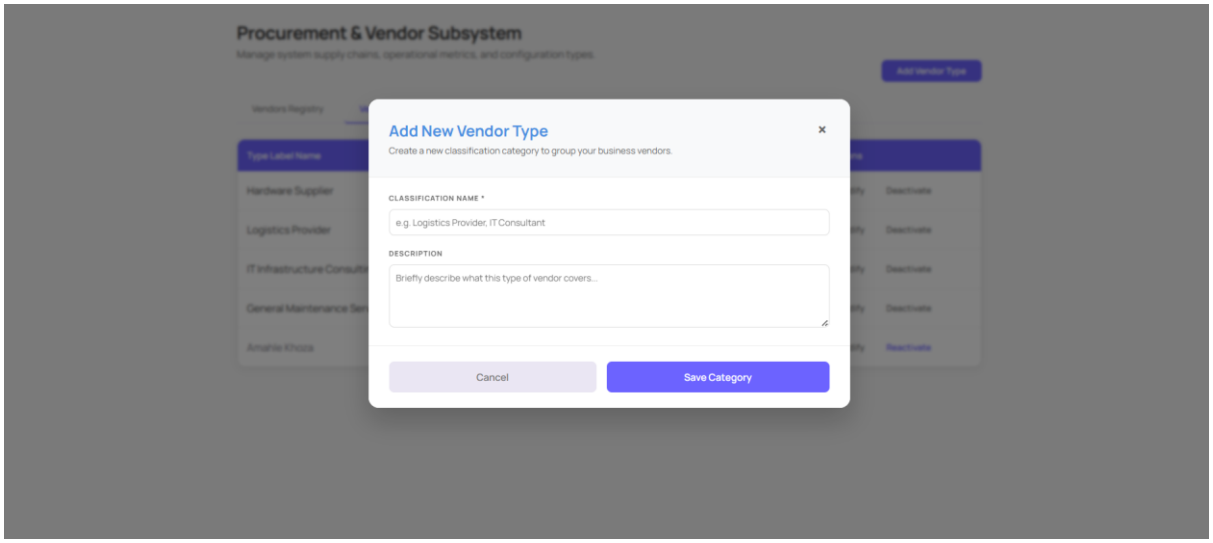


Figure 35: 6.7 Create Vendor Type Screen Design

2.3.5.8 Update Vendor Type: Use Case 6.8

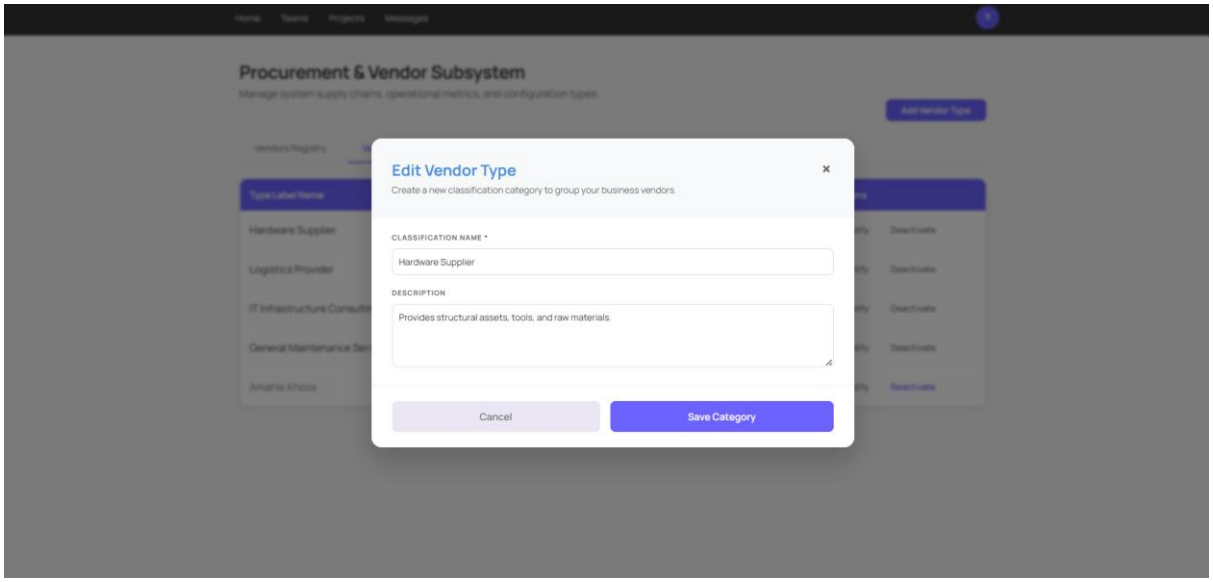


Figure 36: 6.8 Update Vendor Type Screen Design

2.3.5.9 View Vendor Types: Use Case 6.9

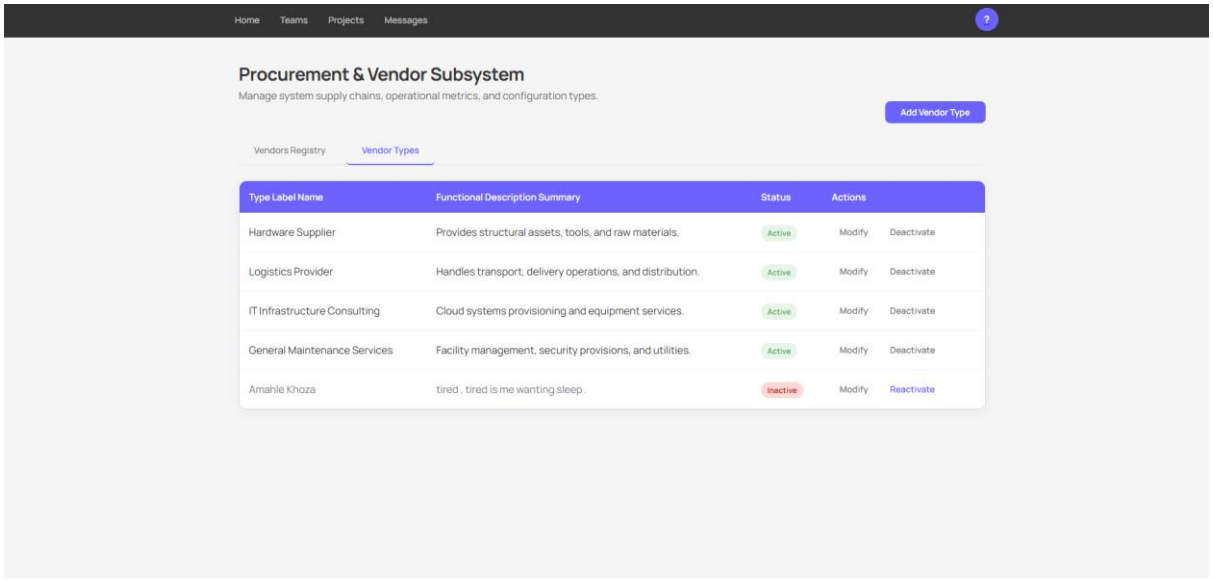


Figure 37: 6.9 View Vendor Types Screen Design

2.3.5.10 Reactivate Vendor Type: Use Case 6.10

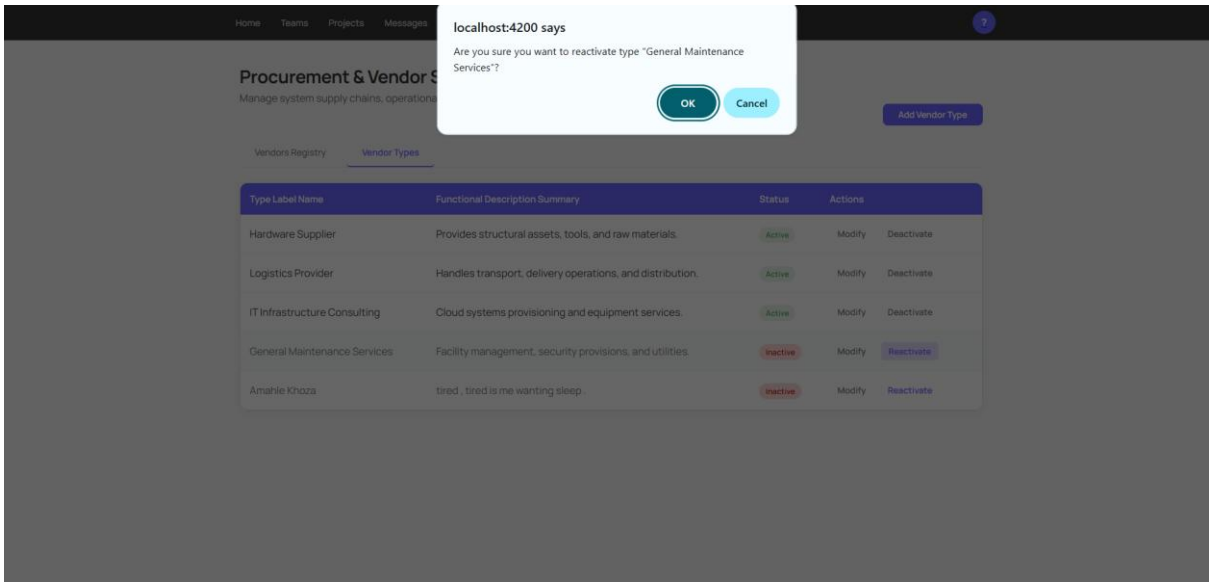


Figure 38: 6.10 Reactivate Vendor Type Screen Design

2.3.5.11 Deactivate Vendor Type: Use Case 6.11

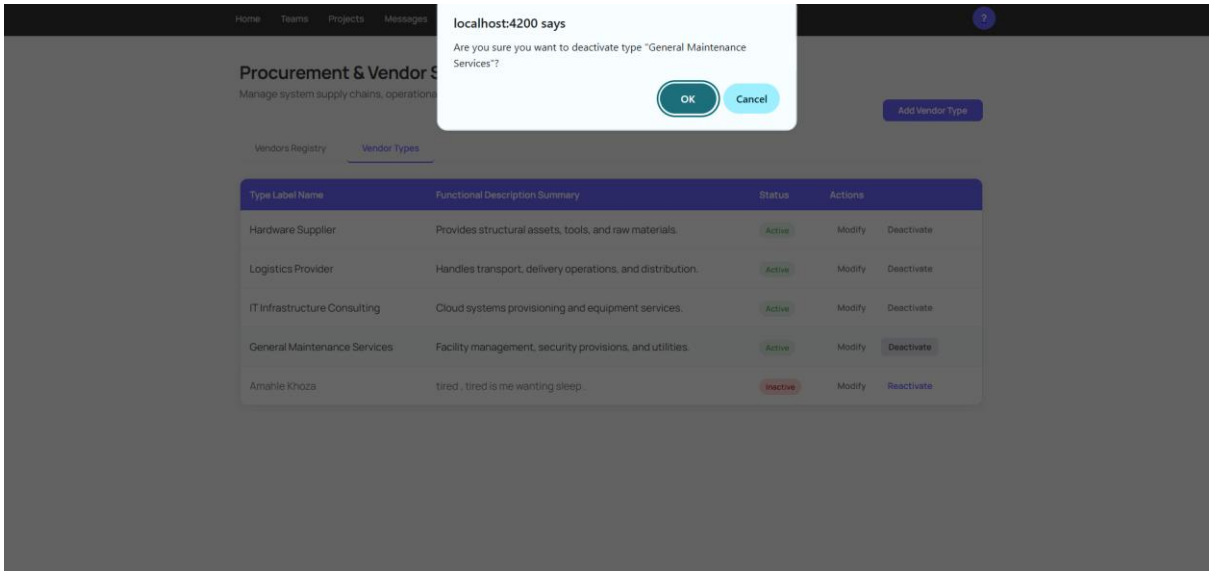


Figure 39: 6.11 Deactivate Vendor Type Screen Design

2.3.6 Project Management Subsystem

2.3.6.1 Capture Project Feedback: Use Case 7.14

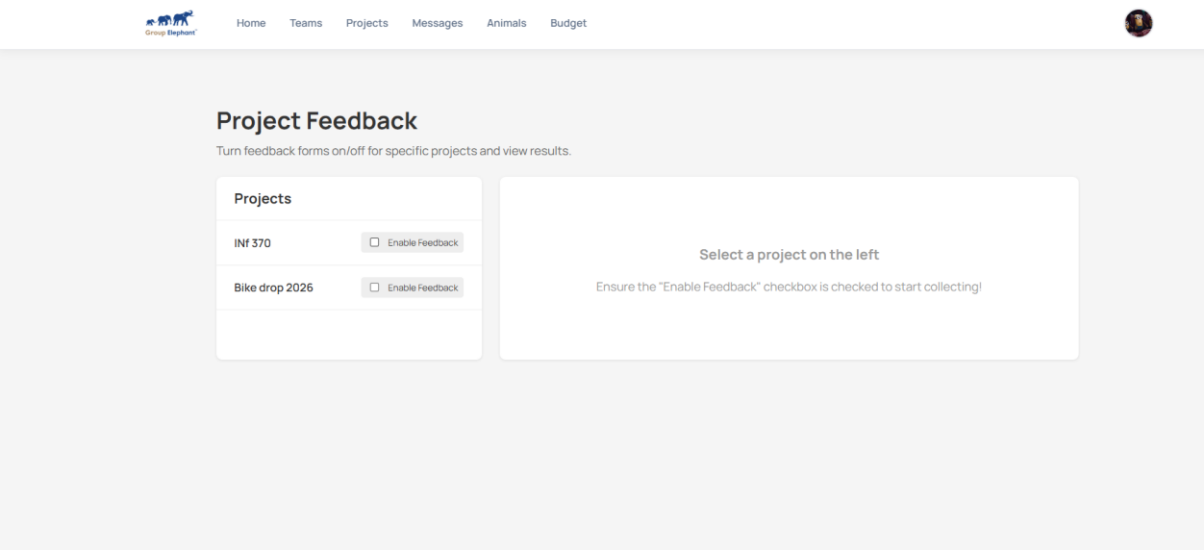


Figure 40: 7.14 Capture Project Feedback Screen Design

2.3.6.2 View Project Feedback: Use Case 7.15

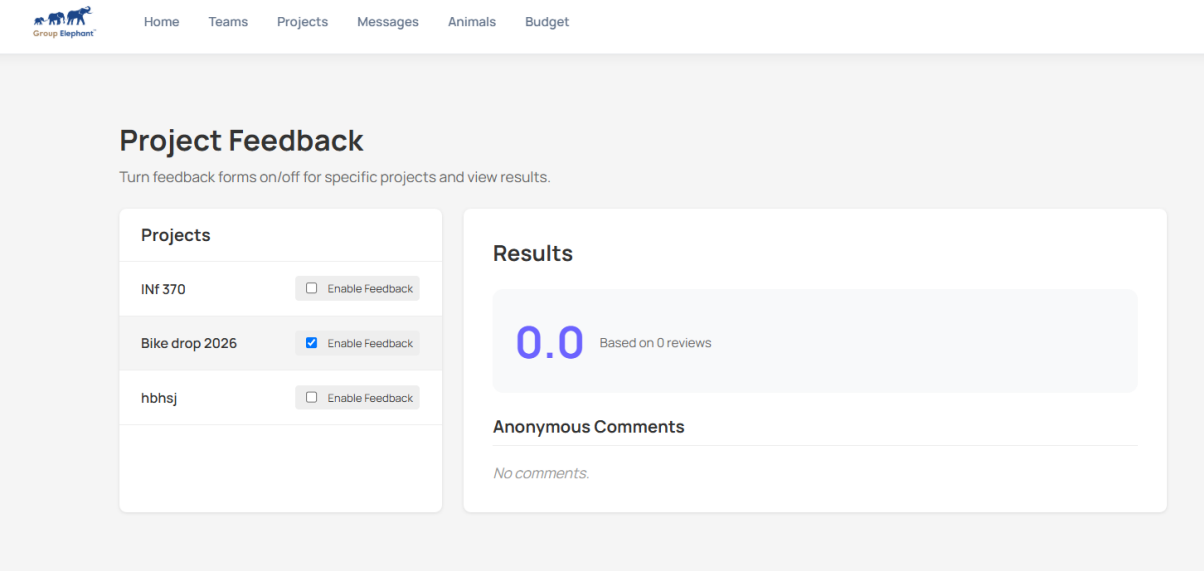


Figure 41: 7.15 View Project Feedback Screen Design

2.3.6.3 Create Impact Feedback Form: Use Case 7.16

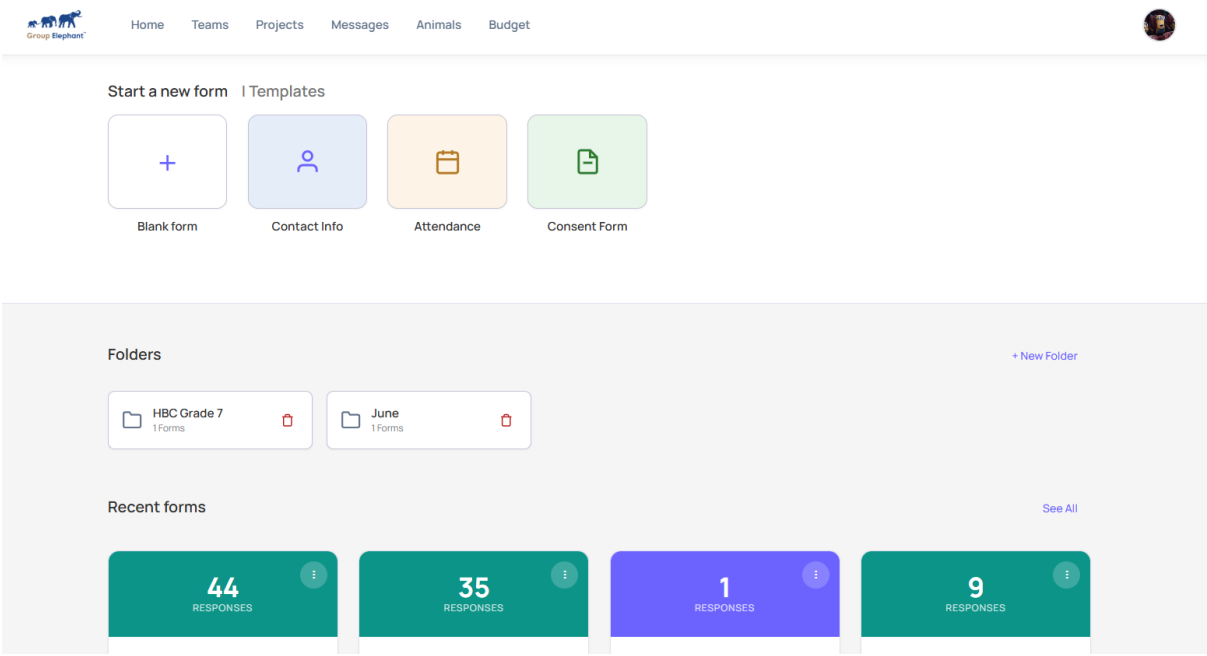


Figure 42: 7.16 Create Impact Feedback Form Screen Design

2.3.6.4 View Impact Feedback Form: Use Case 7.17

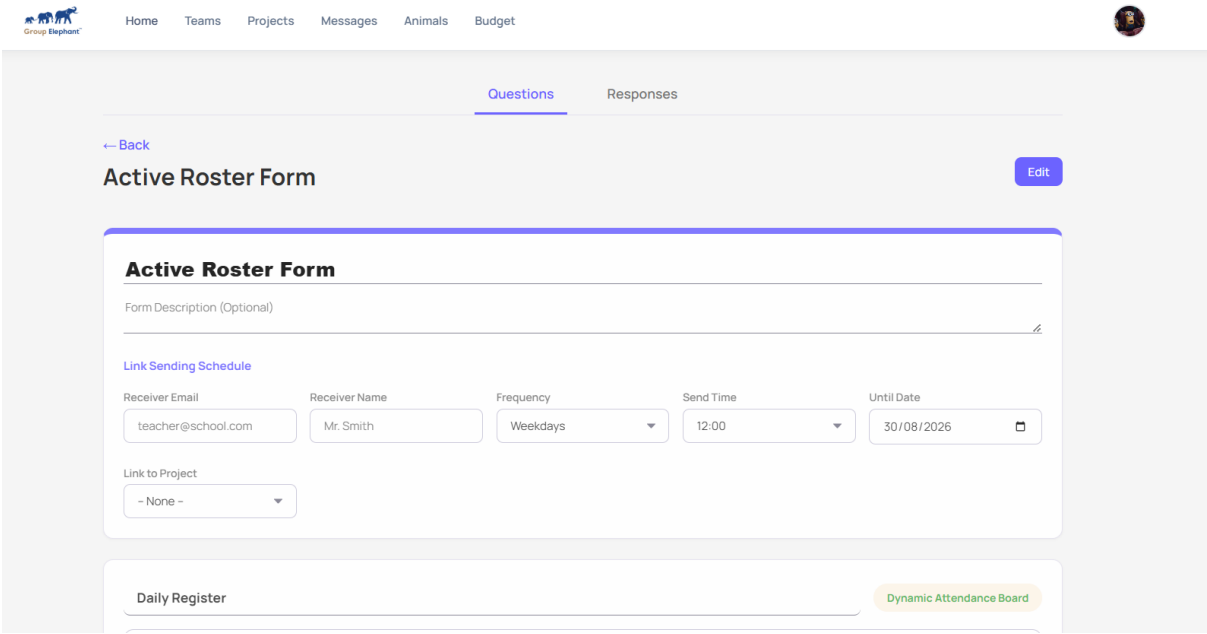


Figure 43: 7.17 View Impact Feedback Form Screen Design

2.3.6.5 Update Impact Feedback Form: Use Case 7.18

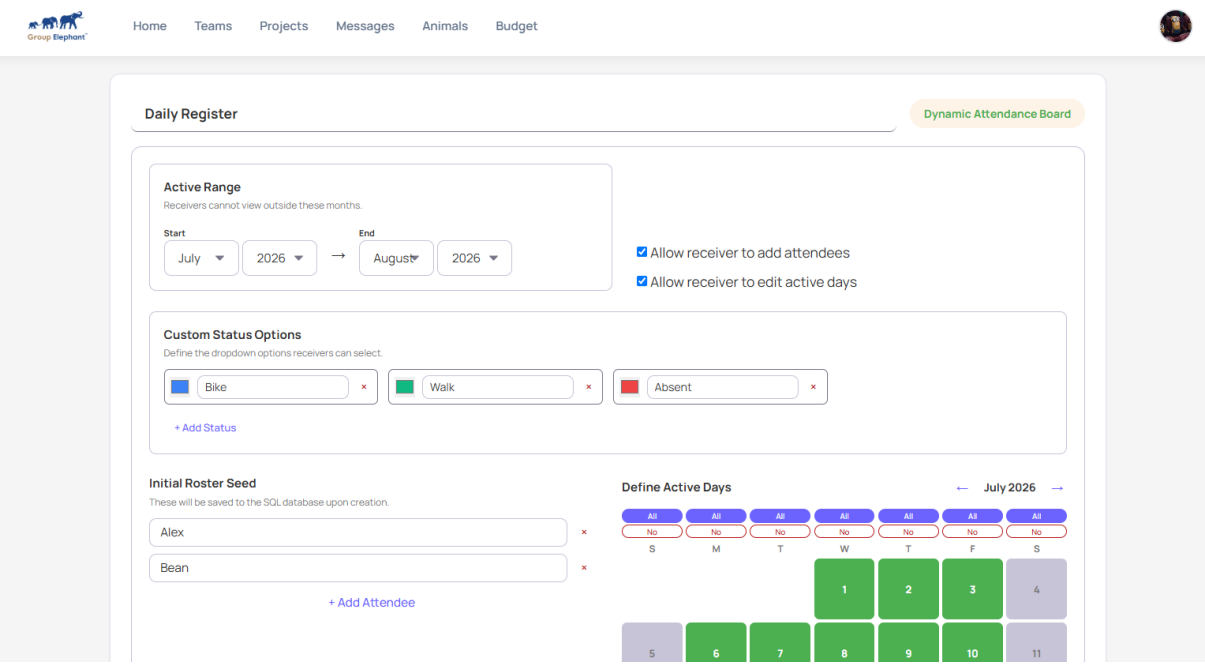


Figure 44: 7.18 Update Impact Feedback Form Screen Design

2.3.6.6 Deactivate Impact Feedback Form: Use Case 7.19

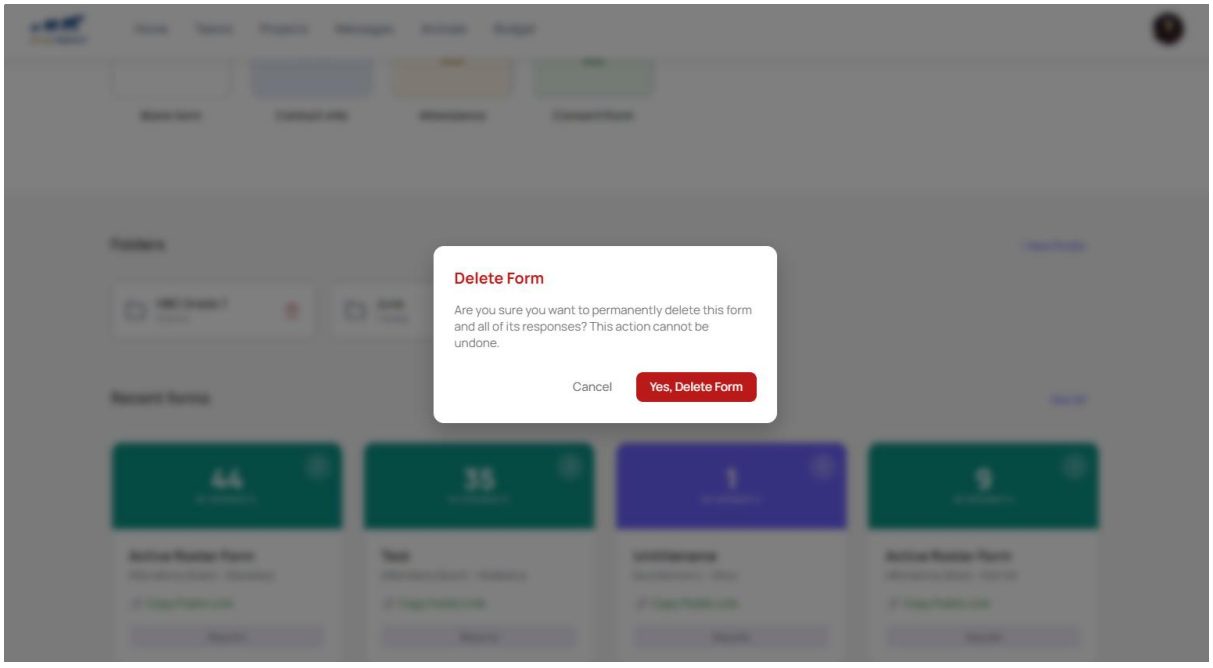


Figure 45: 7.19 Deactivate Impact Feedback Form Screen Design

2.3.6.7 Capture Impact Form: Use Case 7.20

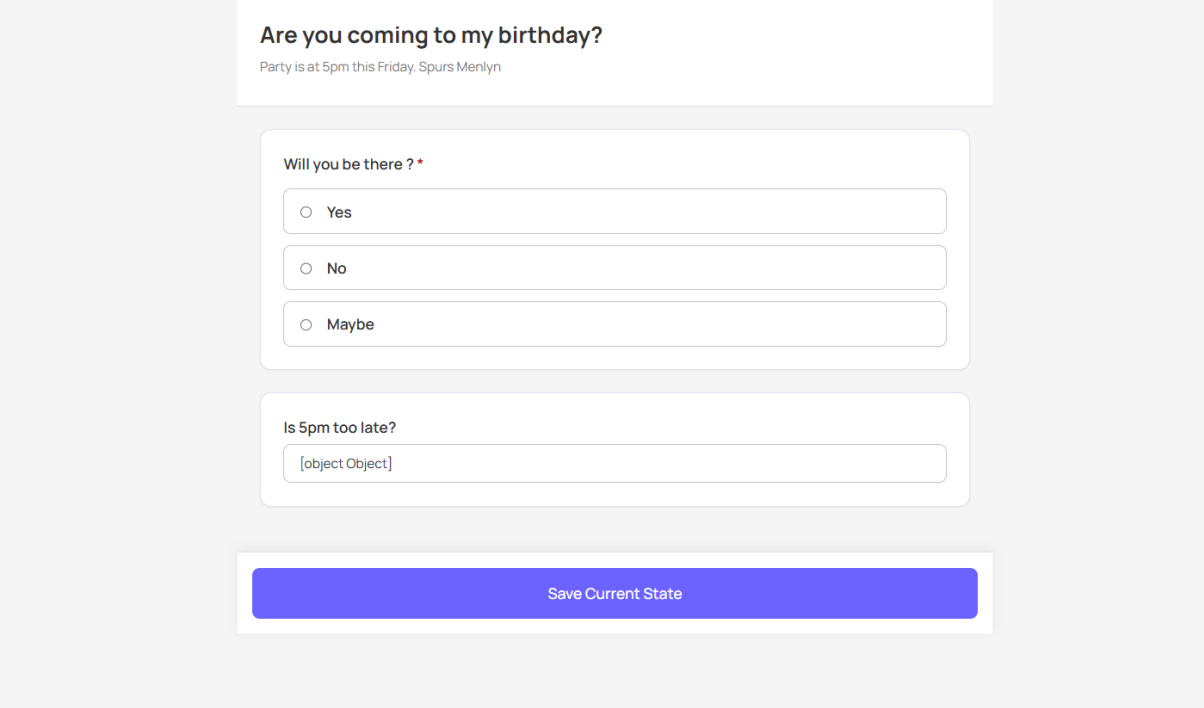


Figure 46: 7.20 Capture Impact Form Screen Design

2.3.6.8 View Impact Form: Use Case 7.21

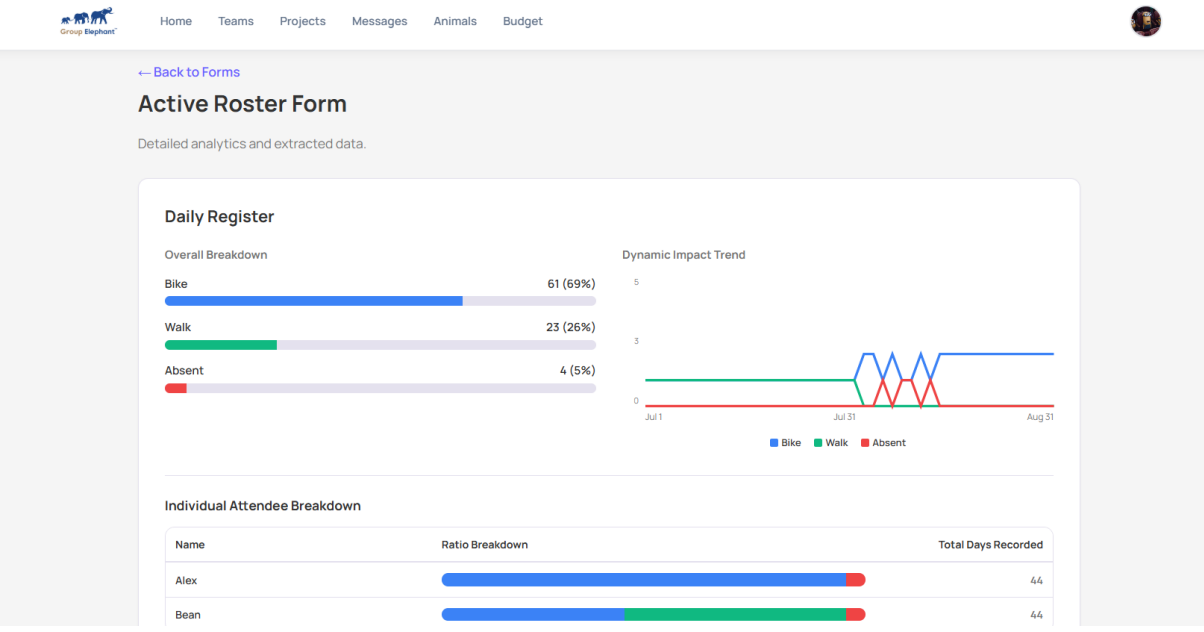


Figure 47: 7.21 View Impact Form Screen Design

2.3.6.9 Add Photo: Use Case 7.22

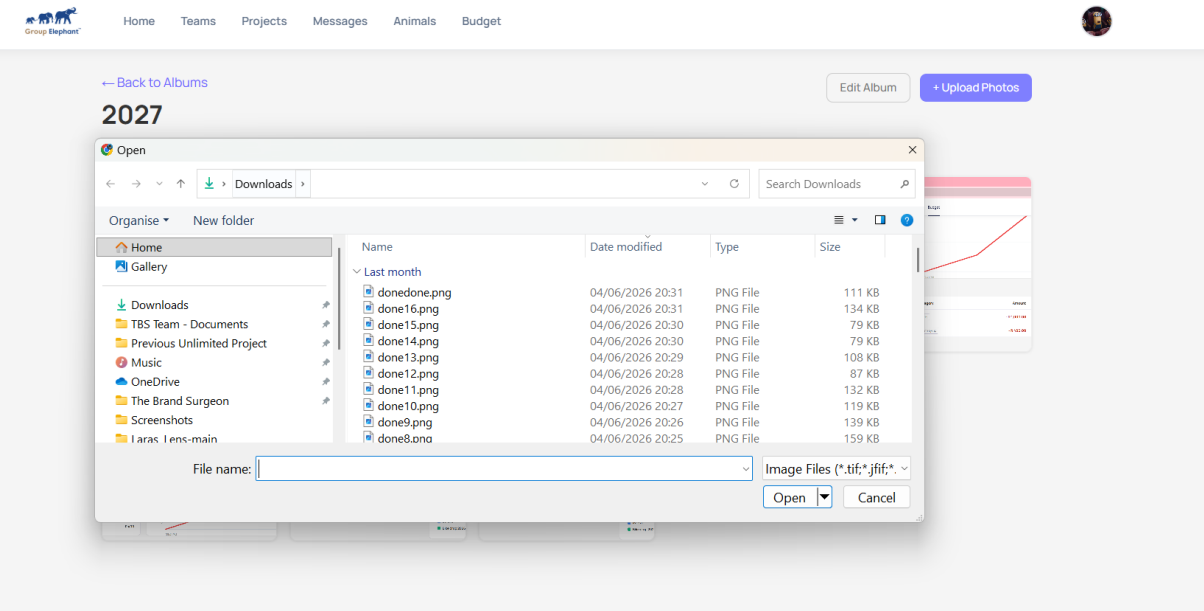


Figure 48: 7.22 Add Photo Screen Design

2.3.6.10 View Photo: Use Case 7.23

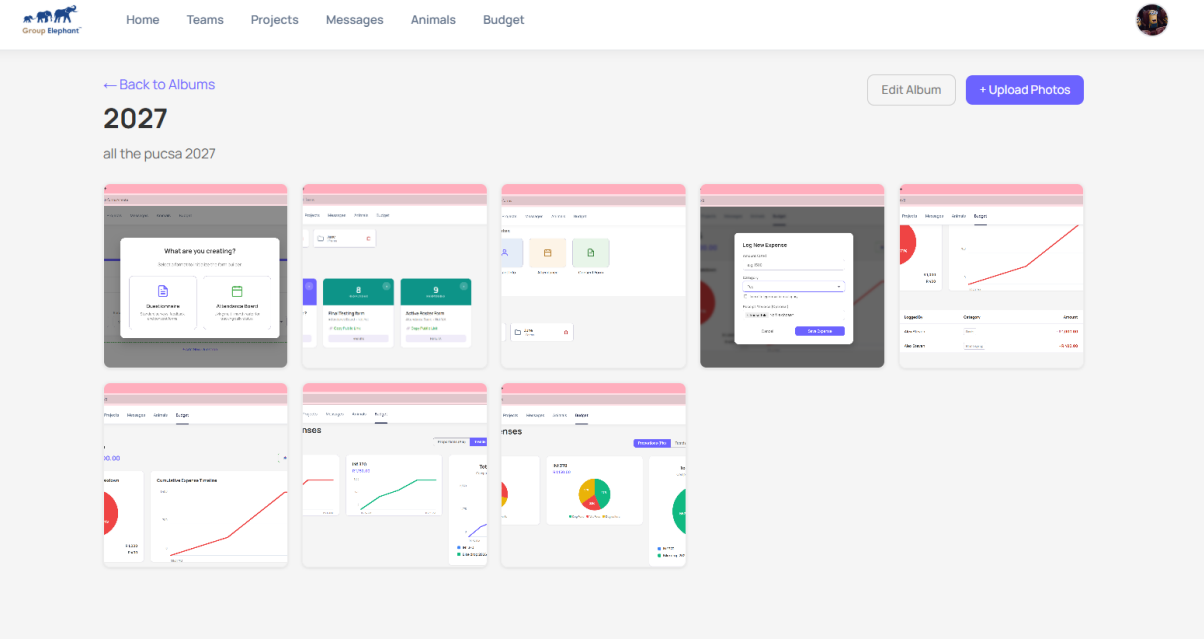


Figure 49: 7.23 View Photo Screen Design

2.3.6.11 Delete Photo: Use Case 7.24

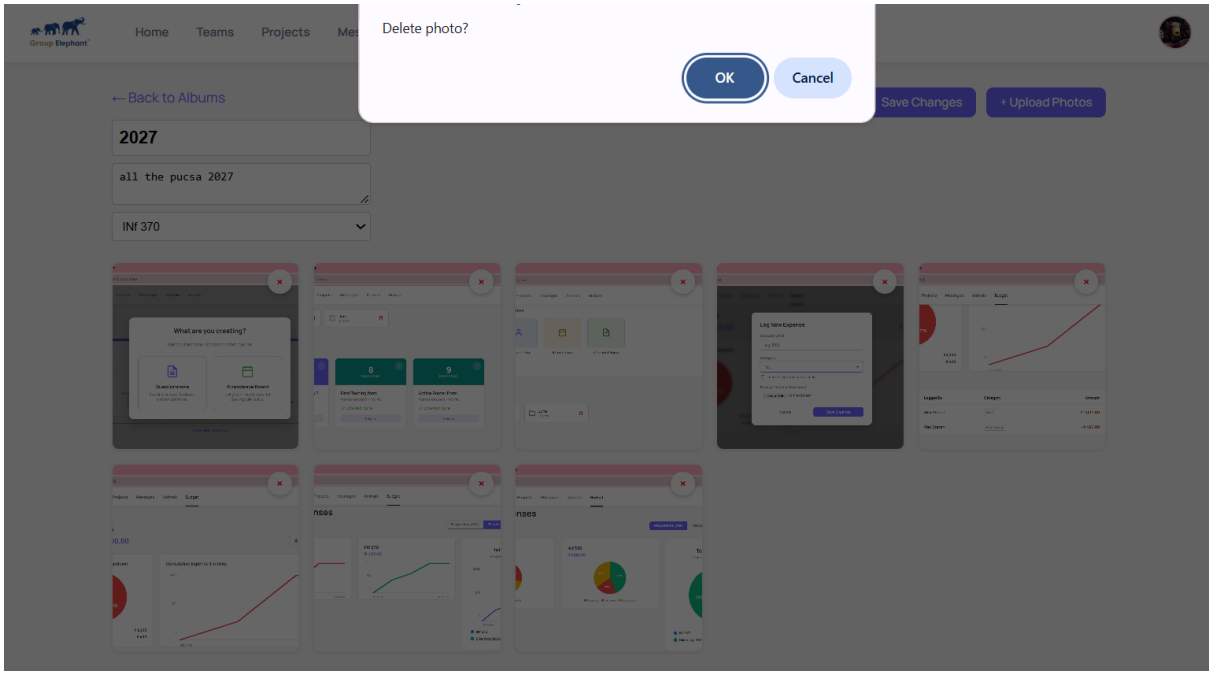


Figure 50: 7.24 Delete Photo Screen Design

2.3.7 K9 Subsystem

2.3.7.1 Add Dog: Use Case 8.1

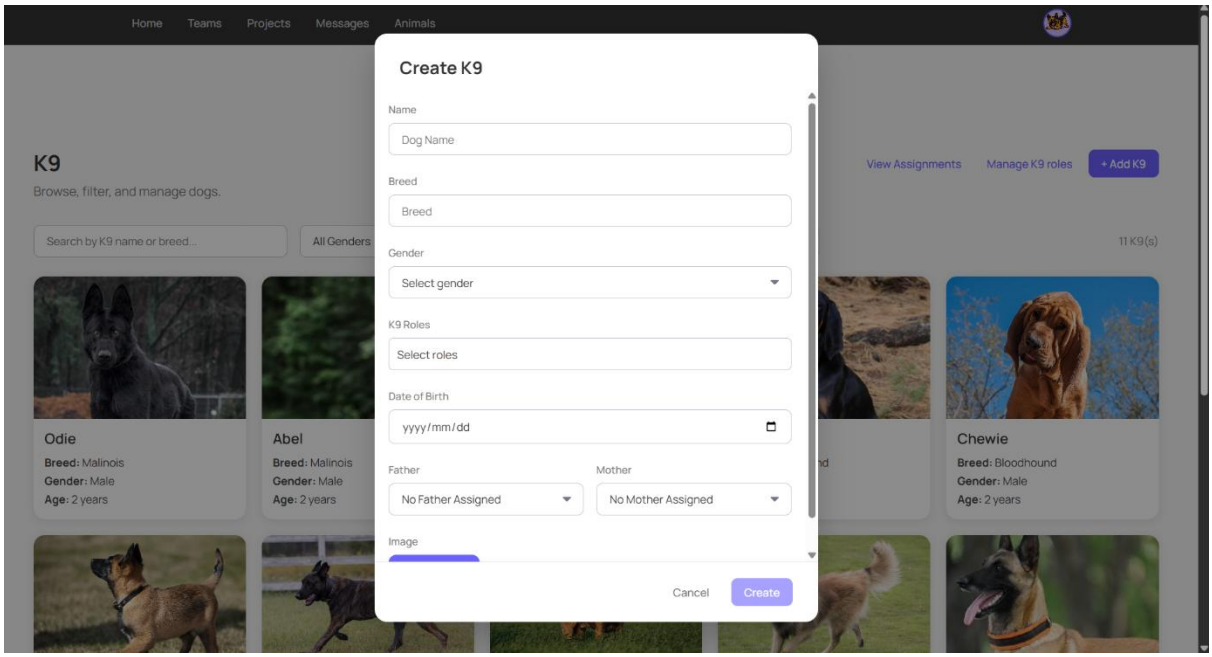


Figure 51: 8.1 Add Dog Screen Design

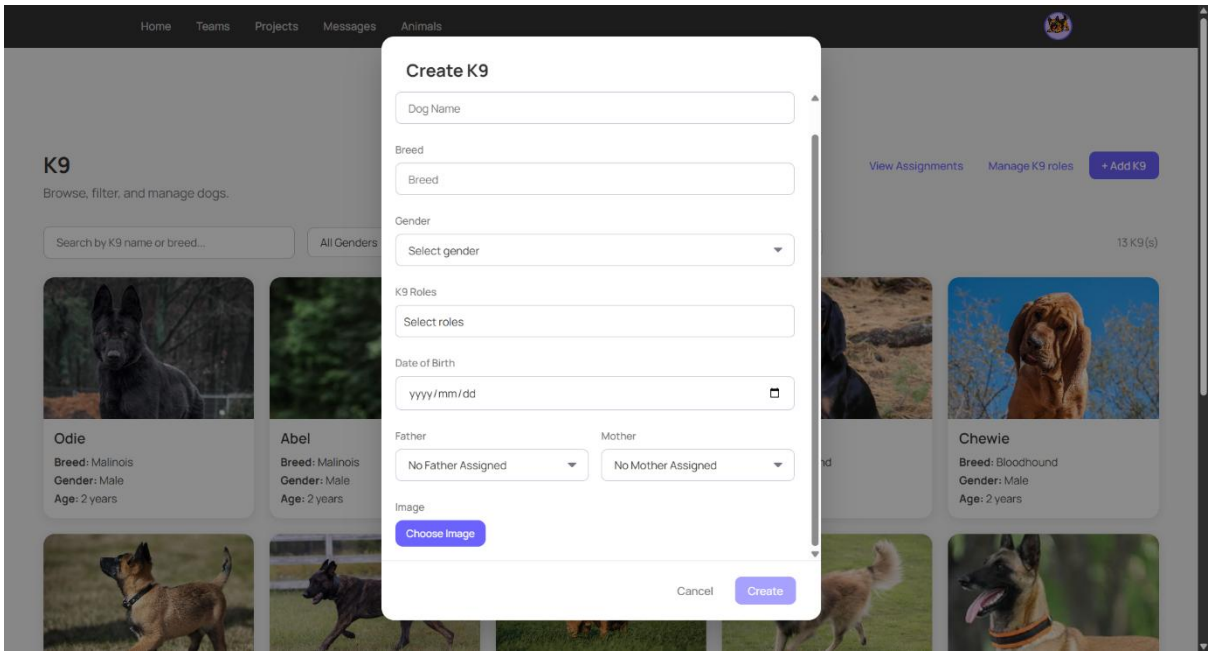


Figure 52: 8.1 Add Dog Screen Design Continued

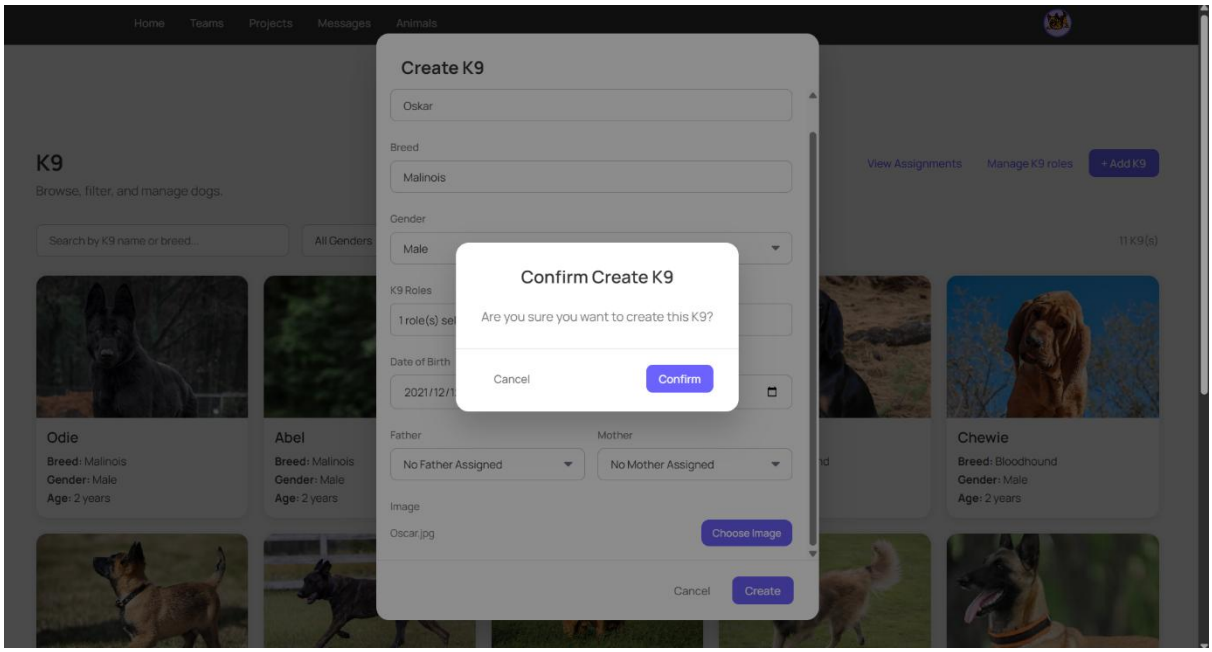


Figure 53: 8.1 Add Dog Confirmation Screen Design

2.3.7.2 View Dog: Use Case 8.2

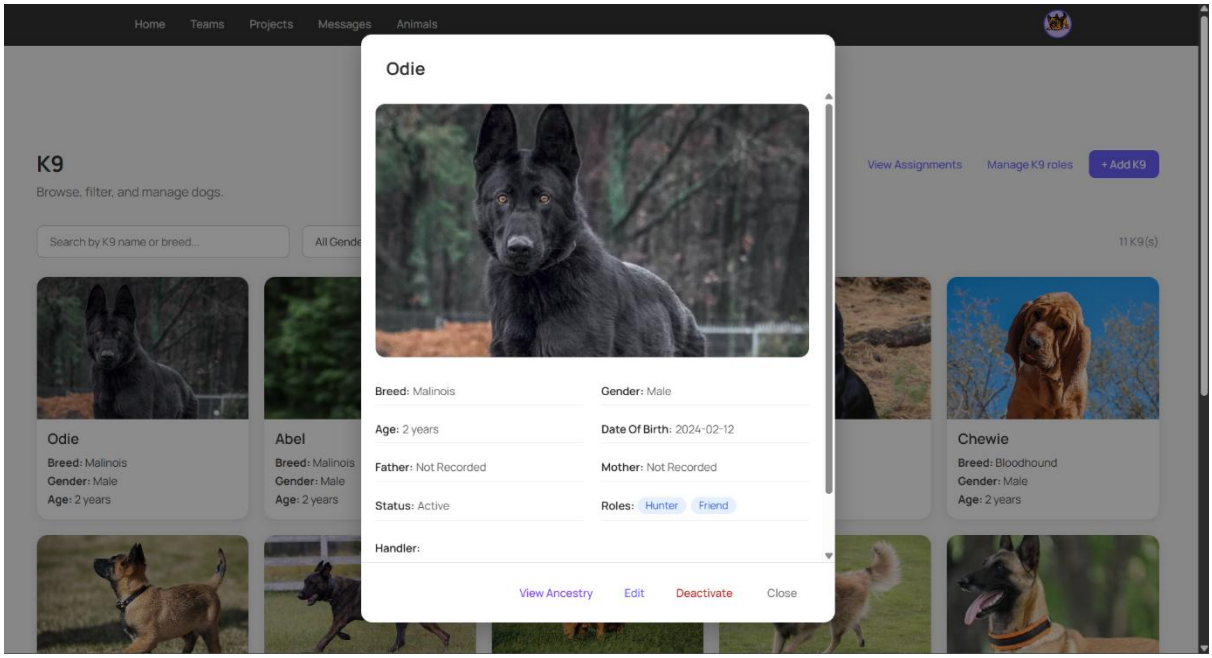


Figure 54: 8.2 View Dog Screen Design

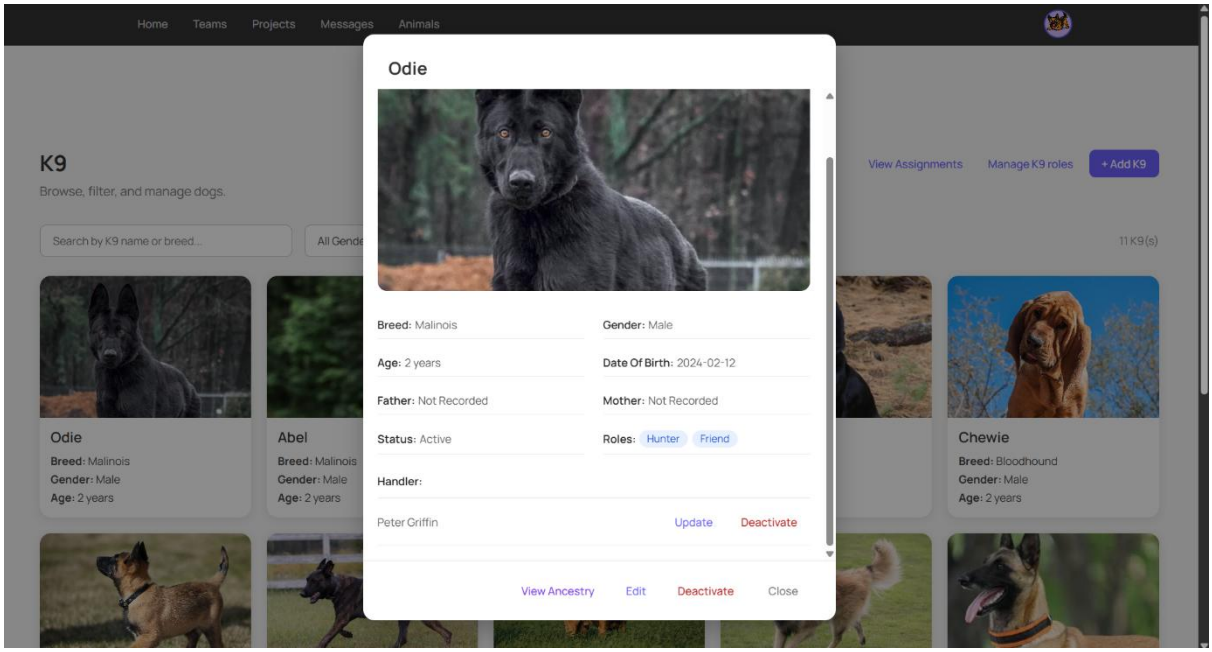


Figure 55: 8.2 View Dog Screen Design Continued

2.3.7.3 View Dogs: Use Case 8.3

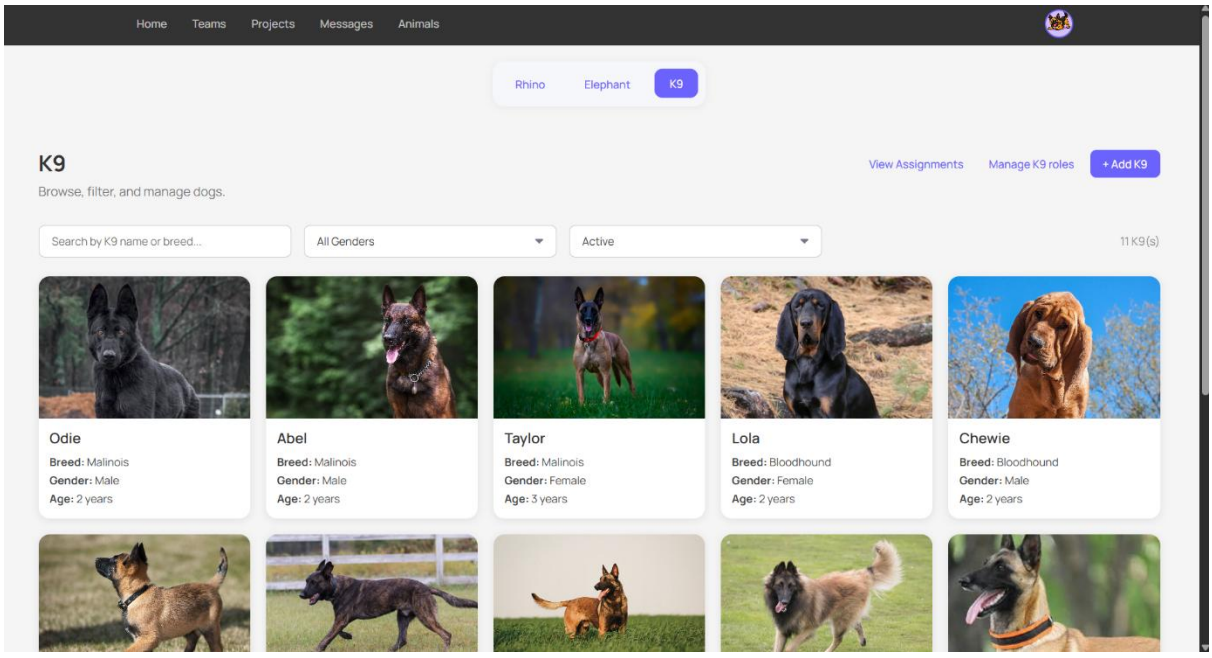


Figure 56: 8.3 View Dogs Screen Design

2.3.7.4 Update Dog: Use Case 8.4

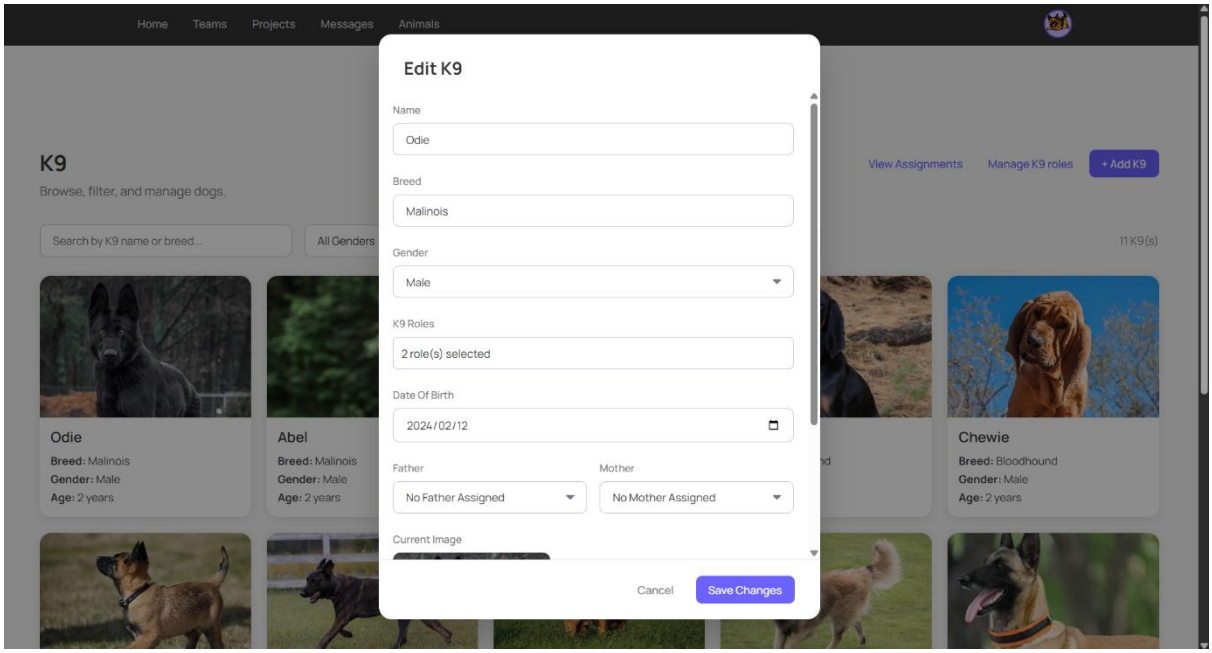


Figure 58: 8.4 Update Dog Screen Design

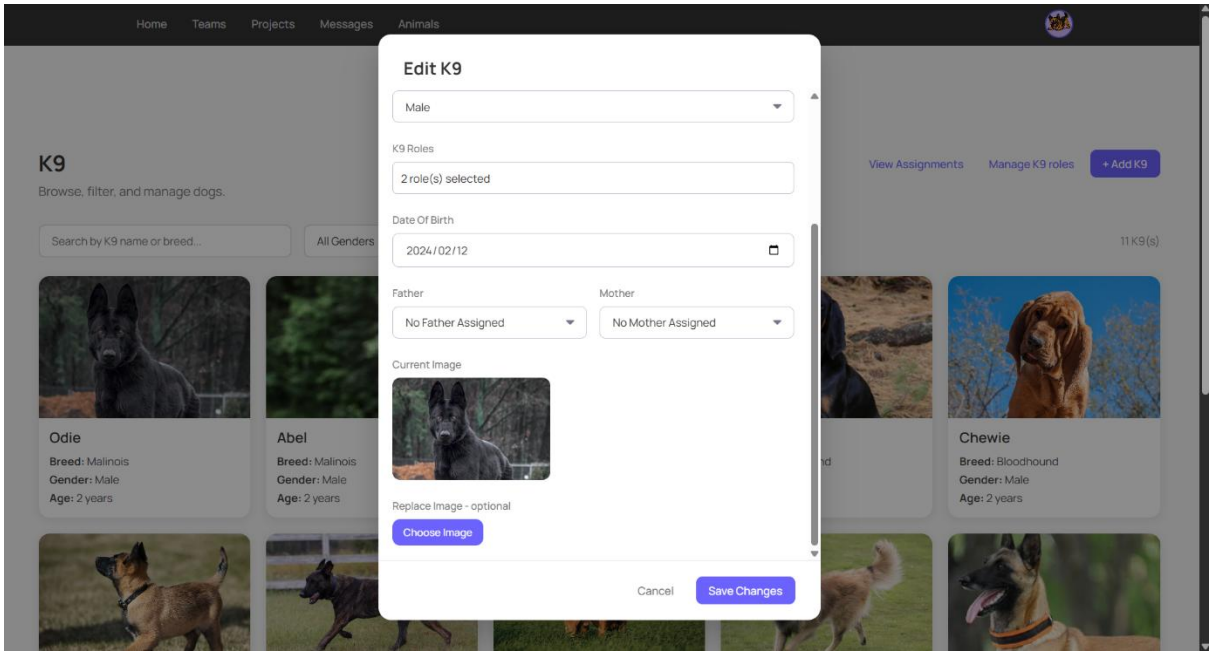


Figure 57: 8.4 Update Dog Screen Design Continued

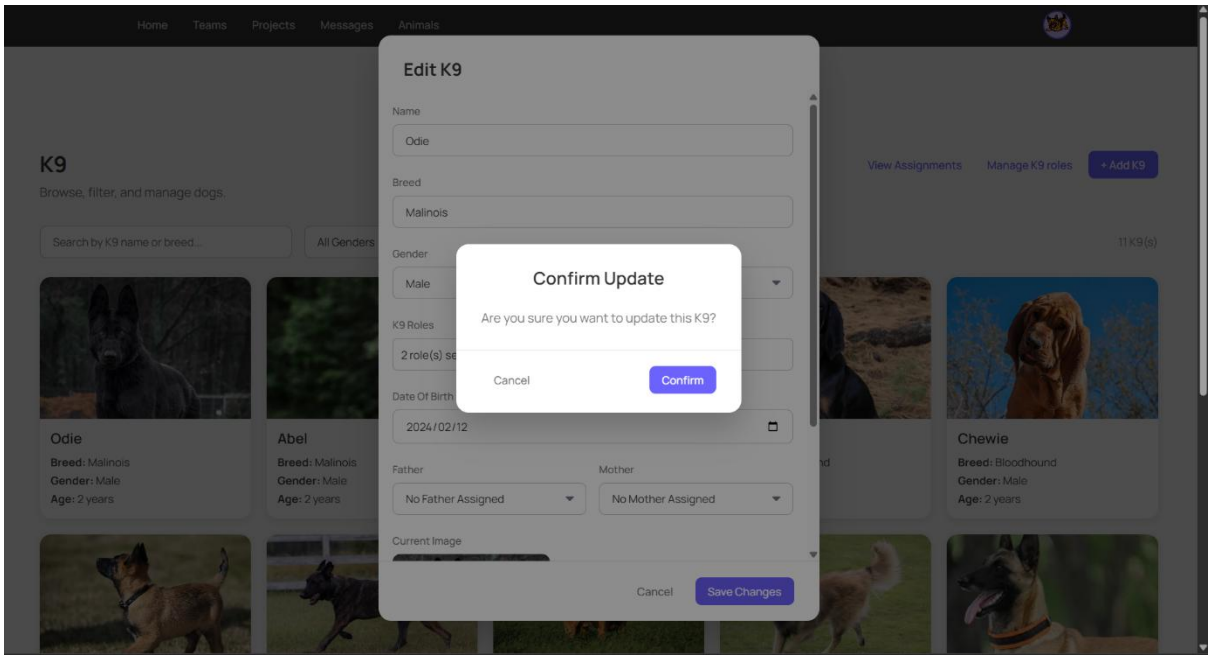


Figure 59: 8.4 Update Dog Confirmation Screen Design

2.3.7.5 Deactivate Dog: Use Case 8.5

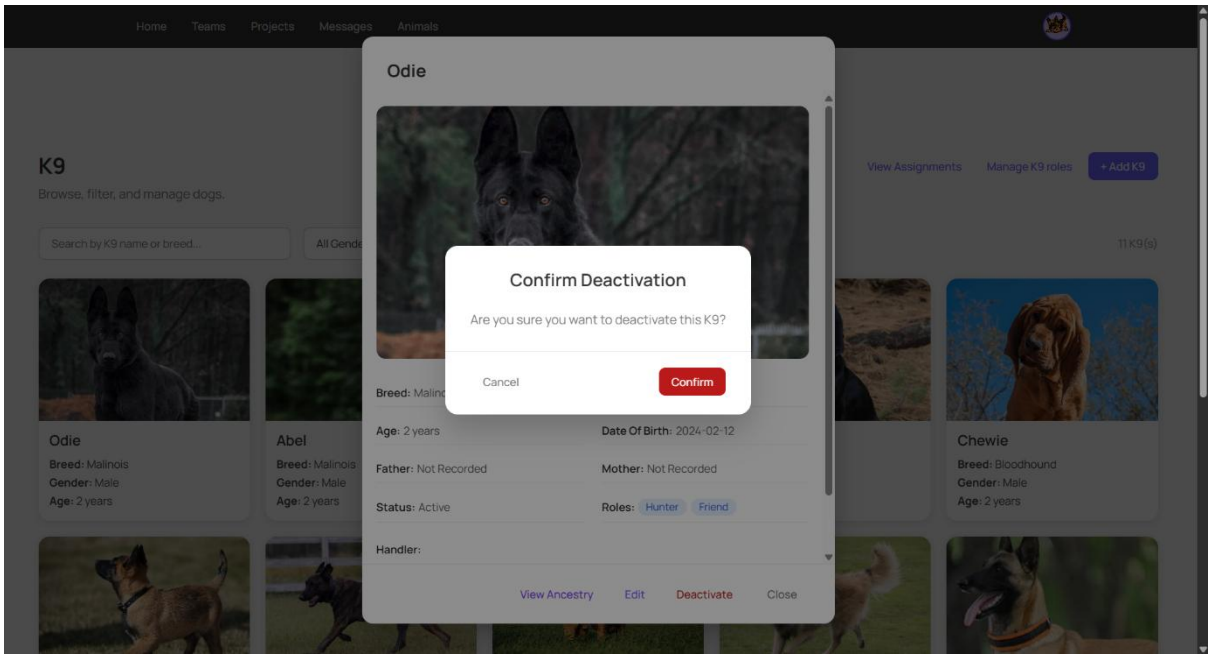


Figure 60: 8.5 Deactivate Dog Screen Design

2.3.7.6 Assign User to Dog: Use Case 8.6

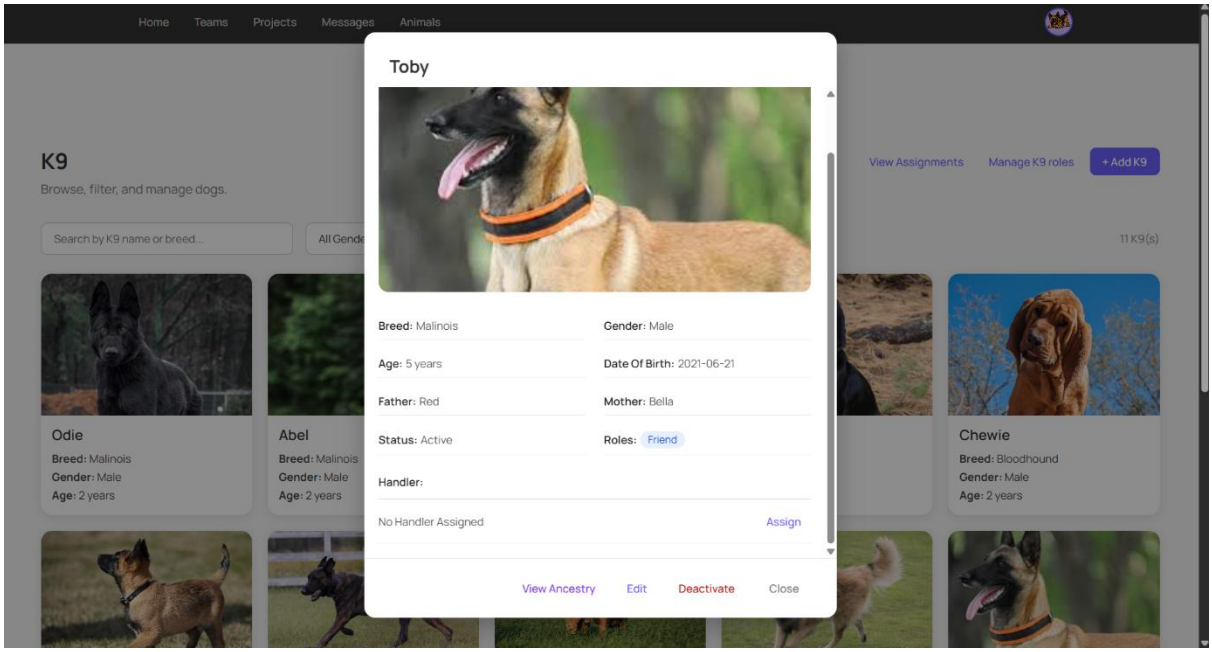


Figure 62: 8.6 Assign User to Dog Screen Design

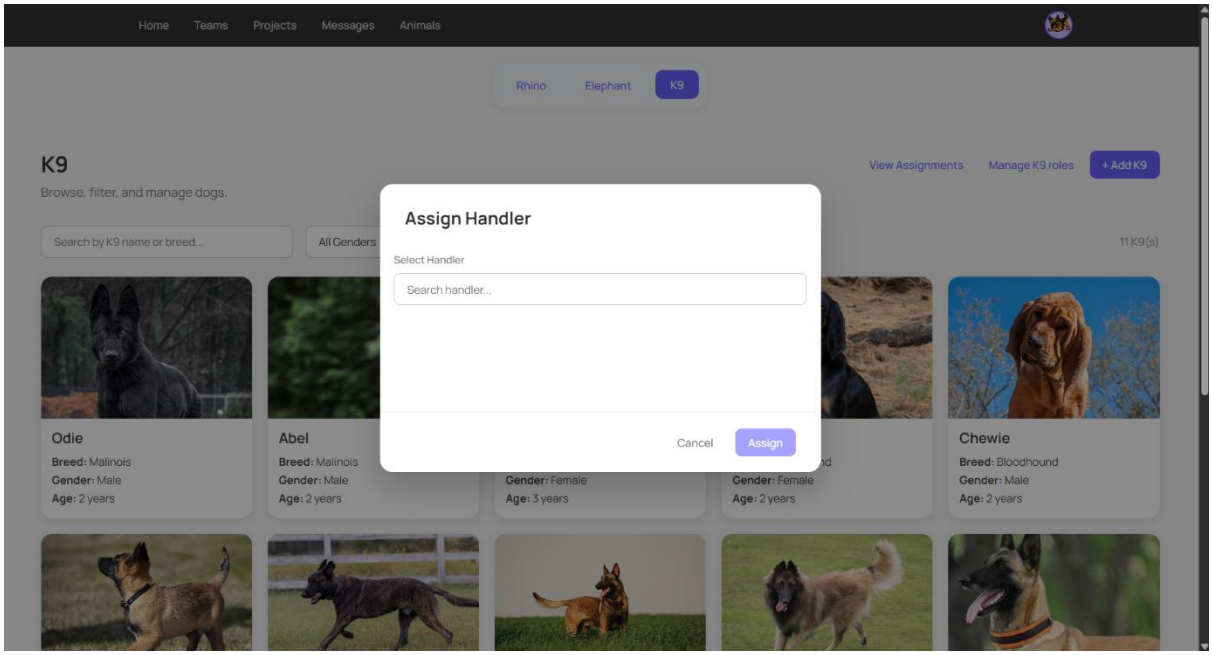


Figure 61: 8.6 Assign User to Dog Screen Design Continued

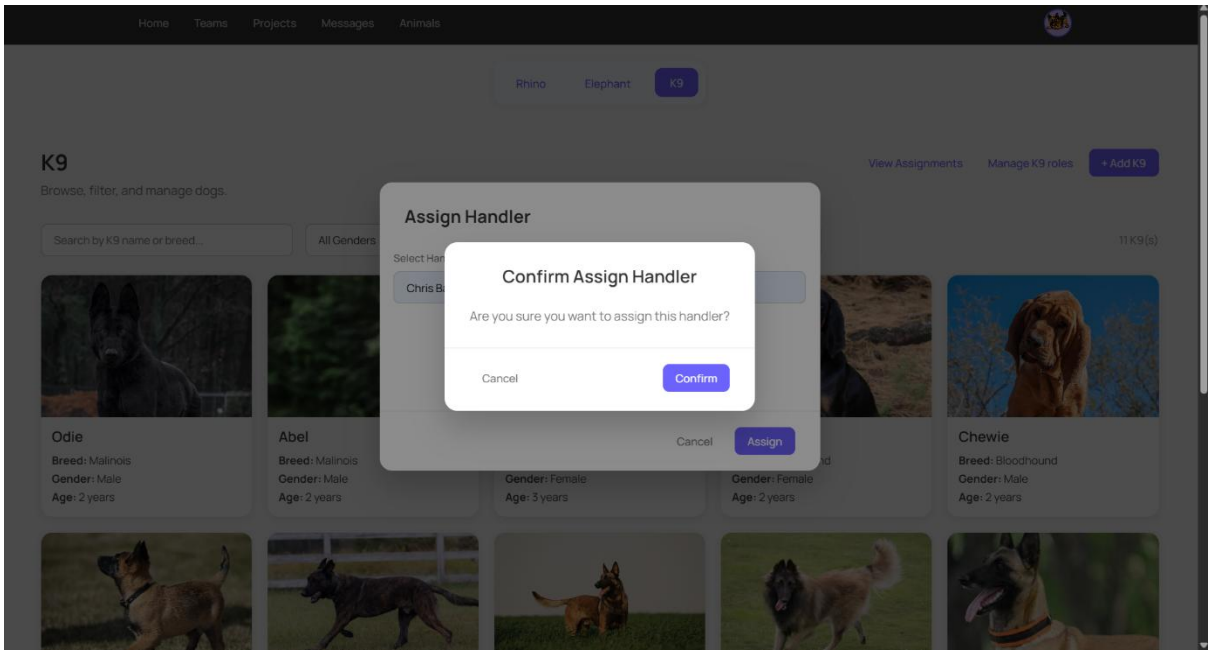


Figure 63: 8.6 Assign User to Dog Confirmation Screen Design

2.3.7.7 View User-Dog Assignment: Use Case 8.7

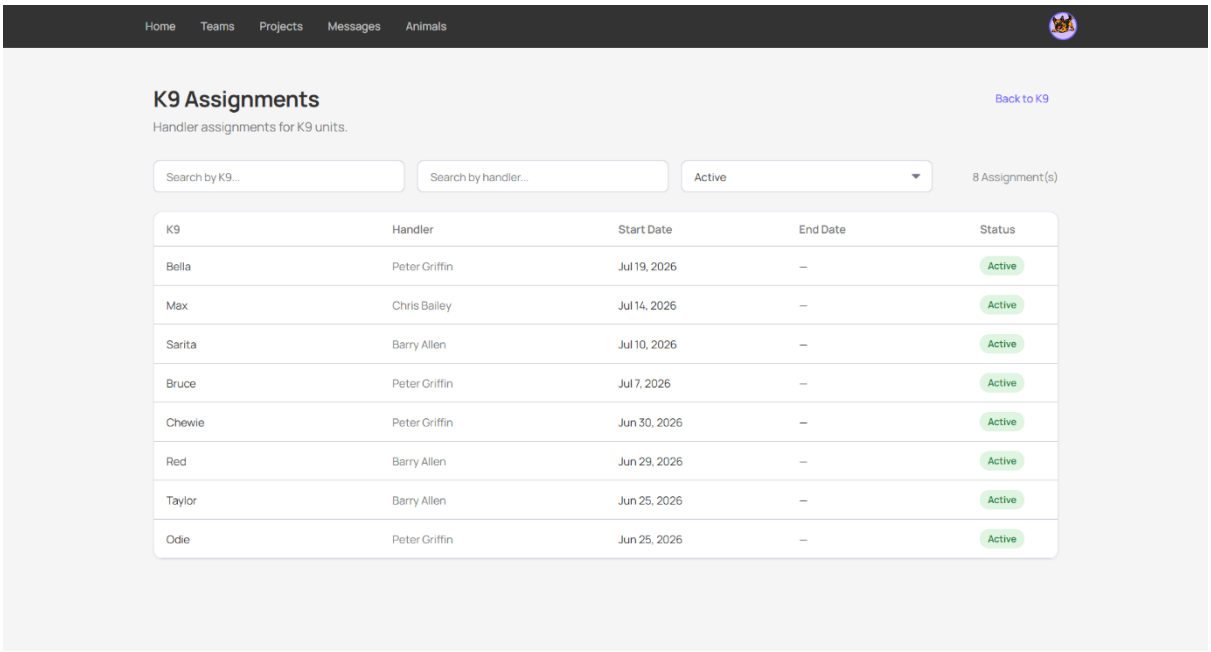


Figure 64: 8.7 View User-Dog Assignment Screen Design

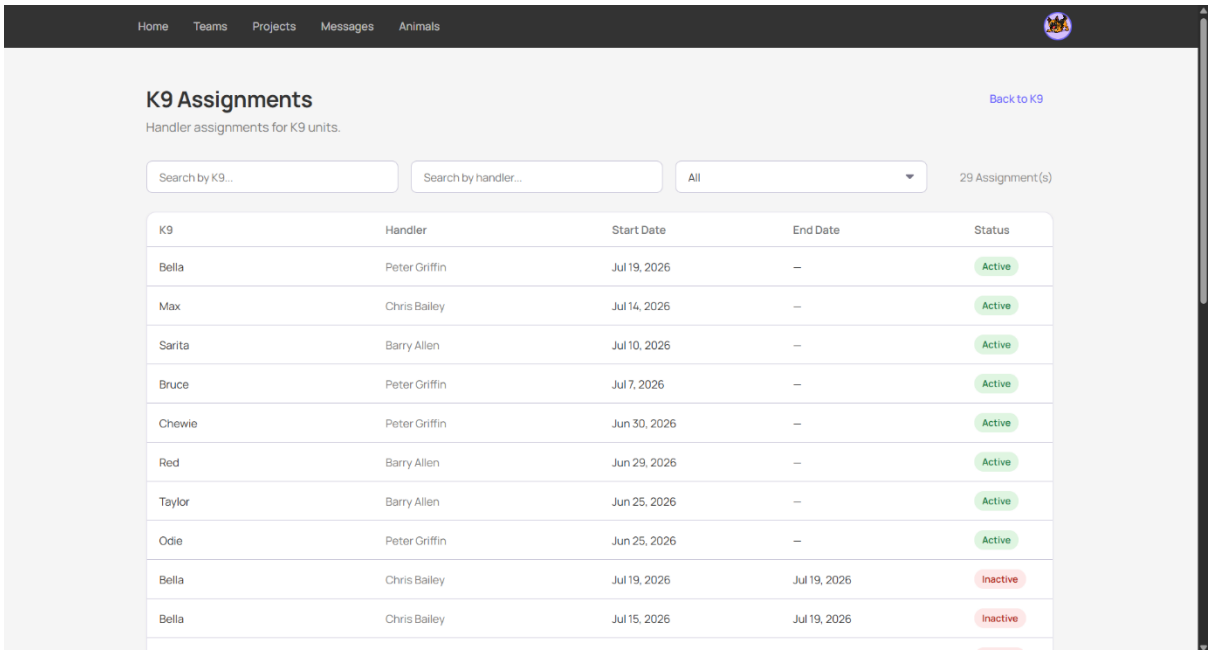


Figure 65: 8.7 View User-Dog Assignment Screen Design Continued

2.3.7.8 Update User-Dog Assignment: Use Case 8.8

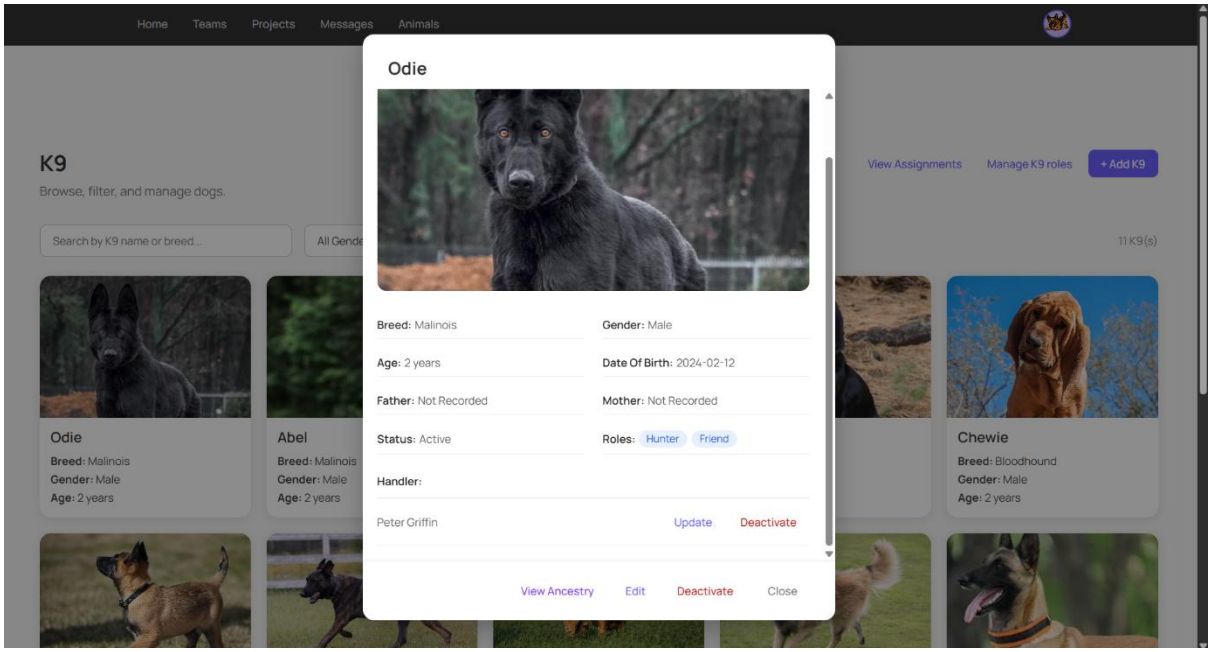


Figure 66: 8.8 Update User-Dog Assignment Screen Design

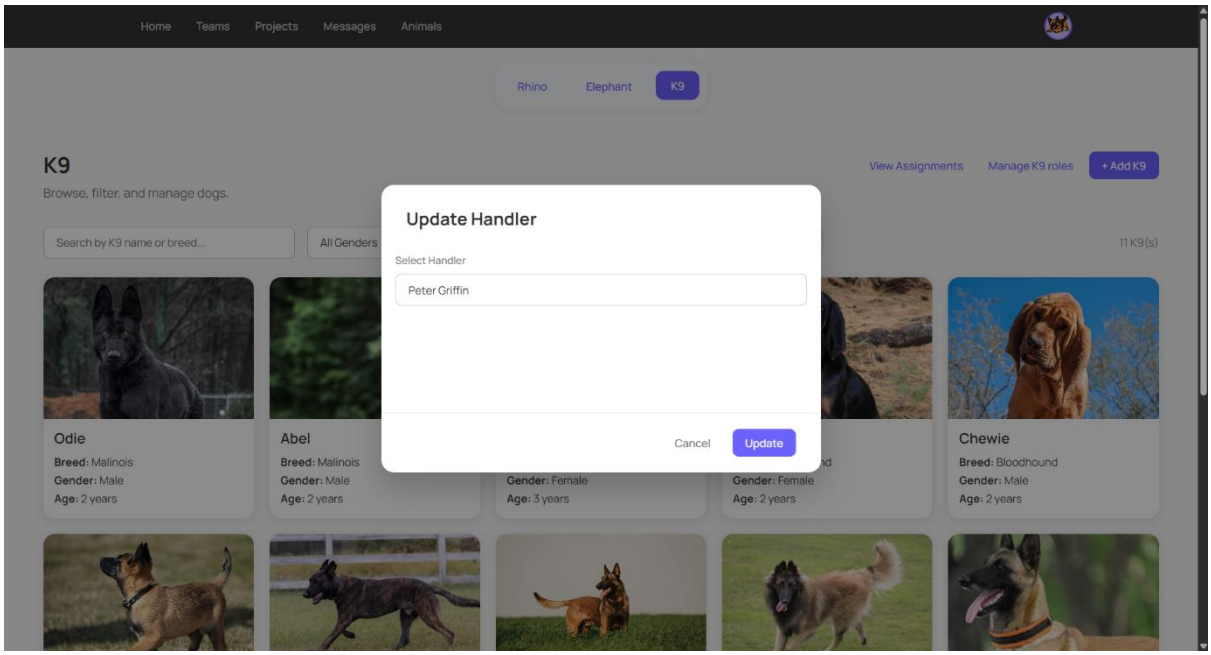


Figure 68: 8.8 Update User-Dog Assignment Screen Design Continued

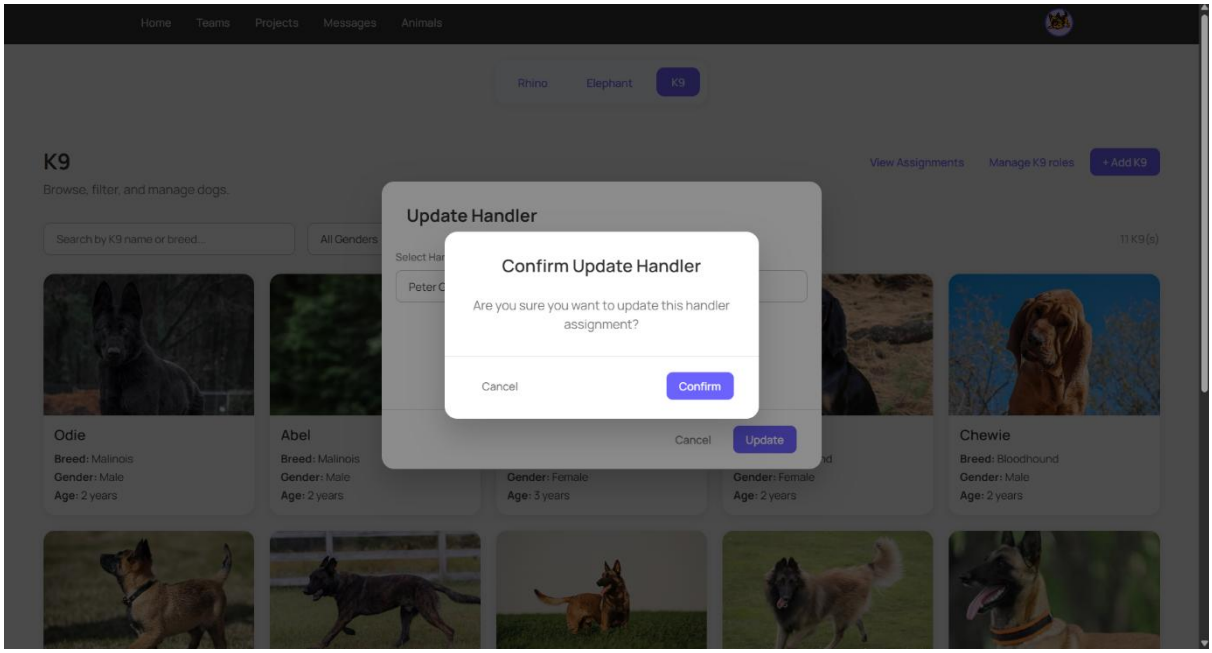


Figure 67: 8.8 Update User-Dog Assignment Confirmation Screen Design

2.3.7.9 Reactivate User-Dog Assignment: Use Case 8.9

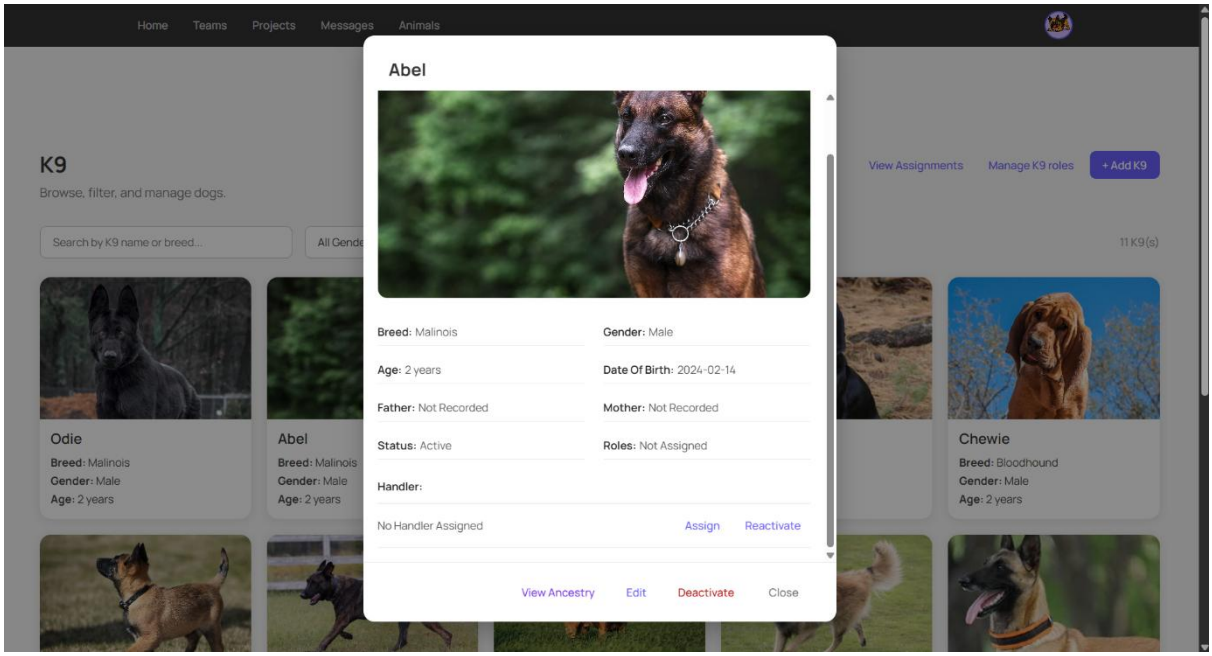


Figure 69: 8.9 Reactivate User-Dog Assignment Screen Design

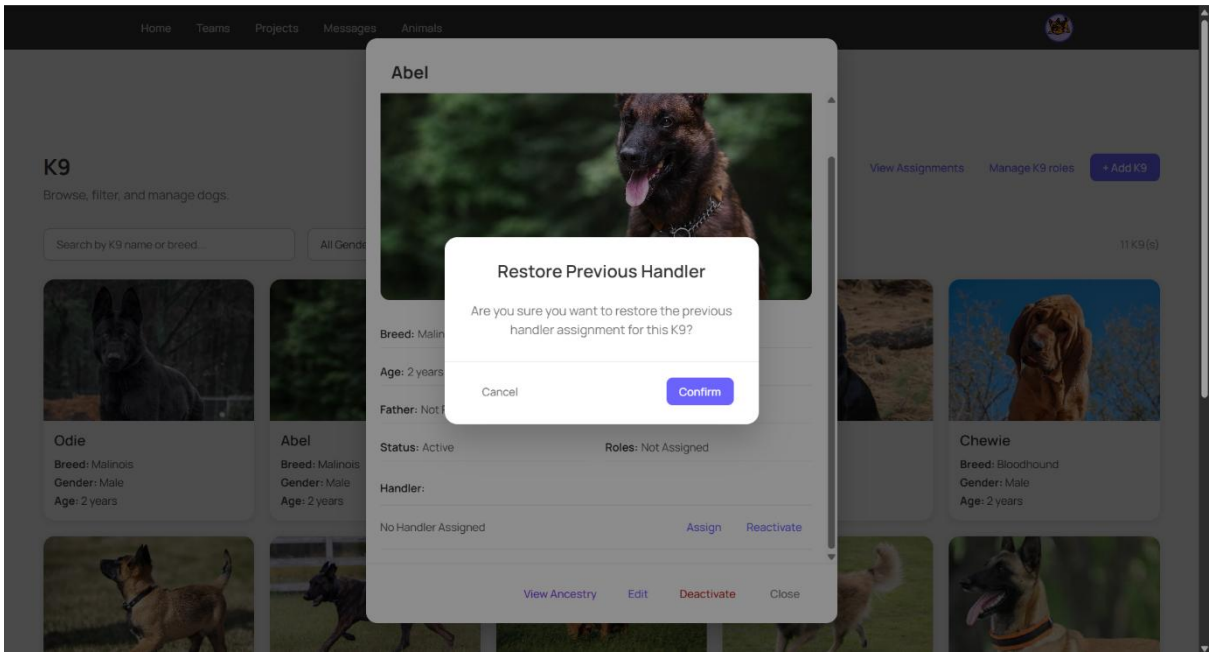


Figure 70: 8.9 Reactivate User-Dog Assignment Confirmation Screen Design

2.3.7.10 Deactivate User-Dog Assignment: Use Case 8.10

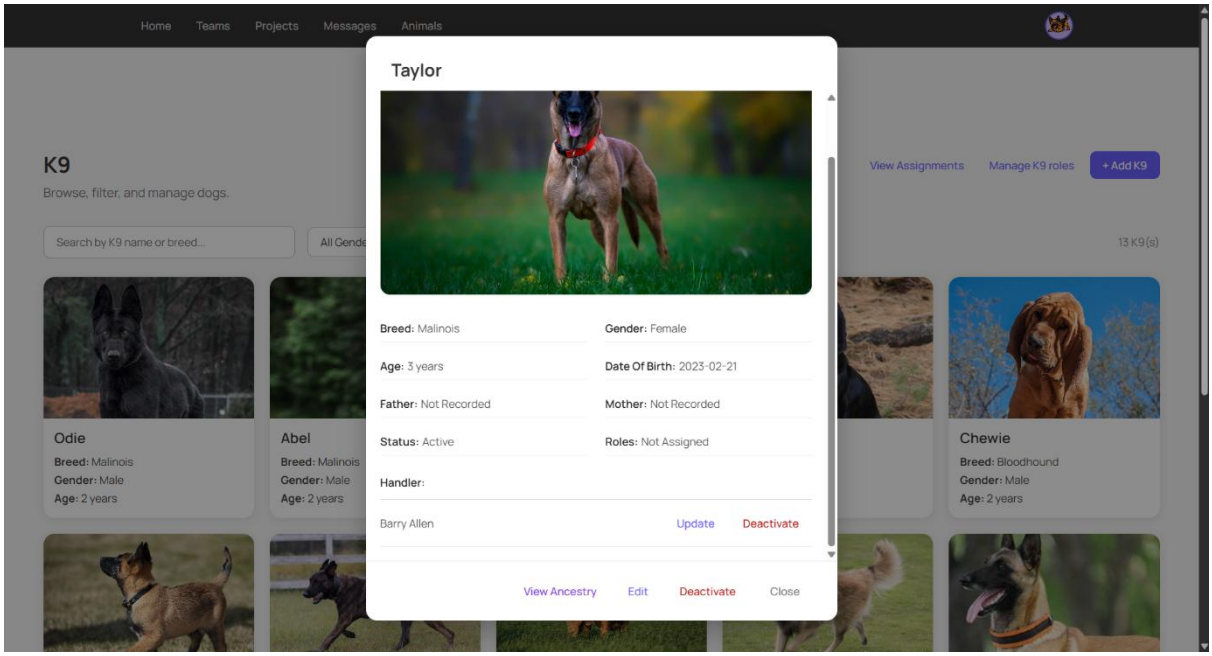


Figure 71: 8.10 Deactivate User-Dog Assignment Screen Design

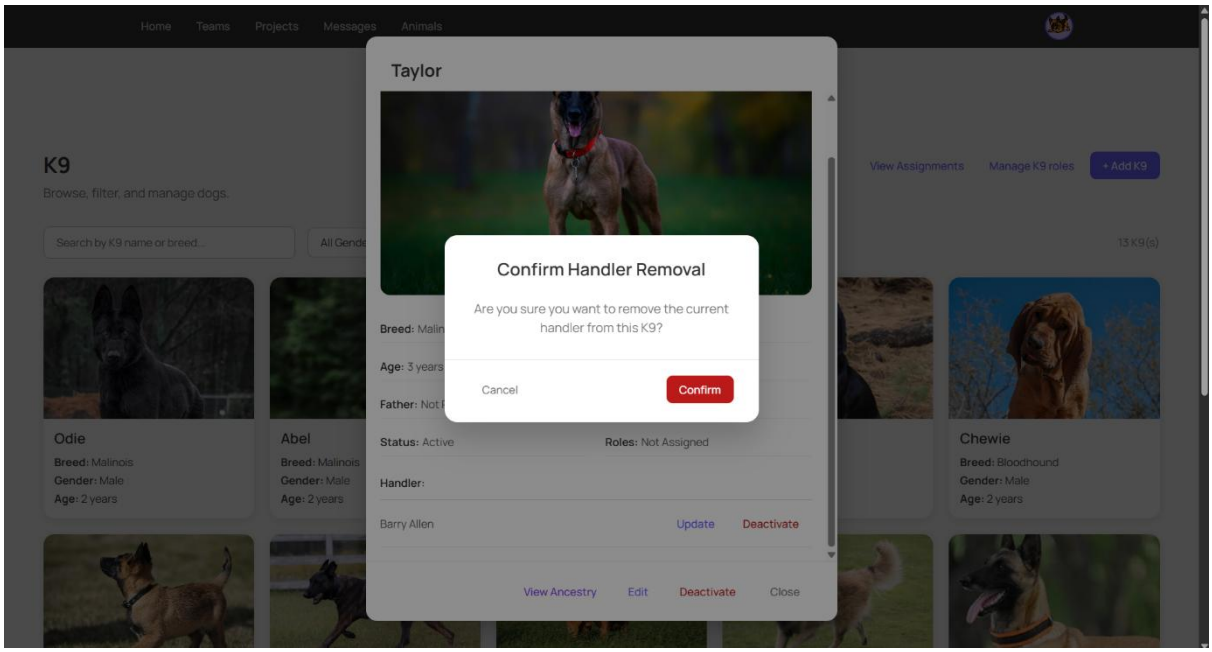


Figure 72: 8.10 Deactivate User-Dog Assignment Confirmation Screen Design

2.3.7.11 Add K9 Role: Use Case 8.11

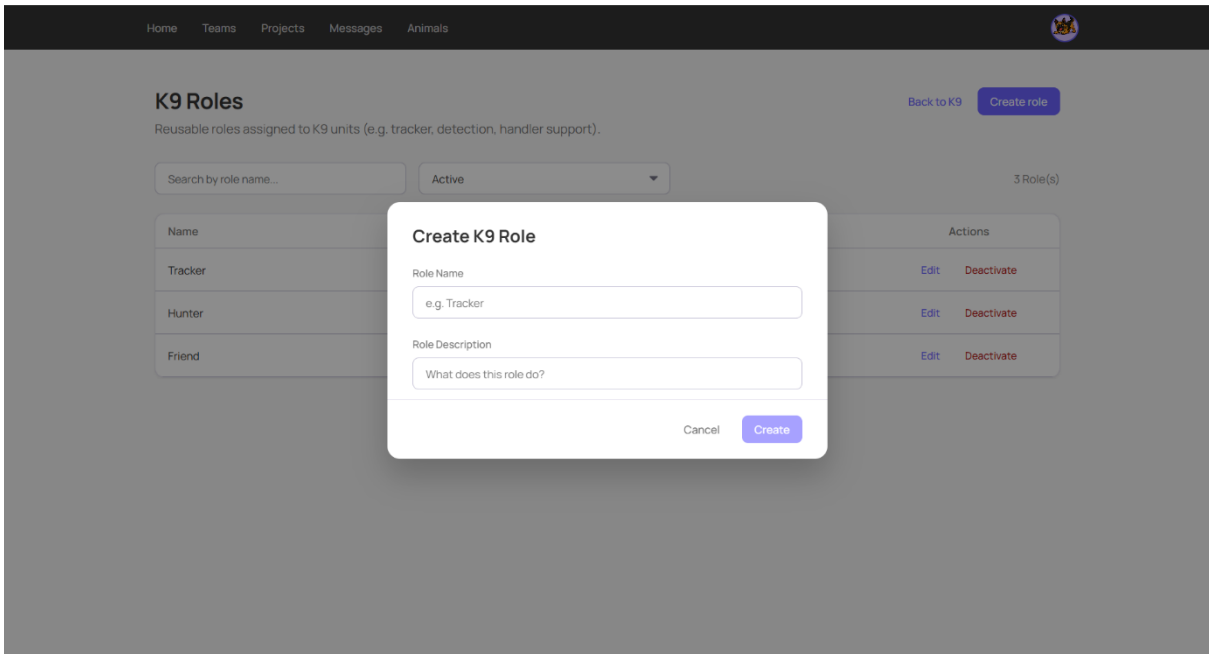


Figure 74: 8.11 Add K9 Role Screen Design

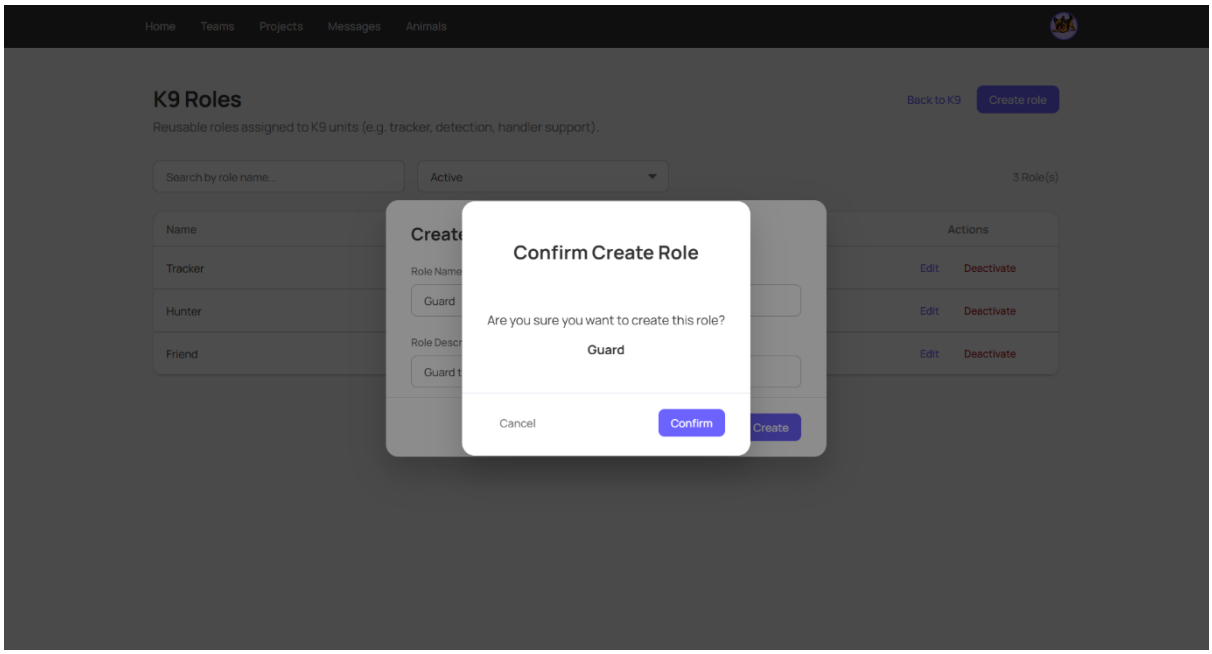


Figure 73: 8.11 Add K9 Role Confirmation Screen Design

2.3.7.12 Update K9 Role: Use Case 8.12

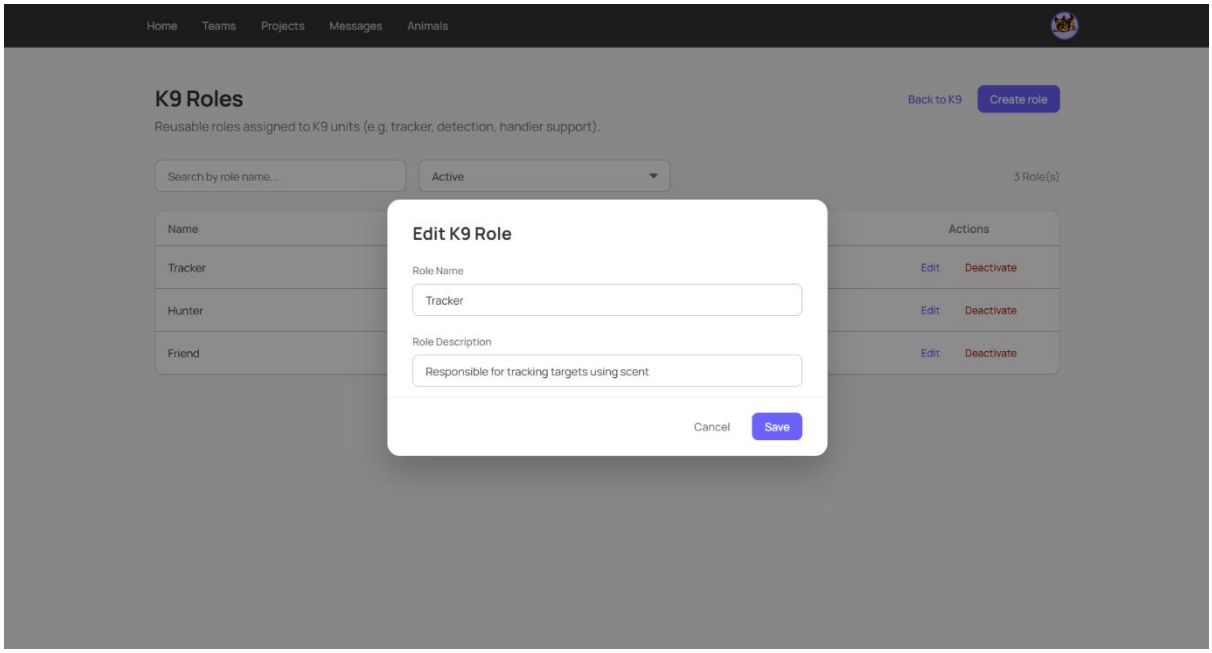


Figure 76: 8.12 Update K9 Role Screen Design

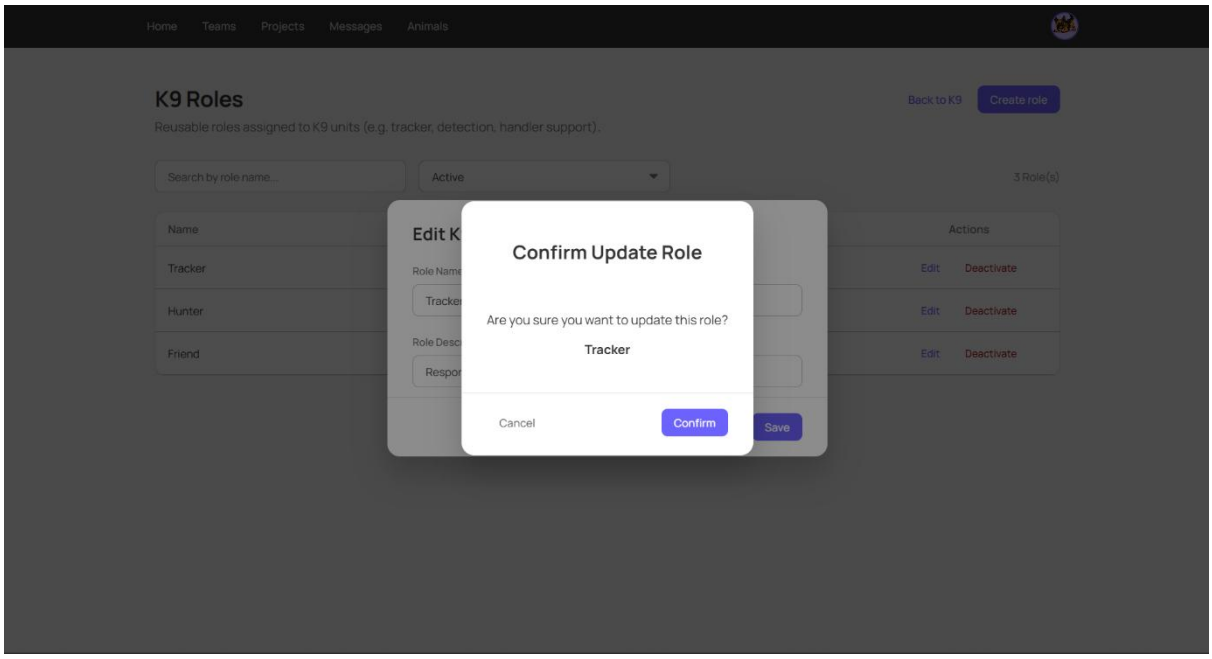


Figure 75: 8.12 Update K9 Role Confirmation Screen Design

2.3.7.13 View K9 Roles: Use Case 8.13

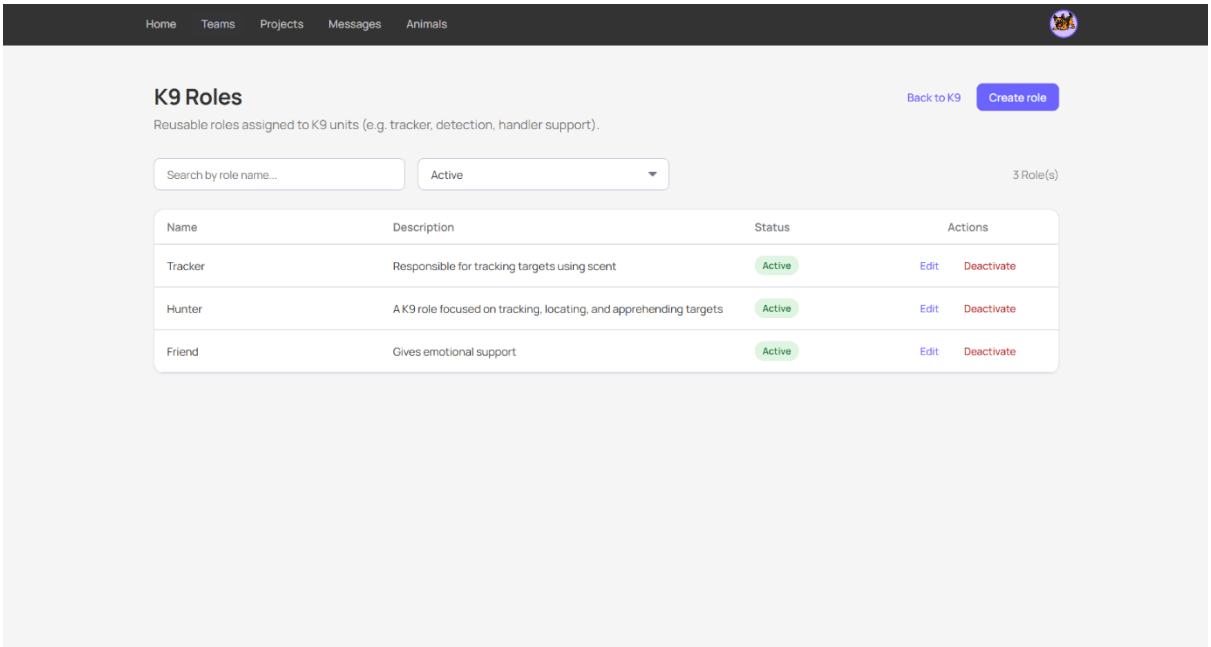


Figure 77: 8.13 View K9 Roles Screen Design

2.3.7.14 Reactivate K9 Role: Use Case 8.14

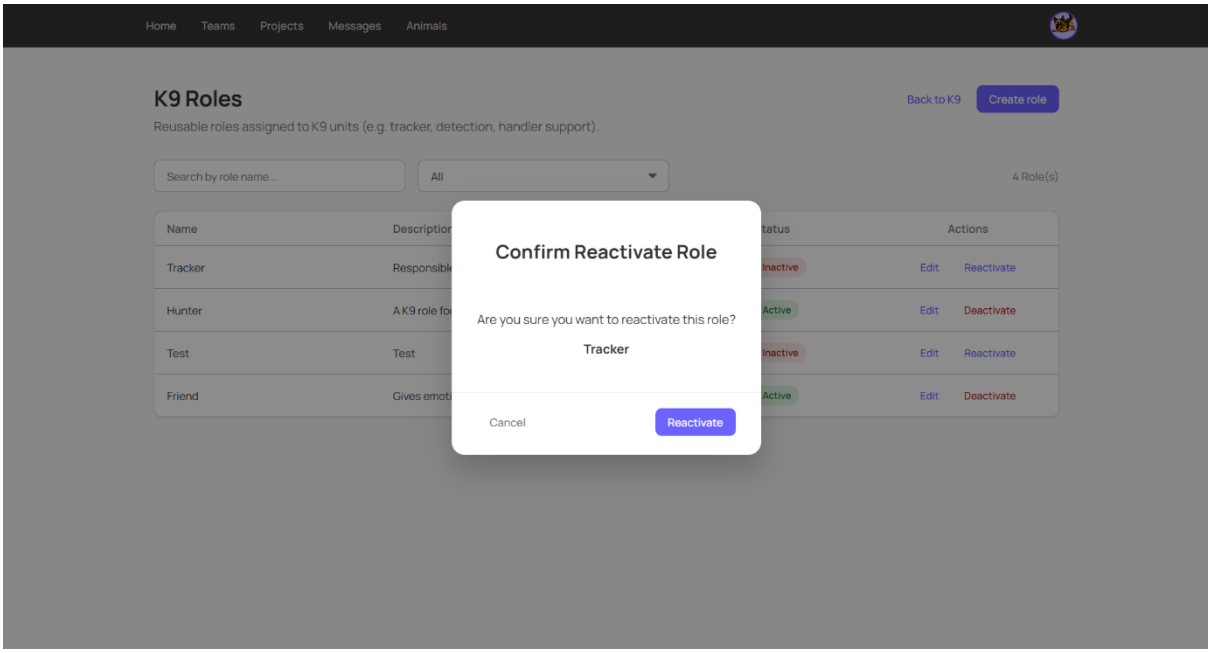


Figure 78: 8.14 Reactivate K9 Role Screen Design

2.3.7.15 Deactivate K9 Role: Use Case 8.15

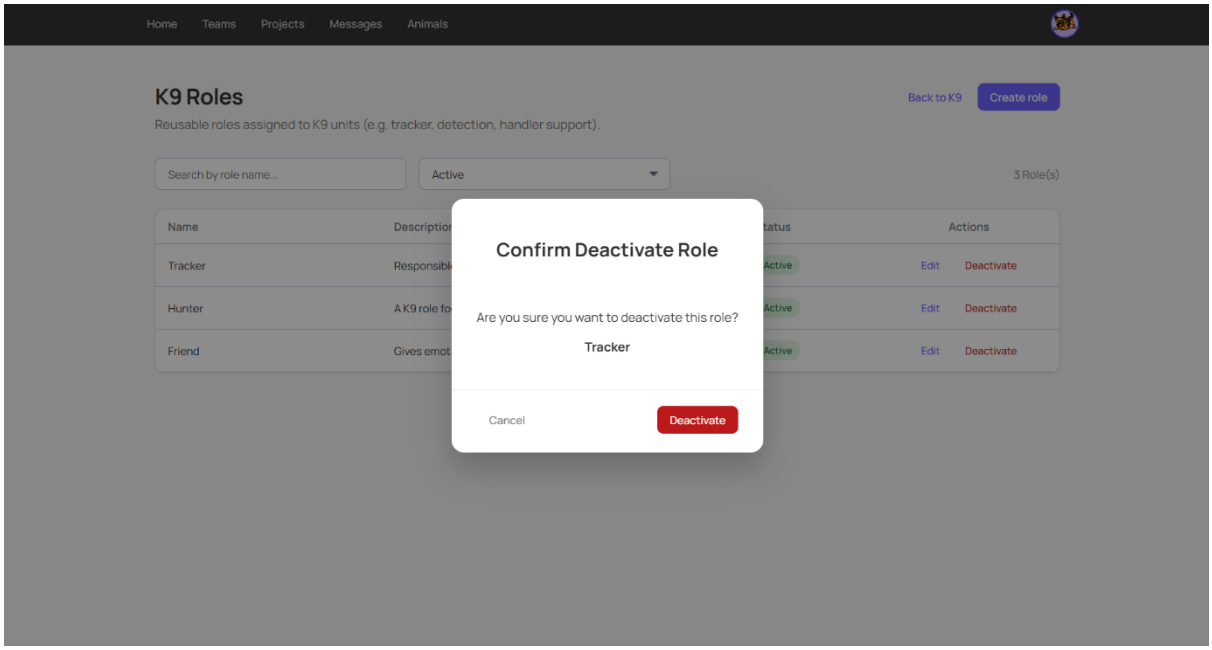


Figure 79: 8.15 Deactivate K9 Role Screen Design

2.3.7.16 View K9 Ancestry: Use Case 8.16

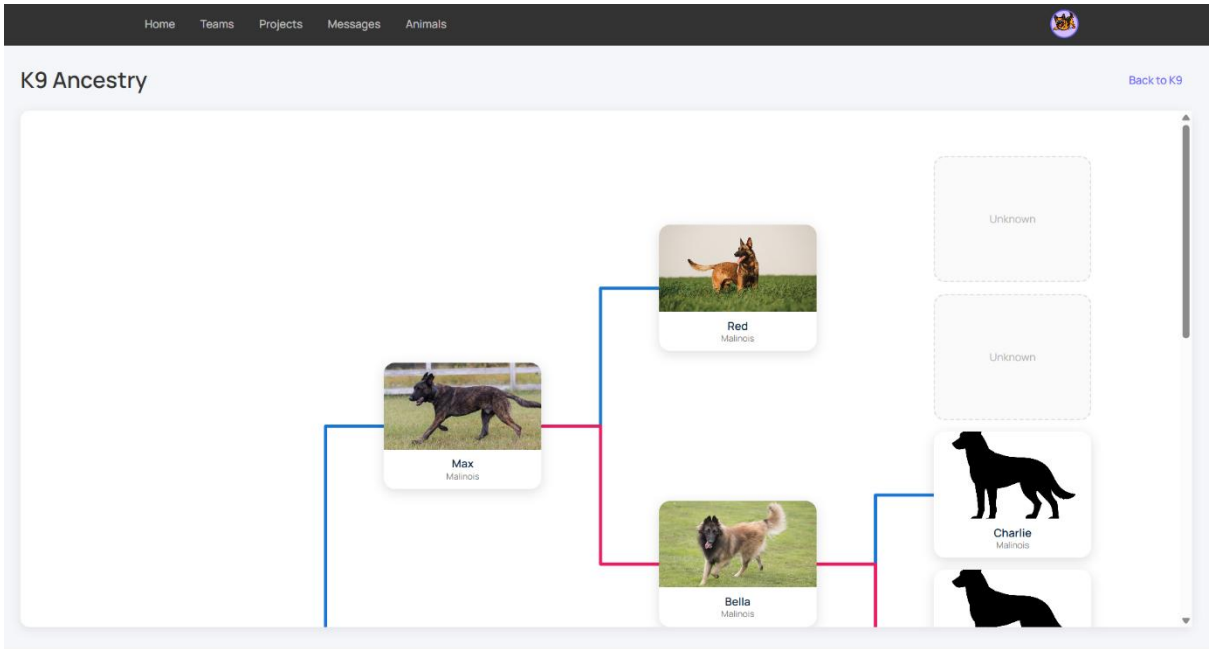


Figure 80: 8.16 View K9 Ancestry Screen Design

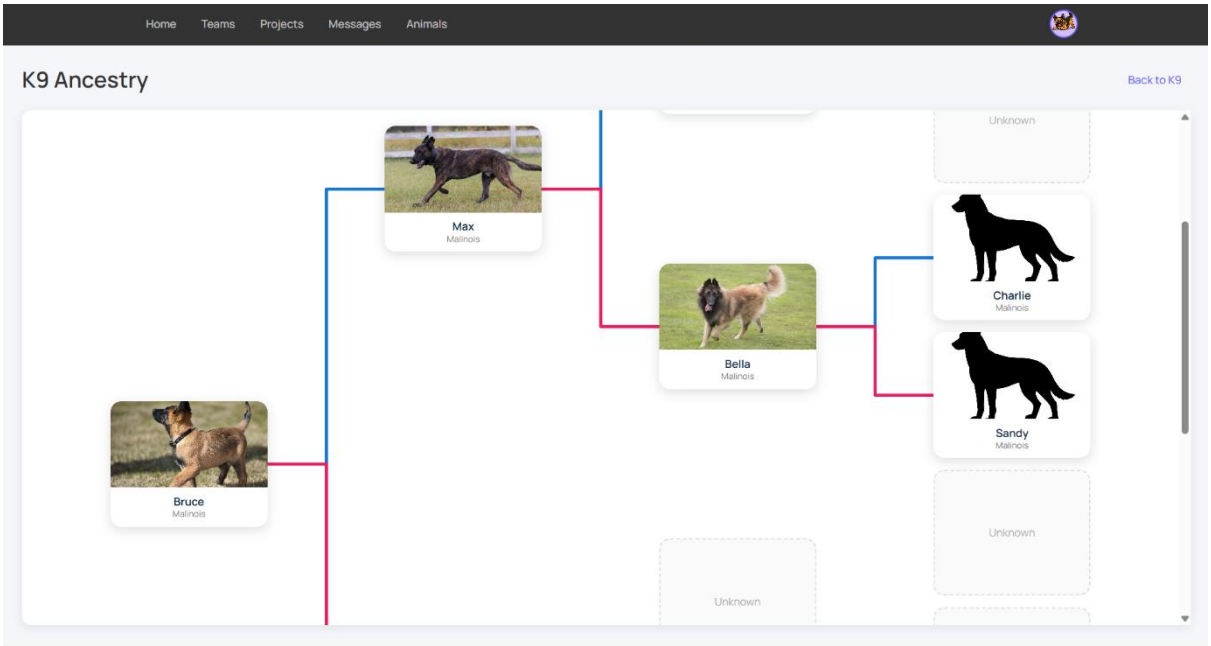


Figure 82: 8.16 View K9 Ancestry Screen Design Continued

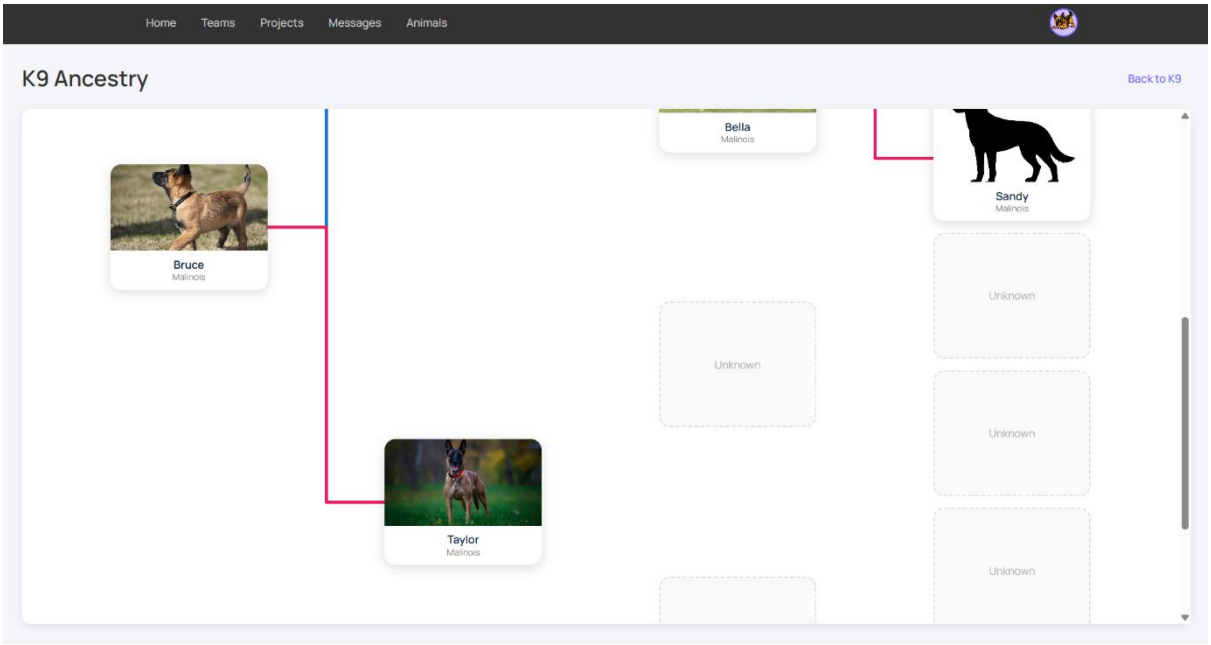


Figure 81: 8.16 View K9 Ancestry Screen Design Continued

2.3.8 Rhino Subsystem

2.3.8.1 Add Rhino: Use Case 9.1

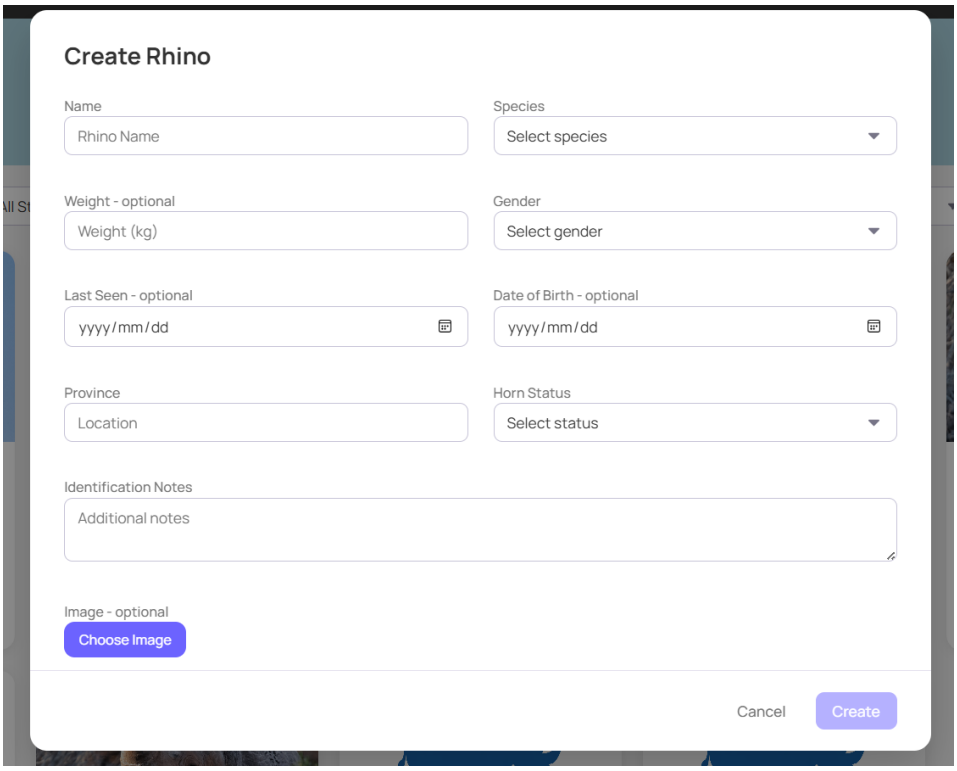


Figure 83: 9.1 Add Rhino Screen Design

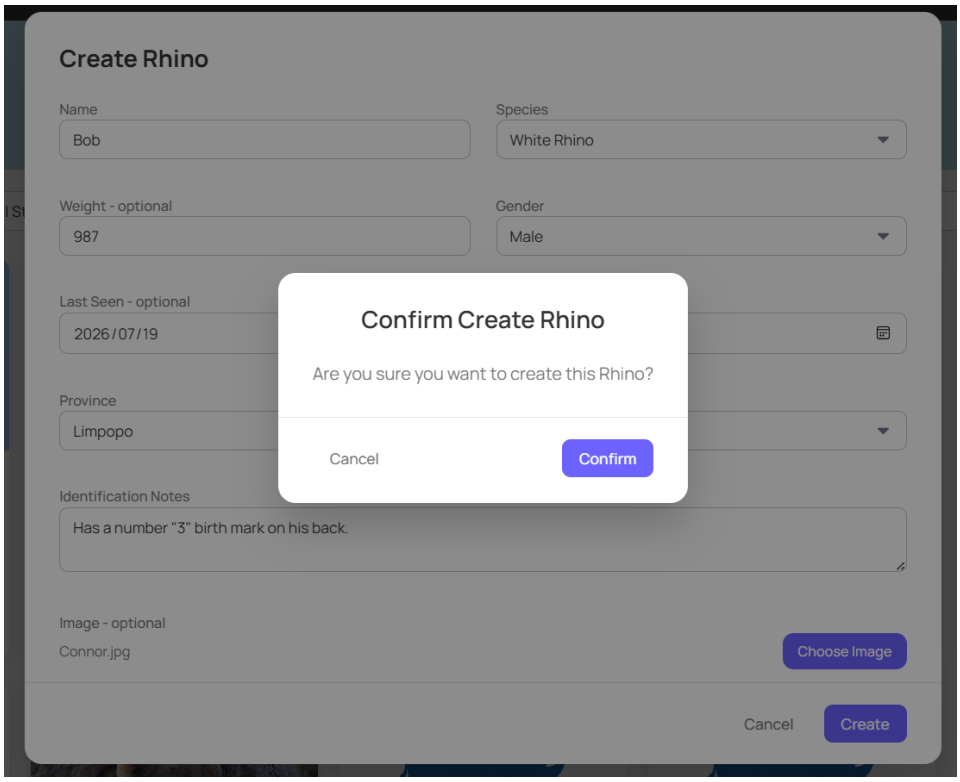


Figure 84: 9.1 Add Rhino Confirmation Screen Design

2.3.8.2 View Rhino: Use Case 9.2

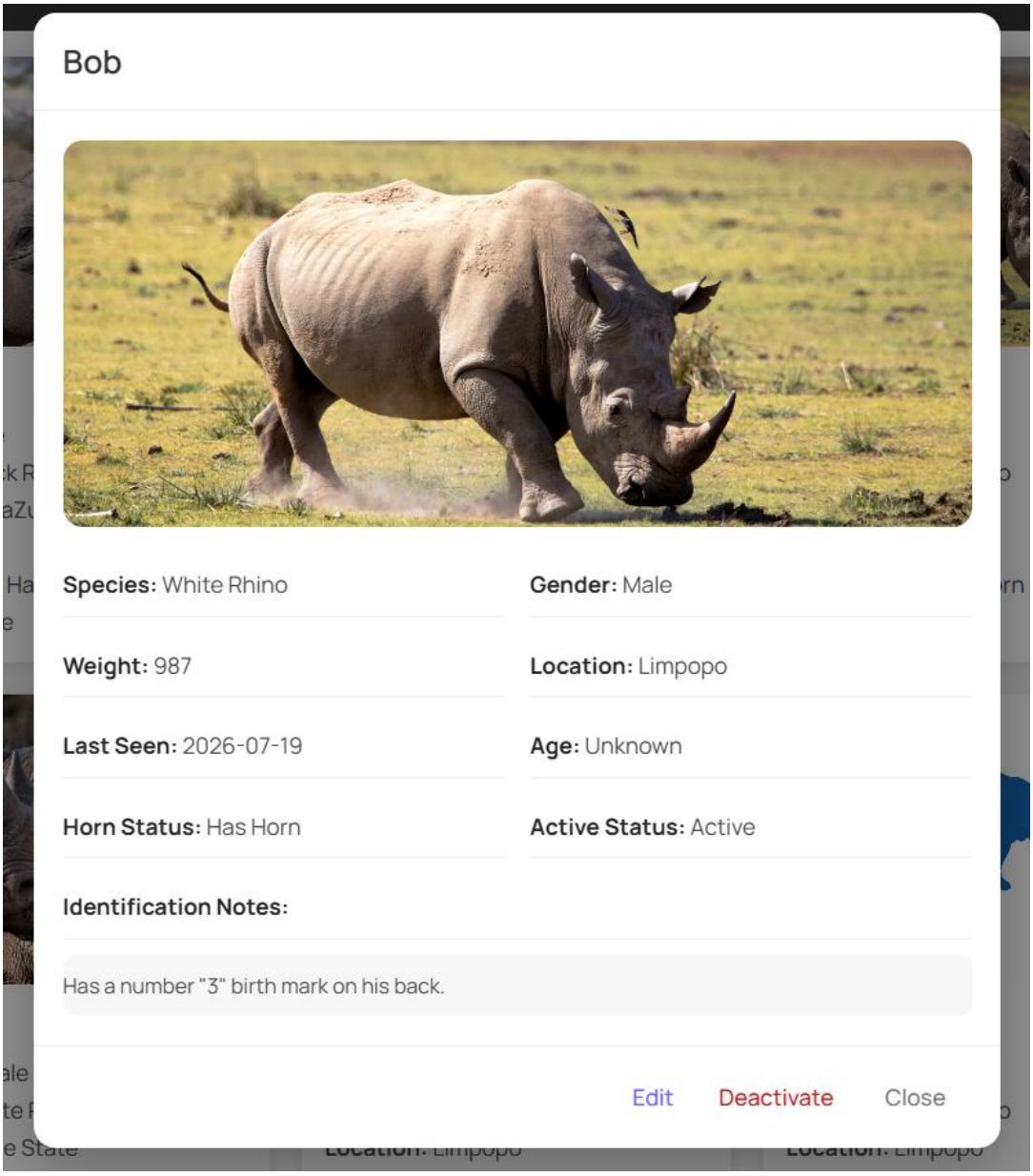


Figure 85: 9.2 View Rhino Screen Design

2.3.8.3 View Rhinos: Use Case 9.3

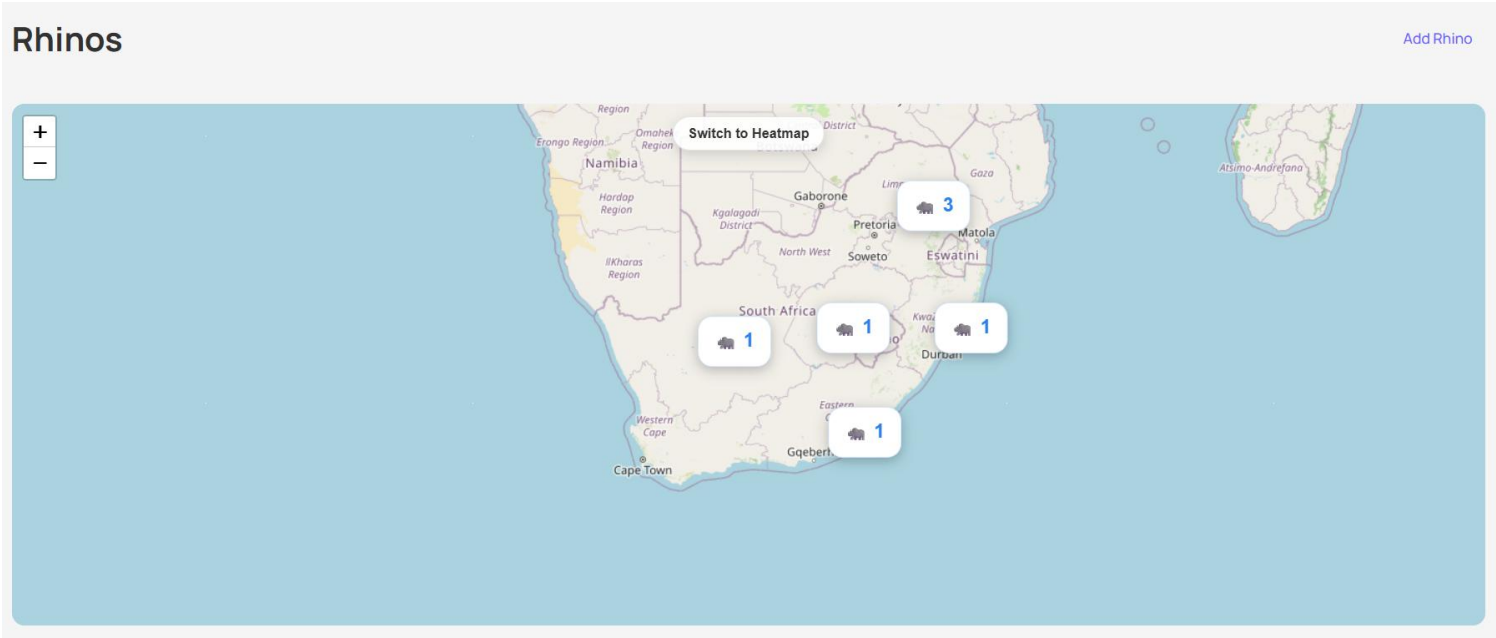


Figure 86: View Rhino Map Screen Design



Figure 87: View Rhinos Heatmap Map Screen Design

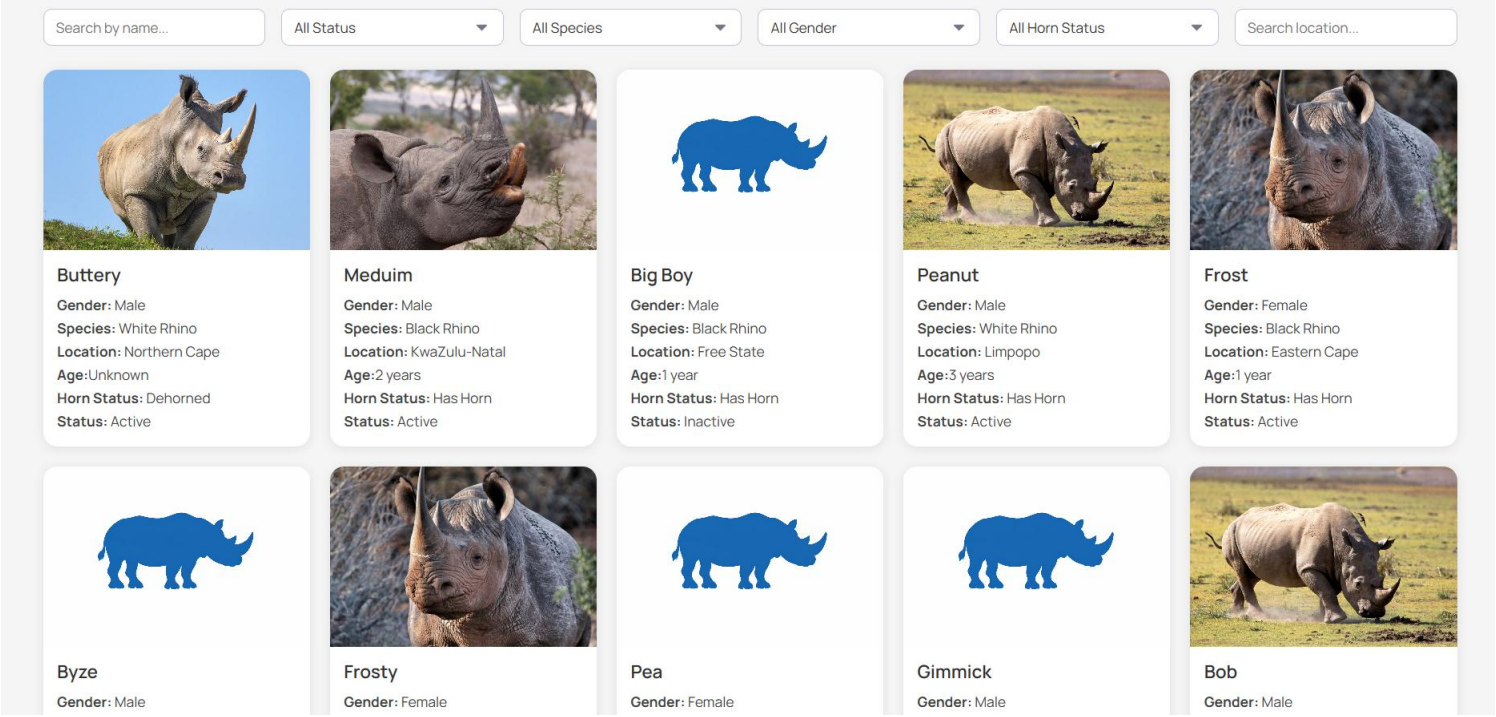


Figure 88: View Rhino Screen Design

2.3.8.4 Update Rhino: Use Case 9.4

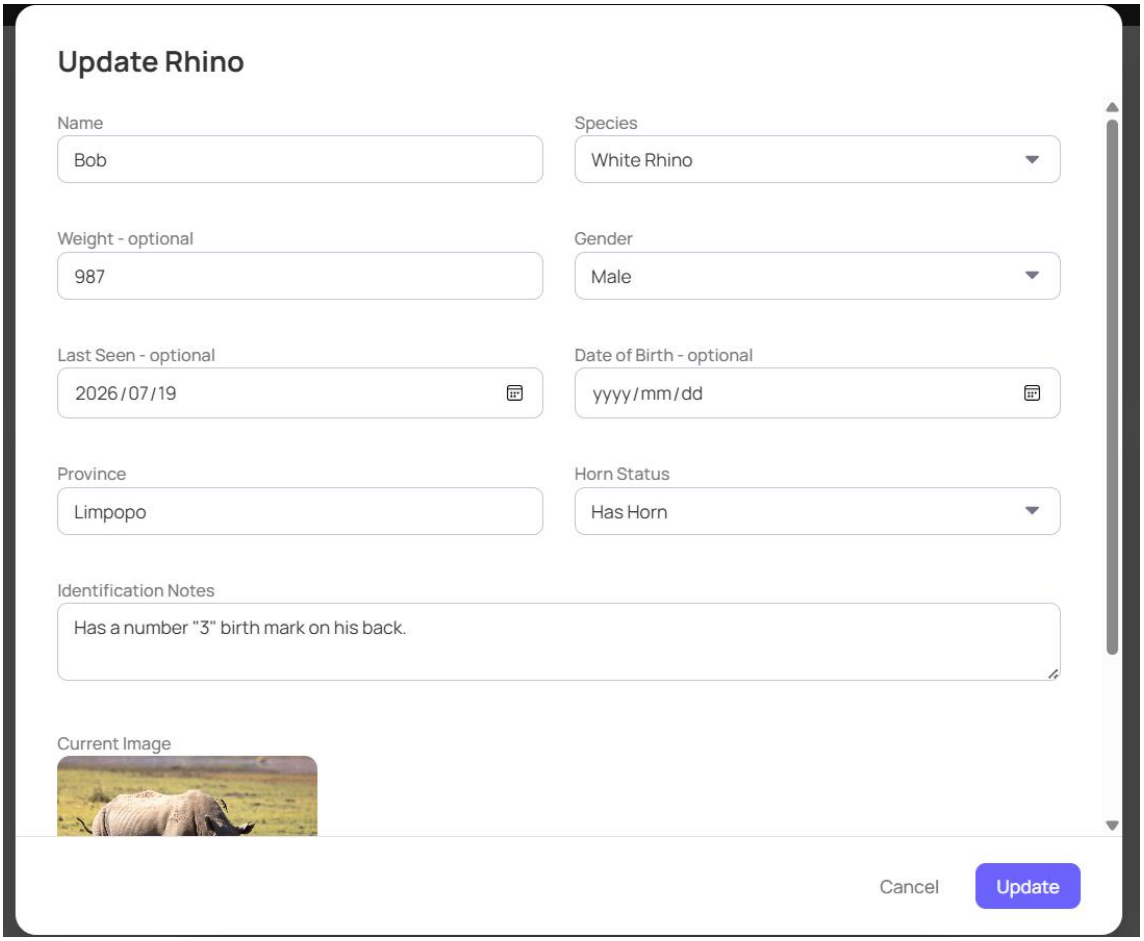


Figure 89: 9.4 Update Rhino Screen Design

2.3.8.5 Deactivate Rhino: Use Case 9.5

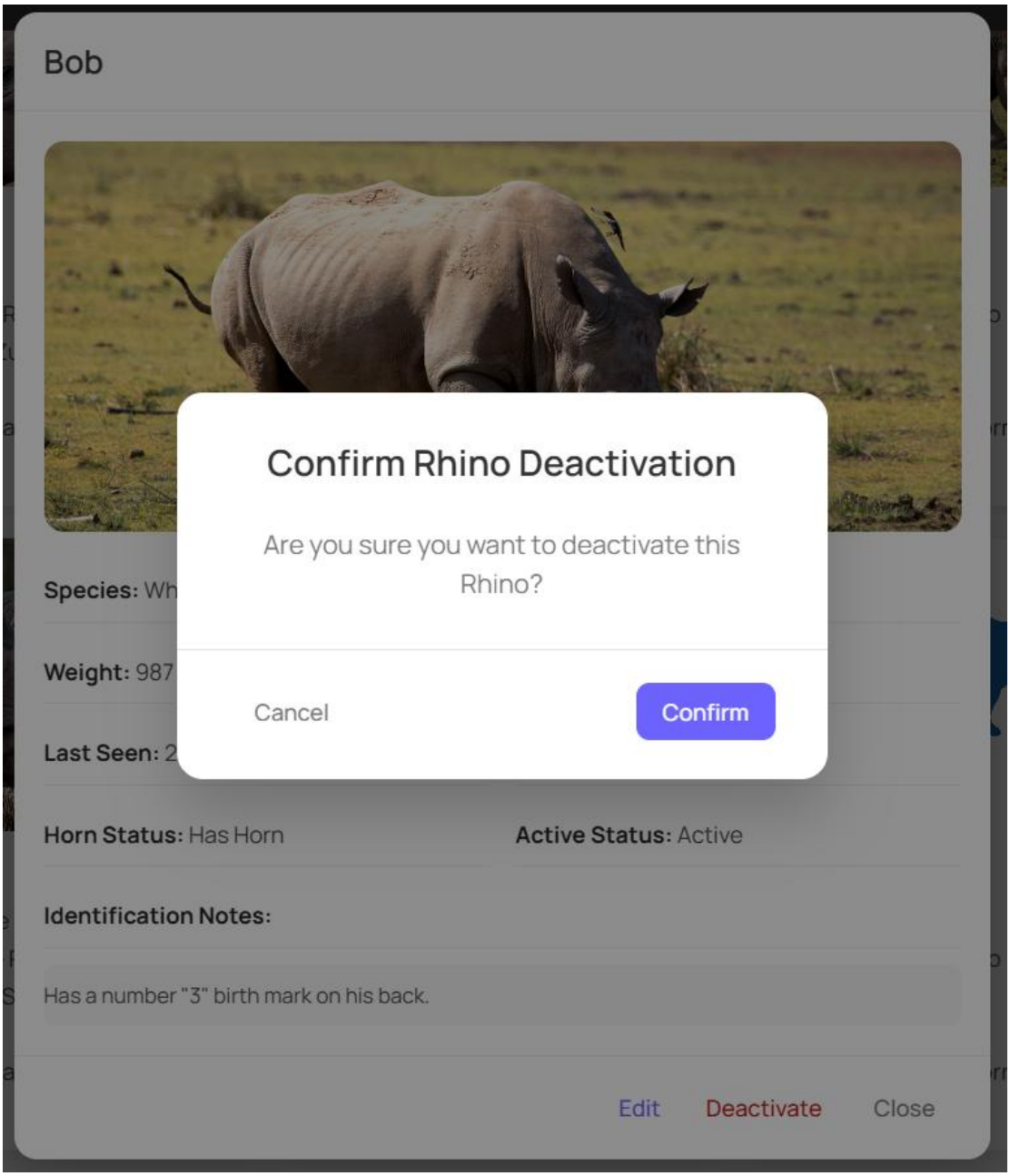


Figure 90: 9.5 Deactivate Rhino Screen Design

2.3.9 Elephant Subsystem

2.3.9.1 Add Elephant: Use Case 10.1

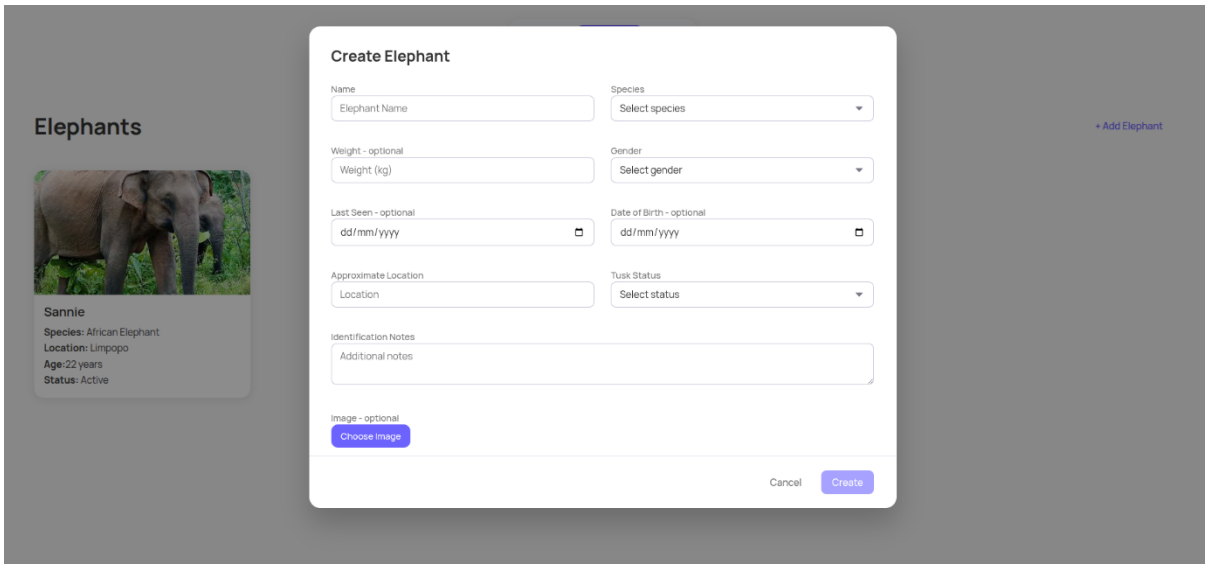


Figure 91: 10.1 Add Elephant Screen Design

2.3.9.2 View Elephant: Use Case 10.2

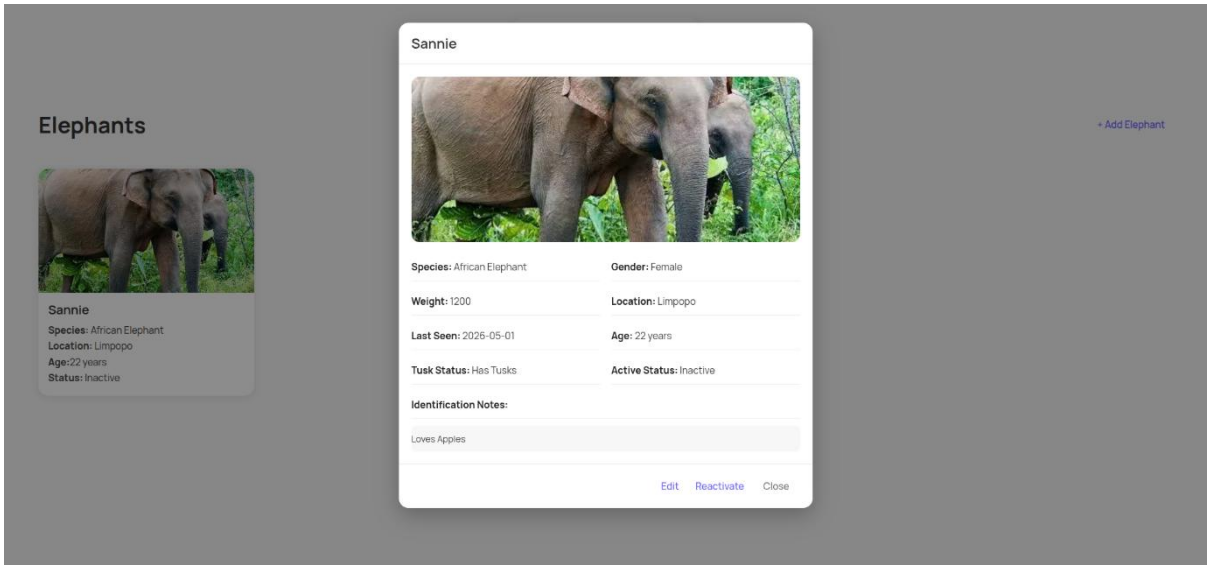


Figure 92: 10.2 View Elephant Screen Design

2.3.9.3 View Elephants: Use Case 10.3

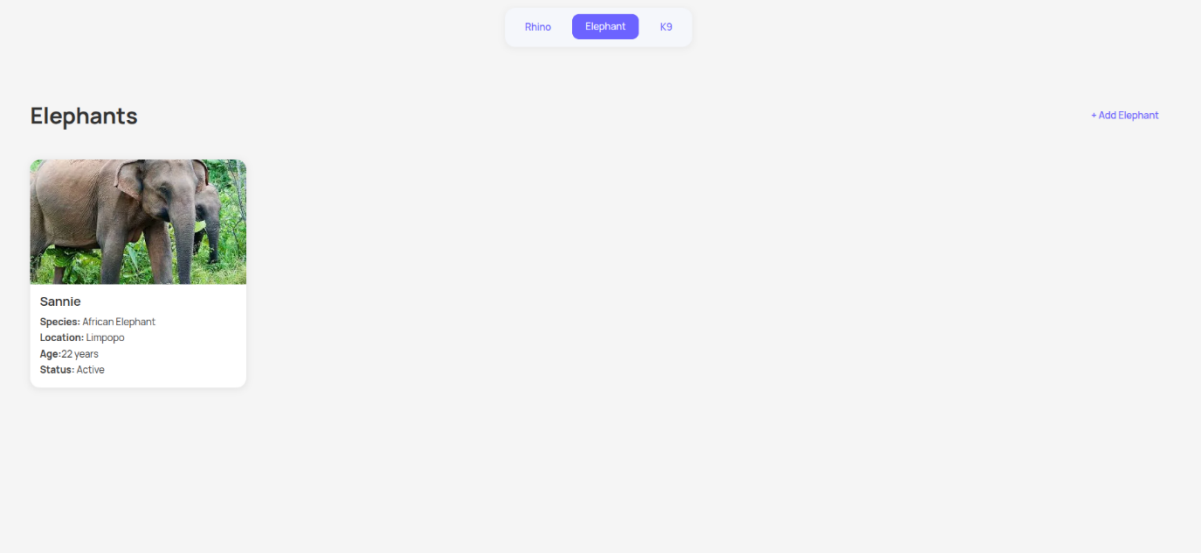


Figure 93: 5.3 View Elephants Screen Design

2.3.9.4 Update Elephant: Use Case 10.4

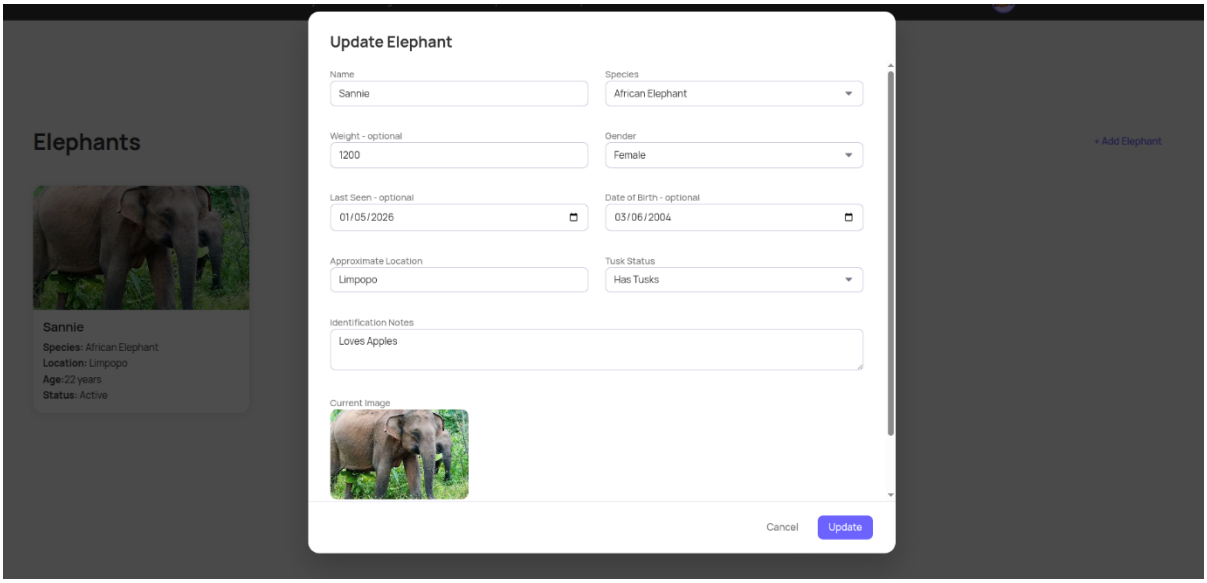


Figure 94: 5.4 Update Elephant Screen Design

2.3.9.5 Deactivate Elephant: Use Case 10.5

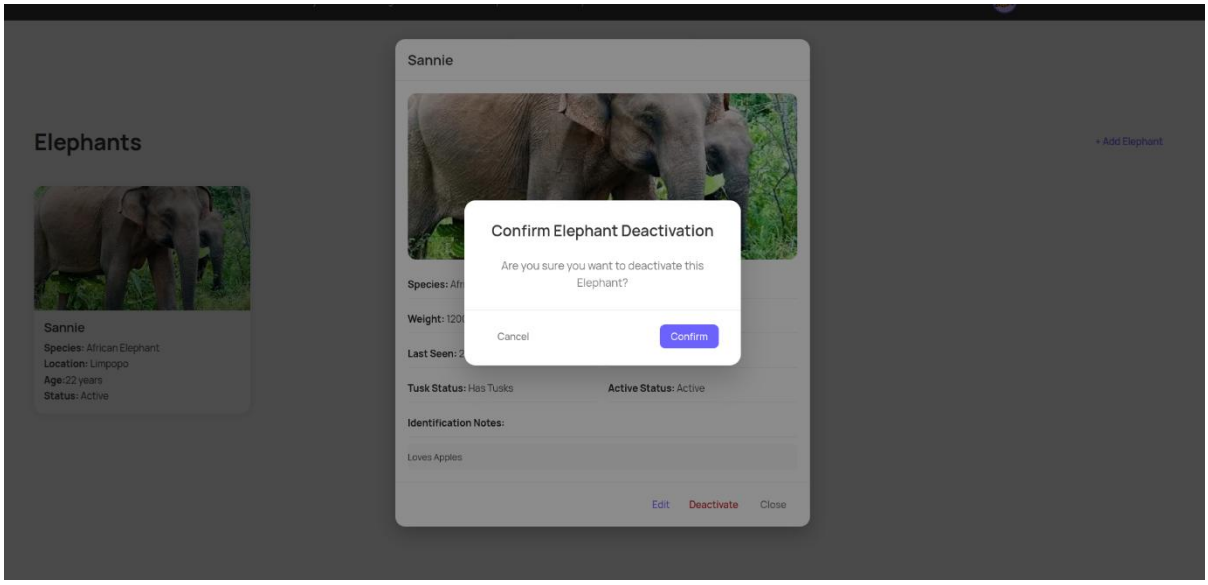


Figure 95: 10.5 Deactivate Elephant Screen Design

2.3.10 Report Subsystem

2.3.10.1 Generate Custom Report: Use Case 12.1

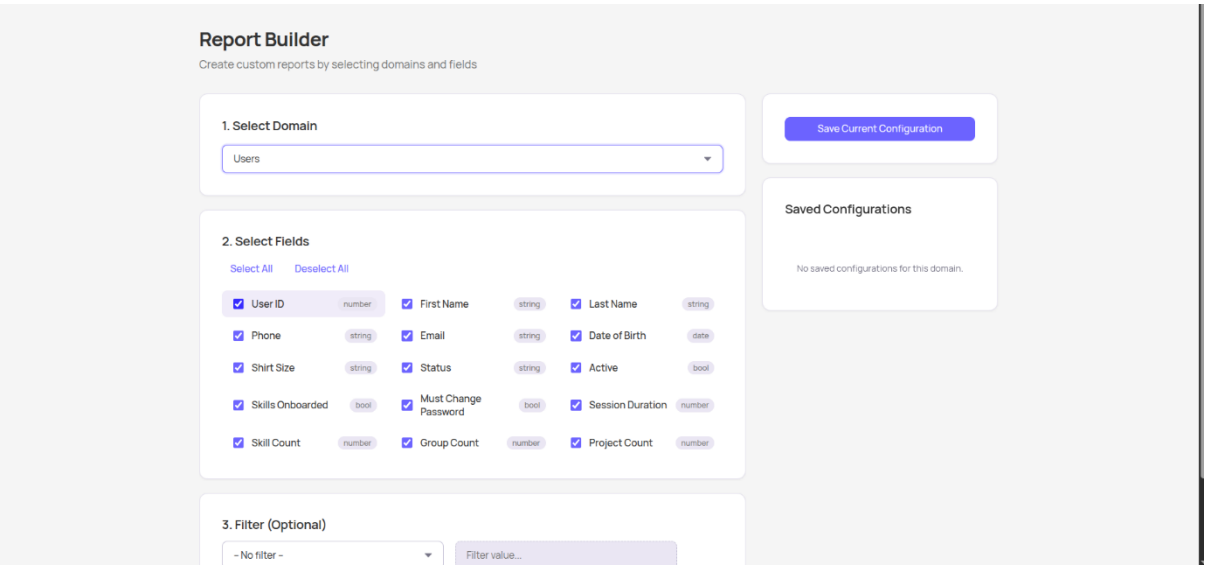


Figure 96: 12.1 Generate Custom Report Screen Design

2.3.10.2 View Custom Report t: Use Case 12.2

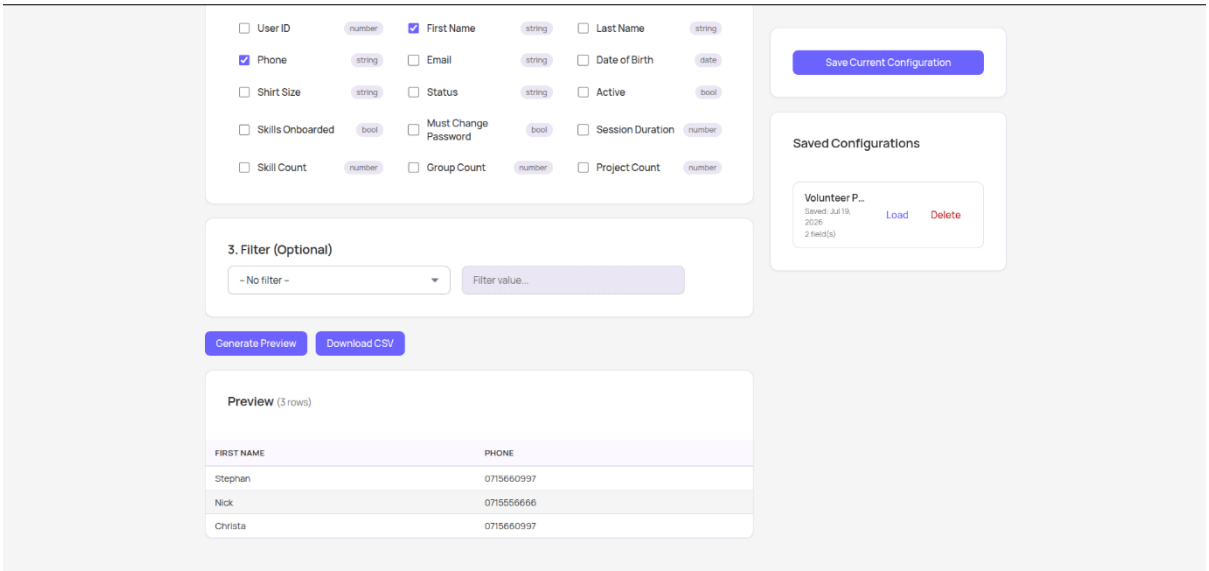


Figure 97:12.2 View Custom Report Screen Design

2.3.10.3 Update Custom Report: Use Case 12.3

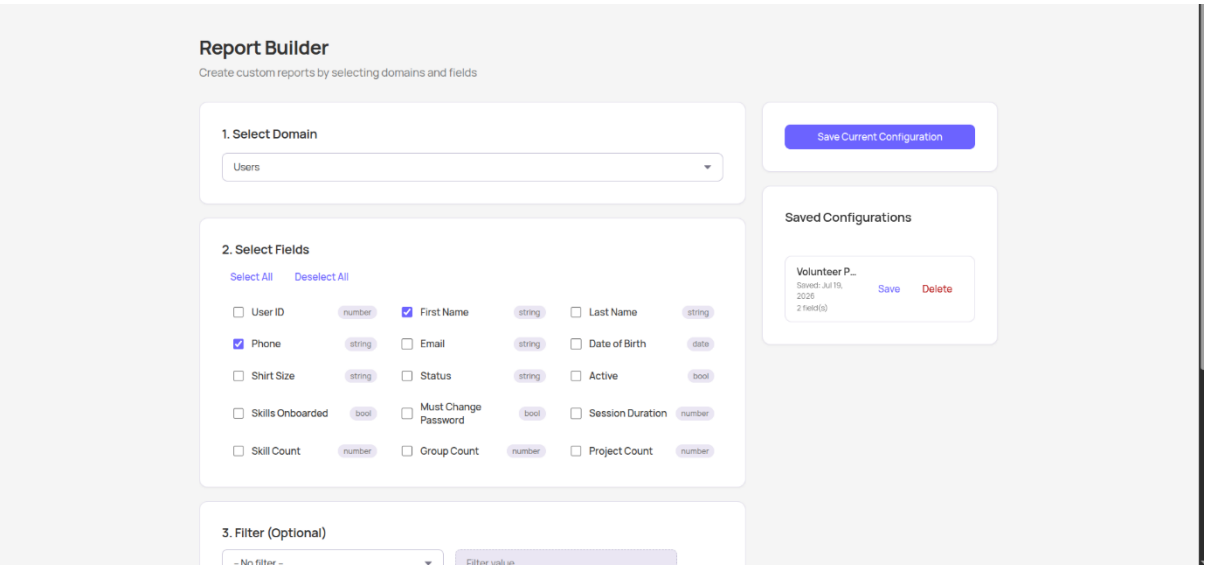


Figure 98: 12.3 Update Custom Report Screen Design

2.3.10.4 Remove Custom Report: Use Case 12.4

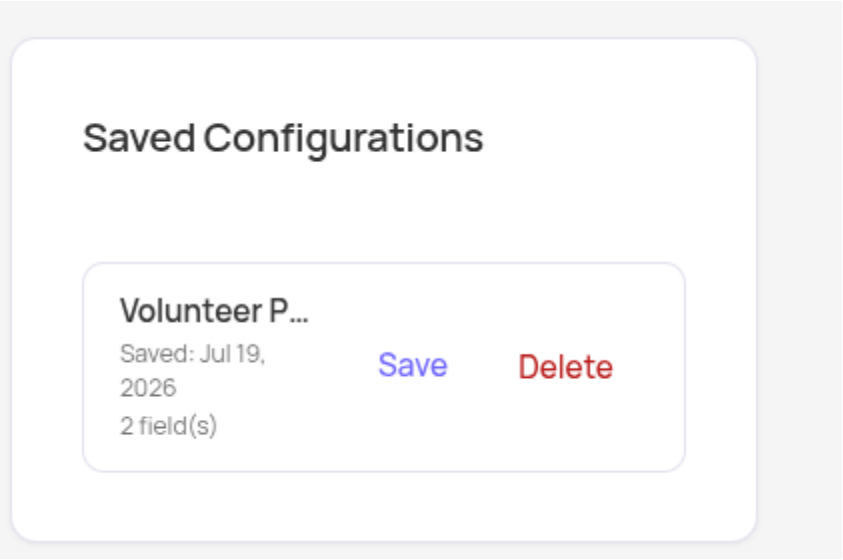


Figure 99: 12.4 Remove Custom Report Screen Design

2.3.11 Budget Subsystem

2.3.11.1 View Project Budget: Use Case 14.1

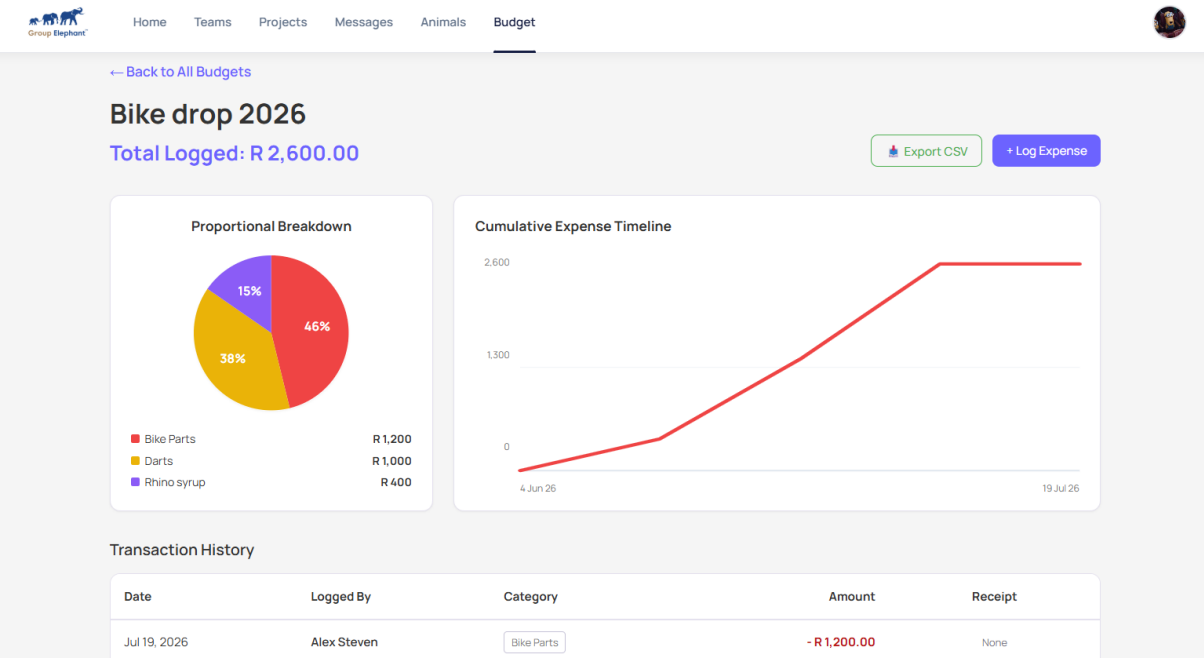


Figure 100: 14.1 View Project Budget Screen Design

2.3.11.2 View Project Budgets: Use Case 14.2

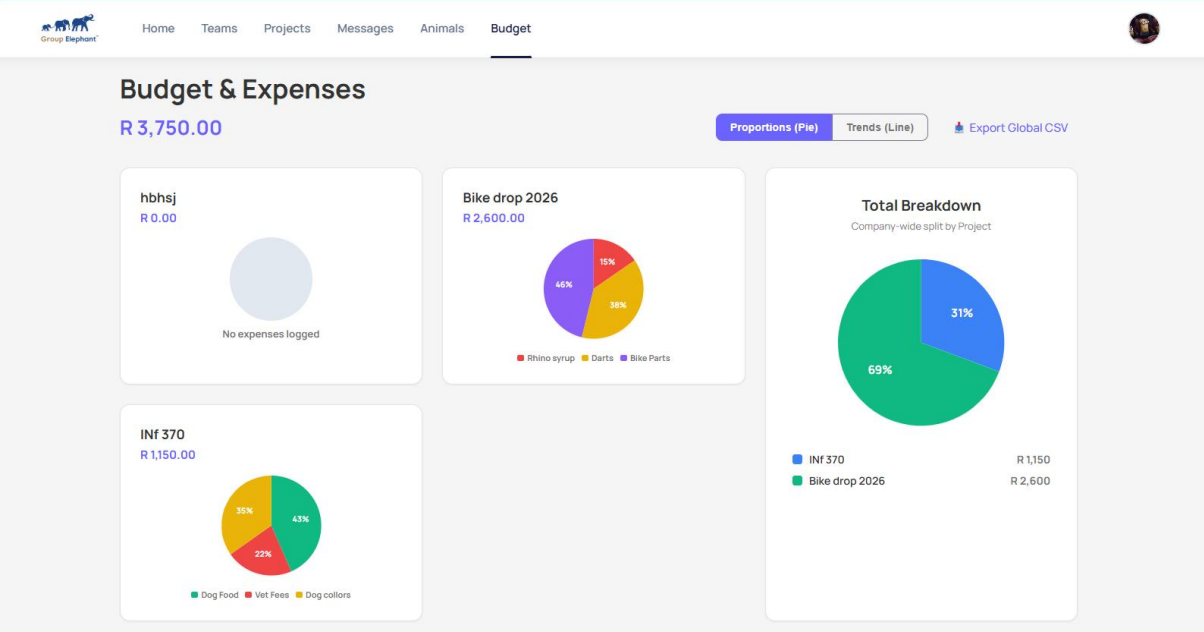


Figure 101: 14.2 View Project Budgets Screen Design

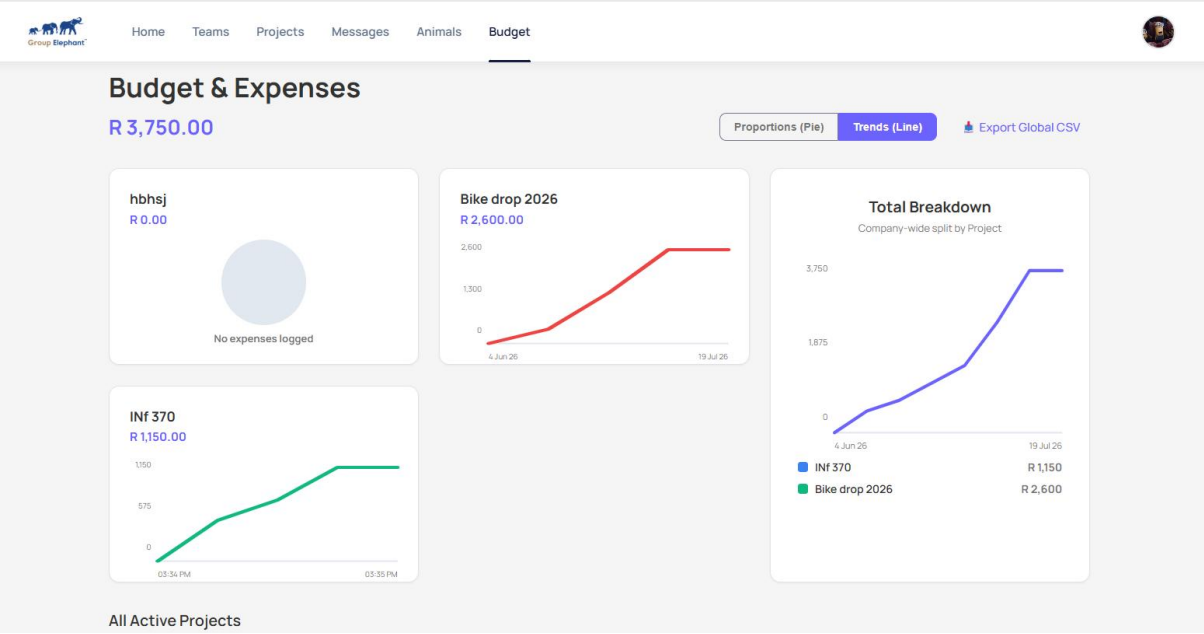


Figure 102: 14.2 View Project Budgets Screen Design Continued

2.3.11.3 Log An Expense: Use Case 14.3

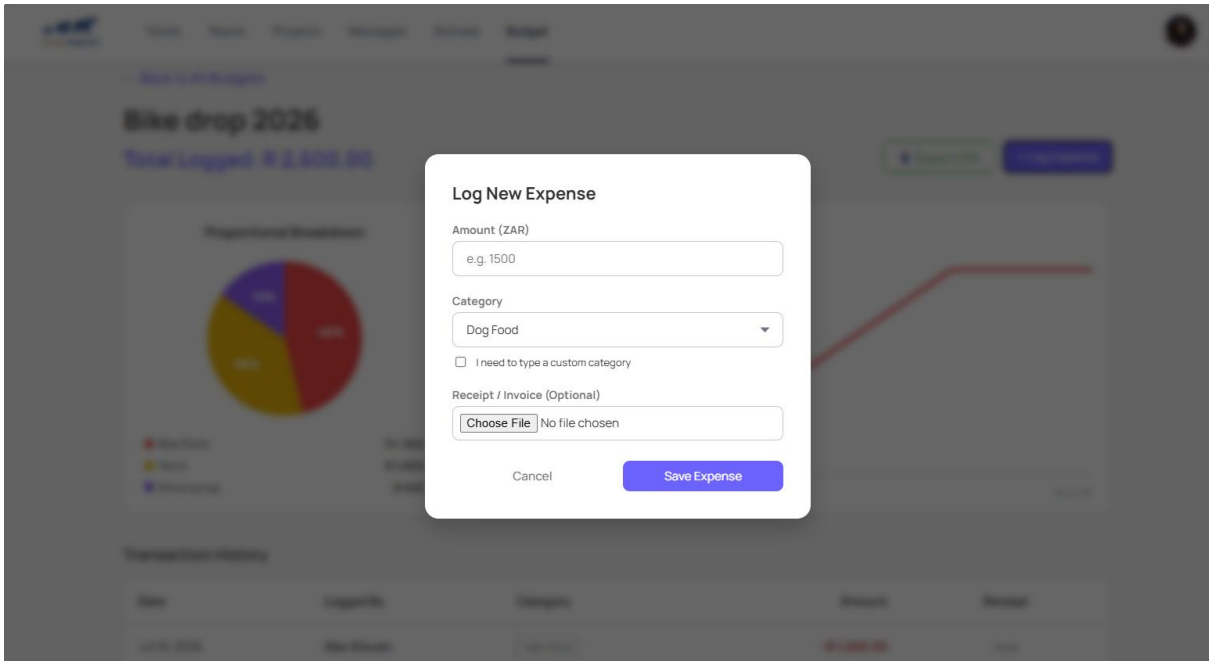


Figure 103: 14.3 Log An Expense Screen Design

2.3.11.4 Update an Expense: Use Case 14.4

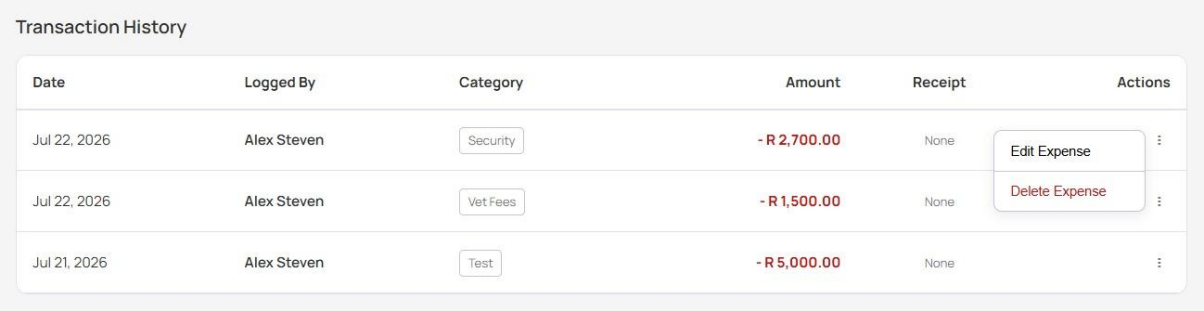


Figure 104: 14.4 Update An Expense Screen Design

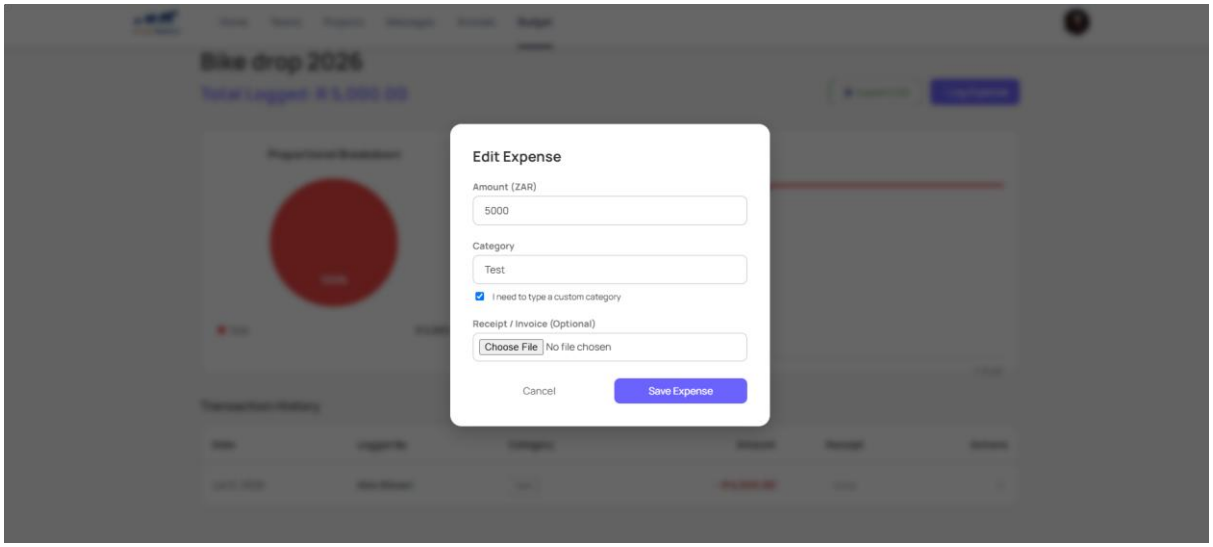


Figure 105: 14.4 Update An Expense Screen Design (Continued)

2.3.11.5 View Expense: Use Case 14.5

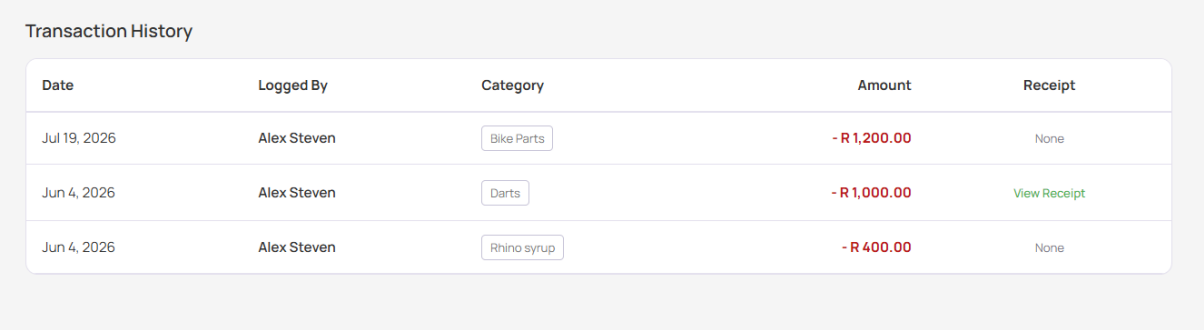


Figure 106: 14.5 View An Expense Screen Design

2.3.11.4 Remove an Expense: Use Case 14.6

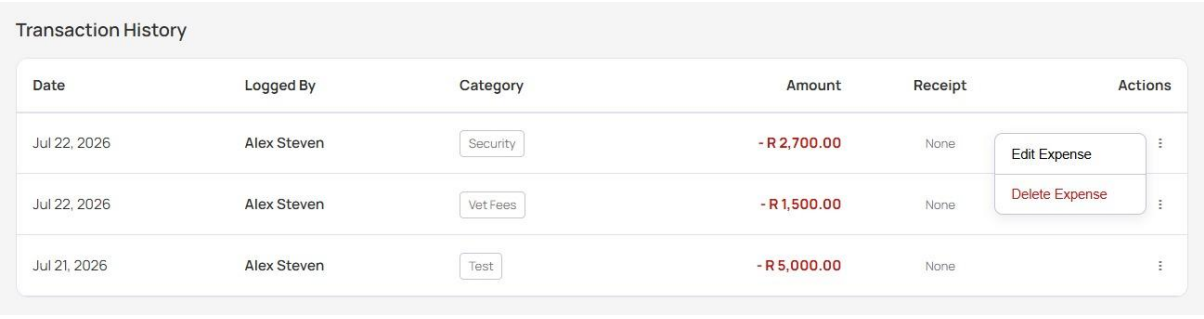


Figure 107: 14.6 Remove An Expense Screen Design

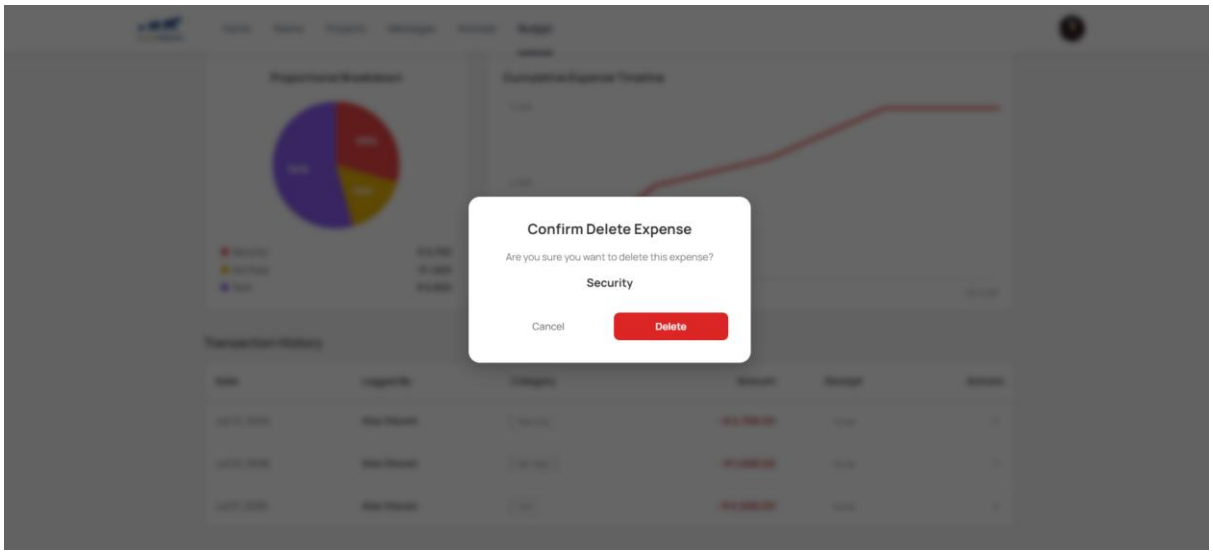


Figure 108: 14.6 Remove An Expense Screen Design (Continued)

2.4 Technical Use Case Narratives

2.4.1 User Subsystem Technical Narratives

2.4.1.1 Use Case 1.9: View FAQs

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Stephan Bredenhann</td><td colspan="2">Date: 17 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>View FAQs</td><td colspan="2" rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>1.9</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">User</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event when an Admin views the platform's frequently asked questions. After navigating to the FAQs page, the system retrieves all FAQ categories and FAQ entries and displays them as a collapsible accordion list. The Admin can narrow the list by typing a keyword or selecting a category, and can expand any entry to read its full answer along with the audit details recorded against it. No data is created, updated, or deleted.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The Admin is authenticated and at least one FAQ category exists in the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The Admin navigates to the FAQs page, either by entering the URL directly or by selecting the FAQs card on the Dashboard.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: Admin navigates to the FAQs page.</td><td></td><td>Step 2: The system loads the FAQs screen and calls “GET api/faqcategory/categories”, which retrieves all FAQ categories from the FAQCategory entity in alphabetical order. The categories populate the filter</td></tr>
</table>

			dropdown, with "All" selected by default. [ALT]
			Step 3: The system calls "GET api/faq", which retrieves every FAQ by joining the FAQ and FAQCategory entities. No filtering, sorting, or paging is applied to the query. [ALT]
			Step 4: The system displays the FAQs page with a search field, a category dropdown, and the accordion list. Each collapsed entry shows a category badge and the question text.
	Step 5: Admin types a keyword into the search field.		**Step 6:** The system filters the list to FAQs whose question or answer contains the keyword and re-renders the accordion. Filtering is performed on the records already retrieved, so no further request is made. [ALT]
	Step 7: Admin selects a category from the dropdown. [ALT]		**Step 8:** The system filters the list to FAQs belonging to the selected category. If a keyword has also been entered, both conditions are applied together. [ALT]
	Step 9: Admin selects an FAQ entry.		**Step 10:** The system expands the entry to show the full answer and an audit footer recording the user who created the FAQ and the date it was created. Selecting the entry again collapses it.
Alternate courses:	[ALT] Step 2: The FAQ categories cannot be retrieved. The system displays "Failed to load FAQ categories." and the category dropdown remains empty.		
	[ALT] Step 3: The FAQs cannot be retrieved. The system displays "Failed to load FAQs." and the accordion list remains empty.		

	[ALT] Step 7: Admin selects "All" from the dropdown. The category condition is bypassed and FAQs from every category are shown, subject to any keyword entered.
	[ALT] Step 6/8: No FAQs match the keyword or the selected category. The system displays "No FAQs match your filters." in place of the accordion list, and the search field and dropdown remain available.
Conclusion:	The use case concludes when the FAQ list is displayed to the Admin, filtered as required, with the selected entries expanded.
Post-condition:	No data is modified. The keyword and category selections remain in effect until the Admin navigates away from the page.
Business rules:	• Only authenticated users can view the FAQs page. • Search is case-insensitive and is applied to both the question and the answer text. • A keyword search and a category selection are applied together when both are active. • Categories are listed alphabetically by name, and FAQs are listed in the order they were created. • Multiple FAQ entries may be expanded at the same time.
Implementation constraints and specifications:	• "FAQ entries are keyboard accessible and expose their expanded state to screen readers.
Assumptions:	• The default FAQ categories are present in the database after installation. • The Admin's browser supports the styling and scripting features used by the page.
Open issues:	• Searching and filtering are performed on the retrieved records, so the entire FAQ table is loaded on every visit.

Table 1: Technical Narrative 1.9 View FAQs

2.4.2 Admin Subsystem Technical Narratives

2.4.2.1 Use Case: 2.5 Add FAQ

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 12 June 2026 **Version: 1.0**	
Use case name:	Add FAQ	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	2.5		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Admin		
Description:	This use case details the step-by-step process where a Super Admin establishes a new Frequently Asked Question entry within the system directory. The operator inputs a targeted category, question, and rich-text solution statement into an Angular Reactive Form layout. Upon clicking submit, the client interface checks form validation criteria and dispatches an asynchronous JSON payload to the corresponding backend .NET 8 API router (POST api/Faq/add). The backend verifies text string properties against duplicate entries before leveraging Entity Framework Core to write the record to the relational database tables. The use case ends when the frontend updates local state array caches, dismisses the editing overlay window, and renders a green success confirmation toast.		
Pre-condition:	• The Super Admin must hold appropriate system configuration management permissions, and at least one functional FAQ classification type category must exist in the database partitions.		
Trigger:	The Super Admin clicks the "Create FAQ" action button trigger element layout component.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The Super Admin requests to create a new FAQ by clicking the "Create FAQ" action trigger button.		**Step 2:** The Angular Router intercepts the route request and initializes the FAQ Creation Component. The framework processes the HTML layout context and updates the DOM elements.

		Step 3: The system displays the user interface window presenting an open Angular Reactive Form Group layout containing an unpopulated category select dropdown array, an empty text input field for the question, and a rich text textarea field for the answer details.	
	Step 4: The Super Admin selects the relevant category, inputs the question text content, captures the detailed answer content, and clicks the primary "Commit Records" submit button. [ALT]		**Step 5:** The client-side application captures the structural data state. The Angular FAQService constructs and dispatches an asynchronous HTTP POST request to the backend API endpoint string api/Faq/add enclosing an AddFaqRequest JSON payload container.The .NET 8 API FaqController intercepts the incoming network payload stream. The backend runtime references the assigned ApplicationDbContext instance and fires an asynchronous verification lookup query against the SQL table partitions via Entity Framework Core to verify that an identical duplicate question is not already saved in that target category mapping.

			Step 6: The backend server engine executes core validation parameters against incoming JSON parameters before authorizing transaction storage updates. The framework performs model validation checks to confirm mandatory rows are present, verify character array length thresholds, map explicit category lookups, and validate Super Admin security credentials.
			Step 7: Upon successful logic validation confirmations, the system writes the new entity record tracking state directly into the DbContext Faq model array collection and executes an asynchronous storage save block routine command (SaveChangesAsync)
			Step 8: The API data pipeline transmits an HTTP 201 Created confirmation response block stream back to the calling client. The Angular frontend catches the execution status code, modifies the local application state FAQ cache arrays, shuts down the UI modal overlay display layers, and fires a green success toast alert saying: "FAQ added successfully."
Alternate courses:	**[ALT] Step 4:** The user chooses to cancel the action and clicks the "Discard" button. The frontend interface closes the editing modal panel immediately, drops all form mutations from active memory buffers, and prevents network calls to the server.		

Conclusion:	The use case concludes once all data rules clear validation, records register successfully inside the database backend tables, form inputs clear, and dashboard lookup lists update.
Post-condition:	• A new FAQ entry is permanently written to the database storage layer, immediately matching it against its target category index for end-user filter selection workflows.
Business rules:	• Only users with explicitly assigned Super Admin security authorization claims can insert new FAQ entities into global datasets. • The system blocks the generation of any new FAQ record that contains an identical question text payload within the exact same category mapping.
Implementation constraints and specifications:	• Alphanumeric text data mapping to the question input field parameters must not exceed a size floor ceiling threshold of 250 characters.
Assumptions:	• None
Open issues:	• None

Table 2: Technical Narrative 2.5 Add FAQ

2.4.2.2 Use Case: 2.6 Update FAQ

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Amahle Khoza</td><td>Date: 12 June 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td colspan="2">Update FAQ</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">2.6</td></tr>
<tr><td>Priority:</td><td colspan="2">High</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">• Super Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">Admin</td></tr>
<tr><td>Description:</td><td colspan="3">This use case details the step-by-step process for modifying an existing FAQ entry's category index, question text, or rich-text solution scope details. The user adjusts configuration parameters within an active frontend reactive layout modal. Upon clicking save, the client interface checks field validity maps and dispatches an asynchronous PUT request to the .NET 8 API endpoint (PUT api/Faq/update/{id}). The backend locates the target entity row via Entity Framework Core tracking contexts and overwrites its structural schema properties with the new values. The use case finishes once mutations write safely to disk, modal popup blocks close, and lookup views across the application refresh.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">• The targeted FAQ entry record must exist in the database partition, and the user must hold necessary administrative configuration write claims.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The user clicks the context "Edit" button layout element located next to a specific FAQ data row line item.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The Super Admin requests to edit an existing FAQ entry row item by hitting the specific contextual "Edit" action button trigger.</td><td></td><td>Step 2: The frontend application catches the click event and launches the edit workflow modal dialog display framework. The Angular service dispatches an automated background HTTP GET request to retrieve the latest backend database metrics.</td></tr>
</table>

		Step 3: The UI component parses the incoming data model values and executes an explicit reactive form patch configuration override command (patchValue) to populate the text inputs automatically with current record attributes for user modification.	
	Step 4: The Super Admin navigates across the active form properties, updates the targeted description inputs or category mapping lists, and hits the "Save Changes" execution control button. [ALT]		**Step 5:** The client framework assesses form validity parameters on the client layer. The Angular FAQService packages the modifications and sends an HTTP PUT request to the route path index api/Faq/update/{id} wrapping the UpdateFaqRequest body details. The .NET API controller intercepts the transmission stream, activates its ApplicationDbContext tracking instance, and invokes an asynchronous data lookup query statement (SingleOrDefaultAsync) against the "Faq" entity table to locate the specific primary key row index.
			Step 6: The system backend architecture executes

			model integrity checking algorithms against the incoming payload to verify the record identity exists, assert string boundary lengths, and ensure the session holds active Super Admin authorisation claims [ALT].
			Step 7: Upon validation success, the system maps the payload attributes over the tracked entity data properties using the ORM tracker and signs off database table record modification updates via an asynchronous save update call (SaveChangesAsync).
			Step 8: The web API returns an HTTP 200 OK completion confirmation payload. The Angular frontend catches the message code, overrides the stale object instance within its active data array cache, removes the modal view wrapper from the DOM, and fires a green success notification alert box reading "FAQ updated successfully."
Alternate courses:	**[ALT] Step 4:** The user aborts the workflow and clicks the "Discard" or "Cancel" button. The frontend interface closes the editing panel immediately, drops all form mutations from memory buffers, and prevents network data calls to the server.		
	[ALT] Step 6: The data validation layer discovers a naming or question text conflict matching another active entry row or detects that the target index has been deleted from memory. The API stops the process pipeline and returns an error response code (HTTP 409 Conflict). The client application catches the exception tracking details and displays a clear red validation error message banner.		

Conclusion:	The use case concludes when the system successfully validates the updated data parameters, saves the transaction metrics to database storage files, and refreshes the main grid layout view.
Post-condition:	• The target FAQ instance's database entity fields are permanently updated with the new text values, ensuring altered parameters reflect across downstream interface views.
Business rules:	• Only users with explicitly assigned Super Admin configuration privileges can modify system FAQ entries.
Implementation constraints and specifications:	• Field modifications targeting the primary question string must strictly maintain compliance with the 250-character threshold limit.
Assumptions:	• None
Open issues:	• None

Table 3: Technical Narrative 2.6 Update FAQ

2.4.2.3 Use Case: 2.7 Remove FAQ

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 12 June 2026 **Version: 1.0**	
Use case name:	Remove FAQ	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	2.7		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Admin		
Description:	This use case details the structural steps executed when a Super Admin deletes an FAQ entry completely from persistent database systems. The user triggers the action from the row context. The frontend generates an explicit validation warning alert overlay modal. Upon confirmation, the client initiates an asynchronous HTTP DELETE transaction directly to the corresponding backend .NET 8 API controller routing path (DELETE api/Faq/remove/{id}). The backend executes validation constraint assessments before completely dropping the targeted row record index from persistent tables via Entity Framework Core context trackers. The use case concludes when the server commits changes, and the client removes the targeted entry from local data grid layers.		
Pre-condition:	• The targeted FAQ tracking record must currently exist inside persistent SQL tables with an active index match.		
Trigger:	The user clicks the specific row deletion contextual action "Delete" icon trigger element layout asset.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The Super Admin requests to drop an FAQ tracking item line row by clicking its specific row deletion contextual action icon trigger.		
		Step 2: The frontend system halts current layout sequences and displays a modal	

		confirmation warning overlay dialog framework, presenting a descriptive message text to the operator: "Are you sure you want to permanently delete this FAQ item? This action cannot be undone."	
	Step 3: The Super Admin assesses the warning text context and hits the primary confirmation "Confirm Delete" action switch inside the modal overlay. [ALT]		**Step 4:** The frontend interface captures the user click intent. The Angular FAQService formats an asynchronous HTTP DELETE request command routing directly to the backend endpoint URL parameter path map api/Faq/remove/{id}. The backend web API intercepts the data deletion stream request. The data runtime environment establishes connections to the ApplicationDbContext unit tracking context and runs an execution query sweep targeting the row index matching the supplied ID property.
			Step 5: The backend system subjects the command parameters to server constraints to verify database row existence and match current user claims data models against required administrative permission sets.
			Step 6: The server marks the identified entity row record for deletion within

			the data set tracker context and permanently drops the row record from persistent storage tables via an asynchronous change statement execution (SaveChangesAsync).
			Step 7: The API transfers an HTTP 200 OK or HTTP 240 No Content message tracking packet block. The client application interceptors catch the response, splice the target item index out of its local collection layout arrays, dismiss the alert modal wrapper layer, and render a green toast confirmation message: "FAQ deleted successfully."
Alternate courses:	**[ALT] Step 3:** The user chooses to cancel the action and clicks the "Cancel" button within the warning dialog block layout. The confirmation modal overlay closes immediately, the workflow cancels, and the FAQ's database status remains completely unchanged.		
Conclusion:	The use case concludes once all data rules clear validation, records drop successfully from database tables, the confirmation dialog closes, and dashboard views refresh to reflect the deletion.		
Post-condition:	• The targeted FAQ entry row is permanently dropped from persistent relational database storage maps, rendering the tracking parameters unrecoverable.		
Business rules:	• Hard-deletion commands targeting FAQ database items are restricted strictly to users with active Super Admin security claims to safeguard data dictionary integrity.		
Implementation constraints and specifications:	• The operation must execute sequentially over an asynchronous pipeline to prevent UI blocking during processing loops.		
Assumptions:	• None		
Open issues:	• None		

Table 4: Technical Narrative 2.7 Remove FAQ

2.4.2.4 Use Case: 2.8 Add FAQ Type

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 24 June 2026 **Version: 1.0**	
Use case name:	Add FAQ Type	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	2.8		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Admin		
Description:	This use case describes the technical steps required to establish a new category label within the FAQ classification lookups system. The user inputs category metrics into an Angular Reactive Form layout overlay modal. Upon clicking submit, the frontend checks form control validation syntax states and dispatches a JSON payload to the corresponding .NET 8 API backend endpoint (POST api/FaqType/create) packing a structured CreateFaqTypeRequest body. The backend processes the incoming package payload context and uses Entity Framework Core to write the new classification record to the relational database tables. The use case finishes once the frontend updates local state lookup arrays, clears active modal containers, and triggers a green confirmation toast notification.		
Pre-condition:	• The executing user must hold appropriate system configuration management privileges, and the system must be deployed ahead of testing cycles.		
Trigger:	The user clicks the "Create FAQ Type" action button.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The Super Admin requests to create a new FAQ category grouping by clicking the "Create FAQ Type" action trigger button.		
		Step 2: The system launches	

		the category generation modal layer overlay. The frontend generates and renders an open Angular Reactive Form Group interface layout containing an alphanumeric text input field for the FAQ Type Name, a text area for the Description Scope, an option selector for status, a "Discard" button, and a primary "Commit Records" submit button.	
	Step 3: The Super Admin populates the category type name input field, enters descriptive details about the scope, selects an initial active configuration state, and clicks the primary "Commit Records" button. [ALT]		**Step 4:** The client application evaluates form field control validation states. The Angular FAQService packages the data components and dispatches an asynchronous HTTP POST payload statement directly to the API path api/FaqType/create enclosing a CreateFaqTypeRequest JSON body. The backend controller receives the incoming request. The framework links directly into the ApplicationDbContext runtime instance and executes an asynchronous

			database search scan against the "FaqType" entity table to verify that an identical category name string does not already exist.
			Step 5: The server backend evaluates data input constraints before authorizing database persistence. The framework runs validations to confirm all mandatory rows are present, verify text string lengths stay within schema boundary structures, and confirm the active session token contains Super Admin operational claim permissions. FaqType Entity Schema Structure: • FaqTypeId (PK) • TypeName • DescriptionScope • IsActiveStatus [ALT]
			Step 6: Upon successful validation passes across all system check loops, the system initializes a new instance tracking entry of the FaqType database model table collection and saves the transaction metrics to disk via an asynchronous database commit command (SaveChangesAsync).
			Step 7: The system endpoint returns an HTTP 201 Created code output tracking confirmation parameters. The frontend

			catches the response message, appends the newly returned category model instance into its master application view display arrays, clears the modal state layer, and triggers a green confirmation toast statement reading: "New FAQ type category successfully added to global system directories."
Alternate courses:	**[ALT] Step 3:** The user chooses to cancel the action and clicks the "Discard" button. The frontend interface closes the editing modal panel immediately, drops all form mutations from memory buffers, and prevents network calls to the server.		
	[ALT] Step 5: The backend service layer triggers a business rule conflict check identifying that an identical classification type name already exists in the table partitions. The system immediately interrupts transaction loops and sends back an HTTP 409 Conflict response code. The frontend layout catches the failure tracking information and renders a prominent red descriptive validation warning layout banner across the modal frame.		
Conclusion:	The use case concludes once all data rules clear validation parameters, records register successfully inside database backend tables, and configuration lookup dashboards refresh.		
Post-condition:	• A new FAQ category classification option is permanently written to the database, making it immediately available for routing and FAQ entry classification workflows.		
Business rules:	• Only users with explicitly assigned Super Admin configuration privileges can create new FAQ lookup categories.		
Implementation constraints and specifications:	• FAQ Type Entity Schema Structure: FaqTypeId (PK), TypeName, DescriptionScope, IsActiveStatus.		
Assumptions:	• None		
Open issues:	• None		

Table 5: Technical Narrative 2.8 Add FAQ Type

2.4.2.5 Use Case: 2.9 Update FAQ Type

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 24 June 2026 **Version: 1.0**	
Use case name:	Update FAQ Type	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	2.9		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Admin		
Description:	This use case details the step-by-step process for modifying an existing FAQ type classification category's structural name and scope parameters. The user adjusts field metrics inside the active frontend editing modal layout panel. Upon clicking save, the client interface assesses client field validation syntax maps and dispatches an asynchronous PUT call directly to the backend endpoint URL parameter path (PUT api/FaqType/update/{id}). The API endpoint binds to the shared ApplicationDbContext instance and runs an asynchronous lookup transaction seeking a matching primary key sequence. The backend overwrites the tracked database entity schema properties with the new payload values and commits state updates via Entity Framework Core. The use case concludes when changes are written to disk, modal containers close, and configuration dashboards refresh.		
Pre-condition:	• The target FAQ type category record must exist inside the database tables, and the operator must hold valid configuration management write claims.		
Trigger:	The user clicks the row "Edit" button layout element asset next to a specific FAQ type listing line item.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The Super Admin requests to alter an FAQ category name or description by clicking the row "Edit" button layout element next to a specific FAQ type listing row.		

		Step 2: The frontend triggers the edit workflow modal overlay window container. The system passes the active row parameters and patches the existing database field metrics directly into the form text inputs via an explicit Angular form patch operation (patchValue).	
	Step 3: The Super Admin modifies the alphanumeric text input values (updating category names or adjusting description scope explanations) and clicks the "Save Changes" execution control button. [ALT]		**Step 4:** The client framework verifies form control validity states on the browser layer. The Angular FAQService dispatches a structured HTTP PUT call directly to the backend endpoint URL parameter path api/FaqType/update/{id} passing the comprehensive updated parameter payload maps. The API endpoint processes data extraction streams. The tracking logic binds to the shared ApplicationDbContext and runs an asynchronous lookup transaction statement (SingleOrDefaultAsync) seeking a matching FAQ type primary key sequence mapping.
			Step 5: The system evaluates payload modifications against

			backend validation constraints. It checks that the target FAQ type record is found in the database layer, assures alphanumeric character arrays do not break size properties, and validates that the active session holds necessary configuration management permissions.
			Step 6: Upon completing verification loops, the system overwrites the tracked database entity schema properties with the new payload values and saves changes to the database storage layer via an asynchronous update command (SaveChangesAsync)
			Step 7: The server backend dispatches an HTTP 200 OK status packet back to the client interface. The frontend application intercepts the success code message, closes the modal popup block container, updates its local state arrays, and triggers a confirmation alert toast reading "Changes committed successfully."
Alternate courses:	**[ALT] Step 3:** The user aborts the workflow and clicks the "Discard" or "Cancel" button. The frontend interface closes the editing panel immediately, drops all form mutations from memory buffers, and prevents network data calls to the server.		
Conclusion:	The use case concludes when the system successfully validates the modified parameter maps, saves updates to the database storage layer, and refreshes the main dashboard grid lookup views.		
Post-condition:	• The target FAQ type category's database entity properties are permanently updated with the new input text parameters.		

Business rules:	• Only users with explicitly assigned system configuration privileges can modify FAQ classification lookup categories
Implementation constraints and specifications:	• Configuration mutations must execute over an asynchronous pipeline leveraging Entity Framework tracking contexts.
Assumptions:	• None
Open issues:	• None

Table 6: Technical Narrative 2.9 Update FAQ Type

2.4.2.6 Use Case: 2.10 Remove FAQ Type

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 24 June 2026 **Version: 1.0**	
Use case name:	Remove FAQ Type	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	2.10		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Admin		
Description:	This use case details the step-by-step process invoked when an authorized user permanently deletes an FAQ category type classification that is no longer needed across systemic directories. The user triggers the action from the lookup grid row. The frontend halts execution flows and presents an explicit modal confirmation warning alert overlay block dialog framework. Upon confirmation, the client interface routes an asynchronous HTTP DELETE request directly to the backend service controller endpoint path (DELETE api/FaqType/remove/{id}). The backend service controller evaluates database dependency constraints before completely marking the tracked record instance for deletion and signing off changes via Entity Framework Core. The use case finishes once table rows are dropped, the modal closes, and dashboard view panels refresh.		
Pre-condition:	• The targeted FAQ type classification category record must exist inside the database context frames and must be free of active downstream dependencies.		
Trigger:	The user clicks the row deletion contextual action icon "Delete" trigger button element asset on a target category row line item.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The Super Admin requests to drop an FAQ tracking category line row by clicking its row deletion contextual action icon trigger button.		

		Step 2: The frontend system halts current layout sequences and displays a modal confirmation warning overlay dialog framework, presenting a descriptive message text to the operator: "Are you sure you want to permanently delete this FAQ item? This action cannot be undone."	
	Step 3: The Super Admin assesses the warning text context and hits the primary confirmation "Confirm Delete" action switch inside the modal overlay. [ALT]		**Step 4:** The frontend interface captures the user click intent. The Angular FAQService formats an asynchronous HTTP DELETE request command routing directly to the backend endpoint URL parameter path map api/FaqType/remove/{id}. The backend web API intercepts the data deletion stream request. The data runtime environment establishes connections to the ApplicationDbContext unit tracking context and runs an execution query sweep targeting the row index matching the supplied ID property.
			Step 5: The backend system runs validation rules before allowing execution. It

			verifies that the targeted row record exists in the tracking context and confirms that no active FAQ entries are currently classified under this category ID to protect structural cascading foreign key dependencies.
			Step 6: Upon successful logic validation confirmations, the server completely marks the tracked record instance for deletion via the DbContext entity tracker and signs off the record sweep via an asynchronous storage update execute call (SaveChangesAsync).
			Step 7: The API transfers an HTTP 200 OK or HTTP 240 No Content message tracking packet block. The client application interceptors catch the response, splice the target item index out of its local collection layout arrays, dismiss the alert modal wrapper layer, and render a green toast confirmation message: "FAQ type category deleted successfully."
Alternate courses:	**[ALT] Step 3:** The operator clicks the "Cancel" or "Discard Deletion" button control asset inside the confirmation dialog panel overlay wrapper layout. The system closes the window immediately, cancels the operational flow, and leaves the database record completely unchanged.		
Conclusion:	The use case concludes once all data integrity validations clear, the target row record drops from persistent system tables, and configurations dashboards refresh local caches.		

Post-condition:	• The specific category row tracking index is permanently removed from system tables, removing it from selection options across systemic forms.
Business rules:	• Deactivation and deletion requests are blocked if any active downstream FAQ profile entries depend on the category classification index to protect data mapping integrity.
Implementation constraints and specifications:	• Relational constraints must strictly block deletion actions unless the dependent array collection size equals zero.
Assumptions:	• None
Open issues:	• None

Table 7: Technical Narrative 2.10 Remove FAQ Type

2.4.2.7 Use Case 2.16: Update Role Permission

System Name: Integrated NGO Management System			
AUTHOR:	Amahle Khoza	**Date:** 22 July 2026 **Version:** 1.0	
USE CASE NAME:	Update Role Permission	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
USE CASE ID:	2.16		
PRIORITY:	High		
SOURCE:	EPI-USE		
Primary Business Actor (PBA):	Administrator		
Primary System Actor (PSA):	• None		
Other Participating Actors:	• None		
Other Interested Stakeholders:	User		
Description:	This use case describes how an Administrator changes the permissions granted by a system-wide role (AppRole). The Administrator opens the role editor, toggles the permissions the role should hold, and saves. The Angular frontend sends the complete desired permission set to the ".NET 8" API endpoint ("PUT api/roles/{id}/permissions"), which replaces the role's entire permission set atomically — whatever is sent is what the role gets. Built-in roles (User, Admin) can be re-permissioned freely, but the Super Admin role always rejects any restriction, since it must always hold every permission in the system.		
Pre-condition:	The Administrator must be authenticated and hold the "Roles.Manage" permission, and must have navigated to the role editor for the role being changed.		
Trigger:	The Administrator toggles one or more permissions for a role and selects "Save Permissions".		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: Administrator navigates to the roles management page.		**Step 2:** The system calls "GET api/roles" and displays each role with

			its name, description, user count, and permission count.
	Step 3: Administrator selects a role to edit.		**Step 4:** The system calls “GET api/roles/{id}” for the role's current permissions and “GET api/roles/permissions” for the full permission catalog, then renders a permission grid grouped by module with the role's current permissions pre-toggled on. [ALT]
	Step 5: Administrator toggles permissions on or off for the role.		**Step 6:** The system tracks the toggle state and enables the “Save Permissions” button once it differs from the saved permission set.
	Step 7: Administrator selects “Save Permissions”.		**Step 8:** The frontend RolesService sends “PUT api/roles/{id}/permissions” with only the permission keys currently toggled on.
			Step 9: The controller confirms the Administrator holds “Roles.Manage”, then AppRoleService.SetRolePermissionsAsync loads the role together with its existing grants from: **RolePermission Entity:** • RolePermissionId (PK) • RoleId (FK) • PermissionKey

			Step 10: The service filters the incoming keys to only those recognised in the permission catalog, normalises their casing, then diffs the current grants against the desired set, removing rows for keys no longer present and inserting rows for newly granted keys.
			Step 11: The API responds with “200 OK” and the updated role, including its new permission list. The frontend refreshes the permission grid and clears the unsaved-changes state.
Alternate courses:	[ALT] Step 4: The selected role is the Super Admin role. Every toggle is shown on, locked, and disabled, with the message “Super Admin always holds every permission and cannot be restricted.” No save is possible for this role.		
	[ALT] Step 9: The role no longer exists. The API returns “404 Not Found” and the frontend shows an error.		
	[ALT] Step 9: The role is the Super Admin role. The service rejects the change with “409 Conflict” (“Super Admin always holds every permission and cannot be restricted.”), and the frontend displays the message inline.		
	[ALT] Step 8: The Administrator's “Roles.Manage” permission was revoked after the page loaded. The API returns “403 Forbidden” and the frontend shows an inline alert.		
Conclusion:	The use case concludes when the role's permission set has been replaced and the updated grants are reflected in the permission grid.		
Post-condition:	The AppRolePermission rows for the role match exactly the submitted permission set. Users assigned to the role gain or lose the corresponding access on their next request, since permissions are resolved live from the database on every call.		

Business Rules:	• The Super Admin role's permission set can never be reduced. • Only permission keys recognised in the catalog are accepted; unrecognised keys are dropped silently. • Saving replaces the role's full permission set rather than merging it with the previous one.
Implementation constraints and specifications:	• "PUT api/roles/{id}/permissions" requires the "Roles.Manage" permission. • Permission keys are matched case-insensitively and normalised to the catalog's casing on save.
Assumptions:	• Administrators understand that saving replaces the role's entire permission set rather than merging individual changes.
Open issues:	• None.

Table 8: Technical Narrative 2.16 Update Role Permission

2.4.2.8 Use Case 2.17: View Role Permission

System Name: Integrated NGO Management System			
AUTHOR:	Amahle Khoza	**Date:** 22 July 2026 **Version:** 1.0	
USE CASE NAME:	View Role Permission	**USE CASE TYPE**	
USE CASE ID:	2.17	Business Requirements: □	
PRIORITY:	Medium	System Analysis: □	
SOURCE:	EPI-USE	System Design: ✓	
Primary Business Actor (PBA):	Super Administrator		
Primary System Actor (PSA):	• None		
Other Participating Actors:	• None		
Other Interested Stakeholders:	User		
Description:	This use case describes how a user inspects what permissions each system-wide role grants. It covers the role list with permission summaries, drilling into a single role's full permission set, and browsing the complete permission catalog grouped by module. It also covers the ".NET 8" API's "GET api/roles/me" call, which any authenticated user relies on to view their own effective permissions and gate their own interface.		
Pre-condition:	The user must be authenticated. Viewing the role list, role detail, and permission catalog additionally requires the "Roles.View" permission; viewing one's own effective permissions requires only a valid session.		
Trigger:	The user navigates to the roles management page, opens a specific role, or logs in.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: User navigates to the roles management page.		**Step 2:** The system calls "GET api/roles" and displays every role as a card or table row, showing its name, a

			"Built-in" badge where applicable, description, user count, and permission count.
	Step 3: User selects a role to inspect.		**Step 4:** The system calls "GET api/roles/{id}" for the role's details and "GET api/roles/permissions" for the full catalog, then renders the role's permissions grouped by module, marking each one granted or not granted. [ALT]
	Step 5: User opens the standalone permission catalog view.		**Step 6:** The system calls "GET api/roles/permissions" and lists every permission the system recognises, grouped by module, without role-specific highlighting.
			Step 7: On login, the frontend calls "GET api/roles/me", which resolves the signed-in user's role and effective permissions from: **RolePermission Entity:** • RolePermissionId (PK) • RoleId (FK) • PermissionKey
			Step 8: The response is cached in the client-side auth store for the session and used to show or hide navigation items and action buttons the user is not permitted to use.

Alternate courses:	[ALT] Step 4: The selected role is the Super Admin role. Every permission is shown as granted, with the note "Super Admin implicitly holds every permission."
	[ALT] Step 4: The role no longer exists. The API returns "404 Not Found" and the page shows "This role does not exist. It may have been deleted."
	[ALT] Step 2: The user lacks "Roles.View". The API returns "403 Forbidden" and the "Roles" navigation item stays hidden.
	[ALT] Step 7: The session's token is missing or expired. The API returns "401 Unauthorized" and the frontend redirects to the login page.
Conclusion:	The use case concludes when the user has viewed the role list, a role's permission grants, the permission catalog, or their own effective permissions.
Post-condition:	No data is changed; the user has viewed the requested role or permission information.
Business Rules:	• "Roles.View" is required for the role list, role detail, and permission catalog endpoints. • "GET api/roles/me" requires only authentication. • The Super Admin role always returns every catalog key, regardless of its stored grants. • Permission keys are returned sorted alphabetically.
Implementation constraints and specifications:	• "GET api/roles/permissions" is served from the in-memory permission catalog; it makes no database call. • A user's role change takes effect only after their next login, since the role is embedded in their JWT.
Assumptions:	• Users understand that a role change made by an Administrator only takes effect for an affected user after that user's next login.
Open issues:	• None.

Table 9: Technical Narrative 2.17 View Role Permission

2.4.3 Inventory Subsystem Technical Narratives

2.4.3.1 Use Case 3.1: Create Procurement

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Christopher Bailey</td><td>Date: 13 July 2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">Create Procurement</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">3.1</td></tr>
<tr><td>Priority:</td><td colspan="2">Medium</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">• None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event in which an admin creates a procurement after receiving inventory. The admin will navigate to and select the Inventory tab and select the create procurement button. The system will load the Procurement screen where the admin will enter the procurements details. The admin will select create procurement and the system will validate the details entered before saving the procurement to the database. The use case ends once the system displays the success toaster and the procurement is saved on the system.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">Admin is logged in.</td></tr>
<tr><td>Trigger:</td><td colspan="3">Inventory is received and the admin requests to create a procurement.</td></tr>
<tr><td rowspan="2">Typical Course of Events:</td><td>Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td></td><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td></td><td>Step 1: Inventory is received and the admin requests to create a procurement.</td><td></td><td>Step 2: The system invokes use case 3.3 (View Inventory).</td></tr>
</table>

	Step 3: The admin selects the "Create procurement" button.		**Step 4:** The Angular frontend calls the Warehouse, Vendor and InventoryTypes service to send a GET request to the C# Web API backend Warehouse, Vendor and InventoryType controllers through the GetWarehouseLookup, GetVendorLookup and GetInventoryTypeLookup endpoints. The backend retrieves active warehouse, vendor and inventory type details from the following entities: Warehouse Entity: • WarehouseId (PK) • Name Vendor Entity: • VendorId(PK) • Name InventoryType Entity: • InventoryTypeId (PK) • Name [ALT]
			Step 5: The system loads and displays the Procurement screen using the Angular frontend with the following controls: A header section containing: • A heading displaying **Procurement**. • A description displaying **Receive inventory**.

			• A **Cancel** button to return to the Inventory screen. • A **Create Procurement** button to submit the procurement details and create the procurement record. A **Procurement Details** card containing: • A **Vendor** drop-down list populated with the available vendors retrieved from the backend. • A **Warehouse** drop-down list populated with the available warehouses retrieved from the backend. • A **Procurement Date** date picker for selecting the procurement date. • An **Invoice (PDF)** file upload control for selecting a PDF invoice. • A label displaying the selected invoice filename or **No file selected** when no file has been selected. An **Inventory Items** section containing: • An **Add Item** button for adding a new inventory item row.

			• A table displaying one or more procurement item rows. Each procurement item row contains: • An **Inventory Type** drop-down list populated with the available inventory types retrieved from the backend. • A **Model Name** text box for entering the inventory model name. • A **Quantity** numeric input field for entering the quantity received. • A **Price** numeric input field for entering the unit price. • A **Total** field displaying the calculated line total (Quantity × Price). • A **Remove** button to remove the procurement item from the list. A summary section containing: • A **Total** display showing the calculated total value of all procurement items. An error message section containing:

			• A validation or processing error message displayed when an error occurs during procurement creation.
	Step 6: The admin enters the procurement's details and selects the "Create Procurement" button. [ALT]		**Step 7:** The system validates the details entered, validating the following: • All fields are completed. • Procurement date is not in the future. • At least one inventory item. • Inventory items are at least one quantity. • Inventory items have a model name. • Invoice is a PDF. [ALT]
			Step 8: The Angular frontend calls the Procurement service to send a POST request to the C# Web API backend Procurement controller through the CreateProcurement endpoint. The backend creates a new procurement in the following entities: Procurement Entity: • ProcurementId (PK) • VendorId (FK) • ProcurementDate • Invoice ProcurementLine Entity: • ProcurementLineId (PK) • ProcurementId (FK)

			• InventoryTypeId (FK) • ModelName • Quantity • Price Inventory • InventoryItemId (PK) • InventoryTypeId (FK) • ProcurementLineId (FK) • WarehouseId (FK) • ModelName • Value • Status [ALT]
			Step 9: The system will then record the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful creation of the procurement. The system will store information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that a procurement has been created. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 10: The system displays a success toaster informing the admin that the

			procurement was successfully created.
Alternate courses:	[ALT] Step 4: The system fails to retrieve the lookup data. The system will display an error message based on the status code.		
	[ALT] Step 6: The admin selects the "Cancel" button. Use case terminates.		
	[ALT] Step 7: The validation fails. The system displays an error message. Go back to step 6.		
	[ALT] Step 8: The system fails to create a new procurement. The system will display an error message based on the status code.		
Conclusion:	The system displays a success toaster informing the admin that the procurement was successfully created.		
Post-condition:	The procurement is saved on the system.		
Business rules:	• Super admins and admins can create procurements.		
Implementation constraints and specifications:	• Each inventory item is stored individually in the Inventory table.		
Assumptions:	• None		
Open issues:	• None		

Table 10: Technical Narrative 3.1 Create Procurement

2.4.3.2 Use Case 3.2: View Procurement

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Christopher Bailey</td><td>Date: 13 July 2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">View Procurement</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">3.2</td></tr>
<tr><td>Priority:</td><td colspan="2">Medium</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">• None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event in which an admin views a procurement. The admin will navigate to the Inventory tab in the navbar. The admin will select a procurement, and the system will retrieve and display the procurement's details on the system. The use case ends once the admin views the procurement.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">Admin is logged into the system and procurement already exists.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The admin requests to view a procurement.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td>Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td></td><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The admin requests to view a procurement.</td><td></td><td>Step 2: The system invokes use case 3.3 (View Inventory).</td></tr>
<tr><td></td><td>Step 3: The admin selects the procurement they wish to view.</td><td></td><td>Step 4: The Angular frontend calls the Procurement service to send a GET request to the C# Web API</td></tr>
</table>

			backend Procurement controller through the GetProcurementById endpoint. The backend retrieves the procurement's details from the following entities: Procurement Entity: • ProcurementId (PK) • ProcurementDate • Invoice ProcurementLine Entity: • ProcurementLineId (PK) • ProcurementId (FK) • InventoryTypeId (FK) • ModelName • Quantity • Price InventoryType Entity: • InventoryTypeId (PK) • Name [ALT]
			Step 5: The system loads and displays the Procurement Details modal using the Angular frontend with the following controls populated with the selected procurement details: A header section containing: • A heading displaying the procurement number. A procurement details section containing:

			• A display of the procurement date. • A display of the total number of procurement items. • An **Invoice** section containing: • A **View Invoice** hyperlink for opening the uploaded invoice PDF in a new browser tab when an invoice exists. An **Inventory Items** section containing a table with the following columns: • **Inventory Type** displaying the inventory type name. • **Model Name** displaying the inventory model name. • **Quantity** displaying the quantity procured. • **Price** displaying the unit price. • **Total** displaying the calculated line total (Quantity × Price). A summary row containing: • A **Grand Total** display showing the calculated total value of all procurement items. A footer section containing:

			• A **Close** button to close the Procurement Details modal and return to the Inventory screen.
Alternate courses:	[ALT] Step 4: The system fails to retrieve the procurement's details. The system will display an error toaster message based on the status code.		
Conclusion:	The system displays the procurement's details.		
Post-condition:	The procurement's details is displayed.		
Business rules:	• Super admins and admins can view procurements.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 11: Technical Narrative 3.2 View Procurement

2.4.3.3 Use Case 3.3: View Inventory

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Christopher Bailey</td><td>Date: 13 July 2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">View Inventory</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">3.3</td></tr>
<tr><td>Priority:</td><td colspan="2">Medium</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event in which an admin views inventory. The admin will navigate to the Inventory tab in the navbar and the system will retrieve and display all inventory on the system where the admin can filter and search inventory. The use case ends once the admin views inventory.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">Admin is logged into the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The admin requests to view inventory.</td></tr>
<tr><td rowspan="4">Typical Course of Events:</td><td>Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td></td><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The admin requests to view inventory.</td><td></td><td></td></tr>
<tr><td>Step 2: The admin navigates to and selects the Inventory tab. [ALT]</td><td></td><td>Step 3: The Angular frontend calls the Warehouse and InventoryTypes service to send a GET request to the C# Web API backend Warehouse and</td></tr>
</table>

			InventoryType controllers through the GetWarehouseLookup, GetInventoryTypeLookup endpoints. The backend retrieves active warehouse, inventory type details from the following entities: Warehouse Entity: • WarehouseId (PK) • Name InventoryType • InventoryTypeId (PK) • Name [ALT]
			Step 4: The Angular frontend calls the Inventory service to send a GET request to the C# Web API backend Inventory controller through the GetAll endpoint. The backend retrieves the inventory from the following entities: Inventory Entity • InventoryItemId (PK) • InventoryTypeId (FK) • ProcurementLineId (FK) • WarehouseId (FK) • ProjectId (FK) • ModelName • Value • Status InventoryType Entity: • InventoryTypeId (PK) • Name

			Warehouse Entity: • WarehouseId (PK) • Name • Location ProcurementLine Entity: • ProcurementLineId (PK) • ProcurementId (FK) [ALT]
			Step 5: The system loads and displays the Inventory screen using the Angular frontend with the following controls: A header section containing: • A heading displaying **Inventory**. • A description displaying **View and manage inventory**. • A **Manage Warehouses** button to navigate to the Warehouse Management screen. • A **Manage Inventory Types** button to navigate to the Inventory Type Management screen. • A **Create Procurement** button to navigate to the Procurement screen. An **Inventory Filters** section containing:

			• A **Search Inventory** search box for filtering inventory records. • A **Status** drop-down list containing the available inventory statuses. • An **Inventory Type** drop-down list populated with the available inventory types. • A **Warehouse** drop-down list populated with the available warehouses. • A **View Quantity Totals** button to switch to the Quantity Totals view. An **Inventory** table containing the following columns: • **Inventory Type** displaying the inventory type name. • **Model Name** displaying the inventory model name. • **Warehouse** displaying the warehouse name. • **Location** displaying the warehouse location. • **Procurement ID** displayed as a button for viewing the associated

			procurement details. • **Value** displaying the monetary value of the inventory item. • **Status** displaying the current inventory status. A fallback message displaying **"No inventory found."** when no inventory records are available.
	Step 6: The admin selects the "View Quantity Totals" button. [ALT]		**Step 7:** The Angular frontend calls the Inventory service to send a GET request to the C# Web API backend Inventory controller through the GetQuantityTotals endpoint. The backend retrieves the inventory from the following entities: Inventory Entity: • InventoryItemId (PK) • InventoryTypeId (FK) • ModelName • Value • Status InventoryType Entity: • InventoryTypeId (PK) • Name The backend groups the inventory records by InventoryTypeId, Inventory Type Name, and Model Name, and calculates the quantity and total value for each inventory status. [ALT]

			Step 8: The system displays the Quantity Totals screen using the Angular frontend with the following controls: A **Quantity Totals Filters** section containing: • An **Inventory Types** multi-select control populated with the available inventory types. • A **Status** drop-down list containing the available inventory statuses. • A **View Inventory** button to return to the Inventory view. A **Quantity Totals** table containing the following columns: • **Inventory Type** displaying the inventory type name. • **Model Name** displaying the inventory model name. • **Total Quantity** displaying the total quantity of inventory items. • **Total Value** displaying the total monetary value of the inventory items. An **Actions** column containing: • An **Allocate** button for allocating

			available inventory items to a project. • A **Write Off** button for writing off available inventory items. The **Actions** column is displayed only when the selected inventory status is **Available**. • A fallback message displaying **"No inventory found."** when no inventory records match the selected filters.
	Step 9: The admin filters the inventory quantities. [ALT]		**Step 10:** The frontend Angular filters the inventory quantities based on the applied criteria. [ALT]
Alternate courses:	[ALT] Step 2: The admin is using a mobile device such as a phone. The admin will select the menu (hamburger) icon and then the inventory option. Continue to step 3.		
	[ALT] Step 3: The system fails to retrieve the lookup data. The system will display an error toaster message based on the status code.		
	[ALT] Step 4: The system fails to retrieve inventory. The system will display an error toaster message based on the status code.		
	[ALT] Step 6a: The admin does not select the View Quantity Totals button. The system remains on the Inventory view. [ALT] Step 6b: The admin filters the inventory. The frontend Angular filters the inventory using the applied filters and displays the inventory matching the criteria.		
	[ALT] Step 7: The system fails to retrieve the inventory quantity totals. The system will display an error toaster message based on the status code.		
	[ALT] Step 9: The admin does not apply any filters.		
	[ALT] Step 10: No inventory matches the applied filters. Go back to step 9.		

Conclusion:	The system displays the inventory.
Post-condition:	Inventory is displayed.
Business rules:	• Super admins and admins can view inventory.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 12: Technical Narrative 3.3 View Inventory

2.4.3.4 Use Case 3.4: Write-Off Inventory

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Christopher Bailey</td><td>Date: 13 July 2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">Write-Off Inventory</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">3.4</td></tr>
<tr><td>Priority:</td><td colspan="2">Medium</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">• None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event in which an admin writes-off inventory. The admin will navigate to the Inventory tab in the navbar. The system displays the inventory records and allows the admin to select inventory for the write-off process. The system then displays the Write Off Inventory modal, where the admin enters the quantity of inventory to be written off. The system validates the entered quantity before updating the status of the selected inventory items to Written Off. The use case ends when a success toast notification is displayed, confirming that the selected inventory items have been successfully updated.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">Admin is logged into the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The admin requests to write-off inventory.</td></tr>
<tr><td rowspan="2">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The admin requests to write-off inventory.</td><td></td><td>Step 2: The system invokes use case 3.3 (View Inventory).</td></tr>
</table>

	Step 3: The admin selects the "Write Off" button.		**Step 4:** The system displays the Write Off modal using the Angular frontend with the following controls populated with the selected inventory details: A header section containing: • A heading displaying **Write Off Inventory**. A form section containing: • An **Inventory Type** read-only text box displaying the selected inventory type. • A **Model Name** read-only text box displaying the selected model name. • An **Available Quantity** read-only text box displaying the quantity of available inventory items. • A **Quantity to Write Off** numeric input field for entering the quantity of inventory items to be written off. An error message section containing: • A validation or processing error message displayed when an error occurs during the write-off process.

			A footer section containing: • A **Cancel** button to close the Write Off Inventory modal without making any changes. • A **Write Off** button to submit the write-off request.
	Step 5: The admin enters the amount of inventory to write-off and selects the "Write Off" button. [ALT]		**Step 6:** The system validates the details entered, validating the following: • The quantity entered does not exceed the quantity available. [ALT]
			Step 7: The Angular frontend calls the Inventory service to send a POST request to the C# Web API backend Inventory controller through the WriteOffInventory endpoint. The backend updates the matching inventory item(s) in the following entity: Inventory Entity: • Status The backend retrieves the inventory items matching the specified InventoryTypeId, ModelName, and Available status, up to the requested quantity and updates their status to Written Off. [ALT]
			Step 8: The system will then record the transaction in the

			AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful update of the inventory item(s) status. The system will store information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that inventory has been written off. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 9: The system displays a success toaster informing the admin that the inventory was successfully written off.
Alternate courses:	[ALT] Step 5: The admin selects the "Cancel" button. Use case terminates.		
	[ALT] Step 6: The validation fails. The system displays an error message. Go back to step 5.		
	[ALT] Step 7: The system fails to update the inventory item(s) status. The system will display an error message based on the status code.		
Conclusion:	A success toaster is displayed informing the admin that inventory was successfully written off.		
Post-condition:	Inventory is successfully written off.		
Business rules:	• Super admins and admins can write-off inventory. • Only available inventory can be written off.		

Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 13: Technical Narrative 3.4 Write-Off Inventory

2.4.3.5 Use Case 3.5: Allocate Inventory

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Christopher Bailey</td><td>Date: 13 July 2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">Allocate Inventory</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">3.5</td></tr>
<tr><td>Priority:</td><td colspan="2">Medium</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">• None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event in which an admin allocates inventory. The admin will navigate to the Inventory tab in the navbar. The system displays the inventory records and allows the admin to select inventory for the allocation process. The system then displays the Allocate Inventory modal, where the admin selects an active project and enters the quantity of inventory to be allocated. The system validates the details before updating the status of the selected inventory items to Allocated. The use case ends when a success toast notification is displayed, confirming that the selected inventory items have been successfully allocated.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">Admin is logged into the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The admin requests to allocate inventory.</td></tr>
<tr><td rowspan="2">Typical Course of Events:</td><td>Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td></td><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td></td><td>Step 1: The admin requests to allocate inventory.</td><td></td><td>Step 2: The system invokes use case 3.3 (View Inventory).</td></tr>
</table>

	Step 3: The admin selects the "Allocate" button.		**Step 4:** The Angular frontend calls the Project service to send a GET request to the C# Web API backend Project controller through the GetProjectLookup endpoint. The backend retrieves active projects from the following entity: Project Entity: • ProjectId (PK) • ProjectName [ALT]
			Step 5: The system displays the Allocate modal using the Angular frontend with the following controls populated with the selected inventory details: A header section containing: • A heading displaying **Allocate Inventory**. A form section containing: • An **Inventory Type** read-only text box displaying the selected inventory type. • A **Model Name** read-only text box displaying the selected model name. • An **Available Quantity** read-only text box displaying the quantity of available inventory items. • A **Project** drop-down list populated with

			the available projects. • A **Quantity** numeric input field for entering the quantity of inventory items to allocate. An error message section containing: • A validation or processing error message displayed when an error occurs during the allocation process. A footer section containing: • A **Cancel** button to close the Allocate Inventory modal without making any changes. • An **Allocate** button to submit the inventory allocation request.
	Step 6: The admin enters the amount of inventory to allocate as well as select a project then selects the "Allocate" button. [ALT]		**Step 7:** The system validates the details entered, validating the following: • A project is selected. • The quantity entered does not exceed the quantity available. [ALT]
			Step 8: The Angular frontend calls the Inventory service to send a POST request to the C# Web API backend Inventory controller through the AllocateInventory endpoint. The backend

			updates the matching inventory item(s) in the following entity: Inventory Entity: • Status • ProjectId (FK) The backend retrieves the inventory items matching the specified InventoryTypeId, ModelName, and Available status, up to the requested quantity and updates their status to Allocated. [ALT]
			Step 9: The system will then record the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful update of the inventory item(s) status. The system will store information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that inventory has been allocated. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 10: The system displays a success toaster informing the admin that the inventory was successfully written off.

Alternate courses:	[ALT] Step 4: The system fails to retrieve the lookup data. The system will display an error message based on the status code.
	[ALT] Step 6: The admin selects the "Cancel" button. Use case terminates.
	[ALT] Step 7: The validation fails. The system displays an error message. Go back to step 6.
	[ALT] Step 8: The system fails to update the inventory item(s) status. The system will display an error message based on the status code.
Conclusion:	The inventory item(s) is successfully allocated and a success toaster is displayed.
Post-condition:	Inventory is successfully allocated to a project.
Business rules:	• Super admins and admins can write-off inventory. • Only available inventory can be allocated. • Inventory can only be allocated to active projects.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 14: Technical Narrative 3.5 Allocate Inventory

2.4.3.6 Use Case 3.6: Create inventory Item Type

System Name: Integrated NGO Management System			
Author:	Christopher Bailey		**Date: 13 July 2026** **Version:** 1
Use case name:	Create Inventory Item Type		**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓
Use case id:	3.6		
Priority:	Low		
Source:	EPI-USE		
Primary business actor (PBA):	Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	• None		
Description:	This use case describes the event in which a super admin creates an inventory item type. The system will display inventory type records and allow the super admin to create a new type. The system will display the Create Inventory Item Type modal and allow the super admin to enter the new inventory type's details. The use case ends once the success toaster is displayed and the new inventory item type has successfully been captured on the system.		
Pre-condition:	Super Admin is logged into the system.		
Trigger:	The super admin requests to create a new inventory item type.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The super admin requests to create a new inventory item type.		**Step 2:** The system invokes use case 3.7 (View Inventory Item Types).
	Step 3: The super admin selects "Create" button.		**Step 4:** The system loads and displays the Create Inventory Item Type modal

			using the Angular frontend with the following controls: A header section containing: • A heading displaying **Create Inventory Type**. A form section containing: • A **Name** text box for entering the inventory type name. • A **Security Level** drop-down list containing the available security levels (**Low**, **Medium**, and **High**). • A **Description** text area for entering the inventory type description. An error message section containing: • A validation or processing error message displayed when an error occurs during the inventory type creation process. A footer section containing: • A **Cancel** button to close the Create Inventory Type modal without creating a new inventory type. • A **Create** button to submit the inventory type details. The

			button remains disabled until all required fields have been completed.
	Step 5: The super admin enters the new inventory item type's details and selects the "Create" button. [ALT]		**Step 6:** The system validates the details entered, validating the following: • All fields are completed [ALT]
			Step 7: The Angular frontend calls the InventoryTypes service to send a POST request to the C# Web API backend InventoryType controller through the CreateInventoryType endpoint. The backend creates the inventory item type in the following entity: InventoryType Entity: • Name • Description • SecurityLevel • Status [ALT]
			Step 8: The system displays a success toaster informing the admin that the inventory item type was successfully created.
			Step 9: The system will then record the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful creation of the inventory item type.

			The system will store information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that an inventory item type was created. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
Alternate courses:	[ALT] Step 5: The super admin selects the "Cancel" button. Use case terminates.		
	[ALT] Step 6: The validation fails. The system displays an error message. Go back to step 5.		
	[ALT] Step 7: The system fails to create the inventory item type. The system will display an error message based on the status code.		
Conclusion:	A success toaster is displayed informing the super admin that the inventory item type was successfully created.		
Post-condition:	The inventory item type is saved on the system.		
Business rules:	• Super admins can create inventory item types.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 15: Technical Narrative 3.6 Create Inventory Item Type

2.4.3.7 Use Case 3.7: View Inventory Item Types

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Christopher Bailey</td><td>Date: 13 July 2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">View Inventory Item Types</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">3.7</td></tr>
<tr><td>Priority:</td><td colspan="2">Low</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Super Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event in which a super admin views inventory item types. The super admin will navigate to the Inventory tab in the navbar. The system displays the inventory records and allows the super admin to select the Manage inventory types button. The system will retrieve and display inventory type records. The use case ends once the inventory item types are displayed.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">Super Admin is logged into the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The super admin requests to view inventory item types.</td></tr>
<tr><td rowspan="2">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td></td><td>Step 1: The super admin requests to view inventory item types.</td><td></td><td>Step 2: The system invokes use case 3.3 (View Inventory).</td></tr>
</table>

	Step 3: The admin selects the "Manage inventory types" button.		**Step 4:** The Angular frontend calls the InventoryTypes service to send a GET request to the C# Web API backend InventoryType controller through the GetAllInventoryTypes endpoint. The backend retrieves the inventory item types from the following entity: **InventoryType Entity:** • InventoryTypeId (PK) • Name • Description • SecurityLevel • Status [ALT]
			Step 5: The system loads and displays the Inventory Type screen using the Angular frontend with the following controls: A header section containing: • A heading displaying **Inventory Types**. • A description displaying **Create, edit, and manage inventory categories**. • A **Back** button to return to the Inventory screen. • A **Create** button to open the **Create**

			Inventory Type modal. An **Inventory Types** table containing the following columns: • **Name** displaying the inventory type name. • **Description** displaying the inventory type description. • **Security Level** displaying the inventory type security level. • **Status** displaying the current inventory type status (**Active** or **Inactive**). An **Actions** column containing: • An **Update** button to open the **Update Inventory Type** modal. The button is disabled when the inventory type is inactive. • A **Deactivate** button to deactivate an active inventory type. • A **Reactivate** button to reactivate an inactive inventory type.
Alternate courses:	[ALT] Step 4: The system fails to retrieve the inventory item types. The system will display an error toaster message based on the status code.		

Conclusion:	The inventory item types are displayed.
Post-condition:	The inventory item types are successfully displayed.
Business rules:	• Super admins can view inventory item types.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 16: Technical Narrative 3.7 View Inventory Item Types

2.4.3.8 Use Case 3.8: Update Inventory Item Type

System Name: Integrated NGO Management System			
Author:	Christopher Bailey	**Date: 13 July 2026** **Version:** 1	
Use case name:	Update Inventory Item Type	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	3.8		
Priority:	Low		
Source:	EPI-USE		
Primary business actor (PBA):	Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	None		
Description:	This use case describes the event in which a super admin updates an inventory item type. The super admin will navigate to and select the Manage inventory types button and the system will retrieve and display inventory type records. The super admin will select the inventory item type they wish to update, and the system will display the Update Inventory Item Type modal. The super admin will then enter the updated details. The use case ends once the success toaster is displayed and the inventory item type is updated on the system.		
Pre-condition:	Super Admin is logged into the system. The inventory item type exists. The inventory item type is active.		
Trigger:	The super admin requests to update an inventory item type.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**

	Step 1: The super admin requests to update an inventory item type.		**Step 2:** The system invokes use case 3.7 (View Inventory Item Types).
	Step 3: The admin selects the "Update" button on the inventory item type they wish to update.		**Step 4:** The Angular frontend calls the InventoryTypes service to send a GET request to the C# Web API backend InventoryType controller through the GetInventoryTypeById endpoint. The backend retrieves the inventory item type from the following entity: InventoryType Entity: • InventoryTypeId (PK) • Name • Description • SecurityLevel • Status [ALT]
			Step 5: The system displays the Update Inventory Item Type modal using the Angular frontend with the following controls populated with the selected inventory type details: A header section containing: • A heading displaying **Update Inventory Type**. A form section containing: • A **Name** text box displaying the inventory type name.

			• A **Security Level** drop-down list displaying the current security level and allowing the super admin to select **Low**, **Medium**, or **High**. • A **Description** text area displaying the inventory type description. An error message section containing: • A validation or processing error message displayed when an error occurs during the inventory type update process. A footer section containing: • A **Cancel** button to close the **Update Inventory Type** modal without saving any changes. • A **Save Changes** button to submit the updated inventory type details. The button remains disabled until all required fields have been completed.
	Step 6: The super admin enters the updated details and selects the "Save Changes" button. [ALT]		**Step 7:** The system validates the details entered, validating the following: • All fields are completed [ALT]

			Step 8: The Angular frontend calls the InventoryTypes service to send a PUT request to the C# Web API backend InventoryType controller through the UpdateInventoryType endpoint. The backend updates the inventory item type in the following entity: InventoryType Entity: • Name • Description • SecurityLevel [ALT]
			Step 9: The system will then record the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful update of the inventory item type. The system will store information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that an inventory item type was updated. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 10: The system displays a success toaster informing

			the admin that the inventory item type was successfully updated.
Alternate courses:	[ALT] Step 4: The system fails to retrieve the inventory item type. The system will display an error toaster message based on the status code.		
	[ALT] Step 6: The super admin selects the "Cancel" button. Use case terminates.		
	[ALT] Step 7: The validation fails. The system displays an error message. Go back to step 6.		
	[ALT] Step 8: The system fails to update the inventory item type. The system will display an error message based on the status code.		
Conclusion:	A success toaster is displayed to inform the super admin that the update was successful.		
Post-condition:	The inventory item type is successfully updated.		
Business rules:	• Super admins can update inventory item types.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 17: Technical Narrative 3.8 Update Inventory Item Type

2.4.3.9 Use Case 3.9: Deactivate Inventory Item Type

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Christopher Bailey</td><td>Date: 13 July 2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">Deactivate Inventory Item Type</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">3.9</td></tr>
<tr><td>Priority:</td><td colspan="2">Low</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Super Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event in which a super admin deactivates an inventory item type. The super admin will navigate to and select the Manage inventory types button and the system will retrieve and display inventory type records. The super admin will select the inventory item type they wish to deactivate, and the system will display the Confirm Inventory Item Type Deactivation modal. The super admin will then confirm the deactivation. The use case ends once the success toaster is displayed and the inventory item type is deactivated on the system.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">Super Admin is logged into the system.
The inventory item type is active.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The super admin requests to deactivate an inventory item type.</td></tr>
<tr><td rowspan="2">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td></td><td>Step 1: The super admin requests to deactivate an inventory item type.</td><td></td><td>Step 2: The system invokes use case 3.7 (View Inventory Item Types).</td></tr>
</table>

	Step 3: The admin selects the "Deactivate" button on the inventory item type they wish to deactivate.		**Step 4:** The system displays the Confirm Inventory Item Type Deactivation modal using the Angular frontend with the following controls: A header section containing: • A heading displaying **Deactivate Inventory Type**. A confirmation section containing: • A confirmation message asking the administrator to confirm the deactivation of the selected inventory type. • A warning message indicating that the inventory type will no longer be available for use when creating new inventory items. An error message section containing: • A validation or processing error message displayed when an error occurs during the deactivation process. A footer section containing: • A **Cancel** button to close the confirmation modal without deactivating the inventory type.

			• A **Confirm** button to deactivate the selected inventory type.
	Step 5: The super admin selects the "Confirm" button. [ALT]		**Step 6:** The Angular frontend calls the InventoryTypes service to send a PUT request to the C# Web API backend InventoryType controller through the DeactivateInventoryType endpoint. The backend updates the selected inventory item type's status in the following entity: InventoryType Entity: • Status [ALT]
			Step 7: The system will then record the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful deactivation of the inventory item type. The system will store information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that an inventory item type was deactivated. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry

			Step 8: The system displays a success toaster informing the admin that the inventory item type was successfully deactivated.
Alternate courses:	[ALT] Step 5: The super admin selects the "Cancel" button. Use case terminates.		
	[ALT] Step 6: The system fails to deactivate the inventory item type. The system will display an error message based on the status code.		
Conclusion:	A success toaster is displayed informing the super admin that the inventory item type was successfully deactivated.		
Post-condition:	The inventory item type is successfully deactivated.		
Business rules:	• Super admins can deactivate inventory item types.		
Implementation constraints and specifications:	• New procurements will no longer be able to use the deactivated inventory item type.		
Assumptions:	• None		
Open issues:	• None		

Table 18: Technical Narrative 3.9 Deactivate Inventory Item Type

2.4.3.10 Use Case 3.10: Reactivate Inventory Item Type

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Christopher Bailey</td><td>Date: 13 July 2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">Reactivate Inventory Item Type</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">3.10</td></tr>
<tr><td>Priority:</td><td colspan="2">Low</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Super Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event in which a super admin reactivates an inventory item type. The super admin will navigate to and select the Manage inventory types button and the system will retrieve and display inventory type records. The super admin will select the inventory item type they wish to reactivate, and the system will display the Confirm Inventory Item Type Reactivation modal. The super admin will then confirm the reactivation. The use case ends once the success toaster is displayed and the inventory item type is reactivated on the system.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">Super Admin is logged into the system.
The inventory item type is inactive.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The super admin requests to reactivate an inventory item type.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The super admin requests to reactivate an inventory item type.</td><td></td><td>Step 2: The system invokes use case 3.7 (View Inventory Item Types).</td></tr>
</table>

	Step 3: The admin selects the "Reactivate" button on the inventory item type they wish to reactivate.		**Step 4:** The system displays the Confirm Inventory Item Type Reactivation modal using the Angular frontend with the following controls: A header section containing: • A heading displaying **Reactivate Inventory Type**. A confirmation section containing: • A confirmation message asking the administrator to confirm the reactivation of the selected inventory type. • An information message indicating that the inventory type will be available for use when creating new inventory items. An error message section containing: • A validation or processing error message displayed when an error occurs during the reactivation process. A footer section containing: • A **Cancel** button to close the confirmation modal without reactivating the inventory type.

			• A **Confirm** button to reactivate the selected inventory type.
	Step 5: The super admin selects the "Confirm" button. [ALT]		**Step 6:** The Angular frontend calls the InventoryTypes service to send a PUT request to the C# Web API backend InventoryType controller through the ReactivateInventoryType endpoint. The backend updates the selected inventory item type's status in the following entity: InventoryType Entity: • Status [ALT]
			Step 7: The system will then record the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful reactivation of the inventory item type. The system will store information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that an inventory item type was reactivated. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry

			Step 8: The system displays a success toaster informing the admin that the inventory item type was successfully reactivated.
Alternate courses:	[ALT] Step 5: The super admin selects the "Cancel" button. Use case terminates.		
	[ALT] Step 6: The system fails to reactivate the inventory item type. The system will display an error message based on the status code.		
Conclusion:	The inventory item type is reactivated.		
Post-condition:	The inventory item type is successfully reactivated.		
Business rules:	• Super admins can reactivate inventory item types.		
Implementation constraints and specifications:	• New procurements will be able to use the reactivated inventory item type.		
Assumptions:	• None		
Open issues:	• None		

Table 19: Technical Narrative 3.10 Reactivate Inventory Item Type

2.4.4 Warehouse Subsystem Technical Narratives

2.4.4.1 Use Case 5.1: Create Warehouse

System Name: Integrated NGO Management System			
Author:	Stephan Bredenhann	**Date:** 13 July 2026 **Version:** 1.0	
Use case name:	Create Warehouse	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	5.1		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	User		
Description:	This use case describes the event when an Admin registers a new warehouse on the Inventory Management page. The Admin opens the "Add Warehouse" form on the Angular frontend and enters the warehouse's name, location, capacity, and security level. Once the form passes validation, the system displays a confirmation prompt. On confirmation, the Angular WarehouseService sends the details to the corresponding ".NET 8" API endpoint ("POST api/Warehouse"). The backend validates the incoming data and, using Entity Framework Core, inserts a new record with an active status. The use case concludes with the frontend refreshing the warehouse list and displaying a success notification.		
Pre-condition:	The Admin must be authenticated and viewing the Warehouses page.		
Trigger:	The Admin requests to add a new warehouse by selecting the "Add Warehouse" button.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: Admin navigates to the Inventory Management page and selects the "Warehouses" tab.		**Step 2:** The system loads the Warehouses screen, displaying a grid of existing warehouses and an "Add Warehouse" button.

	Step 3: Admin selects "Add Warehouse".		**Step 4:** The system opens a form containing fields for Name, Location, Capacity, and Security Level, with the "Create" button disabled until all fields are valid.
	Step 5: Admin completes the form and selects "Create". [ALT]		**Step 6:** The system checks that Name and Location are not empty and that Capacity is a positive number, then displays a confirmation prompt: "Are you sure you want to create this Warehouse?"
	Step 7: Admin selects "Confirm".		**Step 8:** The WarehouseService sends the warehouse details to "POST api/Warehouse". The controller validates the request and Entity Framework Core inserts a new record into the following entity: **Warehouse Entity:** • WarehouseId (PK) • Name • Location • Size • SecurityLevel • Status
			Step 9: The new warehouse is saved with its Status set to active by default. The API responds with an HTTP 201 Created status, and the frontend refreshes the warehouse grid and displays a success notification.

Alternate courses:	[ALT] Step 5: Admin leaves a required field empty or enters an invalid capacity. The system displays an inline error message and keeps the form open.
	[ALT] Step 8: The API returns an HTTP 400 response due to invalid data. The frontend displays "Please check the information entered" and reopens the form.
	[ALT] Step 8: The API returns an HTTP 403 response because the Admin lacks permission. The frontend displays "You do not have permission to add warehouses."
	[ALT] Step 8: The API returns an HTTP 500 response due to a server error. The frontend displays "System error, please try again."
Conclusion:	The use case concludes when the new warehouse is successfully saved to the database and appears in the warehouse grid.
Post-condition:	A new record exists in the Warehouse entity with an automatically generated WarehouseId and an active Status. The warehouse is now available for inventory and stock take operations.
Business rules:	• Name and Location are required fields. • Capacity must be a positive number. • A newly created warehouse is always set to active.
Implementation constraints and specifications:	• Entity Framework Core generates the WarehouseId automatically when the record is saved.
Assumptions:	• The Admin provides accurate warehouse details. There is currently no check preventing duplicate warehouse names.
Open issues:	• No server-side check exists to prevent duplicate warehouse name

Table 20: Technical Narrative 5.1 Create Warehouse

2.4.4.2 Use Case 5.2: View Warehouse

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Stephan Bredenhann</td><td colspan="2">Date: 13 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>View Warehouse</td><td colspan="2" rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>5.2</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">User</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event when an Admin views the details of a specific warehouse. From the Warehouses page, the Admin selects a warehouse card, and the system retrieves the warehouse’s details from the database and displays them in a details window, along with buttons to edit, deactivate, or reactivate the warehouse depending on its current status.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The Admin is authenticated and the selected warehouse exists in the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The Admin selects a warehouse card from the Warehouses grid.</td></tr>
<tr><td rowspan="2">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td></td><td>Step 1: Admin navigates to the Inventory Management page and selects the “Warehouses” tab.</td><td></td><td>Step 2: The WarehousesComponent calls “GET api/Warehouse”, which retrieves all warehouse records from the Warehouse entity. The system displays each warehouse as a card showing its name, location, capacity, security level, and an Active/Inactive status label.</td></tr>
</table>

	Step 3: Admin selects a warehouse card.		**Step 4:** The system calls "GET api/Warehouse/{id}" to retrieve the selected warehouse's details from the Warehouse entity. [ALT]
			Step 5: The system opens a details window showing the warehouse's Name, Location, Capacity, Security Level, and Status, along with "Edit", "Deactivate" (if active) or "Reactivate" (if inactive), and "Close" buttons.
Alternate courses:	[ALT] Step 4: The warehouse cannot be found. The system does not open the details window.		
Conclusion:	The use case concludes when the selected warehouse's details are displayed to the Admin.		
Post-condition:	No data is modified. The warehouse details remain visible until the Admin closes the window.		
Business rules:	• Only authenticated users can view warehouse details. • Active warehouses display a "Deactivate" option; inactive warehouses display a "Reactivate" option.		
Implementation constraints and specifications:	• "GET api/Warehouse/{id}" returns the warehouse's details through a read-only query, since no data is modified.		
Assumptions:	• The warehouse data shown reflects its state at the time it was retrieved.		
Open issues:	• No error message is shown if the warehouse cannot be found.		

Table 21: Technical Narrative 5.2 View Warehouse

2.4.4.3 Use Case 5.3: Update Warehouse

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Stephan Bredenhann</td><td colspan="2">Date: 13 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>Update Warehouse</td><td colspan="2" rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>5.3</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">User</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event when an Admin edits an existing warehouse’s details. From the warehouse details window, the Admin selects “Edit”, updates the pre-filled form, and confirms the change. The system then updates the corresponding record in the database. The warehouse’s active or inactive status is not affected by this use case, since that is handled separately by the deactivate and reactivate use cases.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The Admin is authenticated, the warehouse exists, and its details window is open.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The Admin selects “Edit” in the warehouse details window.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: Admin opens a warehouse’s details window (see Use Case 5.2) and selects “Edit”.</td><td></td><td>Step 2: The system closes the details window and opens an update form pre-filled with the warehouse’s current Name, Location, Capacity, and Security Level.</td></tr>
<tr><td>Step 3: Admin updates the fields and selects “Update”. [ALT]</td><td></td><td>Step 4: The system validates that Name and Location are not empty and Capacity is a positive number, then displays a</td></tr>
</table>

			confirmation prompt: "Are you sure you want to update this Warehouse?"
	Step 5: Admin selects "Confirm".		**Step 6:** The WarehouseService sends the updated details to "PUT api/Warehouse/{id}". The controller checks that the warehouse exists, and Entity Framework Core updates the Name, Location, Size, and SecurityLevel fields on the corresponding Warehouse record. The WarehouseId and Status fields are left unchanged. [ALT]
			Step 7: The API responds with an HTTP 204 status. The frontend refreshes the warehouse grid and displays a success notification.
Alternate courses:	[ALT] Step 3: Admin clears a required field or enters an invalid capacity. The system displays an inline error and keeps the form open.		
	[ALT] Step 6: The warehouse cannot be found. The system displays a generic error message.		
	[ALT] Step 6: The API returns an HTTP 400 response due to invalid data. The system displays a generic error message.		
Conclusion:	The use case concludes when the warehouse's details are successfully updated and reflected in the warehouse grid.		
Post-condition:	The warehouse's Name, Location, Size, and SecurityLevel are updated. The WarehouseId and Status remain unchanged.		
Business rules:	• Only authenticated users can update warehouse details. • The Status field cannot be changed through this use case (see Use Cases 5.4 and 5.5).		
Implementation constraints and specifications:	• "PUT api/Warehouse/{id}" requires the warehouse ID in the request body to match the ID in the URL.		

	• The update is applied directly to the tracked Warehouse entity and saved to the database.
Assumptions:	• None.
Open issues:	• There is no protection against two Admins updating the same warehouse at the same time.

Table 22: Technical Narrative 5.3 Update Warehouse

2.4.4.4 Use Case 5.4: Reactivate Warehouse

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Stephan Bredenhann</td><td colspan="2">Date: 13 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>Reactivate Warehouse</td><td colspan="2" rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>5.4</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">User</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event when an Admin restores an inactive warehouse to active status. From the details window of an inactive warehouse, the Admin selects “Reactivate” and confirms the action. The system updates the warehouse’s Status field to active.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The Admin is authenticated and the warehouse’s Status is currently inactive.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The Admin selects “Reactivate” in the details window of an inactive warehouse.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: Admin opens the details window of an inactive warehouse and selects “Reactivate”.</td><td></td><td>Step 2: The system displays a confirmation prompt: “Are you sure you want to reactivate this Warehouse?”</td></tr>
<tr><td>Step 3: Admin selects “Confirm”.</td><td></td><td>Step 4: The WarehouseService sends a request to “PUT api/Warehouse/{id}/reactivate”. Entity Framework Core updates the Status field on the corresponding Warehouse record to active, leaving all other fields unchanged. [ALT]</td></tr>
</table>

			Step 5: The API responds with an HTTP 204 status. The frontend refreshes the warehouse grid and displays a success notification.
Alternate courses:	[ALT] Step 3: Admin selects “Cancel”. The prompt closes and no changes are made.		
	[ALT] Step 4: The warehouse cannot be found. The system displays a generic error message.		
Conclusion:	The use case concludes when the warehouse’s Status is set to active.		
Post-condition:	The warehouse’s Status is active. Its card displays a green “Active” label, and its details window now shows a “Deactivate” option.		
Business rules:	• Only authenticated users can reactivate a warehouse.		
Implementation constraints and specifications:	• “PUT api/Warehouse/{id}/reactivate” requires no request body.		
Assumptions:	• None.		
Open issues:	• None.		

Table 23: Technical Narrative 5.4 Reactivate Warehouse

2.4.4.5 Use Case 5.5: Deactivate Warehouse

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Stephan Bredenhann</td><td colspan="2">Date: 13 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>Deactivate Warehouse</td><td colspan="2" rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>5.5</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">User</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event when an Admin deactivates an active warehouse, removing it from future inventory and stock take activity while keeping its historical data intact. From the details window, the Admin selects “Deactivate” and confirms the action. The system updates the warehouse’s Status field to inactive. Note that the system does not currently check whether the warehouse still holds inventory before it is deactivated, even though the original design specification calls for this check.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The Admin is authenticated and the warehouse’s Status is currently active.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The Admin selects “Deactivate” in the details window of an active warehouse.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: Admin opens the details window of an active warehouse and selects “Deactivate”.</td><td></td><td>Step 2: The system displays a confirmation prompt: “Are you sure you want to deactivate this Warehouse?”</td></tr>
<tr><td>Step 3: Admin selects “Confirm”.</td><td></td><td>Step 4: The WarehouseService sends a request to “PUT api/Warehouse/{id}/deactivate”. Entity Framework Core updates the Status field on the corresponding Warehouse record to inactive, leaving all</td></tr>
</table>

			other fields unchanged. No check is performed against existing Inventory or Stock Take records. [ALT]
			Step 5: The API responds with an HTTP 204 status. The frontend refreshes the warehouse grid and displays a success notification.
Alternate courses:	[ALT] Step 3: Admin selects "Cancel". The prompt closes and no changes are made.		
	[ALT] Step 4: The warehouse cannot be found. The system displays a generic error message.		
Conclusion:	The use case concludes when the warehouse's Status is set to inactive.		
Post-condition:	The warehouse's Status is inactive. Its card displays a red "Inactive" label, and its details window now shows a "Reactivate" option. Existing inventory records that reference the warehouse are preserved, since deletion is restricted at the database level.		
Business rules:	• Only authenticated users can deactivate a warehouse. • Deactivation is a soft delete; the warehouse record itself is never removed.		
Implementation constraints and specifications:	• "PUT api/Warehouse/{id}/deactivate" requires no request body.		
Assumptions:	• None.		
Open issues:	• None.		

Table 24: Technical Narrative 5.5 Deactivate Warehouse

2.4.4.6 Use Case 5.6: Inventory Stock Take

System Name: Integrated NGO Management System			
AUTHOR:	Stephan Bredenhann	**Date:** 13 July 2026 **Version:** 1.0	
USE CASE NAME:	Inventory Stock Take	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
USE CASE ID:	5.6		
PRIORITY:	High		
SOURCE:	EPI-USE		
Primary Business Actor (PBA):	Super Admin		
Primary System Actor (PSA):	• None		
Other Participating Actors:	• None		
Other Interested Stakeholders:	User		
Description:	The Super Admin performs a physical stock take against a single warehouse. The Super Admin opens the Stock Take page, selects a warehouse, and begins a stock take. The system takes a snapshot of every inventory item in that warehouse whose status is Available and creates one stock take line per item, each with a system quantity of 1 and a counted quantity of 0. The Super Admin then walks the warehouse floor, records the physical count for each line, and submits the stock take. On submission the system stores the counts, calculates the difference for each line, marks the stock take as Completed, and automatically writes off every inventory item that was counted as 0, so that the system inventory is adjusted to match the physical count.		
Pre-condition:	Super Admin is authenticated and holds stock take permissions. At least one active warehouse exists in the system.		
Trigger:	Super Admin clicks "New Stock Take" on the Stock Take page.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: Super Admin navigates to /inventory-		**Step 2:** System renders the StockTakeComponent and calls GET api/StockTake. The

	management and opens the "Stock Take" tab.		StockTakeController returns a list of StockTakeListDto records ordered by DateConducted descending, then StockTakeId descending, and the grid displays the stock take number, warehouse name, date, status and line count.
	Step 3: Super Admin clicks "New Stock Take".		**Step 4:** System opens the new stock take view and calls GET api/Warehouse to populate the warehouse dropdown. The "Begin Stock Take" button stays disabled until a warehouse is selected.
	Step 5: Super Admin selects a warehouse from the dropdown.		**Step 6:** System calls GET api/Inventory?warehouseId={id} and displays the current inventory of the selected warehouse as a read-only preview, together with the total item count, so that the Super Admin can see what is about to be counted.
	Step 7: Super Admin clicks "Begin Stock Take". [ALT]		**Step 8:** StockTakeService sends POST api/StockTake with a CreateStockTakeRequest containing the WarehouseId. The StockTakeController calls WarehouseExists(warehouseId) and returns HTTP 400 "Warehouse does not exist." if the warehouse cannot be found.
			Step 9: On success, the StockTakeRepository reads every Inventory row where WarehouseId matches the request and Status = Available (1). Items with status Allocated (2) or WrittenOff (3) are excluded from the snapshot.

			The repository then persists a new stock take and one line per available item. **StockTake Entity:** • StockTakeId (PK) • WarehouseId (FK) • DateConducted • Status **StockTakeLine Entity:** • StockTakeLineId (PK) • StockTakeId (FK) • InventoryItemId (FK) • SystemQuantity • CountedQuantity
			Step 10: System returns HTTP 201 Created with a StockTakeDetailsDto and a Location header pointing at GET api/StockTake/{newId}. The frontend renders the count entry screen listing every line with its SKU, item name and system quantity.
	Step 11: Super Admin walks the warehouse floor and enters the physical count for each line, then clicks "Submit Stock Take". [ALT]		**Step 12:** System shows a confirmation dialog summarising the total number of items, the number counted, the number missing, and the list of items that will be written off, with the warning that the action cannot be undone.
	Step 13: Super Admin clicks "Confirm & Submit".		**Step 14:** StockTakeService sends PUT api/StockTake/{id}/submit with a SubmitStockTakeRequest containing the StockTakeId and every line with its counted

			quantity. The controller checks that the id in the URL matches the StockTakeId in the body and returns HTTP 400 "StockTakeId mismatch." if they differ. [ALT]
			Step 15: The repository loads the stock take with its lines using Include, updates the CountedQuantity of each matching line, and sets the stock take status to Completed. **StockTakeLine Entity::** • CountedQuantity **StockTake Entity::** • Status
			Step 16: System applies the automatic inventory adjustment. Every line whose CountedQuantity is 0 is treated as a missing item, and the matching inventory record has its status changed to WrittenOff (3). All changes are saved in a single SaveChangesAsync() call. **Inventory Entity::** • Status
			Step 17: System returns HTTP 200 OK with the updated StockTakeDetailsDto. The frontend renders the discrepancy report, showing the system quantity, counted quantity and the difference (CountedQuantity - SystemQuantity) for each line, a totals row, and a warning listing how many items were automatically written off.

Alternate courses:	[ALT] Step 7: The selected warehouse no longer exists. The API returns HTTP 400 "Warehouse does not exist." and the frontend shows "That warehouse could not be found." The stock take is not created.
	[ALT] Step 9: The warehouse contains no items with status Available. The stock take is still created but with zero lines, and the count entry screen shows an empty list with the message "There is nothing to count in this warehouse."
	[ALT] Step 11: The Super Admin leaves lines at a count of 0 without physically checking them. The confirmation dialog lists every one of those items as a pending write-off so that the mistake is visible before the submission is confirmed.
	[ALT] Step 14: The id in the URL does not match the StockTakeId in the request body. The API returns HTTP 400 "StockTakeId mismatch." and the frontend shows "Please check the information entered."
	[ALT] Step 14: The stock take no longer exists. The API returns HTTP 404 and the frontend shows "That stock take could not be found."
	[ALT] Step 14: The API returns HTTP 403. The frontend shows "You do not have permission to perform a stock take."
	[ALT] Step 14: The API returns HTTP 500. The frontend shows "System error, please try again." and the captured counts remain on screen so that the submission can be retried.
Conclusion:	The stock take is stored in the [StockTakes] table with a status of Completed, every line in [StockTakeLines] carries the counted quantity entered by the Super Admin, and every inventory item that could not be found has been written off. The discrepancy report is displayed and the stock take appears at the top of the stock take list.
Post-condition:	A new row exists in [StockTakes] with Status = 'Completed', one row per counted item exists in [StockTakeLines], and every [Inventory] row whose CountedQuantity was 0 now has Status = WrittenOff (3). Written-off items are excluded from future stock takes and from allocation.
Business Rules:	• Only a user with stock take permissions may create or submit a stock take. • A stock take can only be created against an existing warehouse. • Only inventory items with status Available (1) are included in a stock take; Allocated and written-off items are excluded. • A stock take moves through two statuses only: In Progress, then Completed. • Each inventory record represents a single unit, so SystemQuantity is always 1. • The difference on a line is calculated as CountedQuantity minus SystemQuantity.

	• Any line submitted with a counted quantity of 0 causes the matching inventory item to be written off automatically.
Implementation constraints and specifications:	• GET api/StockTake returns List<StockTakeListDto> and uses AsNoTracking() for read-only performance. • GET api/StockTake/{id} returns StockTakeDetailsDto and responds with HTTP 404 when the stock take does not exist. • POST api/StockTake accepts CreateStockTakeRequest (WarehouseId) and responds with HTTP 201 Created. • PUT api/StockTake/{id}/submit accepts SubmitStockTakeRequest (StockTakeId, Lines) and responds with HTTP 200 OK. • StockTakeLine has a cascade delete on StockTakeId and a restrict delete on InventoryItemId, so counted inventory cannot be deleted while a stock take references it. • The Difference field is calculated in the projection and is not stored in the database. • StockTakeRepository and StockTakeService are registered as scoped services.
Assumptions:	• The Super Admin counts the warehouse accurately and submits every line in one request; the physical count is trusted over the system record.
Open issues:	• None • The [Authorize] attributes on the StockTakeController are still commented out, so the endpoints are currently unprotected. • Stock take creation and submission are not written to the audit log, so there is no record of who performed the count. • A submitted stock take cannot be reopened or corrected, and a second submission silently overwrites the first.

Table 25: Technical Narrative 5.6 Inventory Stock Take

2.4.5 Vendor Subsystem Technical Narratives

2.4.5.1 Use Case 6.1: Create Vendor

<table>
<tr><th colspan="3">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Amahle Khoza</td><td>Date: 12 June 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>Create Vendor</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>6.1</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="2">• Super Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="2">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="2">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="2">Admin</td></tr>
<tr><td>Description:</td><td colspan="2">This use case details the system flow where an authorized user registers a new vendor profile within the application. The user captures corporate metrics, type parameters, contact paths, and an image profile asset into an open Angular Reactive Form Group layout interface. Upon clicking save, the frontend interface verifies control validity and dispatches an asynchronous multi-part HTTP POST call payload to the corresponding backend .NET 8 API endpoint router path (POST api/Vendor/create) carrying the text models along with base64 data strings representing the image file. The backend controller binds incoming parameters to an explicit database context tracking instance, executes validation algorithm loops, and commits changes via Entity Framework Core. The use case finishes once the frontend updates directory data grids, closes form panels, and fires a green confirmation toast notification.</td></tr>
<tr><td>Pre-condition:</td><td colspan="2">• The executing user must be authenticated and hold appropriate vendor registry creation and system management claims.</td></tr>
<tr><td>Trigger:</td><td colspan="2">The user requests to register a new vendor within the application by clicking the "Add New Vendor" action button trigger component interface link.</td></tr>
<tr><td rowspan="2">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td></td><td>Step 1: The user requests to register a new vendor within the application by clicking the "Add New Vendor" action button trigger component interface link.</td><td></td><td>Step 2: The Angular framework handles view compilation logic, loading the dedicated Vendor Form template context into the active workspace workspace slot.</td></tr>
</table>

		Step 3: The client interface generates and renders an open, interactive form pane display presenting a structured Angular Reactive Form Group containing empty text boxes, option drop-downs, and file uploader utilities targeting corporate company name, vendor classification type selection, performance metrics score, contact email string details, telephone digits, credit facility caps, physical street address lines, and a profile image stream drop-zone.	
	Step 4: The user populates the relevant descriptive data input targets, links a business type category classification, attaches an initial performance score, triggers an image upload stream file select, and hits the primary "Save Vendor" record		**Step 5:** The client-side framework evaluates active form control validation states. The Angular VendorService builds and dispatches an asynchronous multi-part HTTP POST call payload targeting backend routing path endpoints api/Vendor/create passing the text models along with

	submission controller [ALT]		base64 data strings representing the image file. The backend .NET 8 service intercepts the package payload context. The runtime builds a database connection via the ApplicationDbContext and queries structural repository indices to verify that the target vendor registration numbers or email addresses do not trigger duplication violations.
			Step 6: The API executes model binding constraint evaluations on the application server layers. The system checks data structure parameters against backend fields (checking mandatory items, email string notation properties, numeric limits, and checking telephone formats).
			Step 7: Upon successful validation passes across all system parameters, the backend instantiates a new record row entry instance for the Vendor data model map, appends automated transaction timestamp tracking properties, and issues an asynchronous commit call (SaveChangesAsync).
			Step 8: The application server responds with an HTTP 201 Created action success block stream. The frontend catches the transaction data response,

			inserts the newly returned vendor profile instance into its master directory data display grids, closes the form pane interface drawers, and displays a green confirmation toast notice layout stating: "Vendor record successfully added to global system directories."
Alternate courses:	**[ALT] Step 4:** The user clicks the "Discard" or "Exit" interface option inside the form panel framework drawer layout. The frontend interface stops outbound JSON payload packaging routines, releases active form variables from memory buffers, prevents network data calls to the server, and closes down active panels.		
Conclusion:	The use case concludes when the system successfully validates registration parameters, establishes tracking rows inside the database backend tables, and updates global directory data grids.		
Post-condition:	• A new vendor model row profile is permanently written to the database structure, enabling its selection across lookup picklists and project procurement transaction modules.		
Business rules:	• Every vendor profile record row must contain a completely unique business contact email string notation to prevent directory mapping overlaps. • New vendor profiles must bind to non-negative initial performance tracking score variables.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 26: Technical Narrative 6.1 Create Vendor

2.4.5.2 Use Case 6.2: View Vendor

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 12 June 2026 **Version: 1.0**	
Use case name:	View Vendor	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	6.2		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Admin		
Description:	This use case describes how the system retrieves and displays a comprehensive, standalone read-only view of a selected vendor entity profile line item. When a user selects a record row, the frontend evaluates routing parameters and dispatches an asynchronous HTTP GET request to the matching .NET 8 API endpoint (GET api/Vendor/details/{id}). The backend controller targets the relational rows via the master ApplicationDbContext, executing optimized select query statement operations that perform automated table joins across linked lookup schemas (classification types, contact address arrays, performance records). The server packages variables into a structured Data Transfer Object (DTO) payload and dispatches an HTTP 200 OK code back to the client thread. The use case ends when the frontend maps incoming metrics directly into template interpolations, rendering data arrays into uneditable screen components.		
Pre-condition:	• The target vendor instance record must currently exist inside the database layer partitions with functional active index mapping nodes.		
Trigger:	The user selects a specific vendor entity profile line-item row within the master directory data table list by clicking its corresponding "View Details" text link or row element.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user selects a specific vendor entity profile line-item row within the master directory data table list by clicking its corresponding		**Step 2:** The Angular Router engine captures the user click event stream, evaluates routing templates, and triggers path transitions navigating the layout container over to

	"View Details" text link or row element.		the dedicated vendor single profile view path, appending the unique ID parameter mapping (/vendors/view/:id).
			Step 3: The profile display component launches its initialization lifecycle hooks (ngOnInit). The underlying Angular VendorService builds and fires an asynchronous HTTP GET transaction targeting the backend server resource URL path parameters mapping api/Vendor/details/{id}. The .NET 8 endpoint processes incoming parameters. The underlying architecture builds database calls via the system ApplicationDbContext layer and runs an optimized, compiled read query select statement (FirstOrDefaultAsync) to isolate rows corresponding to the passed identifier [ALT].
			Step 4: The database engine handles automated table joining procedures across associated lookup relational models, fetching tracking variables linked to the primary entity schema including corporate names, telephone parameters, profile images, and historical performance statistics lines.

			Step 5: The backend system architecture transforms the returned raw entities into a structured, lightweight Data Transfer Object (DTO) contract collection and returns an HTTP 200 OK success transmission block to the browser client.
			Step 6: The Angular frontend application intercepts the structured network data. The view component processes the model properties and updates template interpolation states to display complete email paths, ratings, and uploaded imagery files on the interface while locking form inputs into an uneditable, read-only display layout state.
Alternate courses:	**[ALT] Step 3**: The database fetch query evaluates an empty data block stream because the targeted vendor record primary key sequence has been concurrently modified or removed from disk. The backend web API stops the execution loop and delivers an HTTP 404 Not Found error packet. The frontend interceptor captures the status error token, breaks loading states, and displays a red warning dialog stating: "The requested vendor profile record could not be fetched. It may have been deactivated or dropped from system directories."		
Conclusion:	The use case concludes cleanly when data streams transfer successfully from backend databases, parsing tracking metrics directly into uneditable, read-only interface blocks.		
Post-condition:	• The single vendor entity's descriptive statistics, corporate attributes, and performance evaluations are rendered seamlessly on the user interface grid framework.		
Business rules:	• Visibility rules require the system to subject single view detail queries to authorization algorithms ensuring the active user holds clearance claims to browse corporate data repositories.		
Implementation constraints and specifications:	• Processing routines should leverage specialized Data Transfer Objects (DTOs) to enforce network transmission boundaries and strip core infrastructure properties from transmission payloads.		

Assumptions:	• None
Open issues:	• None

Table 27: Technical Narrative 6.2 View Vendor

2.4.5.3 Use Case 6.3: View Vendors

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 12 June 2026 **Version: 1.0**	
Use case name:	View Vendors	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	6.3		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Admin		
Description:	This use case describes how the system extracts and lists all registered vendor tracking profiles from backend database layers. Upon loading the master portal module layout, the client application framework initializes initialization lifecycle code hooks, triggering the internal VendorService to dispatch an asynchronous HTTP GET request transaction to the corresponding server routing URL path (GET api/Vendor/list). The .NET 8 backend controller handles incoming request parameters and queries active rows using Entity Framework Core across global vendor datasets, applying sorting configurations dynamically. The server bundles records into an array list data transfer transfer block and returns an HTTP 200 OK response code. The use case finishes once the frontend loops through returned structures via *ngFor row generation commands, rendering rows inside an interactive table grid layout.		
Pre-condition:	• The user must be authenticated and maintain active data connectivity streams linking host server environments.		
Trigger:	The user navigates to the master Vendor Directory portal layout module via the system dashboard link sidebar menu framework.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user accesses the centralized Vendor Management workspace directory layout by clicking on the designated "Vendors" link button option inside the primary		**Step 2:** The Angular client shell captures routing events, updates active layout slots, and initializes the parent Vendor List Component view container framework within the browser window DOM.

	sidebar application navigation menu.		
			Step 3: The view initialization hooks trigger automated communication sequences. The internal client VendorService builds and dispatches an asynchronous HTTP GET request to the target server API list path route context api/Vendor/list, passing search query filters or status constraint settings. The .NET API controller passes execution commands over to data access contexts. The framework calls into the master ApplicationDbContext instance and prepares an evaluation query string targeted at extracting datasets from global active Vendor tables.
			Step 4: The database instance executes a compiled select query via Entity Framework Core, scanning table rows, sorting records alphabetically by corporate company name parameters, and marshaling records into a clean data collection stream.
			Step 5: The application server packs the assembled record items into an array payload model block structure and issues an HTTP 200 OK network success message package

			back to the client browser app..
	Step 6: The user can adjust dropdown filters or option checkboxes to filter by status (e.g., toggling to view active rows only), which triggers updated compilation data queries.		**Step 7:** The Angular layout engine processes the incoming array block. The framework evaluates template structural looping directives (*ngFor) and loops through the objects to dynamically generate structured table grid rows that present corporate titles, classification indicator pills, and action control asset overlays on screen.
Alternate courses:	None		
Conclusion:	The use case concludes cleanly when serialized collection records compile successfully from databases, updating structural dashboard views to project row grids.		
Post-condition:	• Registered vendor directory data profiles are extracted from backend systems and mapped onto structural interface layouts for query navigation.		
Business rules:	• Security parameters strictly filter dataset records matching security access matrix constraints allocated to the authenticated user profile.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 28: Technical Narrative 6.3 View Vendors

2.4.5.4 Use Case 6.4: Edit Vendor

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 12 June 2026 **Version: 1.0**	
Use case name:	Edit Vendor	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	6.4		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Admin		
Description:	This use case details the step-by-step process where an authenticated operator alters specific descriptive, financial, or contact parameters inside an existing vendor's active registration profile. The user overrides cell properties within an Angular Reactive Form Group layout interface container modal. Upon submission, the client interface runs structural validation checks and routes an asynchronous HTTP PUT network transaction directly to the backend API endpoint (PUT api/Vendor/edit/{id}). The .NET 8 backend references the shared ApplicationDbContext tracker context to isolate the target row entity, runs duplication and constraint verification logic, and flags updates via Entity Framework Core. The use case finishes when mutations write cleanly to relational tables, the editing dialog modal dismisses, and the calling view data layout refreshes.		
Pre-condition:	• The targeted vendor profile instance must currently exist in persistent SQL storage tables with functional, active row mapping indexes, and the operator must possess valid registration write claims.		
Trigger:	The user clicks the specific row context "Edit" action button trigger control asset within the master registry directory table.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user requests to alter specific descriptive parameters inside an existing vendor's registration profile card by clicking the row "Edit" action button trigger.		

		Step 2: The frontend application launches localized editing modal overlay components over the active screen view context, automatically mapping current database structural field parameters into active input boxes via form data control patch value routines.	
	Step 3: The user overrides specific parameter properties within active input controls (modifying phone numbers, expanding credit facility caps, or replacing attachment image files) and clicks the "Save Changes" submit button. [ALT]		**Step 4:** The client application handles validation constraint checks across input properties. The Angular VendorService formats an asynchronous HTTP PUT call dispatched directly to the API routing endpoint template string context api/Vendor/edit/{id}, packing the updated payload map. The backend web API intercepts the data transmission stream. The execution thread references the shared ApplicationDbContext context block and passes asynchronous search statements targeting matching primary key vendor identities across relational database indices.
			Step 5: The server backend processes validation verification checks to

			ensure the target row data item index remains valid, text characters remain inside field type allocations, and modified entry properties do not generate record duplication path collisions.
			Step 6: Upon successful logic validation confirmations, the application context tracks item level changes, overwrites database schema column records with payload parameters, and commits data updates via an asynchronous save update call (SaveChangesAsync).
			Step 7: The .NET API shifts execution pipelines and delivers an HTTP 200 OK response status statement block back to the calling client. The frontend intercepts the success code message, closes down the modal dialogue wrapper overlay panels, reloads view data layouts, and fires a green confirmation toast reading: "Changes committed successfully."
Alternate courses:	**[ALT] Step 3:** The user aborts adjustments and clicks the "Discard" or "Cancel" control option. The frontend interface stops outbound JSON compilation routines, drops all form state alterations from memory buffers, prevents network data calls to the server host, and closes active panels immediately.		
Conclusion:	The use case concludes once all systemic validations clear successfully, updated parameters persist within database disk files and tracking directory layers re-render.		
Post-condition:	• The target vendor instance database row fields are permanently modified with the new configuration parameters, immediately		

	reflecting changes across downstream procurement and accounting workflows.
Business rules:	• Any modification targeting fundamental financial parameters (such as credit facility caps or bank details) must trigger an automated background audit trail log capturing the operator ID and field delta. • Altered contact email string notations must remain completely unique across all active and inactive vendor registry records.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 29: Technical Narrative 6.4 Edit Vendor

2.4.5.5 Use Case 6.5: Reactivate Vendor

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 12 June 2026 **Version: 1.0**	
Use case name:	Reactivate Vendor	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	6.5		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Admin		
Description:	This use case details the workflow executed when an authorized operator restores a previously suspended or deactivated vendor profile entry back to full system operational availability. The user triggers the action from a deactivated row item context, causing the frontend to display a verification prompt overlay dialog warning framework. Upon confirmation, the client browser application issues an asynchronous HTTP PATCH operations statement directly to the API path parameter route (PATCH api/Vendor/reactivate/{id}). The backend .NET 8 controller handles the arrival command, uses the ApplicationDbContext to locate the target entity row, and executes authorization confirmation checks. The system updates the status flag parameter to true, persists changes to disk, closes the overlay, and pushes a green success confirmation toast message banner.		
Pre-condition:	• The targeted vendor record entry must exist inside database partitions with an active status flag currently set to false (IsActiveStatus = false).		
Trigger:	The user hits the specific row context "Reactivate" action switch trigger asset control on a deactivated data grid line item.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user requests to restore a previously suspended vendor profile entry by hitting the row "Reactivate" contextual action button trigger asset control on a deactivated data item row.		

		Step 2: The client framework launches a warning modal prompt overlay display, showing a descriptive verification question string text layout to the system operator: "Are you sure you want to reactivate vendor [Vendor Name]? It will immediately reappear when creating project assignments."	
	Step 3: The user clicks on the "Confirm Reactivate" action switch button located within the active prompt modal window dialog overlay container. [ALT]		**Step 4:** The frontend interface processes the click event intent. The Angular VendorService builds and dispatches an asynchronous HTTP PATCH operations network statement directly targeting the API url parameter path route structure api/Vendor/reactivate/{id}. The .NET backend controller captures the incoming patching command request. The application layer triggers a target data fetch process via the ApplicationDbContext instance to isolate the targeted row record tracking structure inside persistent storage systems.
			Step 5: The system subjects execution calls internal backend authorization validations and database

			checks to confirm the row entity profile index matches valid registry structures and ensure the operator holds valid workspace claim rights.
			Step 6: Upon verification approvals, the ORM entity state tracker context isolates the explicit status parameter target column, flips the data visibility value flag over to true (IsActiveStatus = true), and persists transaction modifications to persistent database disk layers via an asynchronous statement (SaveChangesAsync).
			Step 7: The web API transfers an HTTP 200 OK network success verification payload package. The client browser interceptor handles the code status, updates the data list view layout configuration matrices, closes the dialog prompt modal window context, and pops a green confirmation message alert banner reading: "Vendor profile reactivated successfully."
Alternate courses:	**[ALT] Step 3:** The user decides to cancel reactivation and clicks the "Cancel" or "Dismiss" button control within the prompt layout block container wrapper. The system immediately closes the modal prompt window view context, drops the execution intent, and leaves the database record entry status completely unchanged.		
Conclusion:	The use case concludes when the system completes backend status state validations, successfully updates database table records, closes the active verification overlay dialog wrapper, and refreshes the master data grid cache.		

Post-condition:	• The target vendor entity's data visibility flag is permanently flipped to true (IsActiveStatus = true), making the profile immediately visible across project procurement lookup picklists.
Business rules:	• Status elevation operations are restricted to operators holding valid administrative configuration or high-level procurement workspace claims.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 30: Technical Narrative 6.5 Reactivate Vendor

2.4.5.6 Use Case 6.6: Deactivate Vendor

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 12 June 2026 **Version: 1.0**	
Use case name:	Deactivate Vendor	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	6.6		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Admin		
Description:	This use case details the technical steps executed to suspend a vendor profile line-item row, restricting it from being selected for future project workflows without deleting its historic transactional tracking data. The user triggers the action from the row context. The frontend interrupts workflows to mount a confirmation warning alert overlay modal block framework. Upon user affirmation, the Angular service layer issues an asynchronous HTTP PATCH request to the backend controller path (PATCH api/Vendor/deactivate/{id}). The .NET 8 API handles the tracking token, verifies database row integrity via the master ApplicationDbContext, and evaluates active relational associations. If validations pass, the runtime switches the entity status property flag over to false. The use case finishes once updates save to disk, the modal disappears from the DOM view, and a green success toast fires.		
Pre-condition:	• The targeted vendor profile record must exist inside the database context frames with a functional active index state matching true (IsActiveStatus = true).		
Trigger:	The user requests to suspend a vendor profile line-item row by clicking its specific context "Deactivate" trigger button asset.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user requests to suspend a vendor profile line-item row by clicking its specific row item context "Deactivate" trigger button.		
		Step 2: The frontend	

		interrupts the execution flow and displays a confirmation warning alert overlay modal presenting the text: "Deactivate vendor [Vendor Name]? It will no longer appear when creating projects."	
	Step 3: The user clicks the "Deactivate" confirmation action trigger control within the modal pop-up window block framework. [ALT]		**Step 4:** The client-side interface captures the click event. The Angular VendorService builds and dispatches an asynchronous HTTP PATCH request to the backend service controller path api/Vendor/deactivate/{id} passing the entity's primary key identifier. The API endpoint routes processing workflows to the data layer. The runtime references the master ApplicationDbContext instance to establish database connections and search-targets the matching vendor entity records index row.
			Step 5: The backend server runs internal logic validation rules. The application checks that the targeted vendor record exists in the database table and evaluates active operational data associations to ensure the vendor profile isn't

			currently assigned to an ongoing project workflow.
			Step 6: If all validation rules clear successfully, the system updates the targeted entity tracking field values, changing the status flag column to false (IsActiveStatus = false), and commits the state transaction via an asynchronous database command (SaveChangesAsync).
			Step 7: The API shifts execution pipelines and delivers an HTTP 200 OK network success response payload. The client application catches the response, updates the local directory layout caches, hides the modal overlay window component from the DOM view, and pops a green success toast notification stating: "Vendor record successfully flagged deactivated."
Alternate courses:	**[ALT] Step 3:** The user aborts the suspension and clicks the "Cancel" button control asset inside the confirmation dialog window block layout. The modal container layer closes immediately, the workflow halts, and the vendor's active database status remains completely unchanged.		
Conclusion:	The use case concludes once all data integrity and dependency validations clear, the target tracking column updates successfully, and local frontend directory caches reload.		
Post-condition:	• The specific vendor record status column is permanently flagged as false (IsActiveStatus = false), completely rendering the entry hidden from all future transactional selection picklists.		
Business rules:	• The system strictly blocks deactivation requests targeting any vendor profile with active downstream project ties to prevent relational database mapping breaks.		

Implementation constraints and specifications:	• Relational lookup constraints must evaluate active project arrays via Entity Framework Core tracking contexts ahead of committing state alterations.
Assumptions:	• None
Open issues:	• None

Table 31: Technical Narrative 6.6 Deactivate Vendor

2.4.5.7 Use Case 6.7: Create Vendor Type

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 12 June 2026 **Version: 1.0**	
Use case name:	Create Vendor Type	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	6.7		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Admin		
Description:	This use case details the system flow where an authorized user establishes a new category classification label within the vendor lookup directories. The user populates an open Angular Reactive Form Group interface layout detailing alphanumeric controls for Classification Type Name, Functional Description Mapping Scope, and Initial Status. Upon clicking submit, the client interface checks form control validation syntax states and dispatches an asynchronous HTTP POST request to the backend API endpoint (POST api/VendorType/create) packing a structured CreateVendorTypeRequest JSON payload body. The .NET 8 backend controller handles the arriving payload, performs unique naming verification checks against the database using the ApplicationDbContext, maps parameters into a new VendorType data model row instance, and executes an asynchronous save update command. The use case concludes when the server returns an HTTP 201 Created code, local frontend array caches update, form states clear, and a green confirmation toast notification fires.		
Pre-condition:	• The executing operator must possess valid administrative system configuration privileges, and session authentication tokens must be valid.		
Trigger:	The user requests to add a new classification category by clicking the "Create Vendor Type" or "Manage Types" action button trigger component asset.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user requests to add a new category by clicking the "Create Vendor Type" or "Manage		

	Types" action button trigger component.		
		Step 2: The system launches the type of generation modal wrapper overlay. The frontend generates and renders an open Angular Reactive Form Group interface detailing an alphanumeric text input field for the Classification Type Name, a text area for the Functional Description Mapping Scope, an option selector for Status, a "Discard" button, and a primary "Commit Records" submit button.	
	Step 3: The user populates the classification type name input field, enters descriptive details about the category scope, selects an initial status configuration, and clicks the primary "Commit Records" button. [ALT]		**Step 4:** The frontend framework evaluates active client field validation syntax rules. The Angular VendorService builds and dispatches an asynchronous HTTP POST request to the backend API api/VendorType/create route packing a structured CreateVendorTypeRequest JSON payload body. The backend controller intercepts the incoming payload. The application

			establishes connections to the storage layer via the ApplicationDbContext interface and triggers an asynchronous read check query against the "VendorType" entity rows to verify that the classification name string does not conflict with an existing entry.
			Step 5: The server backend processes validation checks against data model constraints. The system checks that the category name string is populated and unique, verifies that field lengths do not exceed maximum database character allocations, and confirms the active session token contains necessary administrative privileges.
			Step 6: Upon completing verification loops, the system maps the validated payload parameters into a new instance entry of the VendorType database model table collection and saves the tracking metrics via an asynchronous commit statement (SaveChangesAsync).
			Step 7: The API endpoint returns an HTTP 201 Created code success response. The frontend catches the response message, appends the newly returned vendor type category instance into its local cache data arrays, clears the active modal

			form state layer, and triggers a green confirmation toast statement reading: "New vendor type category successfully added to global system directories."
Alternate courses:	**[ALT] Step 3:** The user cancels the category generation and clicks the "Discard" button control asset inside the form panel modal drawer. The frontend interface stops outbound payload packaging routines, releases active form variables from memory buffers, prevents network data calls to the server host, and closes the modal view layer.		
Conclusion:	The use case concludes when data models clear validation parameters, the classification record registers successfully inside database backend tables, and configuration lookup dashboards refresh.		
Post-condition:	• A new vendor classification type entity row is permanently registered to the global lookup directories, making it immediately available for sorting vendor profiles.		
Business rules:	• The system blocks the generation of any new vendor type entity containing a category label string that matches an existing entry exactly (case-insensitive) within the table partitions.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 32: Technical Narrative 6.7 Create Vendor Type

2.4.5.8 Use Case 6.8: Update Vendor Type

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 12 June 2026 **Version: 1.0**	
Use case name:	Update Vendor Type	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	6.8		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Admin		
Description:	This use case details the step-by-step process for modifying an existing vendor type classification category's name string or description scope metrics. The user adjusts field metrics within an active edit workflow modal overlay window container, populated automatically via an explicit Angular form patchValue operation. Upon clicking save, the client interface checks browser-layer validation rules and dispatches a structured HTTP PUT call directly to the endpoint URL parameter path (PUT api/VendorType/update/{id}). The API endpoint binds to the shared ApplicationDbContext instance, runs an asynchronous lookup transaction seeking a matching primary key sequence, evaluates payload modifications against backend validation constraints, and overwrites the tracked database entity schema properties with the new payload values. The use case concludes when changes commit to disk via an asynchronous update command, the modal popup container closes, and a confirmation alert toast fire.		
Pre-condition:	• The targeted vendor type record instance must exist within the database layer partitions, and the operator must hold valid system configuration management write claims.		
Trigger:	The user requests to alter a vendor category type name or description by clicking the row "Edit" button layout element asset next to a specific listing line item.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user requests to alter a vendor category type name or description by clicking the row "Edit" button layout element.		

		Step 2: The frontend triggers the edit workflow modal overlay window container. The system passes the active row parameters and patches the existing database field metrics directly into the form text inputs via an explicit Angular form patch operation (patchValue).	
	Step 3: The user modifies the alphanumeric text input values (updating category names or adjusting functional scope explanations) and clicks the "Save Changes" execution control button. [ALT]		**Step 4:** The client application verifies form control validity on the browser layer. The Angular VendorService dispatches a structured HTTP PUT call directly to the backend endpoint URL parameter path api/VendorType/update/{id}, passing the updated text parameter payload maps. The API endpoint processes data extraction streams. The tracking logic binds to the shared ApplicationDbContext instance and runs an asynchronous lookup transaction statement (SingleOrDefaultAsync) seeking a matching vendor type primary key sequence mapping.
			Step 5: The system evaluates payload modifications against backend validation constraints. It checks that the

			targeted vendor type record is found in the database layer, assures alphanumeric character arrays do not break size properties, and validates that the active session holds necessary configuration management permissions.
			Step 6: Upon completing verification loops, the system overwrites the tracked database entity schema properties with the new payload values and saves changes to the database storage layer via an asynchronous update command (SaveChangesAsync).
			Step 7: The server backend dispatches an HTTP 200 OK status packet back to the client interface. The frontend application intercepts the success code message, closes the modal popup block container, updates its local state arrays, and triggers a confirmation alert toast reading "Changes committed successfully."
Alternate courses:	**[ALT] Step 3:** The user aborts mutations and clicks the "Discard" or "Cancel" execution control button. The frontend interface closes the editing panel layout immediately, drops all form changes from memory buffers, and blocks outbound network data transactions to the server host.		
Conclusion:	The use case concludes when the system successfully validates modified parameters, saves updates to the database storage layer, closes active modal boxes, and refreshes the main dashboard lookup grid views.		
Post-condition:	• The target vendor type category's database entity fields are permanently updated with the new input text parameters, modifying classification text across downstream dashboard views.		
Business rules:	• Configuration mutations targeting global classification directories are restricted strictly to users with active administrative configuration or system management privileges.		

Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 33: Technical Narrative 6.8 Update Vendor Type

2.4.5.9 Use Case 6.9: View Vendor Types

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 12 June 2026 **Version: 1.0**	
Use case name:	View Vendor Types	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	6.9		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Admin		
Description:	This use case describes how the system extracts and lists all registered vendor type classification profiles from backend database layers to render them on dashboard views or fill selection picklists. Upon loading the target lookup view or a vendor form requiring classification picklists, the client framework initializes initialization lifecycle code hooks (ngOnInit). This triggers the internal Angular service layer to build and dispatch an asynchronous HTTP GET transaction to the backend server routing endpoint (GET api/VendorType/list). The .NET 8 backend controller handles the request, references the master ApplicationDbContext, runs an optimized select query statement applying sorting criteria, bundles matching rows into an array data transfer block, and dispatches an HTTP 200 OK status code back to the client thread. The use case finishes once the frontend loops through returned structures via *ngFor row generation commands, rendering rows into an interactive table layout grid.		
Pre-condition:	• The user must be authenticated within the application and maintain active network data connectivity streams linking host server environments.		
Trigger:	The user opens the vendor type category configuration dashboard view, navigating to the lookup panel or loading an administrative input form requiring category arrays.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user opens the vendor type category configuration dashboard view (either by navigating to the lookup panel or		

	loading a vendor creation form that requires category picklists).		
			Step 2: The Angular engine workspace shell loads the component framework layout and triggers automated initialization lifecycle code blocks (ngOnInit).
			Step 3: The internal Angular VendorService constructs and dispatches an asynchronous HTTP GET transaction to the backend server routing endpoint api/VendorType/list. The backend controller routes processing workflows to data layers. The system references the master ApplicationDbContext and constructs a query expression evaluating active rows across the global VendorType data sets.
			Step 4: The database management system runs an optimized select query statement using Entity Framework Core, processing sorting parameters, and assembling list items into a clean dataset transfer block.
			Step 5: The server bundles the matching rows into an array collection dataset transfer block and returns an HTTP 200 OK response to the client.

			Step 6: The Angular template loops through the returned collection using structured *ngFor row generation commands, rendering rows into an interactive table layout grid that displays category names, description text, status badges, and operation management action buttons.
Alternate courses:	None		
Conclusion:	The use case concludes cleanly when serialized collection records compile successfully from databases, updating structural dashboard layouts or picking arrays.		
Post-condition:	• Registered vendor category classification profiles are extracted from backend systems and mapped onto structural interface layouts for query navigation.		
Business rules:	• The collection return payload must include inactive tracking rows when requested inside administrative lookup views but filter out inactive instances when accessed from secondary client picklist creation windows.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 34: Technical Narrative 6.9 View Vendor Types

2.4.5.10 Use Case 6.10: Reactivate Vendor Type

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 12 June 2026 **Version: 1.0**	
Use case name:	Reactivate Vendor Type	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	6.10		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Admin		
Description:	This use case details the process invoked when an authorized operator restores a deactivated vendor type classification category back to system-wide visibility. The user requests reactivation from a target row item layout context. The frontend displays a verification prompt overlay block dialog framework. Upon user confirmation, the client application issues an asynchronous HTTP PATCH update request directly to the backend endpoint URL parameter path (PATCH api/VendorType/reactivate/{id}). The .NET 8 backend controller intercepts the transaction command, sets up connections to the storage layer via the ApplicationDbContext, and evaluates internal backend logic validations. Upon verification approval, the system updates the target entity's status tracking column to true (IsActiveStatus = true) and commits updates to persistent relational disk files. The use case finishes when the server dispatches an HTTP 200 OK status code, the client refreshes category displays, hides active modals, and fires a green success toast notification.		
Pre-condition:	• The targeted vendor classification category record must exist in the database with its operational visibility flag currently set to false.		
Trigger:	The user requests to restore a deactivated vendor category by clicking its row "Reactivate" button trigger layout asset component.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user requests to restore a deactivated vendor category by clicking its row "Reactivate" button trigger layout asset.		

		Step 2: The frontend framework displays a modal verification prompt overlay showing the text: "Are you sure you want to reactivate vendor type [Category Name]? It will immediately reappear when classifying project vendors."	
	Step 3: The user assesses the prompt context and clicks on the "Confirm Reactivate" action control button inside the confirmation dialog panel overlay. [ALT]		**Step 4:** The front end interface captures the execution choice. The Angular VendorService sends an asynchronous HTTP PATCH update request directly to the backend endpoint URL parameter path api/VendorType/reactivate/{id}. The backend controller handles the arriving transaction request. The application establishes connections to the database storage layer via the ApplicationDbContext configuration interface and locates the targeted vendor type record item index row.
			Step 5: The system evaluates internal backend logic validations, verifying that the specific vendor type row exists in the table partitions and assuring the executing user possesses valid configuration workspace privileges.
			Step 6: Upon verification approval, the system updates the target entity tracking field

			values, toggling the explicit schema state column status indicator flag over to true (IsActiveStatus = true), and commits the state transaction via an asynchronous update command (SaveChangesAsync).
			Step 7: The API sends back an HTTP 200 OK status response. The client application catches the code response message, refreshes the category data display lists, removes the modal view layer from the active layout, and renders a green success toast saying, "Vendor type category reactivated successfully."
Alternate courses:	**[ALT] Step 3:** The user dismisses the reactivation prompt and clicks the "Cancel" or "Discard" button control asset inside the confirmation dialog panel layout overlay. The system immediately closes the modal prompt container, terminates the operational intent, and leaves the target database record parameters completely unchanged.		
Conclusion:	The use case concludes once all systemic validations clear successfully, the classification visibility flag updates within database storage files, prompt frames close, and configuration lookup dashboards refresh.		
Post-condition:	• The specific category row tracking index visibility property is permanently flipped to true, making the classification option immediately selectable within vendor profile setup sheets.		
Business rules:	• Only users possessing valid configuration management workspace privileges and explicit system modification security claims can re-elevate classification entity status values.		
Implementation constraints and specifications:	• State adjustments must execute over an asynchronous pipeline leveraging Entity Framework tracking context blocks (SaveChangesAsync).		
Assumptions:	• None		
Open issues:	• None		

Table 35: Technical Narrative 6.10 Reactivate Vendor Type

2.4.5.11 Use Case 6.11: Deactivate Vendor Type

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 12 June 2026 **Version: 1.0**	
Use case name:	Deactivate Vendor Type	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	6.11		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Admin		
Description:	This use case details the step-by-step process executed when an authorized user suspends a vendor classification category type entity row from systemic use, preventing it from classifying future vendor profiles while preserving existing data relationships. The user triggers deactivation from a specific category row item. The frontend generates a confirmation warning alert overlay modal. Upon affirmation, the Angular client interface builds and dispatches an asynchronous HTTP PATCH request to the backend service controller routing path (PATCH api/VendorType/deactivate/{id}). The backend controller intercepts the request and uses the ApplicationDbContext to verify entity status parameters and run dependency evaluation rules. If all rules pass, the context tracker updates the status flag column to false. The use case finishes once table rows update on disk, the modal overlay window closes from the DOM view, and a green success notification statement flashes.		
Pre-condition:	• The targeted vendor type record row must exist inside relational tables with its structural status flag currently matching true (IsActiveStatus = true).		
Trigger:	The user requests to suspend a vendor category by clicking its specific row item context "Deactivate" trigger button layout element asset.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user requests to suspend a vendor category by clicking its specific row item context		

	"Deactivate" trigger button.		
		Step 2: The frontend halts direct executions, generating a confirmation warning alert overlay displaying the message text: "Deactivate vendor type [Category Name]? It will no longer appear when creating or editing vendors."	
	Step 3: The user clicks the "Deactivate" confirmation action trigger control within the confirmation modal pop-up window block framework. [ALT]		**Step 4:** The frontend interface processes the user action event. The Angular VendorService issues an asynchronous HTTP PATCH request to the backend service controller path api/VendorType/deactivate/{id} carrying the primary key parameter. The API endpoint intercept routes processing workflows to the data layer. The master ApplicationDbContext connects to the relational tables and targets the matching vendor type entity record index line row.
			Step 5: The system evaluates business rule validations before proceeding. The application executes structural checks to verify that the specific vendor type record row is found and assesses that no active vendor profiles are currently classified under this category to protect

			downstream system data mapping integrity.
			Step 6: If all validation rules pass, the system updates the record's status flag column to false (IsActiveStatus = false) using the Entity Framework context tracker and commits the state transaction via an asynchronous update command (SaveChangesAsync).
			Step 7: The API sends an HTTP 200 OK status response code. The client interface captures the success message, reloads category directory layouts, hides the modal overlay window, and displays a success notification statement: "Vendor type category successfully flagged deactivated."
Alternate courses:	**[ALT] Step 3:** The operator chooses to cancel the action and clicks the "Dismiss Deactivation" or "Cancel" control within the confirmation modal pop-up window block framework. The overlay layout closes immediately, execution stops, and the category's database fields remain unchanged.		
Conclusion:	The use case concludes once all database validation checks clear, the target row record flag transitions successfully within system storage files, and directory views refresh local state caches.		
Post-condition:	• The target vendor classification row status indicator is permanently written as false (IsActiveStatus = false), removing it from picklist selection filters across systemic vendor creation screens.		
Business rules:	• Deactivation requests targeting any vendor type category are strictly blocked by the system if active downstream vendor database profiles depend on the classification index.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 36: Technical Narrative 6.11 Deactivate Vendor Type

2.4.6 Project Management Subsystem Technical Narratives

2.4.6.1 Use Case: 7.14 Capture Project Feedback

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Alexander Steven</td><td>Date: 15 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>Capture Project Feedback</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>7.14</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">System Admin (Authenticated)</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">Administrator</td></tr>
<tr><td>Description:</td><td colspan="3">A System Admin can attach a project feedback form to a project which will make it so that anyone who has been added to that project will be sent a form 1 week after the project commences asking them for a rating and comments. The system captures the selection for all involved staff. The System Admin navigates to the Feedback screen, sees the list of projects, and toggles the "Enable feedback" checkbox. The Angular frontend instantly sends a PUT request to the .NET 8 API to update the project's feedback configuration in the database.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The System Admin is authenticated and navigating the Feedback screen. The selected project must exist and be active.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The System Admin clicks the "Enable feedback" checkbox next to a project name.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The System Admin clicks the "Feedback" block on the dashboard.</td><td></td><td>Step 2: The system loads the Project Feedback screen, retrieving and displaying the projects and indicating which projects currently have feedback enabled. [ALT]</td></tr>
</table>

	Step 3: The System Admin clicks the "Enable feedback" checkbox on the project they wish to add feedback to.		**Step 4:** The Angular "FeedbackService" captures the toggle event and dispatches a "PUT api/project/{id}/feedback-status" request to the backend with the Boolean payload "isEnabled": true.
			Step 5: The API controller receives the request, validates the System Admin's JWT claims, and invokes Entity Framework Core to locate the target Project entity. [ALT]
			Step 6: The system updates the "FeedbackEnabled" Boolean property on the entity to true and calls "SaveChangesAsync()".
			Step 7: The API returns an HTTP 204 No Content status. The frontend registers the success, leaves the checkbox in the checked state, and displays a success toast notification.
Alternate courses:	[ALT] Step 2: The system fails to retrieve the projects from the database. The system returns an HTTP 500 Internal Server Error and displays an empty list with a red error toast: "Failed to load projects."		
	[ALT] Step 5: The project ID cannot be found in the database. The system returns an HTTP 404 Not Found response. The frontend reverts the checkbox to its previous state and displays a red error toast: "Project not found."		
	[ALT] Step 5: The API returns an HTTP 500 Internal Server Error due to a database connection failure while updating. The frontend reverts the checkbox and displays: "System error, please try again."		
Conclusion:	The project is successfully flagged to collect feedback, ensuring participants receive the form when the project commences.		
Post-condition:	The "FeedbackEnabled" property on the Project entity is set to true. Users in the project receive the review after the project commences		
Business rules:	Only System Admins can enable project feedback or view the results of the feedback.		

	All comments are anonymous.
Implementation constraints and specifications:	• The API endpoint only needs to receive a simple true or false value (a Boolean) to update the status, instead of sending all the project's details over the network. This keeps the system fast and simple.
Assumptions:	• We assume that the project already has staff members assigned to it, and that their email addresses are correct on the system so they can actually receive the feedback form.
Open issues:	• The system does not currently save a record in the "AuditLog" table when a System Admin turns the feedback checkbox on or off.

Table 37: Technical Narrative 7.14 Capture Project Feedback

2.4.6.2 Use Case: 7.15 View Project Feedback

System Name: Integrated NGO Management System			
Author:	Alexander Steven	**Date:** 15 July 2026 **Version:** 1.0	
Use case name:	View Project Feedback	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	7.15		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	System Admin (Authenticated)		
Primary system actor (PSA):	• None		
Other participating actors:	• Project Staff		
Other interested stakeholders:	None		
Description:	If a project has the "FeedbackEnabled" setting turned on, the system automatically sends a feedback survey to all involved users one week after the project starts, or if a System Admin ends the project early. This use case allows a System Admin to view the results of these surveys. The Admin clicks the "Feedback" widget on the home page, which loads a list of projects. When a specific project is clicked, the system retrieves the "ProjectFeedback" records from the database, calculates the average star rating, and displays the rating along with all anonymous comments.		
Pre-condition:	The System Admin is logged in. The project has "FeedbackEnabled" turned on, and users have submitted feedback. The Project has ended, either early by the admin, or by one week and triggered by the time delay built into the system.		
Trigger:	The System Admin clicks the "Feedback" widget on the home dashboard in order to view the results of feedback from projects.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The System Admin clicks the "Feedback" widget on the home page.		**Step 2:** The system loads the Project Feedback screen, showing a list of projects on the left side of the screen.
	Step 3: The System Admin clicks on a specific project from the list.		**Step 4:** The Angular frontend asks the database for the feedback data by sending a GET request to api/feedback/project/{id}. [ALT]

			Step 5: The .NET 8 API checks that the user is a System Admin, and then looks up all the "ProjectFeedback" records for that specific project. [ALT]
			Step 6: The system calculates the average star rating from all the retrieved records and bundles the anonymous comments together to send back.
			Step 7: The API sends back an HTTP 200 OK success message along with the data.
			Step 8: The frontend updates the screen to show a large average star rating (e.g., "4.5 Based on 10 reviews") and a list of the anonymous comments on the right side of the screen.
			Step 12: In the current implementation the "Save
Alternate courses:	[ALT] Step 4: The system has trouble fetching the data, we will just display a "Feedback currently unavailable right now" message with a gif to still look professional and to let the user know they aren't making a mistake.		
	[ALT] Step 5: The database connection fails or the project cannot be found. The API returns an HTTP 500 Internal Server Error or HTTP 404 Not Found. The system leaves the results area empty and shows a red error message: "Failed to load project feedback."		
Conclusion:	The System Admin can clearly read the average rating and anonymous comments to see how the project is performing.		
Post-condition:	The feedback data is displayed on the screen. No information in the database is changed.		
Business rules:	Only System Admins can view the project feedback results.		
Implementation constraints and specifications:	• To protect user privacy, the API must never send user IDs or names back with the feedback comments.		
Assumptions:	• The system correctly counts how many reviews were submitted and calculates the average accurately.		

Open issues:	• If a project has thousands of comments, loading them all at once might slow down the page because there are currently no "next page" buttons (pagination).

Table 38: Technical Narrative 7.15 View Project Feedback

2.4.6.3 Use Case: 7.16 Create Impact Feedback Form

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Alexander Steven</td><td colspan="2">Date: 15 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>Create Impact Feedback Form</td><td colspan="2" rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>7.16</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">System Admin (Authenticated)</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">Community Partners</td></tr>
<tr><td>Description:</td><td colspan="3">A System Admin can create a custom impact form to gather data from community partners. The Admin selects to create a "Questionnaire" (for surveys) or an "Attendance Board" (for tracking daily status). The system provides a form builder where the Admin can continuously add and edit questions (for a questionnaire) or attendees and custom statuses (for an attendance board). The Admin then configures a sending schedule (e.g., "Once" or "Weekdays" with an end date) and saves the form. The system stores the layout as a "FormStructureJson" string in the database and automatically extracts and saves the initial student roster if it is an attendance board.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The System Admin is logged in and navigating the Impact Forms screen.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The System Admin clicks the "Blank form" or a template button on the dashboard in order to create a new impact form.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The System Admin clicks "Blank form" and chooses either "Questionnaire" or "Attendance Board" from the pop-up modal.</td><td></td><td>Step 2: The Angular frontend loads the form builder page matching the chosen type.</td></tr>
</table>

	Step 3: The System Admin enters the form title, description, receiver contact details, and the sending schedule (e.g., Frequency: Weekdays, Send Time: 07:00).		
	Step 4: The System Admin uses the builder to add content. They can loop this step to add multiple questions and options (for Questionnaires) or add multiple attendees and custom statuses (for Attendance Boards).		**Step 5:** The Angular frontend temporarily stores these form blocks and schedule settings in memory.
	Step 6: The System Admin clicks the "Save & Publish" button. [ALT]		**Step 7:** The frontend packages the data into an ImpactFormDto object and sends a POST api/ImpactForms request to the .NET 8 backend.
			Step 8: The API controller receives the request, and saves the ImpactForm record in the database, storing the layout inside the "FormStructureJson" field. [ALT]
			Step 9: The system reads the saved JSON. If the form type is "attendance", the system loops through the JSON to find the initial attendees and saves each one into the FormStudents database table so their attendance can be tracked.
			Step 10: The API returns an HTTP 200 OK response. The frontend takes the user back to the main forms list and displays a success toast message.
Alternate courses:	[ALT] Step 6: The System Admin leaves a required field blank (like the form title). The system highlights the missing field in red and prevents the save.		

	[ALT] Step 8: The database fails to save the record due to a server error. The API returns an HTTP 500 Internal Server Error, and the frontend displays a red error toast: "Failed to save form."
	[ALT] Step 10: We display an error message
Conclusion:	The custom impact form is successfully built, saved, and ready to be filled out by the receiver based on the schedule.
Post-condition:	A new ImpactForm record is saved in the database. If it is an attendance board, FormStudent records are also created.
Business rules:	Only System Admins can create and publish impact forms.
Implementation constraints and specifications:	• The form's visual layout and questions are stored as a single JSON string ("FormStructureJson") to keep the database flexible, rather than creating separate database tables for every question and option.
Assumptions:	• The receiver's email address is valid so they can receive the form link.
Open issues:	• There is currently no way to save a "Draft"; clicking save immediately publishes the form to the schedule. • No way to verify the receiver's email yet

Table 39: Technical Narrative 7.16 Create Impact Feedback Form

2.4.6.4 Use Case: 7.17 View Impact Forms

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Alexander Steven</td><td colspan="2">Date: 15 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>View Impact Forms</td><td colspan="2" rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>7.17</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">System Admin (Authenticated)</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">None</td></tr>
<tr><td>Description:</td><td colspan="3">A System Admin can view all the past projects and see the impact forms questions and answers and see what project they are assigned to. The system retrieves records for impact data and displays it neatly for the admin. The Admin clicks the "Impact Forms" widget on the dashboard, which triggers the Angular frontend to request all available forms from the .NET 8 API. The backend calculates how many responses each form has received. The system then displays the forms as cards, allowing the Admin to click and view the structure, questions, and assigned project details of any specific form.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The System Admin is logged in and is on the main Dashboard. Impact forms must already exist in the database.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The System Admin clicks the "Impact Forms" widget on the home dashboard.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The System Admin clicks the "Impact Forms" widget on the home page.</td><td></td><td>Step 2: The Angular frontend sends a GET request to api/ImpactForms to fetch the forms.</td></tr>
<tr><td></td><td></td><td>Step 3: The API controller receives the request, verifies the Admin's access, and queries the ImpactForms table in the</td></tr>
</table>

			database using Entity Framework Core. [ALT]
			Step 4: For each form, the system checks if it is an attendance board or a questionnaire. It then calculates the "ResponsesCount" by checking either the StudentAttendances or ImpactFormResponses tables.
			Step 5: The API packages this data into an ImpactFormDto list and returns an HTTP 200 OK success response. [ALT]
			Step 6: The Angular frontend receives the data and neatly displays the folders and "Recent forms" as cards. Each card shows the form's name, the number of responses, and the project it is linked to.
	Step 7: The System Admin clicks on a specific form card to view its internal questions and structure.		**Step 8:** The frontend parses the "FormStructureJson" string for that form and visually renders the questions and available options for the Admin to read.
Alternate courses:	[ALT] Step 3: The database is down or an unexpected server error occurs. The API returns an HTTP 500 Internal Server Error. The Angular frontend catches the error, displays an empty dashboard, and shows a red error toast message to the Admin: "Failed to load impact forms."		
	[ALT] Step 5: The query succeeds but no forms exist yet. The API returns an empty list [] with an HTTP 200 OK. The frontend displays a friendly message to the Admin: "No forms available. Create a new one to get started."		
Conclusion:	The System Admin can successfully view all impact forms, see how many responses they have, and inspect the specific questions asked on each form.		

Post-condition:	The impact forms are displayed on the screen. No database records are modified.
Business rules:	Only System Admins can view the impact forms dashboard and structure.
Implementation constraints and specifications:	• The system calculates the "ResponsesCount" directly in the database query to ensure the page loads quickly without pulling down unnecessary response data.
Assumptions:	• The frontend correctly interprets the "FormStructureJson" to display the questions exactly as the community partner will see them.
Open issues:	• If there are hundreds of forms, the system currently fetches all of them at once, which may slow down the initial page load.

Table 40: Technical Narrative 7.17 View Impact Forms

2.4.6.5 Use Case: 7.18 Update Impact Feedback Form

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Alexander Steven</td><td colspan="2">Date: 15 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>Update Impact Feedback Form</td><td colspan="2" rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>7.18</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">System Admin (Authenticated)</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">Community Partners</td></tr>
<tr><td>Description:</td><td colspan="3">The System Admin can update an existing impact form that can be added to projects to make form completions and capturing easier. The system shows fields for the form name and the questions currently on the form as editable content. The Admin opens the form builder, which is pre-filled with the current form details. The Admin can loop through and edit, add, or remove questions, or change the sending schedule. Once saved, the Angular frontend sends a PUT request to the .NET 8 API to overwrite the old form details and "FormStructureJson" in the database.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The System Admin is logged in and viewing the impact forms list. The form being updated must already exist in the database.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The System Admin clicks the "Edit" button on a specific impact form card.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The System Admin clicks "Edit" on a specific impact form.</td><td></td><td>Step 2: The Angular frontend opens the form builder and pre-fills it with the existing form's data (Title, Schedule, and Questions).</td></tr>
<tr><td>Step 3: The System Admin edits the form. They can loop this step to add new questions,</td><td></td><td>Step 4: The Angular frontend temporarily holds these changes in memory, updating the</td></tr>
</table>

	delete old ones, or change the schedule.		local "FormStructureJson".
	Step 5: The System Admin clicks the "Save & Publish" button. [ALT]		**Step 6:** The frontend checks that no required fields are blank, packages the updated data into an ImpactFormDto, and sends a PUT api/ImpactForms/{id} request to the backend.
			Step 7: The .NET API controller receives the request and asks Entity Framework Core to find the existing form in the database. [ALT]
			Step 8: The system overwrites the old details with the new data and calls SaveChangesAsync(). [ALT]
			Step 9: The API returns an HTTP 204 No Content success response.
			Step 10: The frontend takes the user back to the dashboard and displays a green success toast message: "Form updated successfully."
Alternate courses:	[ALT] Step 5: The System Admin clears out a required field (like the form title) and clicks save. The frontend stops the save process and highlights the missing field in red.		
	[ALT] Step 7: The form ID cannot be found in the database (e.g., someone else deleted it). The API returns an HTTP 404 Not Found response. The frontend shows a red error toast: "Form not found."		
	[ALT] Step 8: The database fails to save the updates due to a connection issue. The API returns an HTTP 500 Internal Server Error, and the frontend displays a red error toast: "Failed to update form."		
Conclusion:	The custom impact form is successfully updated with the new questions and scheduling rules.		

Post-condition:	The existing ImpactForm record in the database is modified.
Business rules:	Only System Admins can edit existing impact forms.
Implementation constraints and specifications:	• Modifying the "FormStructureJson" is allowed, but the system must be careful not to break how old responses are displayed if a question is completely deleted.
Assumptions:	• Changing the schedule rules on an existing form will only apply to future send dates, not past ones.
Open issues:	• If a question is deleted, it is unclear if the old answers to that question will still be visible in the results dashboard.

Table 41: Technical Narrative 7.18 Update Impact Feedback Form

2.4.6.6 Use Case: 7.19 Deactivate Impact Feedback Form

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Alexander Steven</td><td colspan="2">Date: 15 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>Deactivate Impact Feedback Form</td><td colspan="2" rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>7.19</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">System Admin (Authenticated)</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• System</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">None</td></tr>
<tr><td>Description:</td><td colspan="3">The System Admin can deactivate an impact form from the list of all the forms being displayed; this can remove unused forms and clear clutter from the system. The system loads the form name for deactivation confirmation before removing it from view. When confirmed, the Angular frontend sends an HTTP DELETE request to the .NET 8 API. The backend uses Entity Framework Core to locate the record in the "ImpactForms" table, permanently removes it, and saves the changes. The system then alerts the Admin that the form was successfully removed.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The System Admin is logged in and viewing the list of recent impact forms. The form must currently exist in the database.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The System Admin clicks the "Deactivate" (or delete) button on a specific impact form card.</td></tr>
<tr><td rowspan="2">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The System Admin clicks the "Deactivate" button on a specific impact form.</td><td></td><td>Step 2: The Angular frontend pauses the action and displays a confirmation modal asking: "Are you sure you want to deactivate this form?"</td></tr>
</table>

	Step 3: The System Admin clicks the "Confirm" button. [ALT]		**Step 4:** The Angular frontend sends a DELETE api/ImpactForms/{id} request to the backend.
			Step 5: The API controller receives the request, verifies the Admin's permissions, and uses Entity Framework Core's FindAsync to locate the target record in the "ImpactForms" table. [ALT]
			Step 6: The system executes the Remove() command on the record and calls SaveChangesAsync() to commit the deletion to the database. [ALT]
			Step 7: The API returns an HTTP 204 No Content success response.
			Step 8: The Angular frontend receives the success response, immediately removes the form's card from the screen, and displays a green success toast: "Form successfully deactivated."
Alternate courses:	[ALT] Step 3: The System Admin changes their mind and clicks "Cancel". The confirmation modal closes, no request is sent, and the form remains on the screen.		
	[ALT] Step 5: The form ID cannot be found in the database. The API returns an HTTP 404 Not Found response. The frontend closes the modal and displays a red error toast: "Form not found. It may have already been removed."		
	[ALT] Step 6: The database fails to delete the form (for example, if there are locked database constraints). The API returns an HTTP 500 Internal Server Error. The frontend leaves the form on the screen and displays a red error toast: "Failed to deactivate form. Please try again."		
Conclusion:	The selected impact form is completely removed from the system and vanishes from the Admin's view.		

Post-condition:	The specific ImpactForm record is deleted from the database.
Business rules:	Only System Admins have the permission to deactivate/delete impact forms.
Implementation constraints and specifications:	• The backend strictly uses the DELETE HTTP method to perform the removal.
Assumptions:	• The frontend automatically updates the list without needing to refresh the entire webpage.
Open issues:	• If a form has already collected responses, deleting it might cause data issues if the database doesn't have "Cascade Delete" turned on for the response tables.

Table 42: Technical Narrative 7.19 Deactivate Impact Feedback Form

2.4.6.7 Use Case: 7.20 Capture Impact Feedback

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Alexander Steven</td><td colspan="2">Date: 15 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>Capture Impact Feedback</td><td colspan="2" rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>7.20</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Community Partner</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">System Administrator</td></tr>
<tr><td>Description:</td><td colspan="3">The system will capture an email address or phone number to send the form to the community partner. When the scheduled time arrives, the community partner receives a unique public link to the form. Clicking the link opens the Angular frontend, which asks the .NET 8 API to retrieve the specific form using its "PublicLinkGuid". The system displays either a Questionnaire or an Attendance Board. The partner fills in the answers or marks attendance statuses (like Bike, Walk, Absent) and clicks submit. The system sends the data back to the API to be saved in the database.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The impact form has been created, published, and the community partner has received their unique public link.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The Community Partner clicks the public link in their email and submits the form.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The Community Partner clicks the public link provided in their email.</td><td></td><td>Step 2: The Angular frontend opens and sends a GET api/ImpactForms/public/{guid} request to the backend.</td></tr>
<tr><td></td><td></td><td>Step 3: The API controller looks up the "ImpactForm" in the database using the GUID, retrieves the</td></tr>
</table>

			"FormStructureJson", and returns it to the frontend. [ALT]
			Step 4: The system reads the JSON and displays the correct layout on the screen (either survey questions or an attendance roster).
	Step 5: The Community Partner answers the questions or marks the daily attendance statuses, then clicks "Submit".		**Step 6:** The Angular frontend packages the answers. If it is a questionnaire, it creates an "ImpactFormResponseDto". If it is an attendance board, it creates a list of "SyncAttendanceDto" records.
			Step 7: The frontend sends a POST request to either the submit endpoint or the attendance/sync endpoint on the .NET API.
			Step 8: The API receives the data and saves the new records into either the "ImpactFormResponses" or "StudentAttendances" database tables using Entity Framework Core. [ALT]
			Step 9: In The API returns an HTTP 200 OK success message and The frontend hides the form and displays a green success message: "Response submitted successfully. Thank you!"
Alternate courses:	[ALT] Step 3: The public link is invalid or the form was deactivated. The API returns an HTTP 404 Not Found error. The frontend displays a red error message: "This form is no longer available or the link is incorrect."		
	[ALT] Step 8: The database fails to save the answers due to a connection error. The API returns an HTTP 500 Internal Server Error, and the frontend displays a red error toast: "Failed to submit response. Please try again."		
Conclusion:	The community partner successfully captures and submits their impact data to the NGO's database.		
Post-condition:	New data is stored in the "ImpactFormResponses" or "StudentAttendances" tables.		

Business rules:	Anyone with the exact public link can submit the form without needing to log in (no authentication required).
Implementation constraints and specifications:	• Questionnaires are saved as raw JSON strings in the responses table, while Attendance records are saved relationally with specific dates and statuses.
Assumptions:	• The community partner has a stable internet connection to load and submit the form.
Open issues:	• Because the public link does not require a login, there is a risk of the link being shared and filled out by unauthorized people

Table 43: Technical Narrative 7.20 Capture Impact Feedback

2.4.6.8 Use Case: 7.21 View Impact Feedback

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Alexander Steven</td><td colspan="2">Date: 15 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>View Impact Feedback</td><td colspan="2" rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>7.21</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">System Admin (Authenticated)</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">None</td></tr>
<tr><td>Description:</td><td colspan="3">A System Admin can view all of the project's impact results and see the impact form questions and answers, the assigned project, and which community partner answered. The system retrieves and displays these records neatly. The Admin clicks the "Results" button on a specific form card on the dashboard. Depending on whether the form is a questionnaire or an attendance board, the Angular frontend requests the appropriate data from the .NET 8 API. The system fetches the data and dynamically renders visual charts, graphs, and breakdowns on the Results Dashboard.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The System Admin is logged in and viewing the impact forms dashboard. The form must have collected responses or attendance data.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The System Admin clicks the "Results" button on a specific impact form card.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The System Admin clicks the "Results" button on an impact form card.</td><td></td><td>Step 2: The Angular frontend opens the Results Dashboard and sends a GET request to the backend. It calls api/ImpactForms/{id}/responses for surveys, or api/ImpactForms/{id}/attendance-data for attendance boards.</td></tr>
<tr><td></td><td></td><td>Step 3: The API controller receives the request, verifies the Admin's JWT token, and queries either the</td></tr>
</table>

			"ImpactFormResponses" or "StudentAttendances" tables in the database using Entity Framework Core. [ALT]
			Step 4: The system gathers the records and returns them in an HTTP 200 OK success response.
			Step 5: The Angular frontend receives the data and uses it to draw the charts. Questionnaires show option breakdowns (like pie charts), while Attendance Boards show a Daily Register graph and Individual Attendee horizontal bar charts. [ALT]
Alternate courses:	[ALT] Step 3: The database query fails due to a server connection issue. The API returns an HTTP 500 Internal Server Error. The frontend displays an empty results pane and shows a red error toast: "Failed to load feedback results."		
	[ALT] Step 5: The query succeeds but no responses have been submitted yet. The API returns an empty list []. The frontend displays an empty state message: "No data available yet. Waiting for responses."		
Conclusion:	The System Admin successfully views the visual charts and breakdowns of the feedback data.		
Post-condition:	The feedback data is displayed on the UI. No records are changed in the database.		
Business rules:	Only System Admins can view the impact feedback results and charts.		
Implementation constraints and specifications:	• The backend uses two different endpoints to keep the data structured properly for the different types of visual charts required.		
Assumptions:	• The frontend charting library can handle large amounts of attendance records to draw the dynamic trend lines correctly.		
Open issues:	• If a form has been running for multiple years, loading all the attendance data at once might slow down the rendering of the line graphs.		

Table 44: Technical Narrative 7.21 View Impact Feedback

2.4.6.9 Use Case: 7.22 Add Photo

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Alexander Steven</td><td>Date: 15 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td colspan="2">Add Photo</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">7.22</td></tr>
<tr><td>Priority:</td><td colspan="2">High</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">System Admin (Authenticated)</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">None</td></tr>
<tr><td>Description:</td><td colspan="3">A user can upload multiple photos to a project's album. The system loads a page where images can be dragged, dropped, or uploaded from the device. The user clicks the "+ Upload Photos" button while inside a specific album and selects multiple image files from their computer. The Angular frontend packages these files and sends them to the .NET 8 API. The backend loops through the files, assigns a unique GUID to each filename to prevent overwriting, physically saves the images to the server's "wwwroot/uploads/photos" folder, and records the file paths in the "Photos" database table.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The user is logged in, viewing a specific photo album, and has image files ready to upload.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The user clicks the "+ Upload Photos" button inside a photo album.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The user clicks the "+ Upload Photos" button.</td><td></td><td>Step 2: The system opens the device's native file explorer, allowing the user to select one or multiple image files.</td></tr>
<tr><td>Step 3: The user selects the files and confirms the upload.</td><td></td><td>Step 4: The Angular frontend appends the files and the user's ID to a FormData object and sends a POST api/PhotoAlbums/{id}/upload request to the backend.</td></tr>
</table>

			Step 5: The API controller receives the request and checks if the album exists in the database. [ALT]
			Step 6: The system loops through the uploaded files. For each file, it generates a unique name using a GUID, creates a file stream, and copies the physical file into the server's "wwwroot/uploads/photos" directory.
			Step 7: For each saved file, the system creates a new "Photo" database record containing the album ID, the new "FilePath", and the ID of the user who uploaded it.
			Step 8: The system calls SaveChangesAsync() to commit the new records to the database and the API returns an HTTP 200 OK success message. [ALT]
			Step 9: The Angular frontend refreshes the album view to display the newly uploaded photos and shows a green success toast: "Photos uploaded successfully."
Alternate courses:	[ALT] Step 5: The album ID cannot be found (e.g., it was deleted by another user). The API returns an HTTP 404 Not Found response with the message "Album not found." The frontend displays a red error toast.		
	[ALT] Step 8: The server runs out of disk space or the database connection fails during saving. The API returns an HTTP 500 Internal Server Error, and the frontend displays a red error toast: "Failed to upload photos."		
Conclusion:	The selected photos are successfully uploaded, safely stored on the server, and displayed in the project's album.		
Post-condition:	Physical image files are saved in the "wwwroot/uploads/photos" folder, and corresponding "Photo" records are added to the database.		
Business rules:	Users can only upload photos to albums they have permission to access.		

Implementation constraints and specifications:	• Files must be appended to a multipart/form-data request so the .NET API can bind them to the List<IFormFile> parameter correctly.
Assumptions:	• The server has write permissions configured for the "wwwroot/uploads/photos" directory.
Open issues:	• There is currently no restriction on file size or file type (e.g., preventing malicious .exe uploads) explicitly shown in the controller code.

Table 45: Technical Narrative 7.22 Add Photo

2.4.6.10 Use Case: 7.23 View Photo

System Name: Integrated NGO Management System			
Author:	Alexander Steven	**Date:** 15 July 2026 **Version:** 1.0	
Use case name:	View Photo	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	7.23		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	User (Authenticated)		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	None		
Description:	A user can see all photos from a specific project. A user can click on a project to see all photos from that project; they can then click the project to see all photos in that album. All projects are displayed in a table with their names and a representative project photo. The Angular frontend first requests all "PhotoAlbum" records from the .NET 8 API and displays them as cards with their "CoverPhotoPath". When the user clicks on an album, the system makes a second request to fetch all the individual "Photo" records linked to that album ID, displaying the physical images neatly on the screen.		
Pre-condition:	The user is logged in. Photo albums and photos must exist in the database and file system.		
Trigger:	The user clicks the "Photos" tab on the main navigation menu.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user clicks the "Photos" tab to open the albums page.		**Step 2:** The Angular frontend sends a GET api/PhotoAlbums request to the backend.
			Step 3: The API controller asks Entity Framework Core to fetch all "PhotoAlbum" records from the database. [ALT]
			Step 4: The system returns an HTTP 200 OK response with the album details, and the frontend

			displays the albums as interactive cards.
	Step 5: The user clicks on a specific album card (e.g., "Bike drop 2026").		**Step 6:** The frontend sends a second request: GET api/PhotoAlbums/{id}/photos using the selected album's ID.
			Step 7: The API queries the "Photos" database table for all images linked to that specific "PhotoAlbumId". [ALT]
			Step 8: The API returns an HTTP 200 OK response containing a list of the photo file paths.
			Step 9: The Angular frontend renders a neat grid layout displaying all the retrieved images to the user.
Alternate courses:	[ALT] Step 3: The database connection fails while loading albums. The API returns an HTTP 500 Internal Server Error. The frontend displays an empty page and a red error toast: "Failed to load albums."		
	[ALT] Step 7: The database connection fails while loading the specific photos inside an album. The API returns an HTTP 500 Internal Server Error. The frontend displays a red error toast: "Failed to load photos for this album."		
Conclusion:	The user successfully navigates the gallery and views the stored photos.		
Post-condition:	The images are displayed on the UI. No records in the database are modified.		
Business rules:	Users can only view photos from albums they have permission to access.		
Implementation constraints and specifications:	• The "FilePath" stored in the database is used by the frontend to fetch the physical images hosted in the "wwwroot/uploads/photos" folder.		
Assumptions:	• The physical image files on the server match the "FilePath" records in the database perfectly (no files were manually deleted from the server folder outside of the system).		
Open issues:	• If an album contains hundreds of high-resolution photos, loading them all at once on a single page might be slow for users with weak internet connections.		

Table 46: Technical Narrative 7.23 View Photo

2.4.6.11 Use Case: 7.24 Delete Photo

System Name: Integrated NGO Management System			
Author:	Alexander Steven	**Date:** 15 July 2026 **Version:** 1.0	
Use case name:	Delete Photo	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	7.24		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	User (Authenticated)		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	None		
Description:	A user can remove photos from a project's image folder. A user can click on a project to see all photos and then click the delete photos button; the system then displays an "X" button on photos they uploaded to remove them. When the user confirms the deletion, the Angular frontend sends a DELETE request to the .NET 8 API. The backend retrieves the "Photo" record, permanently deletes the physical image file from the server's "wwwroot/uploads/photos" directory, removes the record from the database, and saves the changes. The system captures the selection and updates the album.		
Pre-condition:	The user is logged in, has navigated to a project's photo album, and the photo exists on the server.		
Trigger:	The user clicks the "X" (Delete) button on a specific photo.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user clicks the "X" button on a specific photo they wish to remove.		**Step 2:** The Angular frontend pauses the action and displays a confirmation modal asking if they are sure they want to delete the photo.
	Step 3: The user clicks the "Confirm" button. [ALT]		**Step 4:** The Angular frontend sends a DELETE api/PhotoAlbums/photos/{photoId} request to the backend.

			Step 5: The API controller receives the request and uses Entity Framework Core to find the "Photo" record in the database using the provided ID. [ALT]
			Step 6: The system checks the "FilePath" stored in the database, locates the physical file in the "wwwroot/uploads/photos" folder, and executes System.IO.File.Delete() to permanently remove it from the server.
			Step 7: The system removes the "Photo" record from the database and calls SaveChangesAsync().[ALT]
			Step 8: The API returns an HTTP 204 No Content success response.
			Step 9: The Angular frontend removes the deleted photo from the screen's grid and displays a green success toast: "Photo successfully deleted."
Alternate courses:	[ALT] Step 3: The user clicks "Cancel" on the modal. The frontend closes the prompt and stops the deletion process.		
	[ALT] Step 5: The photo ID cannot be found in the database. The API returns an HTTP 404 Not Found response. The frontend closes the modal and displays a red error toast: "Photo not found."		
	[ALT] Step 7: The database connection drops while saving the changes. The API returns an HTTP 500 Internal Server Error. The frontend leaves the photo on the screen and displays a red error toast: "Failed to delete photo."		
Conclusion:	The selected photo is permanently deleted from both the server's physical file system and the SQL database.		
Post-condition:	The "Photo" record is destroyed, and the physical file is permanently erased.		
Business rules:	The user is deleting photos they uploaded or are a system admin.		
Implementation constraints and specifications:	• The physical file must be deleted *before* the database record is removed to ensure no orphaned files are left taking up hard drive space on the server.		
Assumptions:	• The backend server has the correct write/delete permissions configured for the "wwwroot/uploads/photos" folder.		
Open issues:	• There is currently no "Recycle Bin" or soft-delete for photos. • Once the physical file is deleted, it cannot be easily recovered.		

Table 47: Technical Narrative 7.24 Delete Photo

2.4.7 K9 Subsystem Technical Narratives

2.4.7.1 Use Case 8.1: Add Dog

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Nicholas Bailey</td><td>Date: 13/07/2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">Add Dog</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">8.1</td></tr>
<tr><td>Priority:</td><td colspan="2">High</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Super Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">• None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes how a super admin creates a new K9 within the system. The system allows the user to enter the K9's details, upload an optional image, assign K9 roles, and optionally select the K9's parents. The system then validates the entered information, confirms the creation, and saves the K9 to the database. The K9 is stored along with its associated roles and image, the K9 list is refreshed, and a confirmation message is displayed. The transaction is also recorded in the audit log.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The super admin must be logged into the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The super admin wants to add a new dog to the system.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The super admin wants to add a new K9 to the system.</td><td></td><td>Step 2: The system invokes UC 8.3 (View Dogs) to display the K9 page showing all the registered K9s. This is done using the</td></tr>
</table>

			Angular frontend and the K9 Service, which routes the request via the Angular HTTP Client to the corresponding HTTP GET API endpoint in Visual Studio (K9Controller – GetAllK9s).
	Step 3: The super admin selects the "+ Add K9" button.		**Step 4:** The system loads and displays the Create K9 modal with the following components: • Create K9 modal title. • Name section: ○ Name label. ○ Name input field. ○ Placeholder text: "Dog Name". • Breed section: ○ Breed label. ○ Breed input field. ○ Placeholder text: "Breed". • Gender section: ○ Gender label. ○ Dropdown list. ▪ Default option: "Select gender". ▪ Male. ▪ Female. • K9 Roles section: ○ K9 Roles label. ○ Multi-select dropdown displaying all available K9 Roles. ○ Selected role counter. • Date of Birth section:

			○ Date of Birth label. ○ Date selector. • Father section: ○ Father label. ○ Dropdown list. ▪ Default option: "No Father Assigned". ▪ Displays all male K9s retrieved from the system. • Mother section: ○ Mother label. ○ Dropdown list. ▪ Default option: "No Mother Assigned". ▪ Displays all female K9s retrieved from the system. • Image section: ○ Image label. ○ Choose Image button. ○ Displays the selected image file name. • Action buttons: ○ Create button. ○ Cancel button. The Father and Mother dropdown lists are populated using the K9 data that was already retrieved when UC 8.3 (View Dogs) was invoked.

			The K9 Roles dropdown is populated using data retrieved from the K9 Role Entity via the K9 Role Service, which calls the backend API. **K9 Role Entity** • K9RoleId (PK) • RoleName
	Step 5: The super admin enters the K9's details, optionally uploads an image, selects the required K9 roles, and selects the "Create" button. [ALT]		**Step 6:** The system validates the entered details using Angular frontend validation before allowing the K9 to be created. The system validates the following: • The K9 Name field is not null or empty. • The K9 Name contains alphabetic characters only. • The Breed field is not null or empty. • The Breed contains alphabetic characters only. • The Date of Birth is not in the future. • If a father is selected, the father must be at least two years older than the K9. • If a mother is selected, the mother must be at least two years older than the K9. • If an image is selected, only JPG, JPEG, PNG or WEBP image formats are accepted. If any validation fails, an error message is displayed within the Create K9 modal. [ALT]
			Step 7: The system displays a custom Angular confirmation modal after successful validation.

			The confirmation modal contains: • Confirmation modal title ("Confirm Create K9"). • Confirmation message label: ○ Displays the message: "Are you sure you want to create this K9?" • Action buttons: ○ Confirm button. ○ Cancel button.
	Step 8: The super admin selects the "Confirm" button. [ALT]		**Step 9:** The system captures the new K9's details and constructs a FormData object containing the K9 information, optional image, selected parent K9s and selected K9 Roles. The Angular frontend uses the K9 Service and Angular HTTP Client to route the request to the corresponding HTTP POST API endpoint in Visual Studio (K9Controller – CreateK9). The API endpoint passes the request to the K9 Repository where additional business validation is performed before the data is saved using Entity Framework Core in Microsoft SQL Server. The backend validates that: • A K9 cannot be its own parent. • The selected father and mother cannot be the same K9. • The selected father exists and is male. • The selected mother exists and is female. • The selected parents are each at least two years older than the K9. If an image is supplied, the system stores the uploaded image in the application's

			uploads directory and generates a unique file name. If no image is supplied, a default K9 image is assigned. The system creates the new K9 record and stores the selected K9 Roles by creating records in the K9K9Role junction table. The following entities are involved in the process: **K9 Entity** • K9Id (PK) • K9ImageUrl • K9Name • K9Breed • Gender • K9DateOfBirth • ActiveStatus • K9DadId (FK) • K9MomId (FK) **K9K9Role Entity (Junction Table)** • K9Id (PK, FK) • K9RoleId (PK, FK) **K9 Role Entity** • K9RoleId (PK) • RoleName [ALT]
			Step 10: The system will then record the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful creation of the new K9.

			The system will store information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that a new K9 has been created. **AuditLog Entity:** • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 11: The system displays a toaster notification confirming that the K9 has been successfully created. The confirmation message displays the text: "K9 created successfully."
Alternate courses:	ALT-Step 5: The super admin changes their mind and closes the pop-up by selecting the “Cancel” button. Terminate use case.		
	ALT-Step 6: The system finds an error after validating and an error message is displayed within the Create K9 modal. Go back to step 4.		
	ALT-Step 8: The super admin does not confirm and selects the “Cancel” button. Go back to step 4.		
	ALT-Step 9: The system fails to save the new K9 and displays a toaster message: “Failed to create K9”. Terminate use case.		
Conclusion:	The new dog’s details have been saved onto the system, and a confirmation message is shown.		
Post-condition:	The system saves the new K9 in the K9 table, stores any associated K9 role assignments, uploads and stores the selected image (if provided), and makes the newly created K9 available for viewing and management within the system.		
Business rules:	• Only super admin can add a new dog.		

Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 48: Technical Narrative 8.1 Add Dog

2.4.7.2 Use Case 8.2: View Dog

System Name: Integrated NGO Management System			
Author:	Nicholas Bailey		**Date: 13/07/2026** **Version:** 1
Use case name:	View Dog		**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓
Use case id:	8.2		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	Handler		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	• None		
Description:	This use case describes how a handler views the details of a specific K9 within the system. The system allows the handler to search for or filter the available K9s, select a K9 from the displayed list, and view its details. The system retrieves the selected K9's information from the database and displays it in a View K9 Details modal.		
Pre-condition:	The handler must be logged into the system.		
Trigger:	The handler wants to view the specific dog's details on the system.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The handler wants to view a specific K9's details on the system.		**Step 2:** The system invokes UC 8.3 (View Dogs) to display all the current K9s. This is done using the Angular frontend and the K9 Service, which routes the request via the Angular HTTP Client to the corresponding HTTP GET API endpoint

			in Visual Studio (K9Controller – GetAllK9s). The retrieved K9s are then displayed as K9 cards on the K9 page.
	Step 3: The handler searches for a K9 by entering its name or breed in the search field or filters the displayed K9s by gender or status. [ALT]		**Step 4:** The system filters and displays the K9s matching the entered search criteria and selected filters. The filtering is performed within the Angular frontend using the retrieved K9 data. [ALT]
	Step 5: The handler selects a K9 card.		**Step 6:** The system retrieves the selected K9's details through the K9 Service in the Angular frontend. The service sends a request via the Angular HTTP Client to the HTTP GET endpoint in Visual Studio (K9Controller – GetK9ById), which retrieves the K9 based on the selected K9Id. The API then retrieves data from the following entity: **K9 Entity:** • K9Id (PK) • K9ImageUrl • K9Name • K9Breed • Gender • K9DateOfBirth • ActiveStatus • K9DadId (FK) • K9MomId (FK) The API also retrieves the K9's assigned roles from the K9K9Role junction table and the related K9Role entity. **K9K9Role Entity (Junction Table):** • K9Id (PK, FK)

			• K9RoleId (PK, FK) **K9Role Entity:** • K9RoleId (PK) • RoleName The Angular frontend also invokes the K9 Service, which routes a request via the Angular HTTP Client to the corresponding HTTP GET API endpoint in Visual Studio (K9Controller – GetAssignmentSummary). The API retrieves the current handler assignment and determines whether a previous handler exists. This information is used to display the assigned handler and determine which handler action buttons (Assign, Update, Deactivate, or Reactivate) are available. **K9User Entity:** • AssignmentId (PK) • K9Id (FK) • UserId (FK) • StartDate • EndDate **User Entity:** • UserId (PK) • UserName • UserSurname • ActiveStatus
			Step 7: The system displays a View K9 Details modal on the Angular frontend containing the selected K9's details. The modal is triggered when a K9 card is selected.

			The View K9 Details modal contains the following components: • Modal heading: ○ K9 name • K9 image • K9 details section: ○ Breed ○ Gender ○ Age ○ Date of Birth ○ Father ○ Mother ○ Status ○ Assigned K9 Roles ○ Assigned Handler (or "No Handler Assigned" if none exists) • Handler Actions: ○ Assign or Update Handler button (depending on whether a current handler exists) ○ Deactivate Handler button (displayed when a current handler exists) ○ Reactivate Handler button (displayed when no current handler exists but a previous handler exists) • Action buttons section: ○ View Ancestry button

			○ Edit button ○ Deactivate button ○ Close button
	Step 8: The handler can view the selected K9's details.		
Alternate courses:	ALT-Step 3a: The handler does not search or filter for a dog and just selects the dog directly from the ones displayed. Go to Step 6.		
	ALT-Step 4: The system does not find any dogs matching the specified criteria. Go back to step 2.		
Conclusion:	The system displays a modal showing the dog's details.		
Post-condition:	The dog's details are displayed.		
Business rules:	• Handlers can only view the details of dogs. • Admins and super admins can also view the details dogs on the system.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 49: Technical Narrative 8.2 View Dog

2.4.7.3 Use Case 8.3: View Dogs

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Nicholas Bailey</td><td>Date: 13/07/2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">View Dogs</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">8.3</td></tr>
<tr><td>Priority:</td><td colspan="2">High</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Handler</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">• None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes how a handler views all K9s registered in the system. The system retrieves the K9 records from the database and displays them as K9 cards on the K9 page. The handler can then search for K9s by name or breed, filter the displayed K9s by gender or status, and select a K9 to view its detailed information.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The handler must be logged into the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The handler wants to view the specific dog’s details on the system.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The handler wants to view all K9s on the system.</td><td></td><td></td></tr>
<tr><td>Step 2: The handler selects the “Animals”</td><td></td><td>Step 3: The system displays the Animals page containing:</td></tr>
</table>

	tab to navigate to the animals' page.		• A secondary navigation bar with the following animal category tabs: ○ Rhino ○ Elephant ○ K9 • A content area below the navigation bar used to display the selected animal records. • If no animal category has been selected, the content area displays the following message: ○ "Select an animal category" ○ "Choose Rhino, Elephant, or K9 to view records."
	Step 4: The handler then selects "K9" on the secondary navbar to navigate to the K9s page.		**Step 5:** The system retrieves all K9 records using the Angular frontend and the K9 Service, which routes the request via the Angular HTTP Client to the corresponding HTTP GET API endpoint in Visual Studio (K9Controller – GetAllK9s). The API retrieves data from the following entity: **K9 Entity:** • K9Id (PK) • K9ImageUrl • K9Name • K9Breed • Gender • K9DateOfBirth • ActiveStatus

			• K9DadId (FK) • K9MomId (FK)
			Step 6: The system displays the retrieved K9s as K9 cards on the Angular frontend. Each K9 card displays the following information: • Image • Name • Age • Gender • Breed The page also provides the following filtering options: • Search by K9 name or breed. • Filter by gender. • Filter by status (Active, Inactive, or All). • Results counter
	Step 7: The Super Admin can view the K9s.		
Alternate courses:	None		
Conclusion:	The system displays all the dogs on the system.		
Post-condition:	All the dogs on the system are displayed.		
Business rules:	• Handlers can view all dogs. • Admins and super admins can also view all the dogs on the system.		

Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 50: Technical Narrative 8.3 View Dogs

2.4.7.4 Use Case 8.4: Update Dog

System Name: Integrated NGO Management System			
Author:	Nicholas Bailey		**Date: 13/07/2026** **Version:** 1
Use case name:	Update Dog		**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓
Use case id:	8.4		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	• None		
Description:	This use case describes how a super admin updates the details of an existing K9 within the system. The system allows the super admin to modify the K9's information, update its assigned roles, change its parent information, and optionally replace the K9's image. The system validates the updated information, requests confirmation before saving the changes, updates the K9 record and its associated role assignments in the database, records the transaction in the audit log, and displays a confirmation message.		
Pre-condition:	The super admin must be logged into the system.		
Trigger:	The super admin wants to update a dog's details on the system.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The super admin wants to update a K9's details on the system.		**Step 2:** The system invokes UC 8.2 (View Dog) to display the View K9 Details modal containing the selected K9's information. This is done using the Angular frontend and the K9 Service, which routes the

			request via the Angular HTTP Client to the corresponding HTTP GET API endpoint in Visual Studio (K9Controller – GetK9ById).
	Step 3: The super admin selects the "Edit" button. [ALT]		**Step 4:** The system loads and displays the Edit K9 modal with the selected K9's current information. The Edit K9 modal contains the following components: • Edit K9 modal title. • Name section: ○ Name label. ○ Name input field. • Breed section: ○ Breed label. ○ Breed input field. • Gender section: ○ Gender label. ○ Dropdown list. ▪ Male. ▪ Female. • K9 Roles section: ○ K9 Roles label. ○ Multi-select dropdown displaying all available K9 Roles. ○ Selected role counter. • Date of Birth section: ○ Date of Birth label. ○ Date selector. • Father section: ○ Father label.

			○ Dropdown list. ▪ Default option: "No Father Assigned". ▪ Displays all available male K9s except the K9 currently being edited. • Mother section: ○ Mother label. ○ Dropdown list. ▪ Default option: "No Mother Assigned". ▪ Displays all available female K9s except the K9 currently being edited. • Current Image section: ○ Displays the K9's current image. • Replace Image section: ○ Choose Image button. ○ Displays the selected replacement image file name. • Action buttons: ○ Save Changes button. ○ Cancel button. The Father and Mother dropdown lists are populated using the K9 data that was previously retrieved when UC 8.2 (View K9 Details) was invoked.

			The K9 Roles dropdown is populated using data retrieved from the K9 Role Entity. **K9 Role Entity** • K9RoleId (PK) • RoleName
	Step 5: The super admin updates the required K9 details and selects the "Save Changes" button. [ALT]		**Step 6:** The system validates the updated information using Angular frontend validation before allowing the K9 to be updated. The system validates the following: • The K9 Name field is not null or empty. • The K9 Name contains alphabetic characters only. • The Breed field is not null or empty. • The Breed contains alphabetic characters only. • The Date of Birth is not in the future. • If a father is selected, the father must be at least two years older than the K9. • If a mother is selected, the mother must be at least two years older than the K9. • If a replacement image is selected, only JPG, JPEG, PNG or WEBP image formats are accepted. [ALT]
			Step 7: The system displays a custom Angular confirmation modal. The confirmation modal contains:

			• Confirmation modal title ("Confirm Update"). • Confirmation message label: ○ Displays the message: "Are you sure you want to update this K9?" • Action buttons: ○ Confirm button. ○ Cancel button.
	Step 8: The super admin selects the "Confirm" button. [ALT]		**Step 9:** The system captures the updated K9 information and constructs a FormData object containing the modified K9 details, selected K9 Roles, optional replacement image, and parent information. The Angular frontend uses the K9 Service and Angular HTTP Client to route the request to the corresponding HTTP PUT API endpoint in Visual Studio (K9Controller – UpdateK9). The API passes the request to the K9 Repository, where additional business validation is performed before the updated information is saved using Entity Framework Core in Microsoft SQL Server. The backend validates that: • A K9 cannot be assigned as its own parent. • The selected K9 is active and may be modified. • The selected father and mother cannot be the same K9. • The selected father exists and is male. • The selected mother exists and is female.

			• The selected parents are each at least two years older than the K9. If a replacement image is supplied, the existing image is removed and the new image is uploaded and stored. The K9's role assignments are also updated in the K9K9Role junction table before the changes are saved to the database. **K9 Entity** • K9Id (PK) • K9ImageUrl • K9Name • K9Breed • Gender • K9DateOfBirth • ActiveStatus • K9DadId (FK) • K9MomId (FK) **K9K9Role Entity (Junction Table)** • K9Id (PK, FK) • K9RoleId (PK, FK) **K9 Role Entity** • K9RoleId (PK) • RoleName [ALT]
			Step 10: The system will then record the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful update of the K9. The system will store information such as the authenticated user performing the action, the timestamp of the operation,

			and a descriptive log entry indicating that the K9 has been updated. **AuditLog Entity:** • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 11: The system displays a toaster notification confirming that the K9 has been successfully updated. The confirmation message displays the text: "K9 updated successfully."
Alternate courses:	ALT-Step 3: The selected K9 is inactive, the Angular frontend does not display the “Edit” button. Terminate use case.		
	ALT-Step 5: The super admin changes their mind and closes the pop-up by selecting the “Cancel” button. Terminate use case.		
	ALT-Step 6: The system finds an error after validating and an error message is displayed within the Edit K9 modal. Go back to step 4.		
	ALT-Step 8: The super admin does not confirm and selects the “Cancel” button. Go back to step 4.		
	ALT-Step 9: The system fails to save the changes and displays a toaster message: “Failed to update K9”. Terminate use case.		
Conclusion:	The new dog’s details have been saved onto the system, and a confirmation message is shown.		
Post-condition:	The system saves the new K9 details in the K9 table, stores any associated new K9 role assignments, uploads and stores the selected image (if changed), and makes the newly updated K9’s details available for viewing and management within the system.		
Business rules:	• Only super admin can update a dog.		

Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 51: Technical Narrative 8.4 Update Dog

2.4.7.5 Use Case 8.5: Deactivate Dog

System Name: Integrated NGO Management System			
Author:	Nicholas Bailey		**Date: 30/04/2026** **Version:** 1
Use case name:	Deactivate Dog		**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓
Use case id:	8.5		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	• None		
Description:	This use case describes how the super admin deactivates an existing K9. The system displays the selected K9's details, requests confirmation before proceeding, updates the K9's status to inactive, records the action in the audit log, and displays a confirmation message.		
Pre-condition:	The super admin must be logged into the system.		
Trigger:	The super admin wants to deactivate a dog on the system.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The super admin wants to deactivate a dog on the system.		**Step 2:** The system invokes UC 8.2 (View Dog) to display a pop-up showing the selected K9's details. This is done using the Angular frontend and the K9 Service, which routes the request via the Angular HTTP Client to the corresponding HTTP GET API endpoint

			in Visual Studio (K9Controller – GetK9ById).
	Step 3: The super admin selects the “Deactivate” button. [ALT]		**Step 4:** The system displays a custom Angular confirmation modal, which is triggered when the Super Admin clicks the Deactivate button. • Confirmation modal title ("Confirm Deactivation") • Confirmation message label ○ Displays the message: "Are you sure you want to deactivate this K9?" • Action buttons: ○ Confirm button ○ Cancel button
	Step 5: The super admin selects the “Confirm” button. [ALT]		**Step 6:** The system deactivates the selected K9 by routing the request through the Angular frontend and K9 Service, which sends the request via the Angular HTTP Client to the corresponding HTTP PUT API endpoint in Visual Studio (K9Controller – DeactivateK9). The API validates that the K9 exists and performs a soft delete by updating the ActiveStatus field to false using Entity Framework Core in MS SQL Server. The following entity is involved in the process: **K9 Entity** • K9Id (PK) • K9ImageUrl • K9Name • K9Breed • Gender

			• K9DateOfBirth • ActiveStatus • K9DadId (FK) • K9MomId (FK) The deactivated K9 is no longer displayed in the active K9 listings. [ALT]
			Step 7: The system will then record the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful deactivation of the K9. The system will store information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that a K9 has been deactivated. **AuditLog Entity:** • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 8: The system displays a toaster notification confirming that the K9 has been successfully deactivated. The confirmation message displays the text: "K9 deactivated successfully."
Alternate courses:	ALT-Step 3: The selected K9 is inactive, the Angular frontend does not display the “Deactivate” button. Terminate use case.		
	ALT-Step 5: The super admin does not confirm and selects the “Cancel” button. Go back to step 2.		

	ALT-Step 6: The system fails to deactivate the K9 and displays a toaster message: "Failed to deactivate K9". Terminate use case.
Conclusion:	The K9 has been deactivated on the system, and a confirmation message is shown.
Post-condition:	The system sets the K9's status to inactive in the K9 table and the K9 is then no longer displayed in the active K9 listings.
Business rules:	• Only super admin can deactivate a K9.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 52: Technical Narrative 8.5 Deactivate Dog

2.4.7.6 Use Case 8.6: Assign User to Dog

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Nicholas Bailey</td><td>Date: 13/07/2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">Assign User to Dog</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">8.6</td></tr>
<tr><td>Priority:</td><td colspan="2">High</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Super Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• Handler (ERA)</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">• None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes how the super admin assigns a handler to a selected dog. The system displays an assignment form that allows the super admin to search for and select an active handler. Once the assignment is confirmed, the system validates the request, creates a new handler assignment linked to the selected dog, notifies the assigned handler, records the transaction in the audit log, and displays a confirmation message.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The super admin must be logged into the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The super admin wants to assign a handler to a dog on the system.</td></tr>
<tr><td rowspan="2">Typical Course of Events:</td><td>Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td></td><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td></td><td>Step 1: The super admin wants to assign a handler to a dog on the system.</td><td></td><td>Step 2: The system invokes UC 8.2 (View Dog) to display a pop-up showing the selected K9's details. This is done using the Angular frontend and the K9 Service, which routes the request via the Angular HTTP Client to the corresponding HTTP</td></tr>
</table>

			GET API endpoint in Visual Studio (K9Controller – GetK9ById).
	Step 3: The super admin selects the "Assign" button. [ALT]		**Step 4:** The system displays a custom Angular Assign Handler modal, allowing the Super Admin to assign a handler to the selected K9. The assign handler modal contains the following components: • Modal heading ("Assign Handler") • Handler selection section: ○ Search textbox for locating handlers ○ Auto-complete list displaying matching active handlers • Action buttons: ○ Assign button ○ Cancel button The User search list is populated using data retrieved from the User Entity via the User Service, which calls the backend API. **User Entity** • UserId (PK) • UserName • UserSurname • ActiveStatus
	Step 5: The super admin searches for and selects a handler from the list before selecting the "Assign" button. [ALT]		**Step 6:** The system displays a custom Angular confirmation modal requesting confirmation before assigning the selected handler to the K9. The confirmation modal contains the following components:

			• Confirmation modal title ("Confirm Handler Assignment") • Confirmation message label ○ Displays the message: "Are you sure you want to assign this handler to the selected K9?" • Action buttons: ○ Confirm button ○ Cancel button
	Step 7: The super admin selects the "Confirm" button. [ALT]		**Step 8:** The system creates a new handler assignment by routing the request through the Angular frontend and K9 Service, which sends the request via the Angular HTTP Client to the corresponding HTTP POST API endpoint in Visual Studio (K9Controller – AssignUserToDog). The API validates that the selected K9 exists and is active, verifies that the selected user exists and is active, and confirms that the K9 does not already have an active handler. Entity Framework Core then creates a new assignment record in MS SQL Server. The following entity is involved in the process: **K9 Entity** • K9Id (PK) • ActiveStatus **K9User Entity** • AssignmentId (PK) • K9Id (FK) • UserId (FK) • StartDate • EndDate

			User Entity • UserId (PK) • UserName • UserSurname • ActiveStatus The assignment record stores the current date and time as the StartDate, while the EndDate remains null until the assignment is deactivated or updated. [ALT]
			Step 9: The system notifies the assigned handler of the new assignment. This notification is handled by the system's email notification service, which sends an email informing the handler that they have been assigned to the selected K9.
			Step 10: The system records the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful creation of the handler assignment. The system stores information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that a handler has been assigned to a K9. **AuditLog Entity** • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 11: The system displays a toaster notification confirming that the handler has been successfully assigned.

			The confirmation message displays the text: "Handler assigned successfully."
Alternate courses:	ALT-Step 3: The selected K9 is inactive, the Angular frontend does not display the “Assign” button. Terminate use case.		
	ALT-Step 5: The super admin changes their mind and closes the pop-up by selecting the “Cancel” button. Terminate use case.		
	ALT-Step 7: The super admin does not confirm and selects the “Cancel” button. Go back to step 4.		
	ALT-Step 8: The system fails to create the user-dog assignment and displays a toaster message: “Failed to assign handler”. Terminate use case.		
Conclusion:	The dog has been assigned to a handler on the system, and a confirmation message is shown.		
Post-condition:	A new handler assignment is created in the K9User table, linking the selected handler to the selected dog, and the assignment becomes the dog's current active handler.		
Business rules:	• Only super admin can assign a handler to a dog. • A dog can only have one handler.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 53: Technical Narrative 8.6 Assign User to Dog

2.4.7.7 Use Case 8.7: View User-Dog Assignment

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Nicholas Bailey</td><td>Date: 13/07/2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">View User-Dog Assignment</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">8.7</td></tr>
<tr><td>Priority:</td><td colspan="2">High</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Handler</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">• None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes how a handler views all K9 handler assignments recorded in the system. The system retrieves all K9 and handler assignment records from the database and displays them in a table, allowing the handler to search by K9 or handler name and filter assignments by their status (Active, Inactive, or All).</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The handler must be logged into the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The handler wants to view all the dogs and their current handlers on the system.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The handler wants to view all K9 handler assignments on the system.</td><td></td><td>Step 2: The system invokes UC 8.3 (View Dogs) to display the K9 page showing all registered K9s. This is done using the Angular frontend and the K9 Service, which routes the request via the Angular HTTP Client to the corresponding HTTP GET API endpoint</td></tr>
</table>

			in Visual Studio (K9Controller – GetAllK9s).
	Step 3: The super admin selects the "View Assignments" button.		**Step 4:** The Angular frontend navigates to the K9 Assignments page. The system then retrieves the assignments through the K9 Service in the Angular frontend, which routes the request via the Angular HTTP Client to the corresponding HTTP GET API endpoint in Visual Studio (K9Controller – GetK9UserAssignments). The API retrieves all K9 handler assignments from the database using Entity Framework Core in Microsoft SQL Server. The following entity is involved in the process: **K9 Entity** • K9Id (PK) • K9Name **K9User Entity** • AssignmentId (PK) • K9Id (FK) • UserId (FK) • StartDate • EndDate **User Entity** • UserId (PK) • UserName • UserSurname • ActiveStatus
			Step 5: The system displays the K9 Assignments page containing a table of

			all handler assignments. The table displays the following information: • K9 Name • Handler Name • Start Date • End Date • Assignment Status (Active/Inactive) The page also provides the following filtering options: • Search by K9 name. • Search by handler name. • Filter by assignment status (Active, Inactive, or All). [ALT]
	Step 6: The handler can view the K9 handler assignments.		
Alternate courses:	ALT-Step 5: There are currently no assignments, and the system displays a message to the handler. Terminate use case		
Conclusion:	The system displays all the assignments on the system.		
Post-condition:	The current K9 handler assignments are displayed on the K9 Assignments page, and the handler can view, search, and filter the assignment records.		
Business rules:	• Handlers can view assignments. • Admins and super admins can view user-dog assignments.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		

Open issues:	• None

Table 54: Technical Narrative 8.7 View User-Dog Assignment

2.4.7.8 Use Case 8.8: Update User-Dog Assignment

System Name: Integrated NGO Management System			
Author:	Nicholas Bailey		**Date: 13/07/2026** **Version:** 1
Use case name:	Update User-Dog Assignment		**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓
Use case id:	8.8		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• Handler (ERA)		
Other interested stakeholders:	• None		
Description:	This use case describes how the super admin updates the handler assigned to a selected dog. The system displays an update handler form that allows the super admin to search for and select a different active handler. Once the update is confirmed, the system validates the request, ends the current handler assignment, creates a new handler assignment linked to the selected dog, notifies both the previously assigned and newly assigned handlers via email, records the transaction in the AuditLog, and displays a confirmation message.		
Pre-condition:	The super admin must be logged into the system.		
Trigger:	The super admin wants to change a dog's handler on the system.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The super admin wants to update a dog's handler		**Step 2:** The system invokes UC 8.2 (View Dog) to display a pop-up showing the selected K9's details. This is done using the Angular frontend and the K9 Service, which routes the request via the Angular

	assignment on the system.		HTTP Client to the corresponding HTTP GET API endpoint in Visual Studio (K9Controller – GetK9ById).
	Step 3: The super admin selects the "Update" button. [ALT]		**Step 4:** The system displays a custom Angular Update Handler modal, allowing the Super Admin to assign a different handler to the selected K9. The update handler modal contains the following components: • Modal heading ("Update Handler") • Handler selection section: ○ Search textbox for locating handlers ○ Auto-complete list displaying matching active handlers ○ The currently assigned handler is pre-selected • Action buttons: ○ Update button ○ Cancel button The User search list is populated using data retrieved from the User Entity via the User Service, which calls the backend API. **User Entity** • UserId (PK) • UserName • UserSurname • ActiveStatus
	Step 5: The super admin searches for and selects a new handler, then selects		**Step 6:** The system displays a custom Angular confirmation modal requesting confirmation before updating the handler assignment.

	the “Update” button. [ALT]		The confirmation modal contains: • Modal heading ("Confirm Handler Update") • Confirmation message ○ "Are you sure you want to update this handler assignment?" • Action buttons: ○ Confirm ○ Cancel
	Step 7: The super admin selects the “Confirm” button. [ALT]		**Step 8:** The system routes the request through the Angular frontend and K9 Service, which sends the request via the Angular HTTP Client to the corresponding HTTP PUT API endpoint in Visual Studio (K9Controller – UpdateUserDogAssignment). The API validates that the selected K9 exists and is active, verifies that an active assignment exists for the selected K9, confirms that the newly selected handler is different from the current handler, and validates that the selected user exists and is active. Entity Framework Core then closes the current assignment by recording the current date and time as the EndDate and creates a new assignment record with the newly selected handler and the current date and time as the StartDate. The following entities are involved in the process: **K9 Entity** • K9Id (PK) • ActiveStatus **K9User Entity**

			• AssignmentId (PK) • K9Id (FK) • UserId (FK) • StartDate • EndDate **User Entity** • UserId (PK) • UserName • UserSurname • ActiveStatus [ALT]
			Step 9: The system notifies the previously assigned handler that they are no longer assigned to the selected K9. This notification is handled by the system's email notification service, which sends an email informing the handler that the assignment has been removed.
			Step 10: The system notifies the newly assigned handler of the updated assignment. This notification is handled by the system's email notification service, which sends an email informing the handler that they have been assigned to the selected K9.
			Step 11: The system records the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful update of the handler assignment. The system stores information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that a handler has been assigned to a K9.

			AuditLog Entity • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 12: The system displays a toaster notification confirming that the handler assignment has been successfully updated. The confirmation message displays the text: "Handler updated successfully."
Alternate courses:	ALT-Step 3: The selected K9 is inactive, the Angular frontend does not display the “Update” button. Terminate use case.		
	ALT-Step 5: The super admin changes their mind and closes the pop-up by selecting the “Cancel” button. Terminate use case.		
	ALT-Step 7: The super admin does not confirm and selects the “Cancel” button. Go back to step 4.		
	ALT-Step 8: The system fails to create the update the user-dog assignment and displays a toaster message: “Failed to update handler”. Terminate use case.		
Conclusion:	The dog has been assigned to a handler on the system, and a confirmation message is shown.		
Post-condition:	The current handler assignment is updated successfully. The previous assignment is marked as inactive, a new active assignment is created for the selected handler, notification emails are sent to both affected handlers, and the updated assignment is available for viewing on the system.		
Business rules:	• Only super admin can change a dog’s handler. • A dog can only have one handler.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 55: Technical Narrative 8.8 Update User-Dog Assignment

2.4.7.9 Use Case 8.9: Reactivate User-Dog Assignment

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Nicholas Bailey</td><td>Date: 13/07/2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">Reactivate User-Dog Assignment</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">8.9</td></tr>
<tr><td>Priority:</td><td colspan="2">High</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Super Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• Handler (ERA)</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">• None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes how the super admin restores a previous handler assignment for a selected dog. The system allows the super admin to reactivate the most recent previous handler assignment after confirmation. The system validates the request, creates a new active assignment for the previous handler, notifies the restored handler via email, records the transaction in the AuditLog, and displays a confirmation message.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The super admin must be logged into the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The super admin wants to reactivate a user-dog assignment on the system.</td></tr>
<tr><td rowspan="2">Typical Course of Events:</td><td>Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td></td><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td></td><td>Step 1: The super admin wants to reactivate a previous handler assignment</td><td></td><td>Step 2: The system invokes UC 8.2 (View Dog) to display a pop-up showing the selected K9's details. This is done using the Angular frontend and the K9 Service, which routes the request via the Angular HTTP Client to the corresponding HTTP</td></tr>
</table>

	for a selected dog on the system.		GET API endpoint in Visual Studio (K9Controller – GetK9ById).
	Step 3: The super admin selects the "Reactivate" button. [ALT]		**Step 4:** The system displays a custom Angular confirmation modal requesting confirmation before restoring the previous handler assignment. The confirmation modal contains: • Modal heading ("Restore Previous Handler") • Confirmation message ○ "Are you sure you want to restore the previous handler assignment for this K9?" • Action buttons: ○ Confirm button ○ Cancel button
	Step 5: The super admin selects the "Confirm" button. [ALT]		**Step 6:** The system routes the request through the Angular frontend and K9 Service, which sends the request via the Angular HTTP Client to the corresponding HTTP PUT API endpoint in Visual Studio (K9Controller – ReactivateUserDogAssignment). The API validates that the selected K9 exists and is active. It then retrieves the most recent previous handler assignment from the database. Entity Framework Core creates a new assignment record for the previous handler, recording the current date and time as the StartDate and leaving the EndDate as null. The following entities are involved in the process: **K9 Entity** • K9Id (PK)

			• ActiveStatus **K9User Entity** • AssignmentId (PK) • K9Id (FK) • UserId (FK) • StartDate • EndDate **User Entity** • UserId (PK) • UserName • UserSurname • ActiveStatus [ALT]
			Step 7: The system notifies the restored handler that they have been assigned to the selected K9. This notification is handled by the system's email notification service, which sends an email informing the handler that their assignment has been restored.
			Step 8: The system records the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful reactivation of the previous handler assignment. The system stores information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that a handler has been assigned to a K9. **AuditLog Entity** • AuditLogID (PK) • UserID (FK)

			• Timestamp • LogEntry
			Step 9: The system displays a toaster notification confirming that previous handler assignment has been successfully reactivated. The confirmation message displays the text: "Previous handler restored successfully."
Alternate courses:	ALT-Step 3: The selected K9 is inactive, the Angular frontend does not display the "Reactivate" button. Terminate use case.		
	ALT-Step 5: The super admin does not confirm and selects the "Cancel" button. Go back to step 2.		
	ALT-Step 6: The system fails to reactivate the user-dog assignment and displays a toaster message: "Failed to restore handler". Terminate use case.		
Conclusion:	The dog's previous assignment has been reactivated, and a confirmation message is shown.		
Post-condition:	The previous handler assignment is restored successfully. A new active assignment is created for the previous handler, the restored handler is notified via email, and the updated assignment is available for viewing on the system.		
Business rules:	• Only super admin can reactivate a user-dog assignment.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 56: Technical Narrative 8.9 Reactivate User-Dog Assignment

2.4.7.10 Use Case 8.10: Deactivate User-Dog Assignment

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Nicholas Bailey</td><td>Date: 13/07/2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">Deactivate User-Dog Assignment</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">8.10</td></tr>
<tr><td>Priority:</td><td colspan="2">High</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Super Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• Handler (ERA)</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">• None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes how the super admin removes the current handler assignment from a selected dog. The system requests confirmation before removing the assignment. Once confirmed, the system validates the request, deactivates the current user-dog assignment by recording an end date, notifies the removed handler, records the transaction in the audit log, and displays a confirmation message.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The super admin must be logged into the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The super admin wants to deactivate a user-dog assignment on the system.</td></tr>
<tr><td rowspan="2">Typical Course of Events:</td><td>Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td></td><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td></td><td>Step 1: The super admin wants to deactivate a user-dog assignment on the system.</td><td></td><td>Step 2: The system invokes UC 8.2 (View Dog) to display a pop-up showing the selected K9's details. This is done using the Angular frontend and the K9 Service, which routes the request via the Angular HTTP Client to the corresponding HTTP</td></tr>
</table>

			GET API endpoint in Visual Studio (K9Controller – GetK9ById).
	Step 3: The super admin selects the "Deactivate" button. [ALT]		**Step 4:** The system displays a custom Angular confirmation modal requesting confirmation before restoring the previous handler assignment. The confirmation modal contains: • Modal heading ("Confirm Handler Removal") • Confirmation message ○ "Are you sure you want to remove the current handler from this K9?" • Action buttons: ○ Confirm button ○ Cancel button
	Step 5: The super admin selects the “Confirm” button. [ALT]		**Step 6:** The system routes the request through the Angular frontend and K9 Service, which sends the request via the Angular HTTP Client to the corresponding HTTP PUT API endpoint in Visual Studio (K9Controller – DeactivateUserDogAssignment). The API validates that the selected K9 exists and is active. It retrieves the current active handler assignment and records the current date and time as the EndDate, thereby marking the assignment as inactive. Entity Framework Core then saves the updated assignment in Microsoft SQL Server. The following entity is involved in the process: **K9User Entity** • AssignmentId (PK) • K9Id (FK)

			• UserId (FK) • StartDate • EndDate [ALT]
			Step 7: The system notifies the previously assigned handler that they are no longer assigned to the selected K9. This notification is handled by the notification component of the system and informs the handler that their assignment has been removed.
			Step 8: The system records the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful deactivation of the previous handler assignment. The system stores information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that a handler has been removed from a K9. **AuditLog Entity** • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 9: The system displays a toaster notification confirming that the current handler assignment has been successfully deactivated. The confirmation message displays the text: "Handler removed successfully."
Alternate courses:	ALT-Step 3: The selected K9 is inactive, the Angular frontend does not display the “deactivate” button. Terminate use case.		

	ALT-Step 5: The super admin does not confirm and selects the "Cancel" button. Go back to step 2.
	ALT-Step 6: The system fails to deactivate the user-dog assignment and displays a toaster message: "Failed to remove handler". Terminate use case.
Conclusion:	The dog's current assignment has been deactivated, and a confirmation message is shown.
Post-condition:	The current user-dog assignment has been deactivated. The selected dog no longer has an active handler assignment, the previous handler has been notified of the removal, and the updated assignment is available for viewing on the system.
Business rules:	• Only super admin can deactivate a user-dog assignment.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 57: Technical Narrative 8.10 Deactivate User-Dog Assignment

2.4.7.11 Use Case 8.11: Add K9 Role

System Name: Integrated NGO Management System			
Author:	Nicholas Bailey		**Date: 13/07/2026** **Version:** 1
Use case name:	Add K9 Role		**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓
Use case id:	8.11		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	• None		
Description:	This use case describes how the super admin creates a new K9 role within the system. The system displays a form that allows the super admin to enter the role name and description. Once the information is validated and confirmed, the system creates the new K9 role, records the transaction in the audit log, and displays a confirmation message.		
Pre-condition:	The super admin must be logged into the system.		
Trigger:	The super admin wants to add a new K9 role to the system.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The super admin wants to add a new K9 role to the system.		**Step 2:** The system invokes UC 8.13 (View K9 Roles) to display the K9 Roles page showing all registered K9 roles. This is done using the Angular frontend and the K9 Role Service, which routes the request via the Angular HTTP Client to the corresponding HTTP GET API endpoint in

			Visual Studio (K9RoleController – GetAllK9Roles).
	Step 3: The super admin selects the "Create Role" button.		**Step 4:** The system loads and displays the Create K9 Role modal with the following components: • Create K9 Role modal title. • Role Name section: ○ Role Name label. ○ Role Name input field. ○ Placeholder text: *"e.g. Tracker"*. • Role Description section: ○ Role Description label. ○ Role Description input field. ○ Placeholder text: "What does this role do?". • Action buttons: ○ Create button. ○ Cancel button. The Create button remains disabled until both the Role Name and Role Description fields contain values.
	Step 5: The super admin enters the K9 role details and selects the "Create" button. [ALT]		**Step 6:** The system validates the entered details using Angular frontend validation before allowing the K9 role to be created. The system validates the following: • The Role Name field is not null or empty. • The Role Description field is not null or empty. [ALT]

			Step 7: The system displays a custom Angular confirmation modal after successful validation. The confirmation modal contains: • Confirmation modal title ("Confirm Create Role"). • Confirmation message label: ○ Displays the message: ▪ "Are you sure you want to create this role?" • Displays the name of the new role for confirmation. • Action buttons: ○ Confirm button. ○ Cancel button.
	Step 8: The super admin selects the "Confirm" button. [ALT]		**Step 9:** The system creates a new K9 Role by routing the request through the Angular frontend and K9 Role Service, which sends the request via the Angular HTTP Client to the corresponding HTTP POST API endpoint in Visual Studio (K9RoleController – CreateK9Role). The API endpoint passes the request to the K9 Role Repository, where a new K9Role entity is created. The repository automatically assigns the ActiveStatus value as true before the data is saved using Entity Framework Core in Microsoft SQL Server. The following entity is involved in the process: **K9 Role Entity** • K9RoleId (PK) • RoleName

			• RoleDescription • ActiveStatus After the entity has been created, Entity Framework Core generates the primary key and stores the new K9 Role record in the K9Role table. [ALT]
			Step 10: The system will then record the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful creation of the new K9 Role. The system will store information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that a new K9 Role has been created. **AuditLog Entity:** • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 11: The system displays a toaster notification confirming that the K9 Role has been successfully created. The confirmation message displays the text: "Role created successfully."
Alternate courses:	ALT-Step 5: The super admin changes their mind and closes the pop-up by selecting the “Cancel” button. Terminate use case.		
	ALT-Step 6: The system finds an error after validating and an error message is displayed within the Create K9 Role modal. Go back to step 4.		
	ALT-Step 8: The super admin does not confirm and selects the “Cancel” button. Go back to step 4.		

	ALT-Step 9: The system fails to save the new K9 role and displays a toaster message: "Failed to create role". Terminate use case.
Conclusion:	The new K9 role's details have been saved onto the system, and a confirmation message is shown.
Post-condition:	The system saves the new K9 Role in the K9Role table and makes the newly created role available for assignment to K9s and for management within the system.
Business rules:	• Only super admin can add a new K9 role.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 58: Technical Narrative 8.11 Add K9 Role

2.4.7.12 Use Case 8.12: Update K9 Role

System Name: Integrated NGO Management System			
Author:	Nicholas Bailey		**Date: 13/07/2026** **Version:** 1
Use case name:	Update K9 Role		**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓
Use case id:	8.12		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	• None		
Description:	This use case describes how the super admin updates an existing K9 role within the system. The system displays the selected K9 role's details in an editable form, allowing the super admin to modify the role name and description. Once the updated information is validated and confirmed, the system saves the changes, records the transaction in the audit log, and displays a confirmation message.		
Pre-condition:	The super admin must be logged into the system.		
Trigger:	The super admin wants to update a K9 role's details on the system.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The super admin wants to update a K9 role's detail on the system.		**Step 2:** The system invokes UC 8.13 (View K9 Roles) to display the K9 Roles page showing all registered K9 roles. This is done using the Angular frontend and the K9 Role Service, which routes the request via the Angular HTTP Client to the

			corresponding HTTP GET API endpoint in Visual Studio (K9RoleController – GetAllK9Roles).
	Step 3: The super admin selects the "Edit" button for a specific K9 Role.		**Step 4:** The system loads and displays the Edit K9 Role modal with the selected role's existing details pre-populated. The Edit K9 Role modal contains the following components: • Edit K9 Role modal title. • Role Name section: ○ Role Name label. ○ Role Name input field containing the current role name. • Role Description section: ○ Role Description label. ○ Role Description input field containing the current role description. • Action buttons: ○ Save button. ○ Cancel button. The Save button remains disabled until both the Role Name and Role Description fields contain values.
	Step 5: The super admin updates the K9 Role details and selects the "Save" button. [ALT]		**Step 6:** The system validates the entered details using Angular frontend validation before allowing the K9 Role to be updated. The system validates the following: • The Role Name field is not null or empty. • The Role Description field is not null or empty. [ALT]

			Step 7: The system displays a custom Angular confirmation modal after successful validation. The confirmation modal contains: • Confirmation modal title ("Confirm Update Role"). • Confirmation message label: ○ Displays the message: ▪ "Are you sure you want to update this role?" • Displays the updated role name for confirmation. • Action buttons: ○ Confirm button. ○ Cancel button.
	Step 8: The super admin selects the "Confirm" button. [ALT]		**Step 9:** The system updates the selected K9 Role by routing the request through the Angular frontend and K9 Role Service, which sends the request via the Angular HTTP Client to the corresponding HTTP PUT API endpoint in Visual Studio (K9RoleController – UpdateK9Role). The API endpoint passes the request to the K9 Role Repository, where the selected K9 Role is retrieved from the database using its primary key. The repository updates the Role Name and Role Description before saving the changes using Entity Framework Core in Microsoft SQL Server. The following entity is involved in the process: **K9 Role Entity** • K9RoleId (PK)

			• RoleName • RoleDescription • ActiveStatus [ALT]
			Step 10: The system will then record the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful update of the K9 Role. The system will store information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that a K9 Role has been updated. **AuditLog Entity:** • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 11: The system displays a toaster notification confirming that the K9 Role has been successfully updated. The confirmation message displays the text: "Role updated successfully."
Alternate courses:	ALT-Step 5: The super admin changes their mind and closes the pop-up by selecting the “Cancel” button. Terminate use case.		
	ALT-Step 6: The system finds an error after validating and an error message is displayed within the Edit K9 Role modal. Go back to step 4.		
	ALT-Step 8: The super admin does not confirm and selects the “Cancel” button. Go back to step 4.		
	ALT-Step 9: The system fails to save the changes and displays a toaster message: “Failed to update role”. Terminate use case.		

Conclusion:	The updated K9 role's details have been saved onto the system, and a confirmation message is shown.
Post-condition:	The system saves the updated K9 Role details in the K9Role table, and the modified role is immediately available for viewing, assignment to K9s, and management within the system.
Business rules:	• Only super admin can update a K9 role.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 59: Technical Narrative 8.12 Update K9 Role

2.4.7.13 Use Case 8.13: View K9 Roles

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Nicholas Bailey</td><td>Date: 13/07/2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">View K9 Roles</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">8.13</td></tr>
<tr><td>Priority:</td><td colspan="2">High</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Handler</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">• None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes how the handler views the K9 Roles available within the system. The system retrieves all K9 Roles from the database and displays them in a searchable and filterable table showing the role name, description, status, and available actions.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The handler must be logged into the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The handler wants to view the K9 roles on the system.</td></tr>
<tr><td rowspan="2">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td></td><td>Step 1: The handler wants to view the K9 roles on the system.</td><td></td><td>Step 2: The system invokes UC 8.3 (View Dogs) to display the K9 page showing all registered K9s. This is done using the Angular frontend and the K9 Service, which routes the request via the Angular HTTP Client to the corresponding HTTP</td></tr>
</table>

			GET API endpoint in Visual Studio (K9Controller – GetAllK9s).
	Step 3: The handler selects the "Manage K9 Roles" button.		**Step 4:** The system retrieves the K9 Roles by routing the request through the Angular frontend and K9 Role Service, which sends the request via the Angular HTTP Client to the corresponding HTTP GET API endpoint in Visual Studio (K9RoleController – GetAllK9Roles). The API endpoint passes the request to the K9 Role Repository, where all K9 Roles are retrieved from the database using Entity Framework Core in Microsoft SQL Server. The following entity is involved in the process: **K9 Role Entity** • K9RoleId (PK) • RoleName • RoleDescription • ActiveStatus The repository returns the collection of K9 Roles to the controller, which returns the data to the Angular frontend.
			Step 5: The system displays the retrieved K9 Roles in a table on the Angular frontend. Each K9 Role displays the following information: • Role Name • Role Description • Status • Available Actions The page also provides the following filtering options:

			• Search by role name. • Filter by status (Active, Inactive, or All). • Results counter. Each K9 Role also provides the following action buttons: • Edit. • Deactivate (for active roles). • Reactivate (for inactive roles). If no K9 Roles are available, the system displays the message: "No roles found."
	Step 6: The handler can view the K9 roles		
Alternate courses:	None		
Conclusion:	The system displays a pop-up with a table showing all the current roles.		
Post-condition:	The system retrieves and displays the K9 Roles stored in the K9Role table, allowing authorized users to view, search, and filter the available roles within the system		
Business rules:	• Handlers and admins can view all K9 roles on the system. • Super admins can view all K9 roles on the system		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 60: Technical Narrative 8.13 View K9 Roles

2.4.7.14 Use Case 8.14: Reactivate K9 Role

System Name: Integrated NGO Management System			
Author:	Nicholas Bailey		**Date: 30/04/2026** **Version:** 1
Use case name:	Reactivate K9 Role		**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓
Use case id:	8.14		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	• None		
Description:	This use case describes how the super admin reactivates an inactive K9 Role within the system. The system requests confirmation before restoring the role's active status. Once confirmed, the system updates the K9 Role, records the transaction in the audit log, and displays a confirmation message.		
Pre-condition:	The super admin must be logged into the system.		
Trigger:	The super admin wants to reactivate a K9 role on the system.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The super admin wants to reactivate a K9 Role on the system		**Step 2:** The system invokes UC 8.13 (View K9 Roles) to display the K9 Roles page showing all registered K9 Roles. This is done using the Angular frontend and the K9 Role Service, which routes the request via the Angular HTTP Client to the corresponding HTTP GET API

			endpoint in Visual Studio (K9RoleController – GetAllK9Roles).
	Step 3: The super admin selects the "Reactivate" button for a K9 Role with an inactive status.		**Step 4:** The system displays a custom Angular confirmation modal, which is triggered when the Super Admin clicks the Reactivate button. • Confirmation modal title ("Confirm Reactivate Role") • Confirmation message label ○ Displays the message: ▪ "Are you sure you want to reactivate this role?" • Displays the selected K9 Role name. • Action buttons: ○ Reactivate button. ○ Cancel button.
	Step 5: The super admin selects the "Reactivate" button. [ALT]		**Step 6:** The system reactivates the selected K9 Role by routing the request through the Angular frontend and K9 Role Service, which sends the request via the Angular HTTP Client to the corresponding HTTP PUT API endpoint in Visual Studio (K9RoleController – ReactivateK9Role). The API validates that the K9 Role exists and updates the ActiveStatus field to true using Entity Framework Core in Microsoft SQL Server. The following entity is involved in the process: **K9 Role Entity** • K9RoleId (PK) • RoleName

			• RoleDescription • ActiveStatus The reactivated K9 Role is displayed in the active K9 Role listings and becomes available for assignment to K9s. [ALT]
			Step 7: The system will then record the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful reactivation of the K9 Role. The system will store information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that a K9 Role has been reactivated. **AuditLog Entity:** • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 8: The system displays a toaster notification confirming that the K9 Role has been successfully reactivated. The confirmation message displays the text: "Role reactivated successfully."
Alternate courses:	ALT-Step 5: The super admin does not confirm and selects the “Cancel” button. Go back to step 2.		
	ALT-Step 6: The system fails to reactivate the K9 role and displays a toaster message: “Failed to reactivate role”. Terminate use case.		
Conclusion:	The inactive K9 role has been reactivated, and a confirmation message is shown		

Post-condition:	The system updates the K9 Role's status to active in the K9Role table, making the role available for viewing, assignment to K9s, and management within the system.
Business rules:	• Only super admin can reactivate a K9 role.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 61: Technical Narrative 8.14 Reactivate K9 Role

2.4.7.15 Use Case 8.15: Deactivate K9 Role

System Name: Integrated NGO Management System			
Author:	Nicholas Bailey		**Date: 30/04/2026** **Version:** 1
Use case name:	Deactivate K9 Role		**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓
Use case id:	8.15		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	• None		
Description:	This use case describes how the super admin deactivates an active K9 Role within the system. The system requests confirmation before deactivating the role's active status. Once confirmed, the system updates the K9 Role, records the transaction in the audit log, and displays a confirmation message.		
Pre-condition:	The super admin must be logged into the system.		
Trigger:	The super admin wants to deactivate a K9 role on the system.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The super admin wants to deactivate a K9 Role on the system.		**Step 2:** The system invokes UC 8.13 (View K9 Roles) to display the K9 Roles page showing all registered K9 Roles. This is done using the Angular frontend and the K9 Role Service, which routes the request via the Angular HTTP Client to the corresponding HTTP GET API

			endpoint in Visual Studio (K9RoleController – GetAllK9Roles).
	Step 3: The super admin selects the "Deactivate" button for a K9 Role with an active status.		**Step 4:** The system displays a custom Angular confirmation modal, which is triggered when the Super Admin clicks the Deactivate button. • Confirmation modal title ("Confirm Deactivate Role") • Confirmation message label ○ Displays the message: ▪ "Are you sure you want to deactivate this role?" • Displays the selected K9 Role name. • Action buttons: ○ Deactivate button. ○ Cancel button.
	Step 5: The super admin selects the "Deactivate" button. [ALT]		**Step 6:** The system deactivates the selected K9 Role by routing the request through the Angular frontend and K9 Role Service, which sends the request via the Angular HTTP Client to the corresponding HTTP PUT API endpoint in Visual Studio (K9RoleController – DeactivateK9Role). The API validates that the K9 Role exists and updates the ActiveStatus field to false using Entity Framework Core in Microsoft SQL Server. The following entity is involved in the process: **K9 Role Entity** • K9RoleId (PK) • RoleName

			• RoleDescription • ActiveStatus The deactivated K9 Role is no longer displayed in the active K9 Role listings but remains available when viewing all or inactive K9 Roles. [ALT]
			Step 7: The system will then record the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful deactivation of the K9 Role. The system will store information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that a K9 Role has been deactivated. **AuditLog Entity:** • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 8: The system displays a toaster notification confirming that the K9 Role has been successfully deactivated. The confirmation message displays the text: "Role deactivated successfully."
Alternate courses:	ALT-Step 5: The super admin does not confirm and selects the “Cancel” button. Go back to step 2.		
	ALT-Step 6: The system fails to deactivate the K9 role and displays a toaster message: “Failed to deactivate role”. Terminate use case.		

Conclusion:	The active K9 role has been deactivated, and a confirmation message is shown.
Post-condition:	The system updates the K9 Role's status to inactive in the K9Role table, making the role unavailable in the active K9 Role listings while remaining available for viewing and management when inactive roles are displayed.
Business rules:	• Only super admin can deactivate a K9 role.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 62: Technical Narrative 8.15 Deactivate K9 Role

2.4.7.16 Use Case 8.16: View K9 Ancestry

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Nicholas Bailey</td><td>Date: 13/07/2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">View K9 Ancestry</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">8.16</td></tr>
<tr><td>Priority:</td><td colspan="2">Low</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Handler</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">• None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes how the handler views the ancestry of a selected K9 within the system. The system retrieves the selected K9's lineage information from the database and displays it as a graphical ancestry tree showing the current K9, its parents, grandparents, and great-grandparents.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The handler must be logged into the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The handler wants to view the ancestry tree of the dogs on the system.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The handler wants to view the ancestry tree of the dogs on the system.</td><td></td><td>Step 2: The system invokes UC 8.2 (View Dog) to display the View K9 Details modal containing the selected K9's information. This is done using the Angular frontend and the K9 Service, which routes the request via the Angular HTTP Client to the</td></tr>
</table>

			corresponding HTTP GET API endpoint in Visual Studio (K9Controller – GetK9ById).
	Step 3: The super admin selects the "View Ancestry" button.		**Step 4:** The system retrieves the selected K9's ancestry by routing the request through the Angular frontend and K9 Service, which sends the request via the Angular HTTP Client to the corresponding HTTP GET API endpoint in Visual Studio (K9Controller – GetK9Ancestry). The API endpoint passes the request to the K9 Repository, where the selected K9 and its lineage are retrieved using Entity Framework Core in Microsoft SQL Server. The repository recursively traverses the parent relationships stored in the K9 table to construct the ancestry tree for the requested number of generations. The following entity is involved in the process: **K9 Entity** • K9Id (PK) • K9ImageUrl • K9Name • K9Breed • Gender • K9DateOfBirth • ActiveStatus • K9DadId (FK) • K9MomId (FK) [ALT]
			Step 5: The system displays the retrieved ancestry information as a horizontal graphical pedigree tree on the Angular frontend.

			The ancestry tree displays the following information for each available K9: • Image • Name • Breed The ancestry tree displays the following generations: • Current K9 • Parents • Grandparents • Great-grandparents Where no parent information exists, the system displays "Unknown" in the corresponding position. The Angular frontend also draws connector lines between related K9s and highlights the direct ancestral lineage when the handler hovers over a K9 within the ancestry tree.
	Step 6: The handler can view the selected K9's ancestry and navigate through the displayed lineage.		
Alternate courses:	ALT-Step 4: The system fails to retrieve the K9 ancestry and displays a toaster message: “Failed to retrieve ancestry”. Terminate use case.		
Conclusion:	The system generates and displays the family tree of the selected K9.		
Post-condition:	The system retrieves and displays the selected K9's ancestry from the K9 table, allowing the handler to view the K9's lineage up to the configured number of generations.		
Business rules:	• Handlers can view the family tree. • Super admins and admins can view the family tree.		

Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 63: Technical Narrative 8.16 View K9 Ancestry

2.4.8 Rhino Subsystem Technical Narratives

2.4.8.1 Use Case 9.1: Add Rhino

<table>
<tr><td colspan="4">System Name: Integrated NGO Management System</td></tr>
<tr><td>Author:</td><td colspan="2">Christopher Bailey</td><td>Date: 14 July 2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">Add Rhino</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">9.1</td></tr>
<tr><td>Priority:</td><td colspan="2">Medium</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Super Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">• None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event in which a super admin adds a rhino. The system will display the rhinos and allow the super admin to add a new rhino. The system will display the Add Rhino modal and allow the super admin to enter the new rhino’s details after which the system validates the details entered. The use case ends once the success toaster is displayed and the new rhino has successfully been captured on the system.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">Super Admin is logged into the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The super admin requests to add a rhino.</td></tr>
<tr><td rowspan="2">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The super admin requests to add a new rhino.</td><td></td><td>Step 2: The system invokes use case 9.3(View Rhinos).</td></tr>
</table>

	Step 3: The super admin selects the "Add Rhino" button.		**Step 4:** The system loads and displays the Add Rhino modal using the Angular frontend with the following controls: A header section containing: • A heading displaying **Create Rhino**. A form section containing: • A **Name** text box for entering the rhino's name. • A **Species** drop-down list containing the available rhino species. • A **Weight** numeric input field for entering the rhino's weight (optional). • A **Gender** drop-down list containing the available gender options. • A **Last Seen** date picker for selecting the date the rhino was last seen (optional). • A **Date of Birth** date picker for selecting the rhino's date of birth (optional). • A **Province** text box for entering the province where the rhino is located. • A **Horn Status** drop-down list containing the available horn status options.

			• An **Identification Notes** text area for entering additional identification information. • An **Image** file upload control for selecting an image of the rhino (optional). • A display of the selected image filename when an image has been chosen. An error message section containing: • A validation or processing error message displayed when an error occurs during the rhino creation process. A footer section containing: • A **Cancel** button to close the **Create Rhino** modal without creating a rhino. • A **Create** button to proceed with the rhino creation process. The button remains disabled until all required fields have been completed.
	Step 5: The super admin enters the new rhino's details and selects the "Create" button. [ALT]		**Step 6:** The system validates the details entered, validating the following:

			• All fields are completed. • The animal's name must not contain numeric characters. • The date of birth, if provided, cannot be a future date. • The last seen date, if provided, cannot be a future date. • The animal weight, if provided, must be greater than 0. • The uploaded image is of type jpeg, png, jpg or webp. [ALT]
			Step 7: The system loads and displays the Add Rhino Confirmation modal using the Angular frontend with the following controls: A header section containing: • A heading displaying **Confirm Create Rhino**. A confirmation section containing: • A confirmation message asking the super admin to confirm the creation of the rhino. A footer section containing: • A **Cancel** button to close the confirmation modal without creating the rhino.

			• A **Confirm** button to submit the request to create the rhino.
	Step 8: The super admin selects the "Confirm" button. [ALT]		**Step 9:** The Angular frontend calls the Rhino service to send a POST request to the C# Web API backend Rhino controller through the CreateRhino endpoint. The backend creates the Rhino in the following entities: Animal Entity: • AnimalId (PK) • AnimalName • AnimalSpecies • AnimalWeight • AnimalDateOfBirth • AnimalGender • LastSeen • IdentificationNotes • Province • AnimalPhoto • ActiveStatus Rhino Entity: • RhinoId (PK) • AnimalId (FK) • HornStatus [ALT]
			Step 10: The system will then record the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful creation of the rhino. The system will store information such as the authenticated user performing the action, the

			timestamp of the operation, and a descriptive log entry indicating that a rhino was created. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 11: The system displays a success toaster informing the admin that the rhino was successfully added.
Alternate courses:	[ALT] Step 5: The super admin selects the "Cancel" button. Use case terminates.		
	[ALT] Step 6: The validation fails. The system displays an error message. Go back to step 5.		
	[ALT] Step 7: The super admin selects the "Cancel" button. Go back to step 5.		
	[ALT] Step 9: The system fails to create the rhino. The system will display an error message based on the status code.		
Conclusion:	The system displays a success toaster informing the super admin that the rhino was added successfully.		
Post-condition:	The rhino is saved on the system.		
Business rules:	• Super admins can add rhinos.		
Implementation constraints and specifications:	• The user requires internet access to access this feature. • Exact locations of rhinos are not saved for security reasons.		
Assumptions:	• None		
Open issues:	• None		

Table 64: Technical Narrative 9.1 Add Rhino

2.4.8.2 Use Case 9.2: View Rhino

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Christopher Bailey</td><td>Date: 14 July 2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">View Rhino</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">9.2</td></tr>
<tr><td>Priority:</td><td colspan="2">Medium</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Super Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">• None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event in which a super admin views a rhino’s details. The system will display the rhinos and allow the super admin to filter and search for a rhino. The super admin will then select the rhino to view. The system will retrieve and display the selected rhino’s details. The use case ends once the rhino’s details are displayed.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">Super Admin is logged into the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The super admin requests to view a rhino.</td></tr>
<tr><td rowspan="4">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The super admin requests to view a rhino.</td><td></td><td>Step 2: The system invokes use case 9.3 (View Rhinos).</td></tr>
<tr><td>Step 3: The super admin applies the required filters to locate the rhino they wish to view. [ALT]</td><td></td><td>Step 4: The Angular frontend filters the retrieved rhinos based on the applied filters. [ALT]</td></tr>
</table>

	Step 5: The super admin selects the rhino.		**Step 6:** The Angular frontend calls the Rhino service to send a GET request to the C# Web API backend Rhino controller through the GetRhinoById endpoint. The backend retrieves the rhino from the following entities: Rhino Entity: • RhinoId (PK) • AnimalId (FK) • HornStatus • DehornedDate Animal Entity: • AnimalId (PK) • AnimalName • AnimalSpecies • AnimalWeight • AnimalDateOfBirth • AnimalGender • LastSeen • IdentificationNotes • Province • AnimalPhoto • ActiveStatus [ALT]
			Step 7: The system loads and displays the Rhino Details modal using the Angular frontend with the following controls populated with the selected rhino details: A header section containing: • A heading displaying the selected rhino's name.

			A details section containing: • A display of the rhino's image. • A display of the rhino's species. • A display of the rhino's gender. • A display of the rhino's weight. • A display of the rhino's location (province). • A display of the date the rhino was last seen. • A display of the rhino's age. • A display of the rhino's horn status. • A display of the rhino's active status. • A display of the dehorned date, if applicable. • A display of the rhino's identification notes. A footer section containing: • An **Edit** button to open the **Update Rhino** modal. The button is disabled when the rhino is inactive. • A **Deactivate** button to open the **Deactivate Rhino** confirmation modal. The button is disabled when the rhino is inactive.

			• A **Close** button to close the **Rhino Details** modal.
Alternate courses:	[ALT] Step 3: The super admin does not apply any filters and selects one of the rhinos already displayed. Continue from step 5.		
	[ALT] Step 4: No rhinos match the applied filters, display "No rhinos found" message and go back to step 3.		
	[ALT] Step 6: The system fails to retrieve the selected rhino's details. The system will display an error toaster message based on the status code.		
Conclusion:	The selected rhino's details are displayed.		
Post-condition:	The rhino's details are successfully displayed.		
Business rules:	• Super admins can view a rhino.		
Implementation constraints and specifications:	• The user requires internet access to access this feature.		
Assumptions:	• None		
Open issues:	• None		

Table 65: Technical Narrative 9.2 View Rhino

2.4.8.3 Use Case 9.3: View Rhinos

System Name: Integrated NGO Management System			
Author:	Christopher Bailey		**Date: 14 July 2026** **Version:** 1
Use case name:	View Rhinos		**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓
Use case id:	9.3		
Priority:	Medium		
Source:	EPI-USE		
Primary business actor (PBA):	Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	• None		
Description:	This use case describes the event in which a super admin views all rhinos on the system. The admin will navigate to and select the Animals tab and then select the Rhino tab. The system will load and display the Rhino screen. The use case ends once the system displays all the rhinos on the system.		
Pre-condition:	Super admin is logged in.		
Trigger:	Super admin requests to view all rhinos.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: Super admin requests to view all rhinos.		
	Step 2: The super admin navigates to and selects the Animals tab. [ALT]		

	Step 3: The super admin selects the "Rhino" tab.		**Step 4:** The Angular frontend calls the Projects service to send a GET request to the C# Web API backend Projects controller through the GetProjectLookup endpoint. The backend retrieves active project details from the following entities: Project Entity: • ProjectId (PK) • ProjectName [ALT]
			Step 5: The Angular frontend calls the Rhino service to send a GET request to the C# Web API backend Rhino controller through the GetAllRhinos endpoint. The backend retrieves the rhinos from the following entities: Rhino Entity: • RhinoId (PK) • AnimalId (FK) • HornStatus Animal Entity: • AnimalId (PK) • AnimalName • AnimalDateOfBirth • AnimalGender • AnimalSpecies • Province • AnimalPhoto • ActiveStatus [ALT]
			Step 6: The Angular frontend calls the Rhino service to retrieve the rhino distribution data from the C# ASP.NET Web API backend through the

			GetRhinoDistribution endpoint. The backend retrieves the data from the following entities: Rhino Entity: • RhinoId (PK) • AnimalId (FK) Animal Entity: • AnimalId (PK) • Province • ActiveStatus The backend retrieves active rhinos, groups them by **Province**, and counts the number of rhinos in each province. [ALT]
			Step 7: The system loads and displays the Rhino screen using the Angular frontend with the following controls: A header section containing: • A heading displaying **Rhinos**. • An **Add Rhino** button to open the **Create Rhino** modal. An interactive map that is implemented using the Leaflet JavaScript mapping library. Rhino location data retrieved from the backend is displayed either as map markers or as a heatmap using the Leaflet.heat plugin, allowing users to visualize both individual rhino locations and areas with higher rhino concentrations. The map section contains:

			• An interactive map displaying the geographical distribution of active rhinos. • A **Switch to Heatmap/Switch to Markers** button to toggle between the marker view and heatmap view of the rhino distribution. A filter section containing: • A **Search by Name** search box for filtering rhinos by name. • A **Project** dropdown list containing active projects retrieved from the backend. • A **Status** drop-down list containing **All Status**, **Active**, and **Inactive** options. • A **Species** drop-down list containing **All Species**, **Black Rhino**, and **White Rhino** options. • A **Gender** drop-down list containing **All Gender**, **Male**, and **Female** options. • A **Horn Status** drop-down list containing **All Horn Status**, **Has Horn**, and **Dehorned** options. • A **Search Location** text box for filtering rhinos by province.

			A rhino display section containing a collection of rhino cards. Each card displays: • The rhino's image. • The rhino's name. • The rhino's gender. • The rhino's species. • The rhino's location. • The rhino's age. • The rhino's horn status. • The rhino's active status. • Each rhino card is selectable and opens the **Rhino Details** modal for the selected rhino
Alternate courses:	[ALT] Step 2: The super admin is using a mobile device such as a phone. The super admin will select the menu (hamburger) icon and then the Animals option. Continue to step 3.		
	[ALT] Step 4: The system fails to retrieve the lookup data. The system will display an error toaster message based on the status code.		
	[ALT] Step 5: The system fails to retrieve the rhinos. The system will display an error toaster message based on the status code.		
	[ALT] Step 6: The system fails to retrieve the rhino distribution data. The system will display an error toaster message based on the status code.		
Conclusion:	The rhinos are successfully displayed.		
Post-condition:	All rhinos are displayed.		
Business rules:	• Super admins can view rhinos.		
Implementation constraints and specifications:	• The user requires internet access to access this feature.		

Assumptions:	• None
Open issues:	• None

Table 66: Technical Narrative 9.3 View Rhinos

2.4.8.4 Use Case 9.4: Update Rhino

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Christopher Bailey</td><td>Date: 15 July 2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">Update Rhino</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">9.4</td></tr>
<tr><td>Priority:</td><td colspan="2">Medium</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Super Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">• None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event in which a super admin updates a rhino on the system. The system will display the rhino’s details and allow the super admin to update the rhino’s details on the Update Rhino modal. The system will validate the updated details and ask the super admin for confirmation. Once confirmed the system will save the rhino’s updated details. The use case ends once the success toaster is displayed and the rhino’s updated details are successfully saved on the system.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">Super admin is logged in.
Rhino already exists.</td></tr>
<tr><td>Trigger:</td><td colspan="3">Super admin requests to update a rhino’s details.</td></tr>
<tr><td rowspan="2">Typical Course of Events:</td><td>Actor Action</td><td>System Response</td><td></td></tr>
<tr><td></td><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td></td><td>Step 1: Super admin requests to update a rhino’s details.</td><td></td><td>Step 2: The system invokes use case 9.2 (View Rhino).</td></tr>
</table>

	Step 3: The super admin selects the "Edit" button.		**Step 4:** The system loads and displays the Update Rhino modal using the Angular frontend with the following controls populated with the selected rhino's current details: A modal header containing: • A heading displaying **Update Rhino**. A modal body containing: • An error message displayed when validation or processing errors occur. • A **Name** text box for editing the rhino's name. • A **Species** drop-down list containing Black Rhino and White Rhino options. • A **Weight** numeric input field for editing the rhino's weight (optional). • A **Gender** drop-down list containing Male and Female options. • A **Last Seen** date picker for editing the rhino's last seen date (optional). • A **Date of Birth** date picker for editing the rhino's date of birth (optional). • A **Province** text box for editing the rhino's location. • A **Horn Status** drop-down list containing

			Has Horn and Dehorned options. • An **Identification Notes** text area for editing additional identification information. • A **Current Image** section displaying the rhino's existing image. • A **Replace Image** file upload control for selecting a new image (optional). • A display showing the name of the selected replacement image file after it has been chosen. A modal footer containing: • A **Cancel** button to close the modal without saving changes. • An **Update** button to validate the entered information and open the Update Rhino Confirmation modal. The Update button remains disabled until all required fields contain valid values.
	Step 5: The super admin enters the rhino's updated details and selects the "Update" button. [ALT]		**Step 6:** The system validates the details entered, validating the following: • All required fields are completed. • The rhino's name does not contain numbers. • Last seen and date of birth are not in the future. • The rhino's weight is greater than 0.

			[ALT]
			Step 7: The system loads and displays the Confirm Rhino Update modal using the Angular frontend with the following controls: A modal header containing: • A heading displaying **Confirm Update Rhino**. A modal body containing: • A confirmation message asking the super admin whether they wish to update the selected rhino. A set of confirmation actions containing: • A **Cancel** button to close the confirmation modal without updating the rhino and return to the Update Rhino modal. • A **Confirm** button to submit the updated rhino details to the system.
	Step 8: The super admin selects the "Confirm" button. [ALT]		**Step 9:** The Angular frontend calls the Rhino service to send a PUT request to the C# ASP.NET Web API backend through the UpdateRhino endpoint. The backend updated the rhino in the following entities: Rhino Entity: • RhinoId (PK) • AnimalId (FK) • HornStatus • DehornedDate

			Animal Entity: • AnimalId (PK) • AnimalName • AnimalSpecies • AnimalWeight • AnimalDateOfBirth • AnimalGender • LastSeen • IdentificationNotes • Province • AnimalPhoto If the rhino's **HornStatus** changes from **Has Horn** to **Dehorned**, the backend sets the **DehornedDate** to the current date. If a new image is provided, the backend updates the **AnimalPhoto**. [ALT]
			Step 10: The system will then record the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful update of the rhino. The system will store information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that a rhino was updated. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry

			Step 11: The system displays a success toaster informing the admin that the rhino was successfully updated.
Alternate courses:	[ALT] Step 5: The super admin selects the "Cancel" button. Use case terminates.		
	[ALT] Step 6: The validation fails. The system displays an error message. Go back to step 5.		
	[ALT] Step 8: The super admin selects the "Cancel" button. Go back to step 5.		
	[ALT] Step 9: The system fails to update the rhino's details. The system will display an error message based on the status code.		
Conclusion:	The success toaster is displayed informing the super admin that the rhino's details were successfully updated.		
Post-condition:	Rhino's updated details are successfully saved.		
Business rules:	• Super admins can update rhinos.		
Implementation constraints and specifications:	• The user requires internet access to access this feature.		
Assumptions:	• None		
Open issues:	• None		

Table 67: Technical Narrative 9.4 Update Rhino

2.4.8.5 Use Case 9.5: Deactivate Rhino

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Christopher Bailey</td><td>Date: 15 July 2026
Version: 1</td></tr>
<tr><td>Use case name:</td><td colspan="2">Deactivate Rhino</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">9.5</td></tr>
<tr><td>Priority:</td><td colspan="2">Medium</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Super Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">• None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event in which a super admin deactivates a rhino on the system. The system will display the rhino’s details and allow the super admin to deactivate the rhino. The system will then ask the super admin for confirmation. Once confirmed the system will deactivate the rhino. The use case ends once the success toaster is displayed and the rhino is successfully deactivated on the system.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">Super admin is logged in.
Rhino is active.</td></tr>
<tr><td>Trigger:</td><td colspan="3">Super admin requests to deactivate a rhino.</td></tr>
<tr><td rowspan="2">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: Super admin requests to deactivate a rhino.</td><td></td><td>Step 2: The system invokes use case 9.2 (View Rhino).</td></tr>
</table>

	Step 3: The super admin selects the "Deactivate" button.		**Step 4:** The system loads and displays the Confirm Rhino Deactivation modal using the Angular frontend with the following controls: A modal header containing: • A heading displaying **Confirm Rhino Deactivation**. A modal body containing: • An error message displayed if an error occurs during the deactivation process. • A confirmation message asking the super admin whether they wish to deactivate the selected rhino. A set of confirmation actions containing: • A **Cancel** button to close the confirmation modal without deactivating the rhino and return to the **Rhino Details** modal. • A **Confirm** button to deactivate the selected rhino.
	Step 5: The super admin selects the "Confirm" button. [ALT]]		**Step 6:** The Angular frontend calls the Rhino service to send a PUT request to the C# ASP.NET Web API backend through the DeactivateRhino endpoint. The backend updates the rhino's status in the following entity: Rhino Entity: • RhinoId (PK)

			• AnimalId (FK) Animal Entity: • AnimalId (PK) • ActiveStatus [ALT]
			Step 7: The system will then record the transaction in the AuditLog table to maintain a history of actions performed within the system. This is handled in the backend after the successful deactivation of the rhino. The system will store information such as the authenticated user performing the action, the timestamp of the operation, and a descriptive log entry indicating that a rhino was deactivated. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 8: The system displays a success toaster informing the admin that the rhino was successfully deactivated.
Alternate courses:	[ALT] Step 5: The super admin selects the "Cancel" button. Use case terminates.		
	[ALT] Step 6: The system fails to deactivate the rhino. The system will display an error message based on the status code.		
Conclusion:	The success toaster is displayed informing the super admin that the rhino was successfully deactivated.		
Post-condition:	The rhino was successfully deactivated.		

Business rules:	• Super admins can deactivate rhinos.
Implementation constraints and specifications:	• The user requires internet access to access this feature.
Assumptions:	• None
Open issues:	• None

Table 68: Technical Narrative 9.5 Deactivate Rhino

2.4.9 Elephant Subsystem Technical Narratives

2.4.9.1 Use Case 10.1: Add Elephant

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Stephan Bredenhann</td><td colspan="2">Date: 13 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>Add Elephant</td><td colspan="2" rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>10.1</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event when a Admin adds a new elephant to the system. After navigating to the Animals page and selecting the Elephant tab, the Admin selects “Add Elephant” and fills in a form covering the elephant’s name, species, gender, tusk status, location, and other details, with an optional photo. Once confirmed, the system saves the new elephant across the Animal and Elephant entities, and stores the photo if one was provided.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The Admin is authenticated and on the Animals page.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The Admin selects “Add Elephant” from the Elephant grid.</td></tr>
<tr><td rowspan="4">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: Admin navigates to the Animals page and selects the “Elephant” tab.</td><td></td><td>Step 2: The system calls “GET api/Elephant” to retrieve all existing elephants by joining the Elephant and Animal entities, then displays them as a grid of cards.</td></tr>
<tr><td>Step 3: Admin selects “Add Elephant”.</td><td></td><td>Step 4: The system opens a form with fields for Name, Species, Weight (optional), Gender, Last Seen (optional), Date of Birth (optional), Approximate Location, Tusk</td></tr>
</table>

			Status, Identification Notes, and an optional photo upload. The “Create” button is disabled until the form is valid.
	Step 5: Admin completes the form and selects “Create”. [ALT]		**Step 6:** The system validates the entered details, for example checking that the name contains no digits and that no dates are set in the future, then displays a confirmation prompt: “Are you sure you want to create this Elephant?”
	Step 7: Admin selects “Confirm”. [ALT]		**Step 8:** The ElephantService sends the form data, including the photo if provided, to “POST api/Elephant”. If a photo was uploaded, the system stores it on the server. The controller then creates new records in the following entities: **Animal Entity:** • AnimalId (PK) • AnimalName • AnimalSpecies • AnimalWeight • AnimalDateOfBirth • AnimalGender • LastSeen • IdentificationNotes • ApproximateLocation • AnimalPhoto • ActiveStatus **Elephant Entity:** • ElephantId (PK) • AnimalId (FK) • TuskStatus
			Step 9: Both records are saved together, with the new elephant’s ActiveStatus set to active by default. The API

			responds with an HTTP 201 Created status, and the frontend refreshes the grid and displays a success notification reading "Successfully created elephant".
Alternate courses:	[ALT] Step 5: Admin selects "Cancel". The form closes and resets.		
	[ALT] Step 6: Validation fails. The system displays an inline error message and keeps the form open.		
	[ALT] Step 7: Admin selects "Cancel" on the confirmation prompt. The prompt closes and the form data is kept.		
	[ALT] Step 8: The API returns an HTTP 400 response due to invalid data. The system displays "Please check the information entered."		
	[ALT] Step 8: The API returns an HTTP 500 response due to a server error. The system displays "System error, please try again."		
Conclusion:	The use case concludes when the new elephant is successfully saved and appears in the elephant grid.		
Post-condition:	A new record exists in the Animal entity with an active status, along with a linked record in the Elephant entity. The photo is stored on the server if one was provided.		
Business rules:	• Only Admins can add elephants. • Name, species, gender, tusk status, identification notes, and location are required fields.		
Implementation constraints and specifications:	• "POST api/Elephant" accepts the form data, including the optional photo, as a multipart request. • Uploaded photos are limited to JPEG, PNG, and WEBP formats.		
Assumptions:	• None.		
Open issues:	• None.		

Table 69: Technical Narrative 10.1 Add Elephant

2.4.9.2 Use Case 10.2: View Elephant

System Name: Integrated NGO Management System			
Author:	Stephan Bredenhann	**Date:** 13 July 2026 **Version:** 1.0	
Use case name:	View Elephant	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	10.2		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Admin		
Description:	This use case describes the event when an Admin views the full details of a single elephant. From the Elephant grid, the Admin selects a card, and the system retrieves and displays all of that elephant's details in a details window, along with relevant action buttons.		
Pre-condition:	The Admin is authenticated and the Elephant grid has loaded.		
Trigger:	The Admin selects an elephant card.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: Admin navigates to the Animals page, selects the "Elephant" tab, and the grid loads (see Use Case 10.3).		**Step 2:** The grid displays each elephant as a card showing its name, species, location, age, and status.
	Step 3: Admin selects an elephant card. [ALT]		**Step 4:** The system calls "GET api/Elephant/{id}" to retrieve the elephant's details by joining the Elephant and Animal entities. [ALT]
			Step 5: The system opens a details window showing the elephant's photo, Species, Gender,

			Weight, Location, Last Seen, Age (calculated from date of birth), Tusk Status, Active Status, and Identification Notes, along with "Edit", "Deactivate" (if active) or "Reactivate" (if inactive), and "Close" buttons. [ALT]
	Step 6: Admin reviews the elephant's details.		
Alternate courses:	[ALT] Step 3: Admin does not select a card, since the grid currently has no built-in search feature.		
	[ALT] Step 4: The elephant cannot be found. The system does not open the details window and does not display an error message.		
	[ALT] Step 5: A network or server error occurs while retrieving the details. The system does not display an error message to the Admin.		
Conclusion:	The use case concludes when the elephant's details are displayed to the Admin.		
Post-condition:	No data is modified. The elephant's details remain visible until the Admin closes the window.		
Business rules:	• Admins can view the details of any elephant.		
Implementation constraints and specifications:	• "GET api/Elephant/{id}" returns the elephant's details through a read-only query. • The elephant's age is calculated on the frontend from its date of birth.		
Assumptions:	• None.		
Open issues:	• There is no search or filter option in the elephant grid.		

Table 70: Technical Narrative 10.2 View Elephant

2.4.9.3 Use Case 10.3: View Elephants

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Stephan Bredenhann</td><td colspan="2">Date: 13 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>View Elephants</td><td colspan="2" rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>10.3</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event when an Admin views all elephants recorded in the system. After navigating to the Animals page and selecting the Elephant tab, the system retrieves every elephant record and displays them as cards in a grid. Both active and inactive elephants are shown, and the list is not paginated, sorted, or filtered.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The Admin is authenticated.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The Admin navigates to the Animals page and selects the “Elephant” tab.</td></tr>
<tr><td rowspan="4">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: Admin navigates to the Animals page.</td><td></td><td>Step 2: The system displays a tab bar with Rhino, Elephant, and K9 options.</td></tr>
<tr><td>Step 3: Admin selects the “Elephant” tab.</td><td></td><td>Step 4: The system calls “GET api/Elephant”, which retrieves every elephant by joining the Elephant and Animal entities. No filtering, sorting, or paging is applied to the query.</td></tr>
<tr><td></td><td></td><td>Step 5: For any elephant without a stored photo,</td></tr>
</table>

			the system assigns a default image. The API returns the full list of elephants, and the frontend renders them as a grid of cards showing each elephant's photo, name, species, location, age, and Active/Inactive status.
	Step 6: Admin views the list of elephants.		
Alternate courses:	None.		
Conclusion:	The use case concludes when all elephants, both active and inactive, are displayed in the grid.		
Post-condition:	No data is modified. The elephant grid reflects every elephant record in the system.		
Business rules:	• Admins can view all elephants.		
Implementation constraints and specifications:	• Photo file paths are converted to full URLs by the server before being returned.		
Assumptions:	• None.		
Open issues:	• None.		

Table 71: Technical Narrative 10.3 View Elephants

2.9.4.4Use Case 10.4: Update Elephant

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td colspan="2">Stephan Bredenhann</td><td>Date: 13 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td colspan="2">Update Elephant</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td colspan="2">10.4</td></tr>
<tr><td>Priority:</td><td colspan="2">High</td></tr>
<tr><td>Source:</td><td colspan="2">EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">Admin</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">None</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event when an Admin edits an existing elephant’s details. From the elephant details window, the Admin selects “Edit”, updates the pre-filled form, and confirms the change. The system updates the corresponding records across the Animal and Elephant entities, and replaces the elephant’s photo if a new one was uploaded.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The Admin is authenticated and the elephant’s details window is open (see Use Case 10.2).</td></tr>
<tr><td>Trigger:</td><td colspan="3">The Admin selects “Edit” in the elephant details window.</td></tr>
<tr><td rowspan="2">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td></td><td>Step 1: Admin views an elephant’s details (Use Case 10.2) and selects “Edit”.</td><td></td><td>Step 2: The system closes the details window and opens an update form pre-filled with the elephant’s current Name, Species, Weight, Gender, Last Seen, Date of Birth, Approximate Location, Tusk Status, and Identification Notes, along with the current photo and an option to replace it.</td></tr>
</table>

	Step 3: Admin updates the fields and selects "Update". [ALT]		**Step 4:** The system validates the entered details, then displays a confirmation prompt: "Are you sure you want to update this Elephant?"
	Step 5: Admin selects "Confirm". [ALT]		**Step 6:** The ElephantService sends the updated form data to "PUT api/Elephant/{id}". The controller checks that the elephant exists and handles the optional photo upload. [ALT]
			Step 7: Entity Framework Core updates the elephant's TuskStatus on the Elephant entity, and its remaining details on the Animal entity. The photo is only replaced if a new file was uploaded; otherwise the existing photo is kept. Both updates are saved together. [ALT]
			Step 8: The API responds with an HTTP 204 status. The frontend refreshes the grid, closes the form, and displays a success notification reading "Successfully updated elephant".
Alternate courses:	[ALT] Step 3: Admin selects "Cancel". The form closes and no changes are made.		
	[ALT] Step 4: Validation fails. The system displays an inline error and keeps the form open.		
	[ALT] Step 5: Admin selects "Cancel" on the confirmation prompt. The prompt closes and the form data is kept.		
	[ALT] Step 6: The elephant cannot be found. The system displays a generic error message.		

	[ALT] Step 7: The API returns an HTTP 500 response due to a server error. The system displays “System error, please try again.”
	[ALT] Step 8: The API returns an HTTP 403 response because the Admin lacks permission. The system displays “You do not have permission to update elephants.”
Conclusion:	The use case concludes when the elephant’s updated details are saved and reflected in the grid.
Post-condition:	The elephant’s Animal and Elephant records are updated with the new details. The photo is only changed if a new file was uploaded.
Business rules:	• Only Admins can update elephant details. • The elephant’s Active Status cannot be changed through this use case (see the deactivate and reactivate use cases).
Implementation constraints and specifications:	• “PUT api/Elephant/{id}” accepts the updated form data, including an optional new photo, as a multipart request. • Field validation is currently performed only on the frontend.
Assumptions:	• None.
Open issues:	• None.

Table 72: Technical Narrative 10.4 Update Elephant

2.9.4.5 Use Case 10.5: Deactivate Elephant

System Name: Integrated NGO Management System			
Author:	Stephan Bredenhann	**Date:** 13 July 2026 **Version:** 1.0	
Use case name:	Deactivate Elephant	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	10.5		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	None		
Description:	This use case describes the event when an Admin deactivates an active elephant using a soft delete. From the elephant details window, the Admin selects "Deactivate" and confirms the action. The system sets the elephant's ActiveStatus to false on the Animal entity, while keeping the underlying record intact.		
Pre-condition:	The Admin is authenticated, the elephant's details window is open, and its ActiveStatus is currently true.		
Trigger:	The Admin selects "Deactivate" in the elephant details window.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: Admin views an active elephant's details (Use Case 10.2), where the details window shows a "Deactivate" button.		
	Step 2: Admin selects "Deactivate".		**Step 3:** The system displays a confirmation prompt: "Are you sure you want to deactivate this Elephant?"
	Step 4: Admin selects "Confirm". [ALT]		**Step 5:** The ElephantService sends a request to "PUT api/Elephant/{id}/deactivate". Entity Framework Core sets

			the linked Animal record's ActiveStatus to false, without changing any Elephant entity fields. [ALT]
			Step 6: The API responds with an HTTP 204 status. The frontend refreshes the grid, closes the form, and displays a success notification reading "Successfully deactivated elephant".
			Step 7: The deactivated elephant now shows an "Inactive" label in the grid, and its details window shows a "Reactivate" option instead of "Deactivate".
Alternate courses:	[ALT] Step 4: Admin selects "Cancel". The prompt closes and no changes are made.		
	[ALT] Step 5: The elephant cannot be found. The system displays a generic error message.		
	[ALT] Step 5: The API returns an HTTP 500 response due to a server error. The system does not display an error message to the Admin.		
Conclusion:	The use case concludes when the elephant's ActiveStatus is set to false and reflected in the grid.		
Post-condition:	The elephant's ActiveStatus is false, while all of its other details remain unchanged. The grid shows the elephant as "Inactive", and its details window offers a "Reactivate" option.		
Business rules:	• Only Admins can deactivate elephants. • Deactivation is a soft delete; the elephant record itself is never removed.		
Implementation constraints and specifications:	• "PUT api/Elephant/{id}/deactivate" requires no request body.		
Assumptions:	• None.		
Open issues:	• None.		

Table 73: Technical Narrative 10.5 Deactivate Elephant

2.4.10 Report Subsystem Technical Narratives

2.4.10.1 Use Case 12.1: Generate Custom Report

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Stephan Bredenhann</td><td colspan="2">Date: 13 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>Generate Custom Report</td><td colspan="2" rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>12.1</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">User (Authenticated)</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">Administrator</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event when a User builds and generates a custom report. The User selects a domain, chooses which fields to include, optionally applies a filter, and generates an on-screen preview. The report can then be downloaded as a CSV file, and the field selection can be saved as a named configuration for later reuse.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The User is authenticated and the selected domain contains existing records.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The User navigates to the Reports page and selects a domain to begin building a report.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: User navigates to the Reports page.</td><td></td><td>Step 2: The system calls “GET api/reports/domains”, which returns the list of reportable domains and the fields available for each one. The domains populate the selection dropdown. [ALT]</td></tr>
<tr><td>Step 3: User selects a domain from the dropdown.</td><td></td><td>Step 4: The system displays the available fields for that domain as a checkbox grid, with every field selected by default, and loads the</td></tr>
</table>

			user's saved configurations for the domain into the sidebar.
	Step 5: User deselects any unwanted fields and, if required, chooses a filter field and enters a filter value.		**Step 6:** The system records the field selection and filter. No request is made at this point.
	Step 7: User selects "Generate Preview".		**Step 8:** The system checks that a domain is selected and at least one field is chosen, then calls "POST api/reports/generate" with the domain, the selected fields, and any filter. [ALT]
			Step 9: The system retrieves the records for the selected domain, returns only the requested fields, and applies the filter if one was supplied. The results are displayed in a scrollable preview table using the field display names as column headings.
	Step 10: User selects "Download CSV".		**Step 11:** The system calls "POST api/reports/generate/download", which regenerates the report and returns it as a CSV file named after the domain and the current date and time. The browser saves the file and the system displays a success notification. [ALT]
	Step 12: User selects "Save Current Configuration".		**Step 13:** The system opens a window containing a name field, a summary of the selected domain and field count, and "Save" and "Cancel" buttons. The "Save" button remains disabled until a name is entered.
	Step 14: User enters a configuration name and selects "Save". [ALT]		**Step 15:** The system calls "POST api/reports/configurations", which inserts a new record into the ReportConfiguration entity holding the name, domain, and field selection against the user. The window closes, the saved

			configurations sidebar refreshes, and a success notification is displayed. [ALT]
Alternate courses:	[ALT] Step 2: The domains cannot be retrieved. The system displays “Failed to load report domains” and the dropdown remains empty.		
	[ALT] Step 8: The report cannot be generated, either because the domain is not recognised or because the database is unavailable. The system displays “Failed to generate report preview”.		
	[ALT] Step 11: The report cannot be downloaded. The system displays “Failed to download report” and no file is saved.		
	[ALT] Step 14: The configuration name is left blank. The system displays “Please enter a configuration name” and no request is made.		
	[ALT] Step 15: The configuration cannot be saved. The system displays “Failed to save configuration” and the window remains open.		
Conclusion:	The use case concludes when the report preview is displayed to the User and, if selected, the report is downloaded or the field selection is saved as a named configuration.		
Post-condition:	The report preview is displayed. If the report was downloaded, a CSV file is saved to the User’s device. If a configuration was saved, a new record exists in the ReportConfiguration entity holding the field selection.		
Business rules:	• Only authenticated users can generate reports. • At least one field must be selected before a report can be generated. • A configuration must be given a name before it can be saved.		
Implementation constraints and specifications:	• “POST api/reports/generate” accepts the domain, the selected fields, and an optional filter, and returns only the requested fields. • The CSV file is encoded so that it opens correctly in Microsoft Excel. • The preview table scrolls horizontally so that wide reports do not break the page layout.		
Assumptions:	• Each reportable domain has been configured with the fields available to it. • The User’s browser supports downloading files generated by the page.		
Open issues:	• None.		

Table 74: Technical Narrative 12.1 Generate Custom Report

2.4.10.2 Use Case 12.2: View Custom Report

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Stephan Bredenhann</td><td colspan="2">Date: 13 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>View Custom Report</td><td colspan="2" rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>12.2</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">User (Authenticated)</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">Administrator</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event when a User views previously saved report configurations. The User can browse all saved configurations, narrow them by domain, and review the name, domain, selected fields, and dates recorded against each one. A configuration can also be loaded into the Report Builder to display its report preview. Only configurations belonging to the User are shown, and no data is created, updated, or deleted.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The User is authenticated and has at least one saved report configuration.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The User navigates to the Saved Configurations page, or opens the saved configurations sidebar on the Report Builder page.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: User navigates to the Saved Configurations page.</td><td></td><td>Step 2: The system calls “GET api/reports/configurations”, which retrieves the configurations belonging to the User, ordered by the date they were last updated. [ALT]</td></tr>
<tr><td></td><td></td><td>Step 3: The system displays each configuration as a card showing its name, domain, the fields it includes, and the dates it was created and last updated, together with “Load”, “Edit”, and “Delete” buttons. [ALT]</td></tr>
</table>

	Step 4: User selects a domain from the filter dropdown.		**Step 5:** The system narrows the list to configurations belonging to the selected domain. Selecting "All Domains" restores the full list.
	Step 6: User selects "Load" on a configuration card.		**Step 7:** The system calls "GET api/reports/configurations/{id}", which retrieves the configuration only if it belongs to the User. The system opens the Report Builder page with the saved domain and field selection restored. [ALT]
			Step 8: The system generates the report preview from the restored field selection, as described in Use Case 12.1, and displays the results in the preview table.
Alternate courses:	[ALT] Step 3: The User has no saved configurations. The system displays a message inviting the User to create a report.		
	[ALT] Step 2: The User is not authenticated or the session has expired. The system redirects the User to the login page.		
	[ALT] Step 7: The configuration cannot be found because it was deleted or belongs to another user. The system displays "Failed to load configuration".		
	[ALT] Step 7: The configuration refers to a domain that no longer exists. The system displays "Domain not found for this configuration" and no report is generated.		
Conclusion:	The use case concludes when the saved configurations are displayed to the User, with the selected configuration loaded into the Report Builder if requested.		
Post-condition:	No data is modified. If a configuration was loaded, the Report Builder displays the report preview generated from its field selection.		
Business rules:	• Only authenticated users can view report configurations. • Configurations that have been removed are excluded from the list. • Configurations are listed with the most recently updated first.		
Implementation constraints and specifications:	• Domain and field labels are resolved from the stored report definitions, so no further request is needed to display them.		
Assumptions:	• The reportable domains and their fields remain consistent with the entities they describe.		

	• A configuration saved against a domain remains valid while that domain exists.
Open issues:	• None.

Table 75: Technical Narrative 12.2 View Custom Report

2.4.10.3 Use Case 12.3: Update Custom Report

System Name: Integrated NGO Management System			
Author:	Stephan Bredenhann	**Date:** 13 July 2026 **Version:** 1.0	
Use case name:	Update Custom Report	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	12.3		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	User (Authenticated)		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Administrator		
Description:	This use case describes the event when a User updates a saved report configuration. From the Saved Configurations page the User renames a configuration, and the system updates the record only after confirming the configuration belongs to the User. A User who instead wants to change which fields a report includes loads the configuration into the Report Builder, adjusts the field selection, and saves it as a new configuration.		
Pre-condition:	The User is authenticated and has at least one saved report configuration.		
Trigger:	The User selects “Edit” on a configuration card on the Saved Configurations page.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: User navigates to the Saved Configurations page.		**Step 2:** The system displays the User’s saved configurations as cards, each showing the configuration name, domain, and selected fields, with “Load”, “Edit”, and “Delete” buttons.
	Step 3: User selects “Edit” on a configuration card.		**Step 4:** The system opens an edit window containing a name field pre-filled with the current configuration name, together with “Save” and “Cancel” buttons. The “Save” button

			remains disabled while the name is blank.
	Step 5: User changes the configuration name and selects "Save". [ALT]		**Step 6:** The system calls "PUT api/reports/configurations/{id}", which retrieves the configuration only if it belongs to the User and updates the name and the last updated date. The domain and field selection are left unchanged. [ALT]
			Step 7: The system closes the edit window, refreshes the configurations list with the new name, and displays a success notification.
	Step 8: User selects "Load" on a configuration to change the fields it includes.		**Step 9:** The system restores the saved domain and field selection in the Report Builder and generates the report preview, as described in Use Case 12.2.
	Step 10: User adds or removes fields and selects "Save Current Configuration".		**Step 11:** The system updates the preview and the selected field count as each field is changed, then saves the revised selection as a new configuration, as described in Use Case 12.1. [ALT]
Alternate courses:	[ALT] Step 5: User selects "Cancel" or closes the edit window. No changes are made and no request is sent.		
	[ALT] Step 6: The configuration cannot be found because it was removed or belongs to another user. The system displays "Configuration not found" and the edit window remains open.		
	[ALT] Step 6: The User is not authenticated or the session has expired. The system redirects the User to the login page.		
	[ALT] Step 6: The configuration cannot be updated. The system displays "Failed to update configuration" and the edit window remains open.		
	[ALT] Step 11: The name entered matches an existing configuration. The system saves it regardless, so two configurations with the same name can exist for the same domain.		
Conclusion:	The use case concludes when the configuration is renamed and the new name appears in the list, or when a revised field selection is saved as a new configuration.		

Post-condition:	The selected configuration has an updated name and last updated date. If the field selection was changed instead, a new configuration exists holding the revised fields.
Business rules:	• Only authenticated users can update report configurations. • The last updated date is set every time a configuration is changed. • Renaming a configuration does not change the fields it includes.
Implementation constraints and specifications:	• "PUT api/reports/configurations/{id}" updates the name and the last updated date only; the domain cannot be changed once a configuration has been saved.
Assumptions:	• The User can identify which configuration to update from its name, domain, and listed fields. • Configuration names do not need to be unique within a domain.
Open issues:	• None.

Table 76: Technical Narrative 12.3 Update Custom Report

2.4.10.4 Use Case 12.4: Remove Custom Report

System Name: Integrated NGO Management System			
Author:	Stephan Bredenhann	**Date:** 13 July 2026 **Version:** 1.0	
Use case name:	Remove Custom Report	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	12.4		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	User (Authenticated)		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Administrator		
Description:	This use case describes the event when a User removes a saved report configuration. The configuration is marked as inactive rather than being deleted from the database, so the record is retained for audit purposes while no longer appearing to the User. Removal can be started from the Saved Configurations page or from the Report Builder sidebar, and both paths require confirmation before the configuration is removed.		
Pre-condition:	The User is authenticated and has at least one saved report configuration.		
Trigger:	The User selects “Delete” on a configuration card on the Saved Configurations page, or on a configuration in the Report Builder sidebar.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: User navigates to the Saved Configurations page.		**Step 2:** The system displays the User’s saved configurations as cards, each showing the configuration name, domain, and selected fields, with “Load”, “Edit”, and “Delete” buttons.
	Step 3: User selects “Delete” on a configuration card. [ALT]		**Step 4:** The system displays a confirmation prompt reading “Are you sure you want to delete the configuration?”, naming the selected configuration, with “Delete” and “Cancel” buttons.

	Step 5: User selects "Delete" to confirm. [ALT]		**Step 6:** The system calls "DELETE api/reports/configurations/{id}", which retrieves the configuration only if it is active and belongs to the User. [ALT]
			Step 7: The system marks the configuration as inactive rather than removing the record, leaving all other details intact. [ALT]
			Step 8: The system refreshes the configurations list, which no longer includes the removed configuration, and displays a success notification.
Alternate courses:	[ALT] Step 3: User selects "Delete" on a configuration in the Report Builder sidebar instead. The system displays a confirmation prompt and, once confirmed, follows the same removal process from Step 6.		
	[ALT] Step 5: User selects "Cancel" or closes the confirmation prompt. The configuration is not removed and no request is sent.		
	[ALT] Step 6: The configuration cannot be found because it was already removed or belongs to another user. The system displays "Configuration not found" and refreshes the list.		
	[ALT] Step 6: The User is not authenticated or the session has expired. The system redirects the User to the login page.		
	[ALT] Step 7: The configuration cannot be updated. The system displays "Failed to delete configuration" and the configuration remains in the list.		
Conclusion:	The use case concludes when the configuration is marked as inactive, no longer appears in the configurations list, and a success notification is displayed.		
Post-condition:	The selected configuration is marked as inactive and is no longer returned to the User. The record is retained with its name, domain, field selection, and dates intact.		
Business rules:	• Only authenticated users can remove report configurations. • Removal is not destructive; the record is retained and marked as inactive. • Only active configurations can be removed. • The User must confirm the removal before it is carried out.		
Implementation constraints and specifications:	• "DELETE api/reports/configurations/{id}" marks the configuration as inactive rather than deleting the record, and no other details are changed. • A configuration that has already been removed returns the same result as one that does not exist.		
Assumptions:	• The confirmation prompt is sufficient to prevent accidental removals.		

Open issues:	• None.

Table 77: Technical Narrative 12.4 Remove Custom Report

2.4.11 Budget Subsystem Technical Narratives

2.4.11.1 Use Case: 14.1 View Project Budget

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Alexander Steven</td><td colspan="2">Date: 15 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>View Project Budget</td><td colspan="2" rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>14.1</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">User (Authenticated)</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">None</td></tr>
<tr><td>Description:</td><td colspan="3">A user with the appropriate permissions should be able to view a project budget. The system will allow the user to select and view a budget’s details including expenses. The user clicks on a specific project from the main budget dashboard. The Angular frontend sends a request to the .NET 8 API to fetch the financial data. The backend queries the "Project" and "Expense" database tables, calculates the total amount spent, and groups the data so it can be drawn as charts. The frontend then displays a clean dashboard showing a proportional breakdown (pie chart), a cumulative timeline (line chart), and a full transaction history table.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The user is logged in, has the correct permissions to view financial data, and is on the main Budget screen.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The user clicks on a specific project card or table row on the Budget dashboard.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The user clicks on a specific project to view its budget details.</td><td></td><td>Step 2: The Angular frontend navigates to the detailed view and sends a GET api/Expenses/project/{projectId} request to the backend.</td></tr>
<tr><td></td><td></td><td>Step 3: The API controller receives the request and uses Entity Framework Core to verify the</td></tr>
</table>

			"Project" exists in the database. [ALT]
			Step 4: The system queries the "Expense" table for all records linked to that specific project ID, ordered by the date they were incurred. [ALT]
			Step 5: The backend calculates the total spent, groups the expenses by category for the pie chart, and calls a helper function to generate the cumulative timeline data for the line chart.
			Step 6: The API packages this formatted data into a "ProjectBudgetDetailDto" and returns an HTTP 200 OK success response.
			Step 7: The Angular frontend receives the data and renders the visual elements: a "Total Logged" number, a "Proportional Breakdown" pie chart, an "Expense Timeline" line graph, and the "Transaction History" table.
Alternate courses:	[ALT] Step 3: The requested project ID cannot be found in the database. The API returns an HTTP 404 Not Found response. The frontend redirects the user back to the main dashboard and displays a red error toast: "Project budget not found."		
	[ALT] Step 4: The database connection drops while trying to retrieve the expenses. The API returns an HTTP 500 Internal Server Error. The frontend leaves the charts empty and displays a red error toast: "Failed to load financial data."		
Conclusion:	The user successfully views a detailed, visual breakdown of the project's financial expenses.		
Post-condition:	The financial data is displayed on the screen. No database records are modified during this view process.		
Business rules:	Financial data is privileged; only users assigned to the project or with administrative roles may view it.		
Implementation constraints and specifications:	• The backend controller formats the chart data (labels, values, and hex colors) before sending it to the frontend, which keeps the Angular code lightweight.		

Assumptions:	• The frontend charting library can handle dynamically plotting the data slices sent by the backend.
Open issues:	• While the user interface shows a "Transaction History" table, the ability to edit or delete existing expenses directly from this table may need to be mapped out in future use cases.

Table 78: Technical Narrative 14.1 View Project Budget

2.4.11.2 Use Case: 14.2 View Project Budgets

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Alexander Steven</td><td colspan="2">Date: 15 July 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>View Project Budgets</td><td colspan="2" rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>Use case id:</td><td>14.2</td></tr>
<tr><td>Priority:</td><td>High</td></tr>
<tr><td>Source:</td><td>EPI-USE</td></tr>
<tr><td>Primary business actor (PBA):</td><td colspan="3">System Admin (Authenticated)</td></tr>
<tr><td>Primary system actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other participating actors:</td><td colspan="3">• None</td></tr>
<tr><td>Other interested stakeholders:</td><td colspan="3">None</td></tr>
<tr><td>Description:</td><td colspan="3">A user with the appropriate permissions should be able to view all project budgets. The system will retrieve all project budgets on the system and display them to the user. The user navigates to the "Budget" tab, which triggers the Angular frontend to request the dashboard data from the .NET 8 API. The backend retrieves the user's assigned projects, fetches all related expenses across the company, and calculates global totals. It packages the proportions, cumulative timelines, and table data into a "DashboardSummaryDto", which is then dynamically rendered on the frontend as interactive charts and a comprehensive project list.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The user is logged into the system and has the correct role permissions to view the financial dashboard.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The user clicks the "Budget" tab in the main navigation menu.</td></tr>
<tr><td rowspan="3">Typical Course of Events:</td><td rowspan="2">Actor Action</td><td colspan="2">System Response</td></tr>
<tr><td>Manual Action</td><td>Automated Action</td></tr>
<tr><td>Step 1: The user clicks the "Budget" tab on the navigation bar.</td><td></td><td>Step 2: The Angular frontend detects the navigation and sends a GET api/Expenses/dashboard/{userId} request to the backend.</td></tr>
<tr><td></td><td></td><td>Step 3: The API controller receives the request and uses Entity Framework Core to fetch the user's</td></tr>
</table>

			project assignments from the "UserProjects" table. [ALT]
			Step 4: The system queries the "Expenses" table to get all financial logs, and the "Projects" table to get active project details.
			Step 5: The backend calculates the grand total, groups the expenses into a "BigChart" (pie chart) and a "BigTimeline" (line chart), and formats the data for the recent projects cards and the "TableProjects" list.
			Step 6: The API returns an HTTP 200 OK response containing the fully populated "DashboardSummaryDto" JSON object.
			Step 7: The Angular frontend parses the data and renders the overall Budget & Expenses dashboard, including the interactive charts and the "All Active Projects" data table.
Alternate courses:	[ALT] Step 3: The database connection fails while trying to fetch the user's projects or expenses. The API returns an HTTP 500 Internal Server Error. The frontend stops rendering the charts and displays a red error toast: "Failed to load financial data."		
Conclusion:	The user is presented with a comprehensive, visual overview of all project budgets they are authorized to see.		
Post-condition:	The aggregated financial data is displayed. No database records are altered.		
Business rules:	Financial data is privileged, so the dashboard must only display project data that the user's role explicitly permits them to view.		
Implementation constraints and specifications:	• The backend controller assigns distinct colors from a predefined "Palette" array to ensure the pie charts and line charts remain visually consistent across the dashboard.		
Assumptions:	• The frontend charting libraries can dynamically bind to the nested arrays (like List<ChartSliceDto>) provided in the response payload.		
Open issues:	• Fetching all expenses across all projects simultaneously might cause performance bottlenecks as the database grows over the years		

Table 79: Technical Narrative 14.2 View Project Budgets

2.4.11.3 Use Case: 14.3 Log An Expense

System Name: Integrated NGO Management System			
Author:	Alexander Steven	**Date:** 15 July 2026 **Version:** 1.0	
Use case name:	Log An Expense	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	14.3		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	User (Authenticated)		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	None		
Description:	A user with the appropriate role can log a new expense for a specific project. The user clicks the "+ Log Expense" button on the project's budget dashboard, opening a modal to capture the amount (ZAR), category, and an optional receipt file. The Angular frontend packages this into an "AddExpenseDto" and sends a POST request to the .NET 8 API. The backend verifies that the user is an active member of the project, creates the "Expense" record (storing a snapshot of the user's name at that exact moment), and saves it to the database.		
Pre-condition:	The user is authenticated and navigating the Budget dashboard for a specific project.		
Trigger:	The user clicks the "+ Log Expense" button.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user clicks the "+ Log Expense" button.		**Step 2:** The Angular frontend opens the "Log New Expense" modal.
	Step 3: The user enters the amount, selects a category, optionally uploads a receipt, and clicks "Save Expense". [ALT]		**Step 4:** The Angular frontend packages the "AddExpenseDto" and sends a POST api/Expenses request to the backend.
			Step 5: The API controller receives the request and

			queries the "Users" and "UserProjects" database tables to ensure the user exists and is assigned to the project. [ALT]
			Step 6: The system initializes a new "Expense" entity, setting the "LoggedByName" snapshot and mapping the DTO fields.
			Step 7: The system adds the record and calls SaveChangesAsync().[ALT]
			Step 8: The API returns an HTTP 200 OK success response.
			Step 9: The Angular frontend closes the modal, refreshes the budget charts to reflect the new total, and displays a success toast.
Alternate courses:	[ALT] Step 3: The user clicks "Cancel". The modal closes and no data is submitted.		
	[ALT] Step 5: The user is not assigned to the project. The API returns an HTTP 403 Forbidden response. The frontend displays a red error toast: "You are not authorized to log expenses for this project."		
	[ALT] Step 7: The database connection fails. The API returns an HTTP 500 Internal Server Error, and the frontend displays a red error toast.		
Conclusion:	The financial transaction is successfully logged and instantly reflected on the project's financial dashboard.		
Post-condition:	A new record is inserted into the "Expenses" table.		
Business rules:	Users can only log expenses for projects they are actively participating in.		
Implementation constraints and specifications:	• The backend captures the "LoggedByName" string permanently so historical records remain accurate even if a user deletes their account later.		
Assumptions:	• The frontend handles the file upload for the receipt beforehand and passes the resulting "ReceiptUrl" in the DTO.		
Open issues:	• None		

Table 80: Technical Narrative 14.3 Log An Expense

2.4.11.4 Use Case: 14.4 Update An Expense

System Name: Integrated NGO Management System			
Author:	Alexander Steven	**Date:** 15 July 2026 **Version:** 1.0	
Use case name:	Update An Expense	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	14.4		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	User (Authenticated)		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	None		
Description:	A user with the appropriate permissions should be able to edit the details of an expense. The system will display the expense details in a modal, allowing the user to modify the amount, category, or update the receipt URL. Once the user clicks "Save", the Angular frontend validates the input and sends a PUT request to the .NET 8 API. The backend queries Entity Framework Core to verify the expense exists, applies the modifications to the database record, and responds with a success status.		
Pre-condition:	The user is authenticated, navigating the budget dashboard, and an expense has already been logged.		
Trigger:	The user clicks the "Edit" action button on a specific expense in the Transaction History table.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user clicks the "Edit" button on a specific expense row.		**Step 2:** The Angular frontend opens the Edit Expense modal, pre-populated with the current amount and category.
	Step 3: The user modifies the details and clicks "Save". [ALT]		**Step 4:** The Angular frontend validates the data and sends a PUT api/Expenses/{id} request with the updated DTO to the backend.

			Step 5: The API controller receives the request and uses FindAsync() to locate the "Expense" record in the database. [ALT]
			Step 6: The system overwrites the relevant properties (AmountZAR, Category) and calls SaveChangesAsync().[ALT]
			Step 7: The API returns an HTTP 204 No Content response.
			Step 8: The Angular frontend closes the modal, refreshes the budget timeline and pie charts, and displays a green success toast.
Alternate courses:	[ALT] Step 3: The user clicks "Cancel". The modal closes and no changes are made.		
	[ALT] Step 5: The expense cannot be found in the database. The API returns an HTTP 404 Not Found response. The frontend displays a red error toast: "Expense not found."		
	[ALT] Step 6: The database update fails. The API returns an HTTP 500 Internal Server Error, and the frontend displays a red error toast.		
Conclusion:	The expense details are corrected, and the financial graphs update automatically to reflect the accurate amounts.		
Post-condition:	The "Expense" record is modified in the database.		
Business rules:	Users can only edit expenses they logged themselves, or must possess administrative permissions.		
Implementation constraints and specifications:	• The "LoggedByName" property is treated as an immutable snapshot and cannot be modified during this update process.		
Assumptions:	• The frontend automatically performs basic validation (e.g., preventing negative currency amounts) before sending the request.		
Open issues:	• Loading time for the data is slow		

Table 81: Technical Narrative 14.4 Update An Expense

2.4.11.5 Use Case: 14.5 View Expense

System Name: Integrated NGO Management System			
Author:	Alexander Steven	**Date:** 15 July 2026 **Version:** 1.0	
Use case name:	View Expense	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	14.5		
Priority:	Medium		
Source:	EPI-USE		
Primary business actor (PBA):	User (Authenticated)		
Primary system actor (PSA):	• System		
Other participating actors:	• None		
Other interested stakeholders:	None		
Description:	A user with appropriate permissions can view the full details of a specific logged expense. When the user clicks on a row in the Transaction History table or clicks "View Receipt", the system opens a detailed modal. Since the expense data was already passed to the frontend in the "ProjectBudgetDetailDto" payload during dashboard load, the Angular client seamlessly renders the details (Date, Category, Amount, Logged By, and the attached receipt image) without making an additional database query.		
Pre-condition:	The user is authenticated, navigating the budget dashboard, and expenses exist in the Transaction History table.		
Trigger:	The user clicks a specific expense row or the "View Receipt" button in the table.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user clicks "View Receipt" on a specific transaction row.		**Step 2:** The Angular frontend intercepts the click and retrieves the specific "ExpenseItemDto" from its local memory state.
			Step 3: The system launches a detailed view modal and populates it with the expense data, rendering the image

			from the "ReceiptUrl". [ALT]
	Step 4: The user reviews the details and clicks "Close".		**Step 5:** The Angular frontend closes the modal.
Alternate courses:	[ALT] Step 3: The "ReceiptUrl" is empty or the image file cannot be fetched from the server. The frontend displays a placeholder icon and the text "No receipt uploaded."		
Conclusion:	The user successfully views the full details and physical receipt image for a transaction.		
Post-condition:	The modal is closed and the user returns to the main budget dashboard.		
Business rules:	Users can only view expense receipts for projects they have permission to access.		
Implementation constraints and specifications:	• Because the data is already contained in the "ProjectBudgetDetailDto" from Use Case 15.1, this action does not trigger a separate GET request to the .NET API, heavily optimizing server load.		
Assumptions:	• The browser has access to load the image path stored in the "ReceiptUrl".		
Open issues:	• None		

Table 82: Technical Narrative 14.5 View Expense

2.4.11.6 Use Case: 14.6 Remove Expense

System Name: Integrated NGO Management System			
Author:	Alexander Steven	**Date:** 15 July 2026 **Version:** 1.0	
Use case name:	Remove Expense	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
Use case id:	14.6		
Priority:	Medium		
Source:	EPI-USE		
Primary business actor (PBA):	user (Authenticated)		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	None		
Description:	A user with appropriate permissions can permanently delete an expense from the project budget. The user clicks the "Delete" button on a transaction row. The Angular frontend displays a confirmation prompt. Upon confirmation, the system sends a DELETE api/Expenses/{id} request to the .NET 8 API. The backend uses Entity Framework Core to locate the "Expense" record, removes the associated receipt file from the server (if applicable), deletes the database record, and saves the changes. The financial charts are then updated to reflect the removed amount.		
Pre-condition:	The user is authenticated, viewing the budget dashboard, and an expense exists.		
Trigger:	The user clicks the "Delete" icon/button on an expense row.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user clicks the "Delete" button on a specific expense.		**Step 2:** The Angular frontend pauses the action and displays a confirmation modal: "Are you sure you want to delete this expense?"
	Step 3: The user clicks "Confirm". [ALT]		**Step 4:** The Angular frontend sends a DELETE api/Expenses/{id} request to the backend.
			Step 5: The API controller receives the request and

			locates the "Expense" record in the database using FindAsync(). [ALT]
			Step 6: The system checks if a "ReceiptUrl" exists. If so, it locates the physical file on the server and executes System.IO.File.Delete() to remove it.
			Step 7: The system calls _context.Expenses.Remove() and SaveChangesAsync() to commit the database deletion. [ALT]
			Step 8: The API returns an HTTP 204 No Content response.
			Step 9: The Angular frontend removes the row from the table, recalculates the budget charts dynamically, and displays a green success toast.
Alternate courses:	[ALT] Step 3: The user clicks "Cancel". The modal closes and the deletion is aborted.		
	[ALT] Step 5: The expense cannot be found in the database. The API returns an HTTP 404 Not Found response. The frontend displays a red error toast.		
	[ALT] Step 7: The database connection fails during saving. The API returns an HTTP 500 Internal Server Error, and the frontend displays a red error toast: "Failed to delete expense."		
Conclusion:	The erroneous expense is permanently erased, restoring accuracy to the project's financial budget.		
Post-condition:	The "Expense" database record is destroyed, and the physical receipt file (if any) is deleted.		
Business rules:	Users can only delete expenses they created themselves, unless they are a System Admin.		
Implementation constraints and specifications:	• The physical receipt file must be deleted before the database record is removed to prevent orphaned files consuming storage space.		
Assumptions:	• The backend API is programmed with a DELETE endpoint in the ExpensesController that mirrors the standard architecture.		
Open issues:	• None		

Table 83: Technical Narrative 14.6 Remove Expense

2.5 Technical Context Diagram

2.5.1 User Subsystem Context Diagram



Figure 109: User Technical Context Diagram

2.5.2 Admin Subsystem Context Diagram

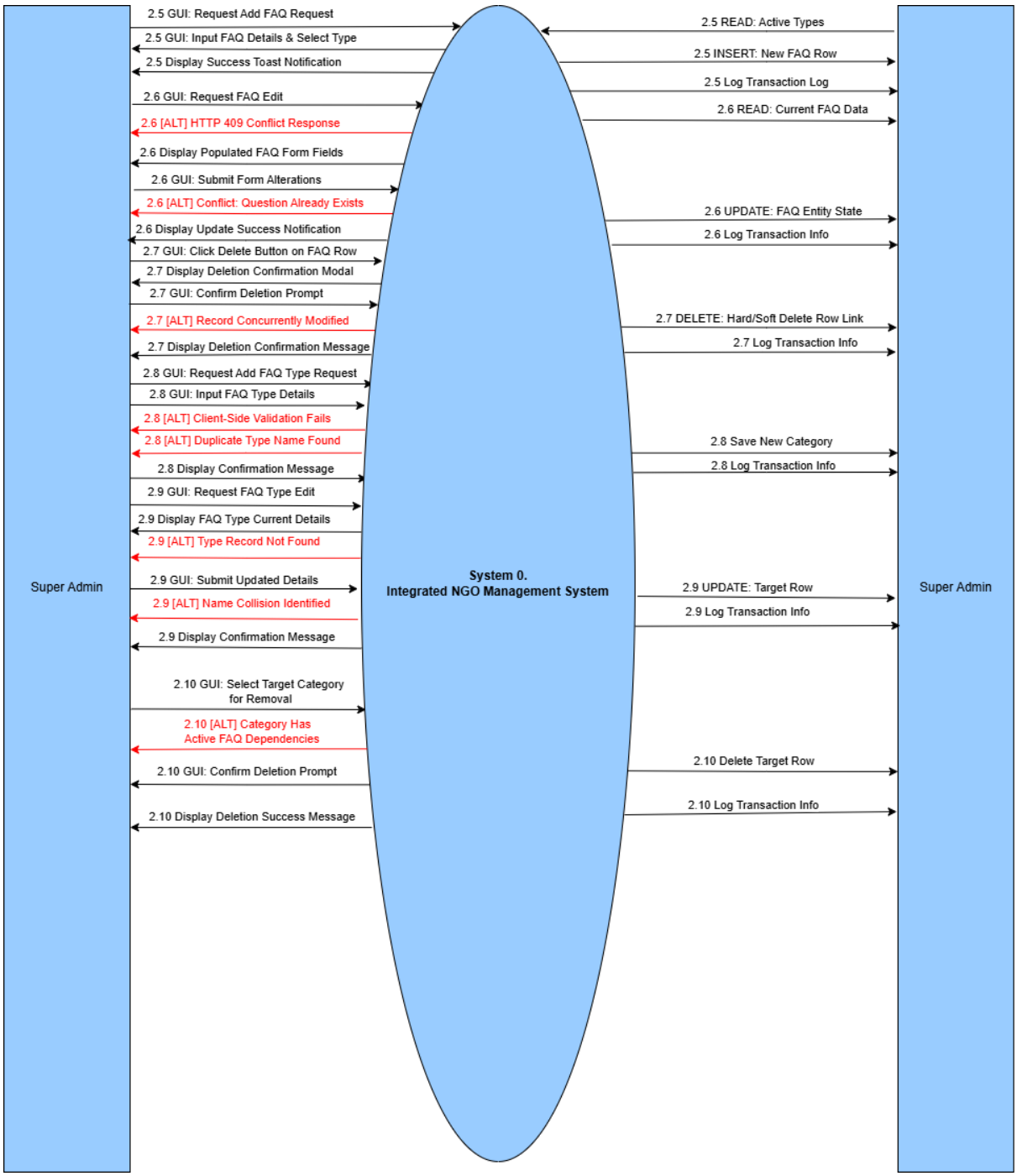


Figure 110: Admin Technical Context Diagram

2.5.3 Inventory Subsystem Context Diagram

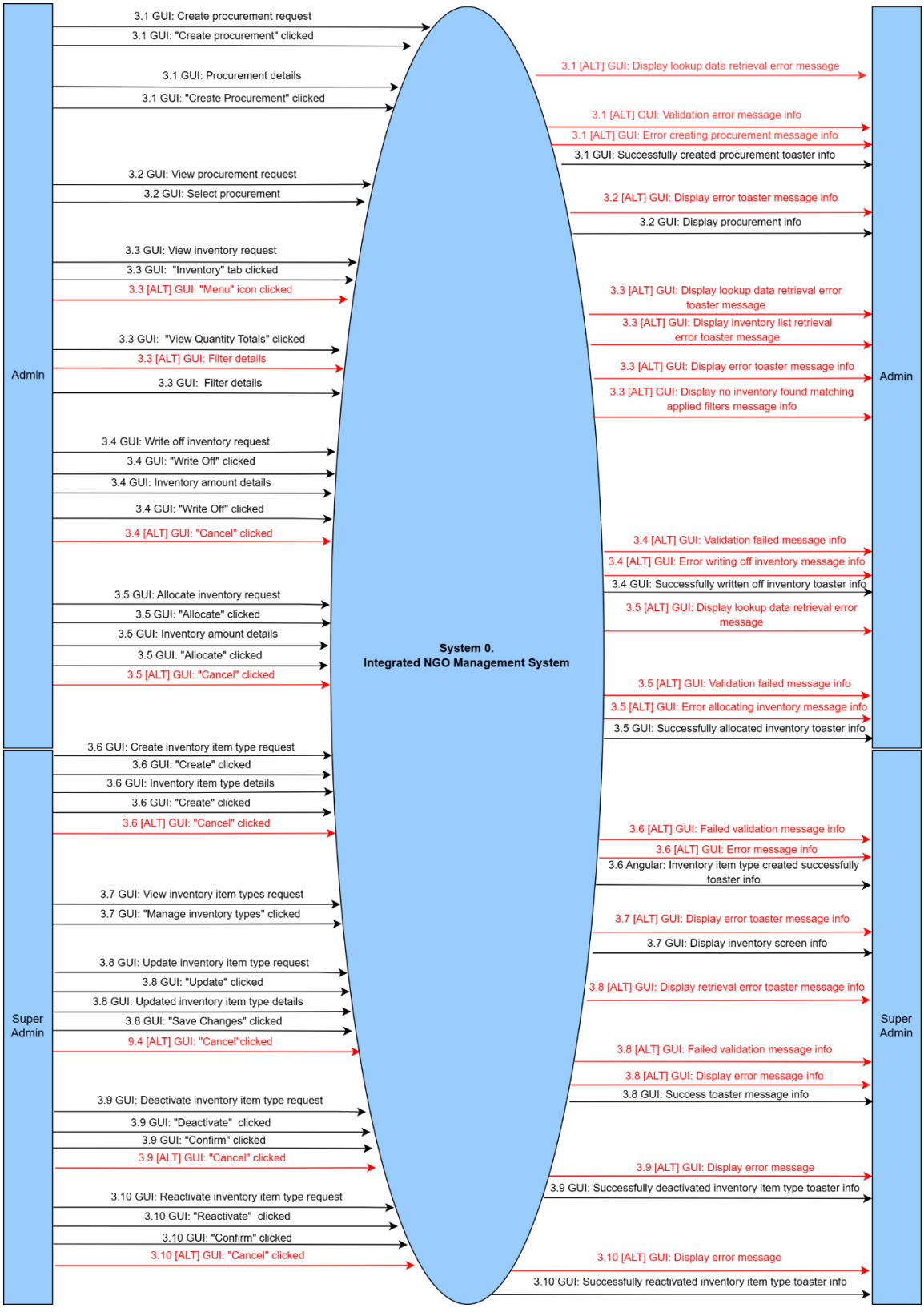


Figure 111: Inventory Technical Context Diagram

2.5.4 Warehouse Subsystem Context Diagram

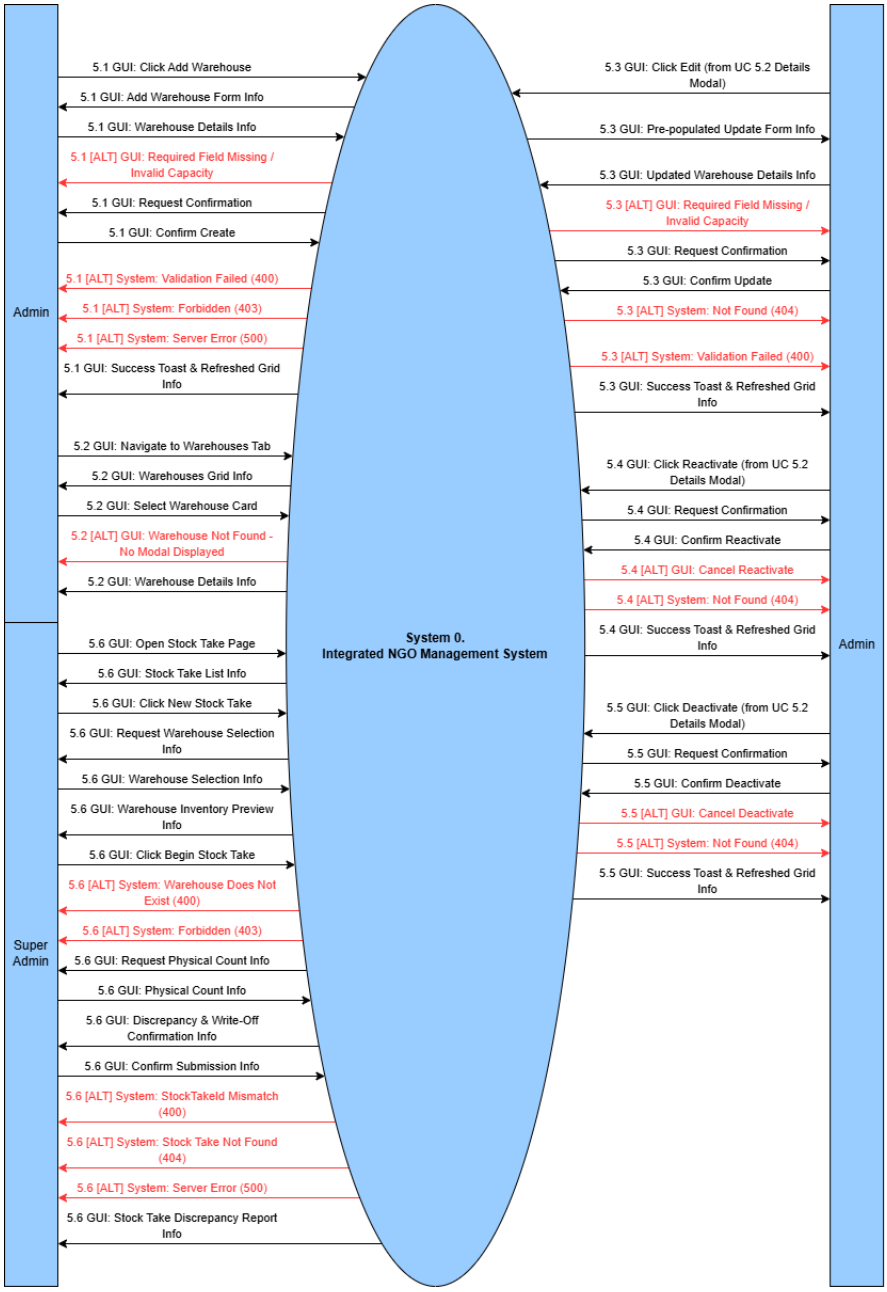


Figure 112: Warehouse Technical Context Diagram

2.5.5 Vendor Subsystem Context Diagram

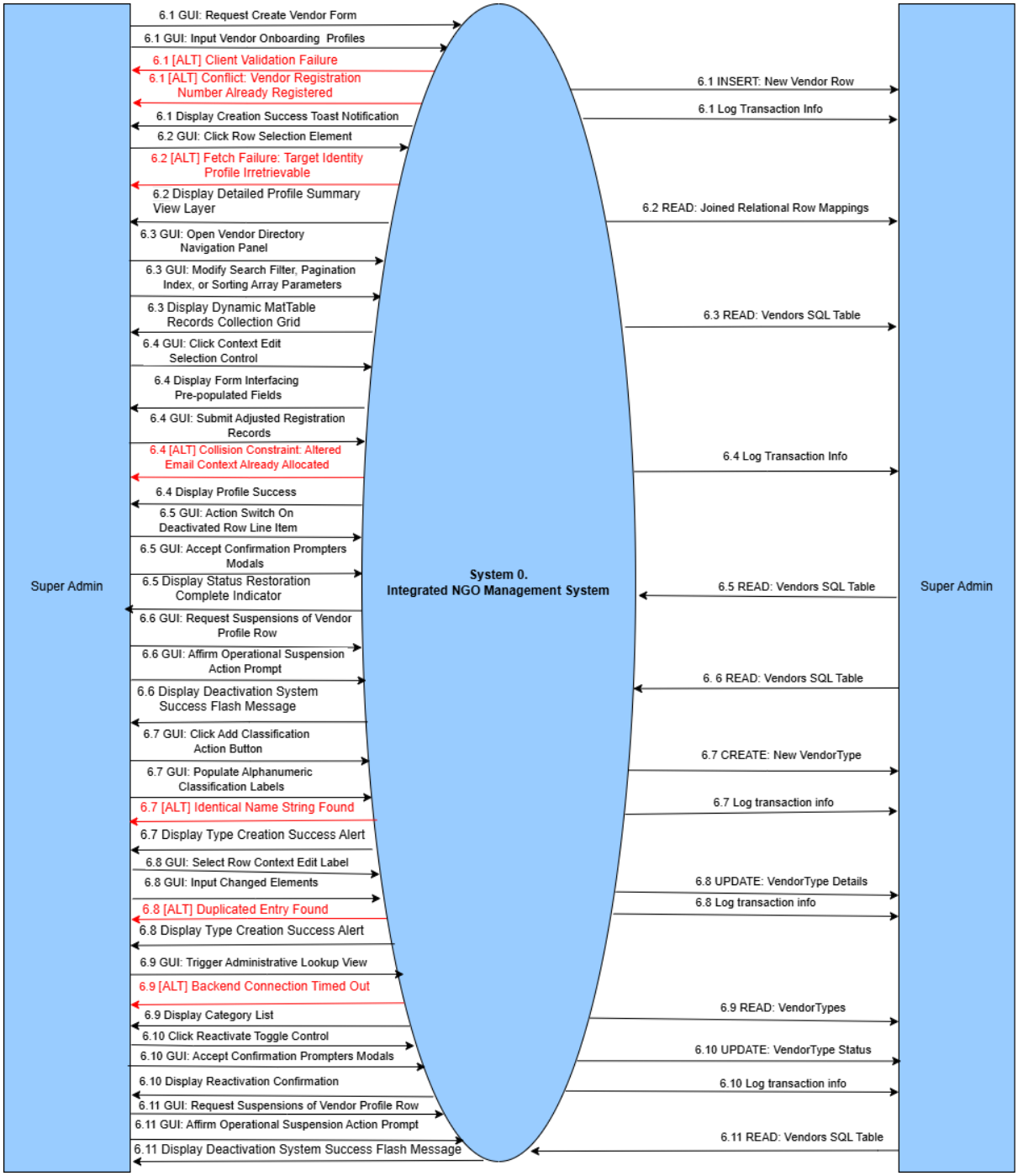


Figure 113: Vendor Technical Context Diagram

2.5.6 Project Management Context Diagram

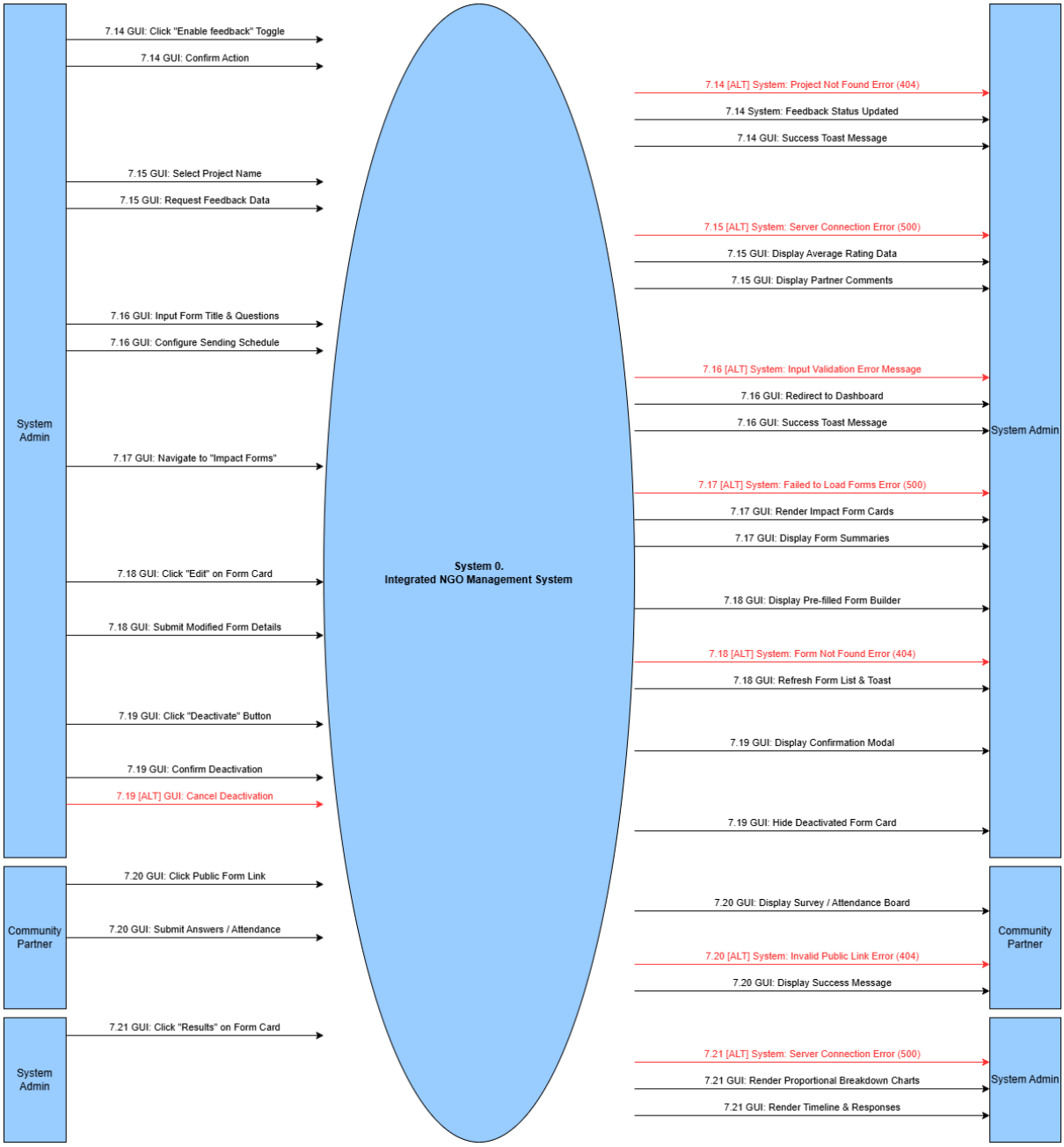


Figure 114: Project Management Technical Context Diagram

2.5.7 K9 Subsystem Context Diagram

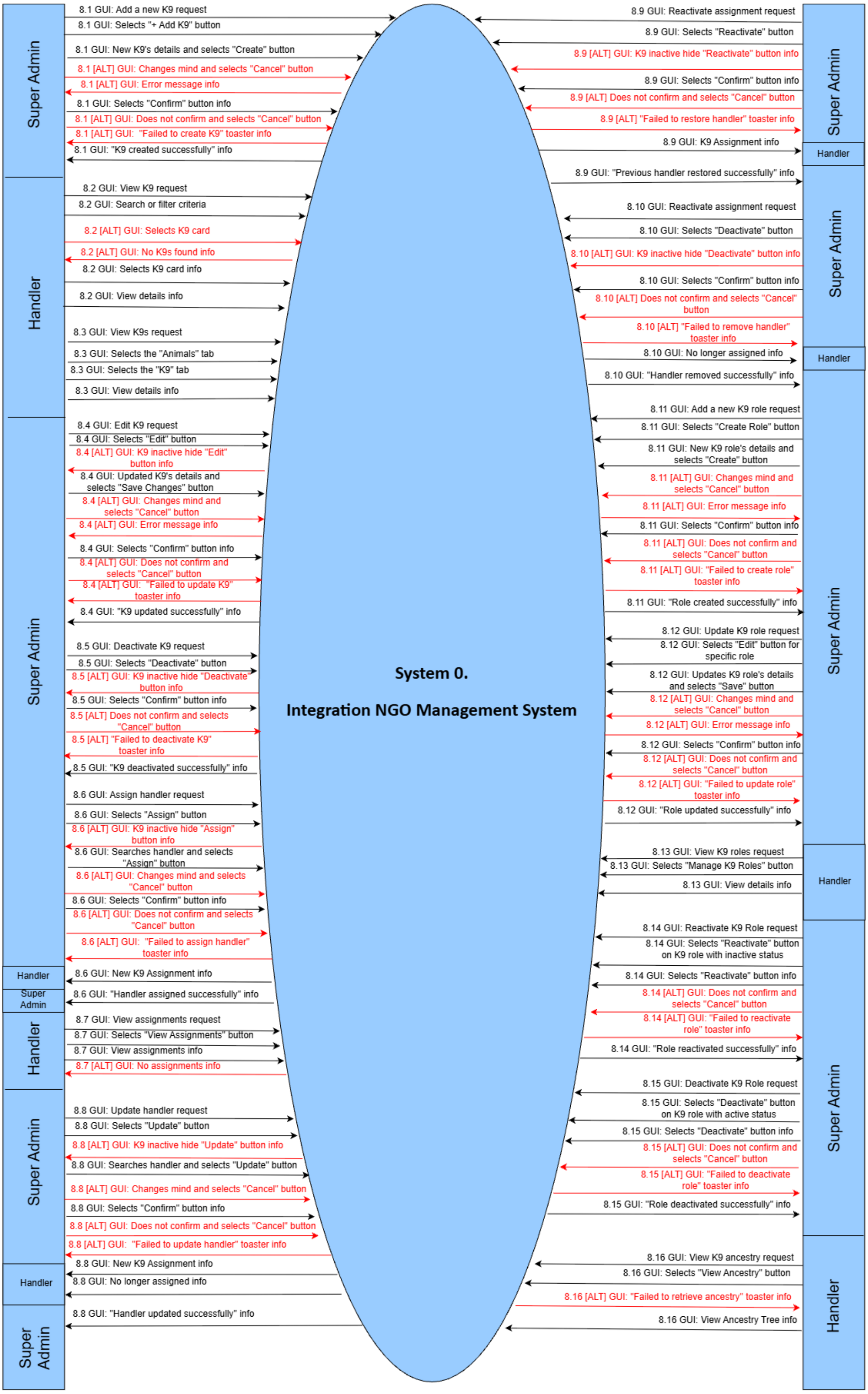


Figure 115: K9 Technical Context Diagram

2.5.8 Rhino Subsystem Context Diagram

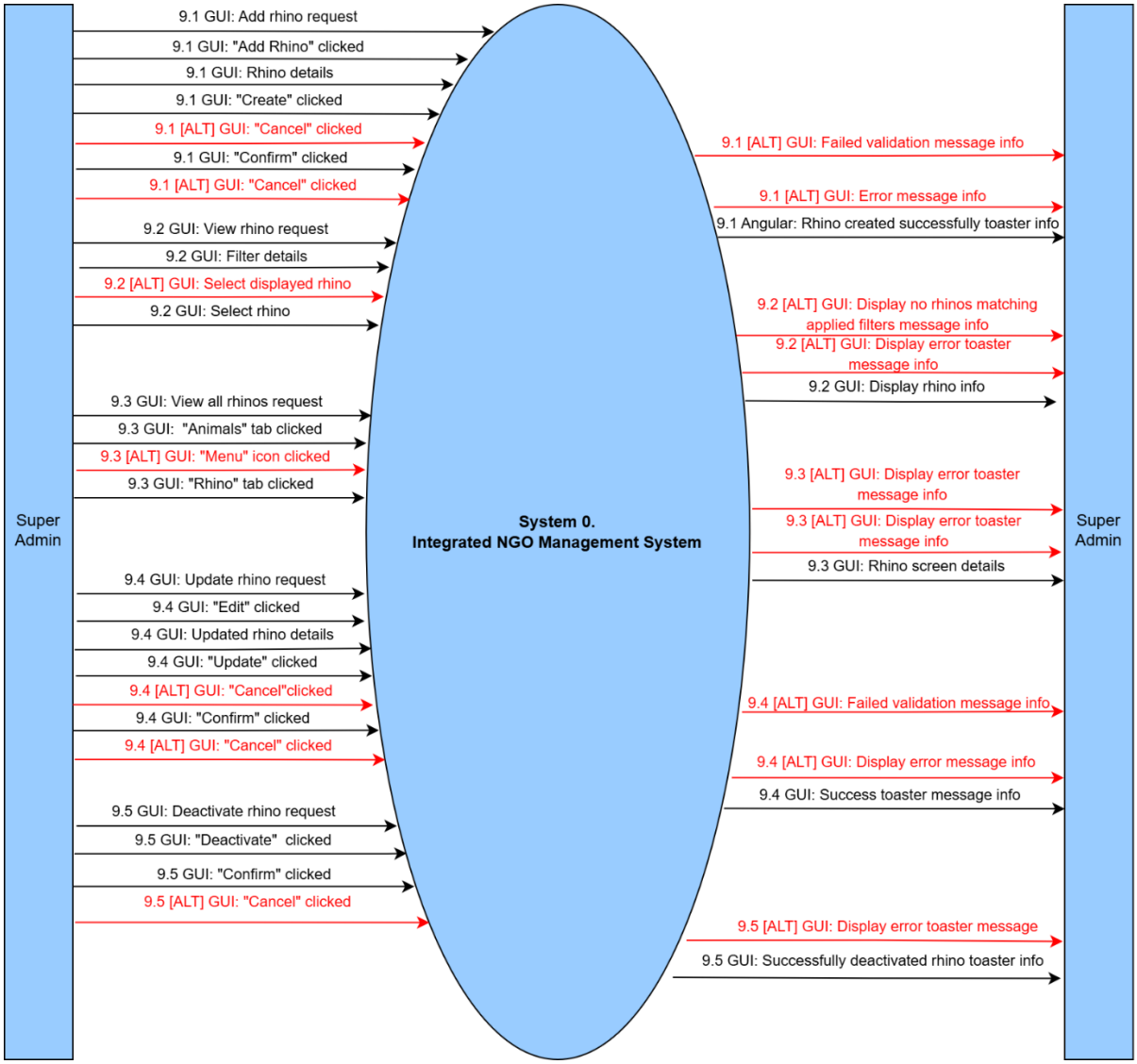


Figure 116: Rhino Technical Context Diagram

2.5.9 Elephant Subsystem Context Diagram

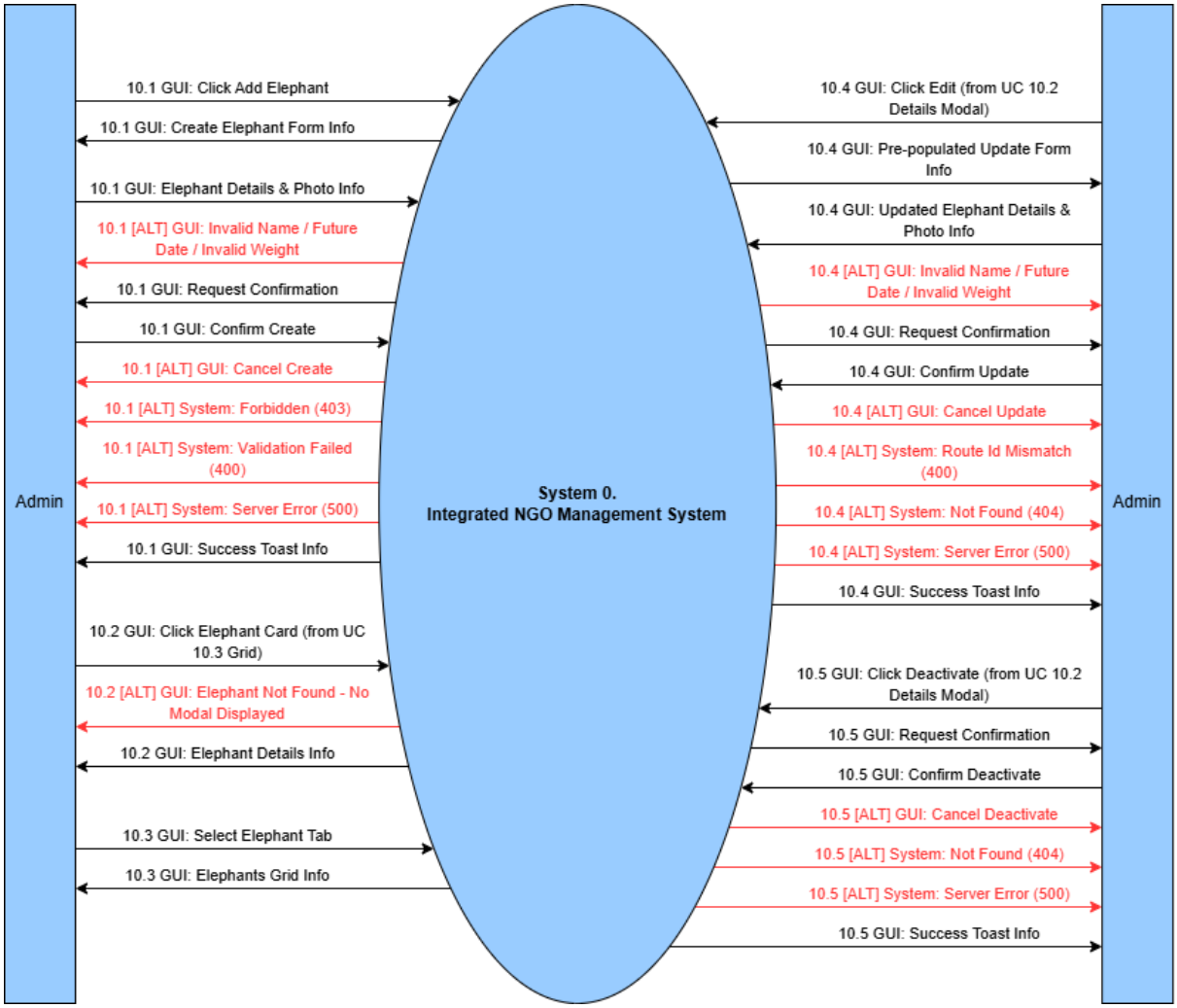


Figure 117: Elephant Technical Context Diagram

2.5.10 Report Subsystem Context Diagram

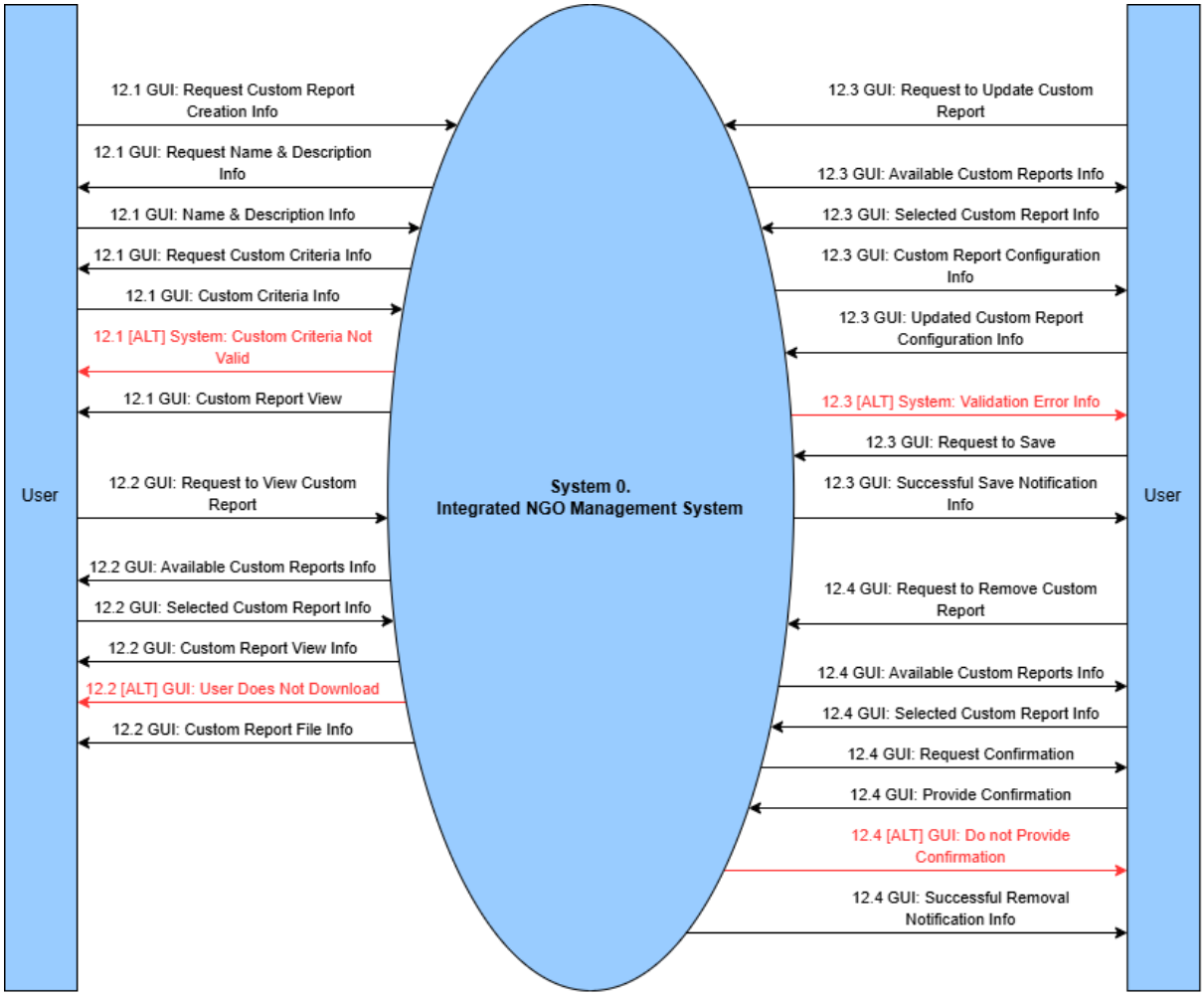


Figure 118: Report Technical Context Diagram

2.5.11 Budget Subsystem Context Diagram

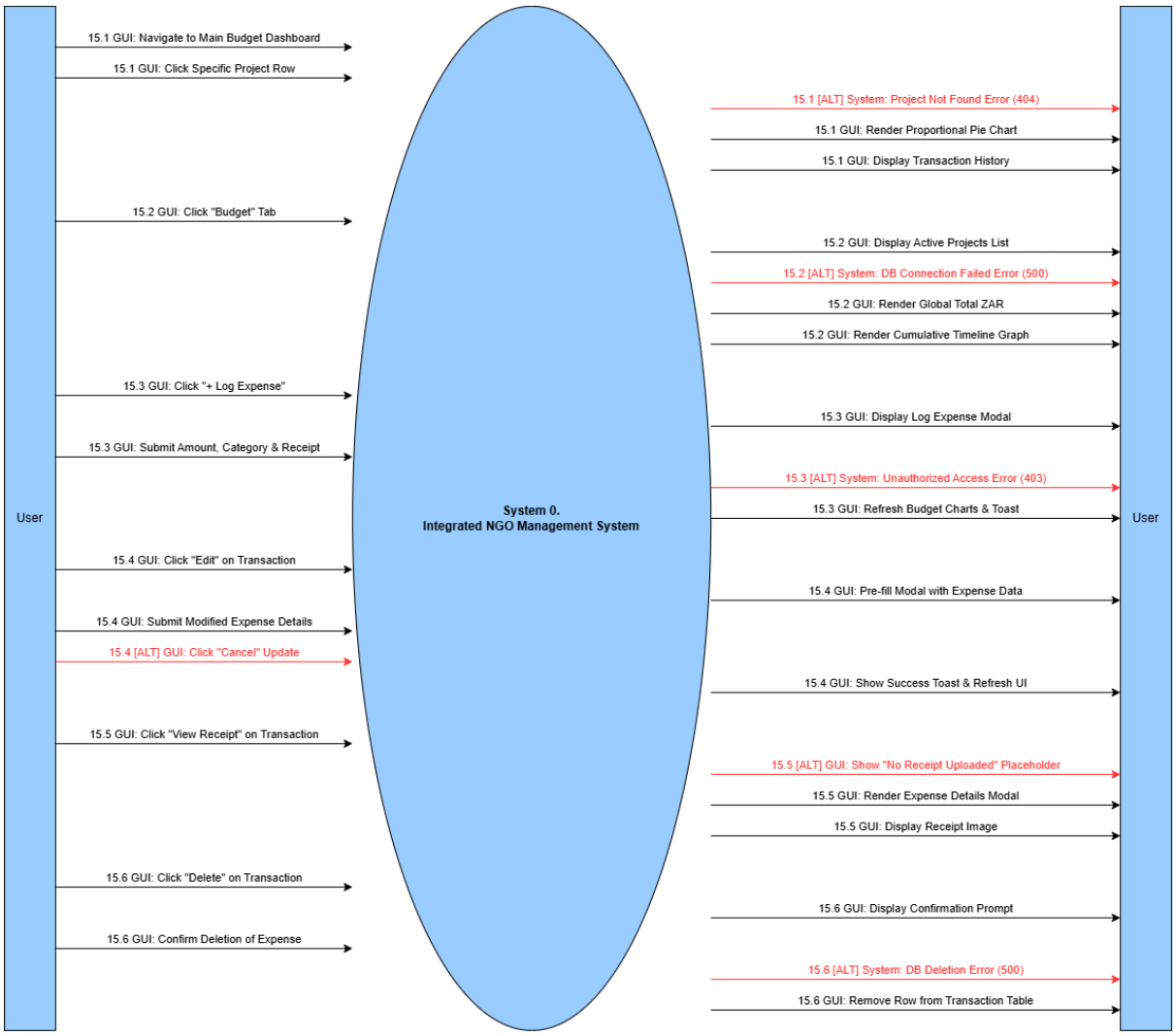


Figure 119: Budget Technical Context Diagram

2.6 Technical Primitive Level Diagram

2.6.1 User Subsystem Primitive Diagrams

2.6.1.1 View FAQ: Use Case 1.9

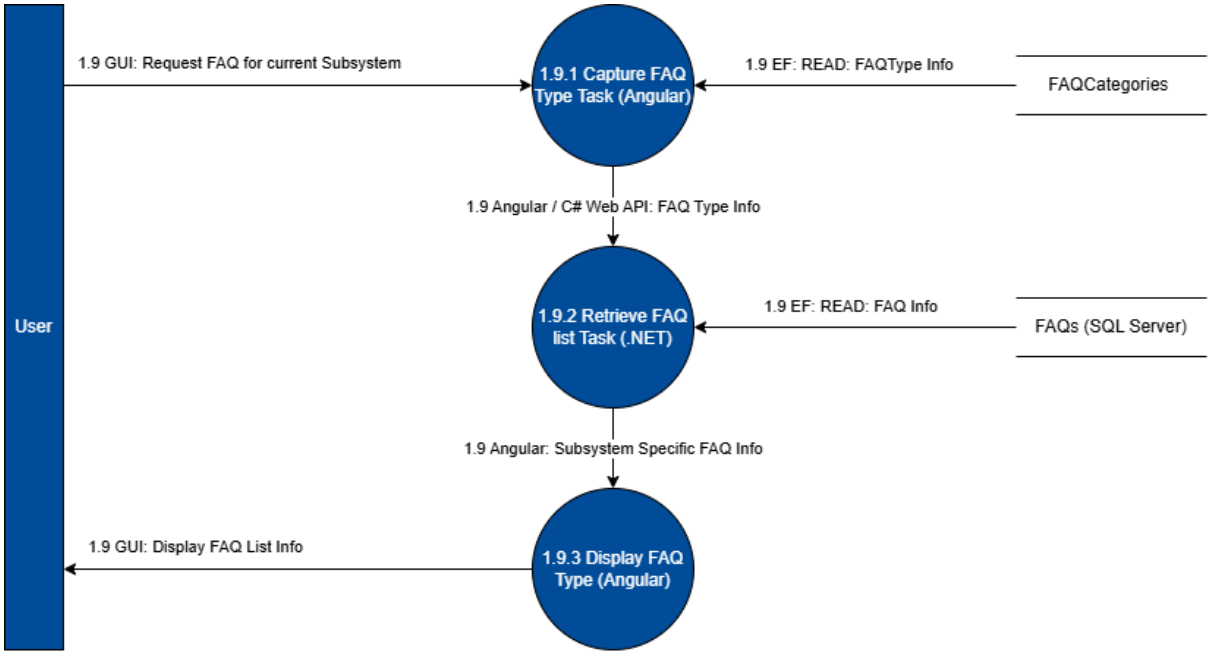


Figure 120: Technical Primitive Level Diagram 1.9 View FAQ

2.6.2 Admin Subsystem Primitive Diagrams

2.6.2.1 Add FAQ: Use Case 2.5

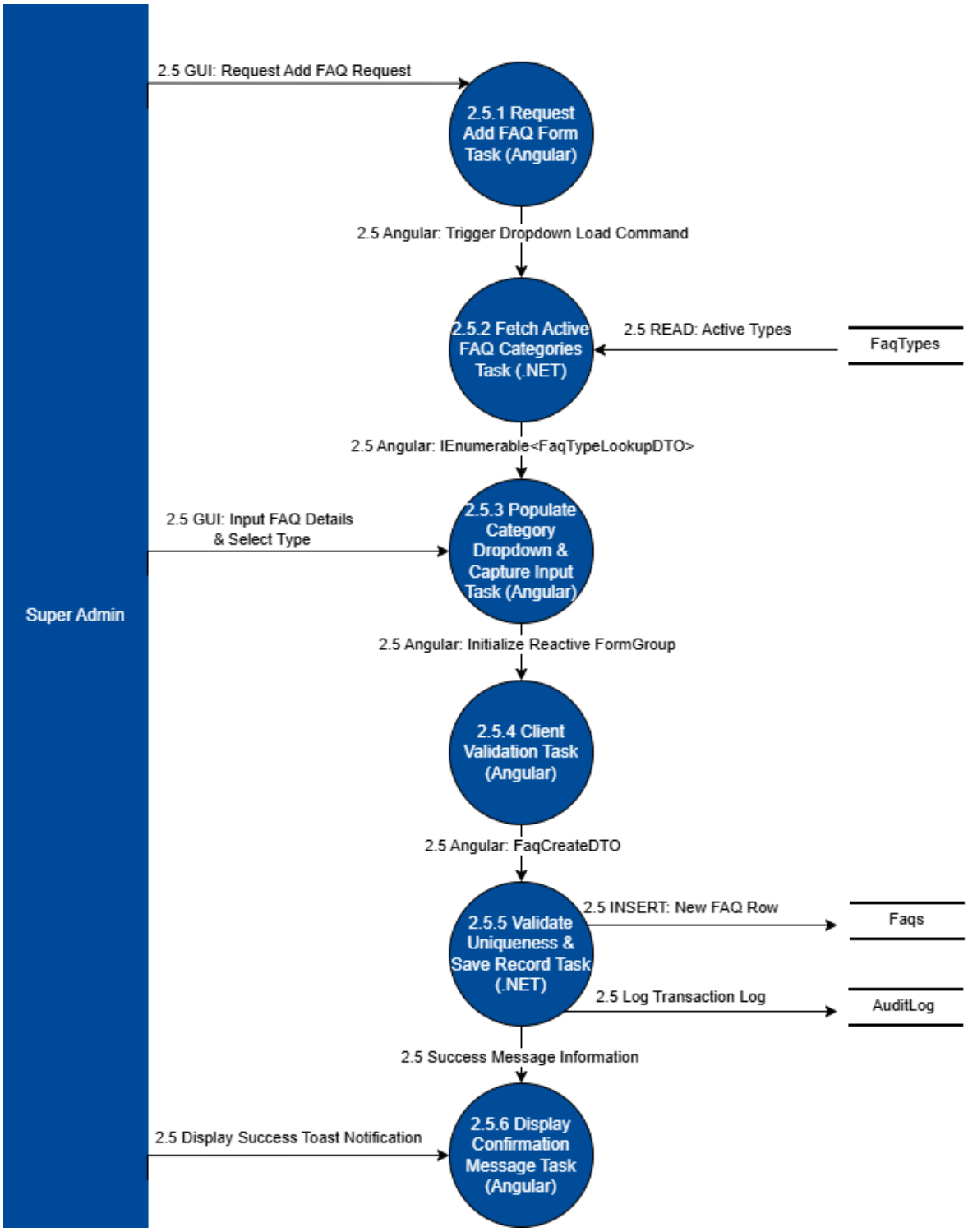


Figure 121: Technical Primitive Level Diagram 2.5 Add FAQ

2.6.2.2 Update FAQ: Use Case 2.6

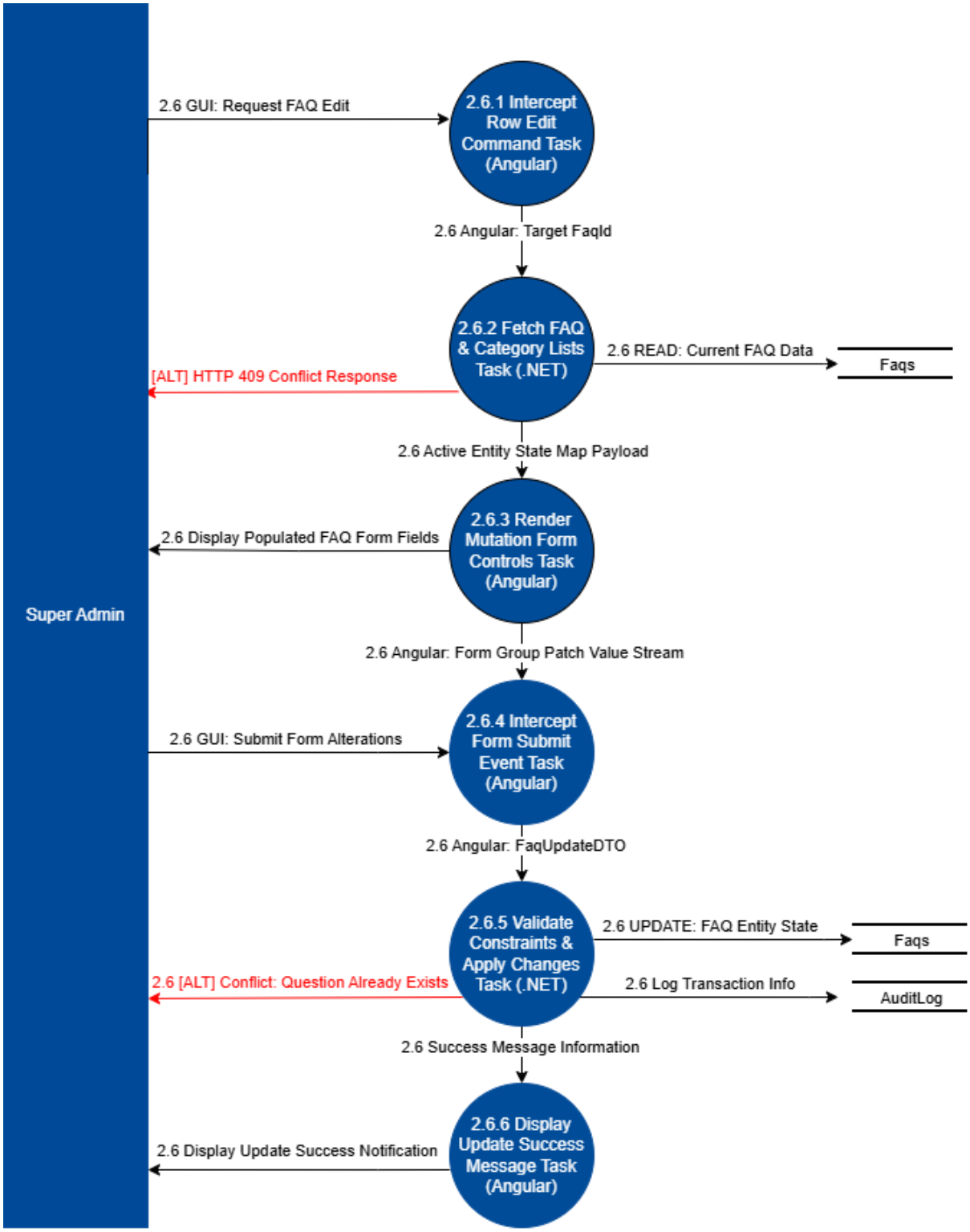


Figure 122: Technical Primitive Level Diagram 2.6 Update FAQ

2.6.2.3 Remove FAQ: Use Case 2.7

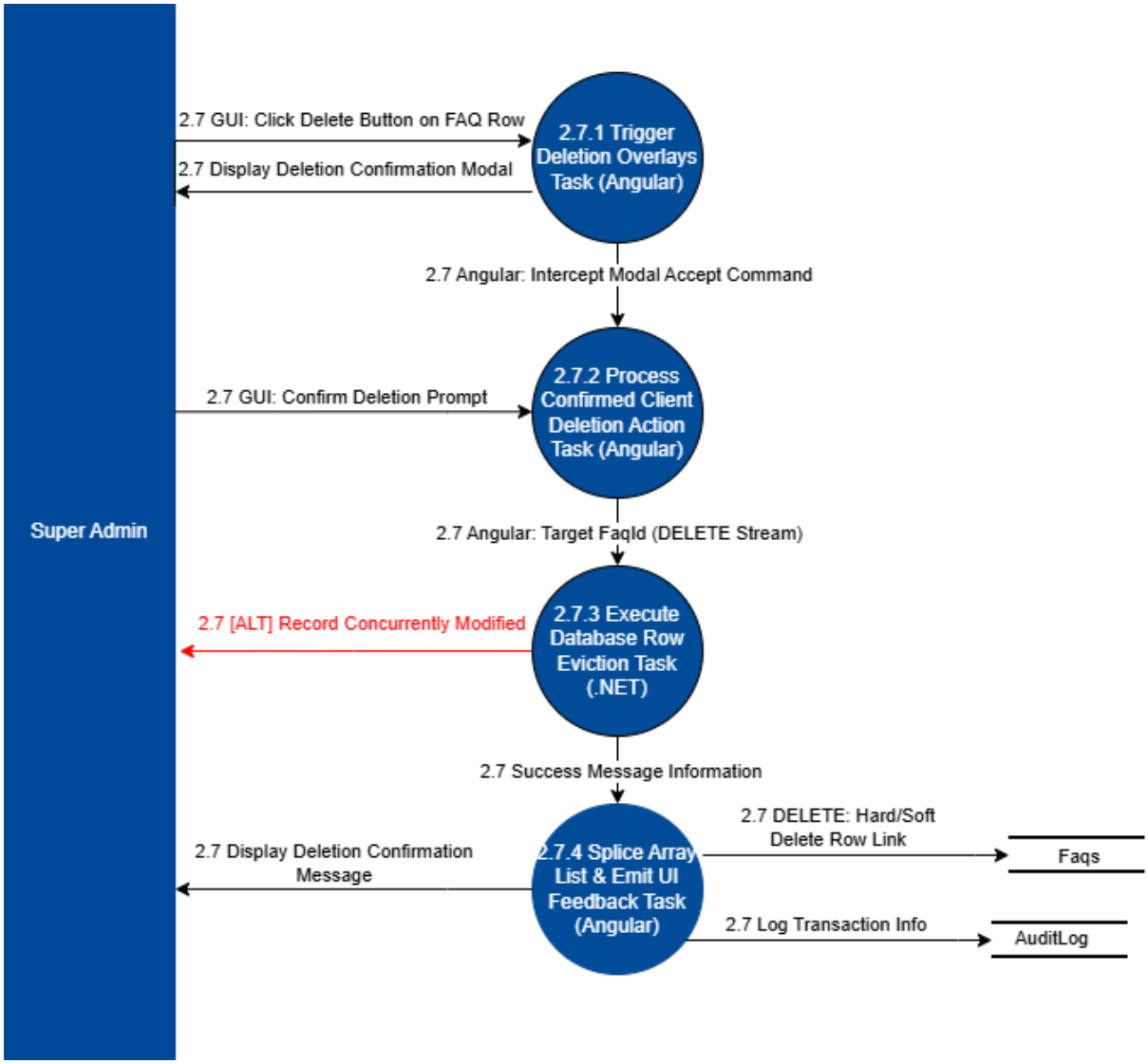


Figure 123: Technical Primitive Level Diagram 2.7 Remove FAQ

2.6.2.4 Add FAQ Type: Use Case 2.8

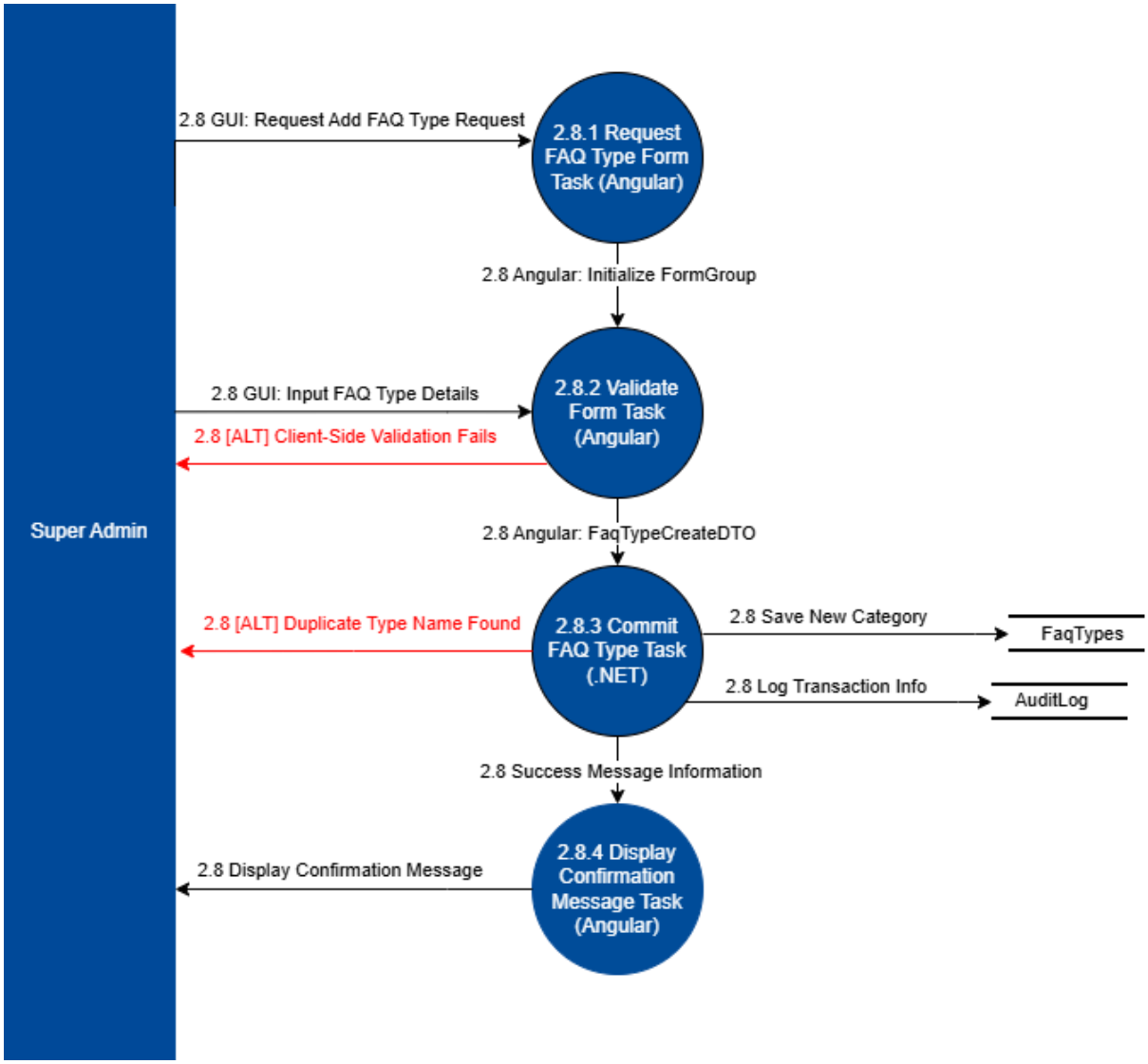


Figure 124: Technical Primitive Level Diagram 2.8 Add FAQ Type

2.6.2.5 Update FAQ Type: Use Case 2.9

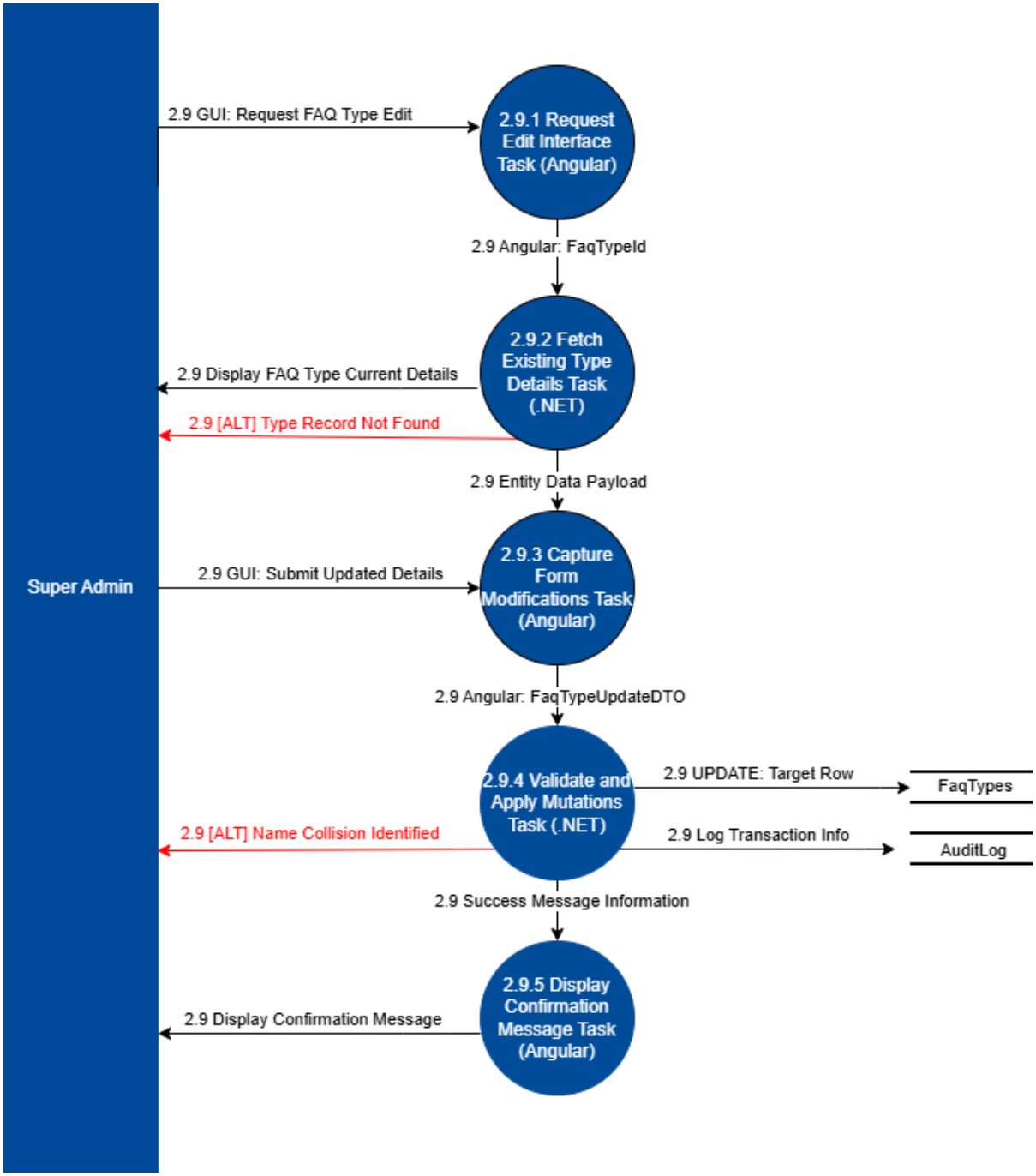


Figure 125: Technical Primitive Level Diagram 2.9 Update FAQ Type

2.6.2.6 Remove FAQ Type: Use Case 2.10

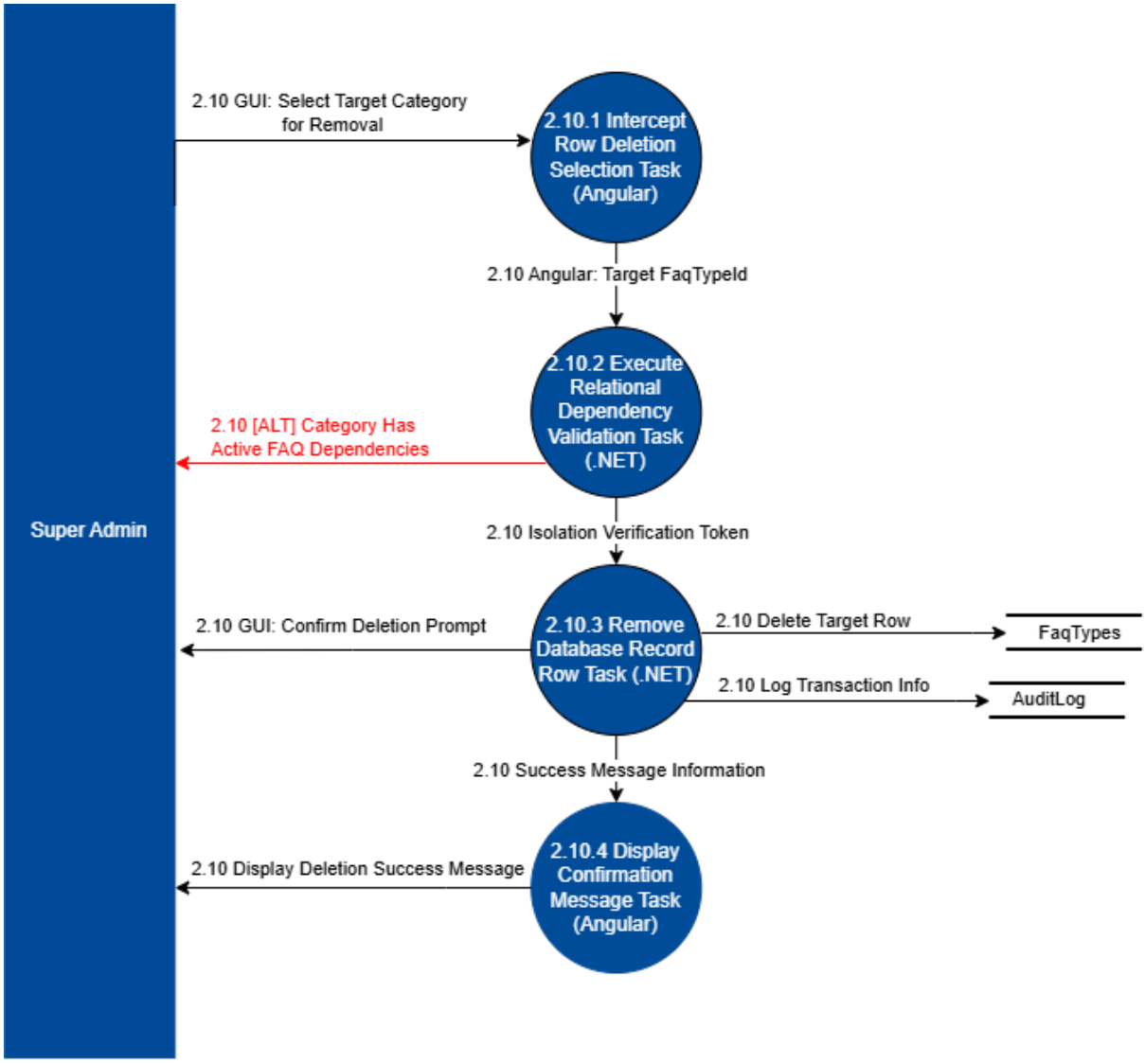


Figure 126: Technical Primitive Level Diagram 2.10 Remove FAQ Type

2.6.2.7 Update Role Permission: Use Case 2.16

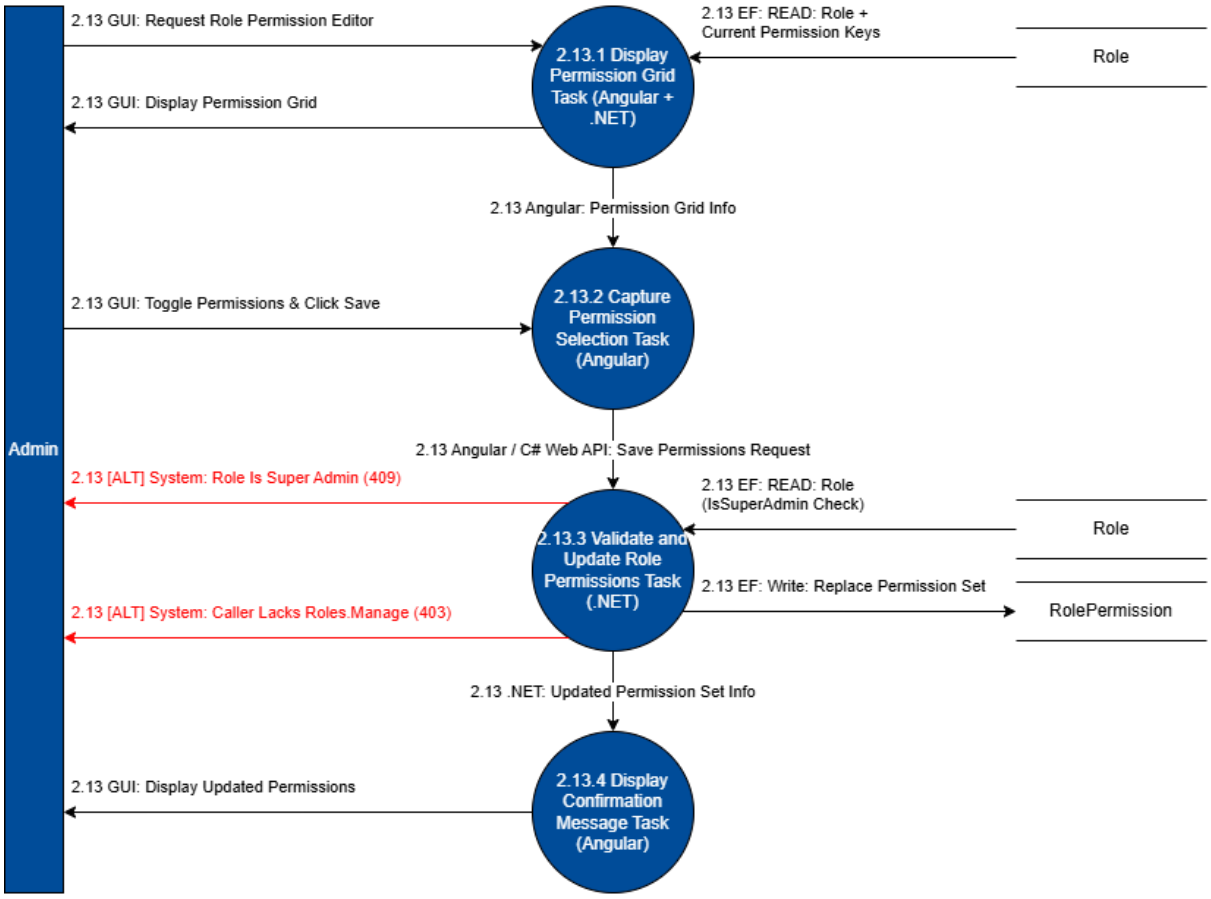


Figure 127: Technical Primitive Level Diagram 2.16 Update Role Permission

2.6.2.8 View Role Permission: Use Case 2.17

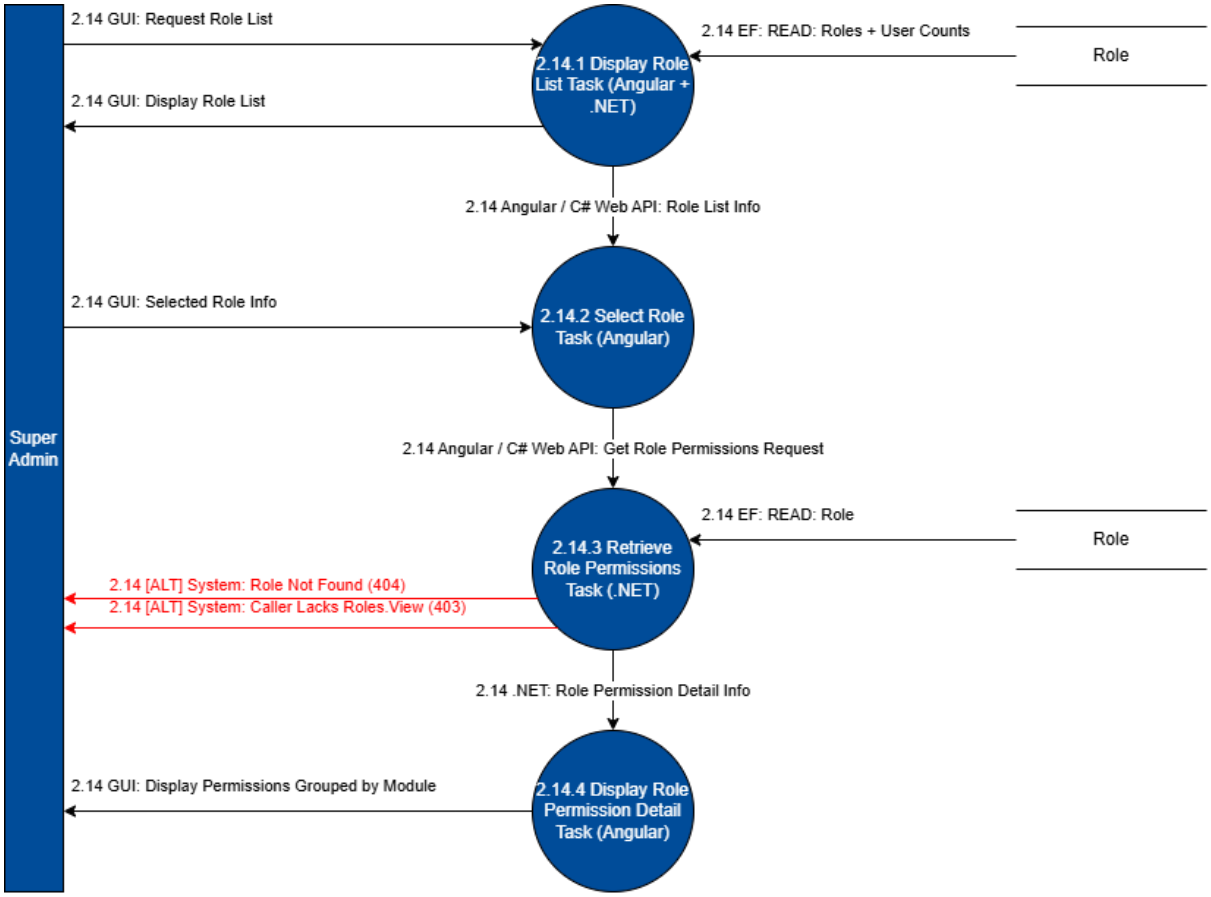


Figure 128: Technical Primitive Level Diagram 2.17 View Role Permission

2.6.3 Inventory Subsystem Primitive Diagrams

2.6.3.1 Create Procurement: Use Case 3.1

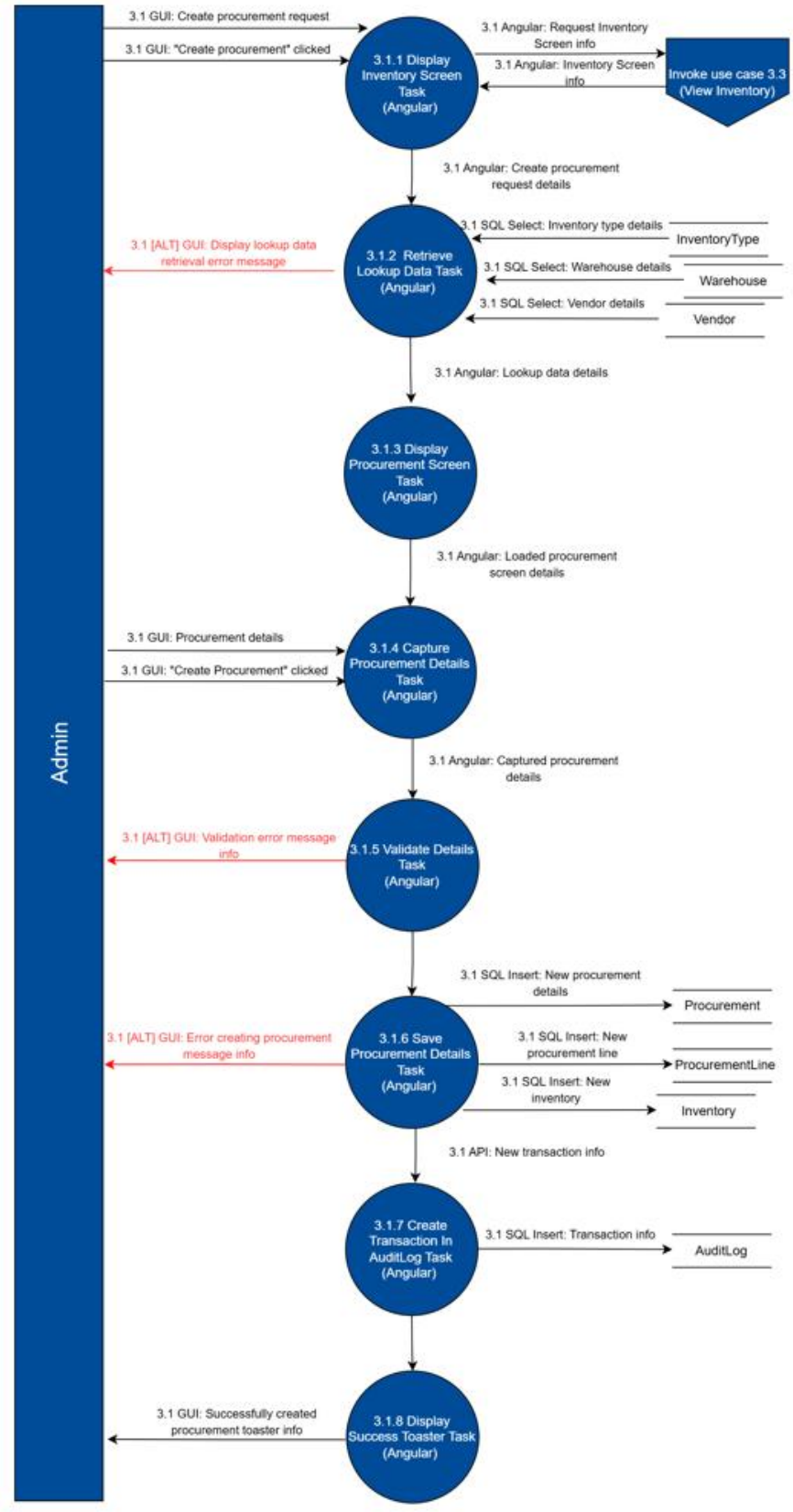


Figure 129: Technical Primitive Level Diagram 3.1 Create Procurement

2.6.3.2 View Procurement: Use Case 3.2

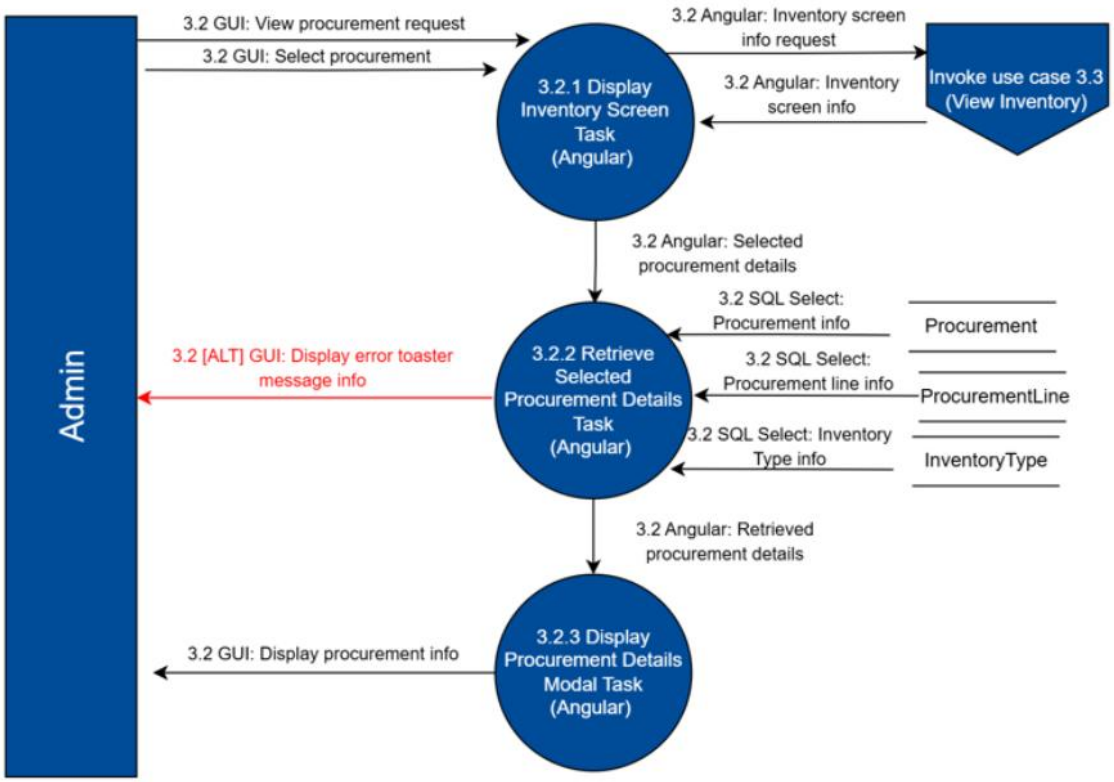


Figure 130: Technical Primitive Level Diagram 3.2 View Procurement

2.6.3.3 View Inventory: Use Case 3.3

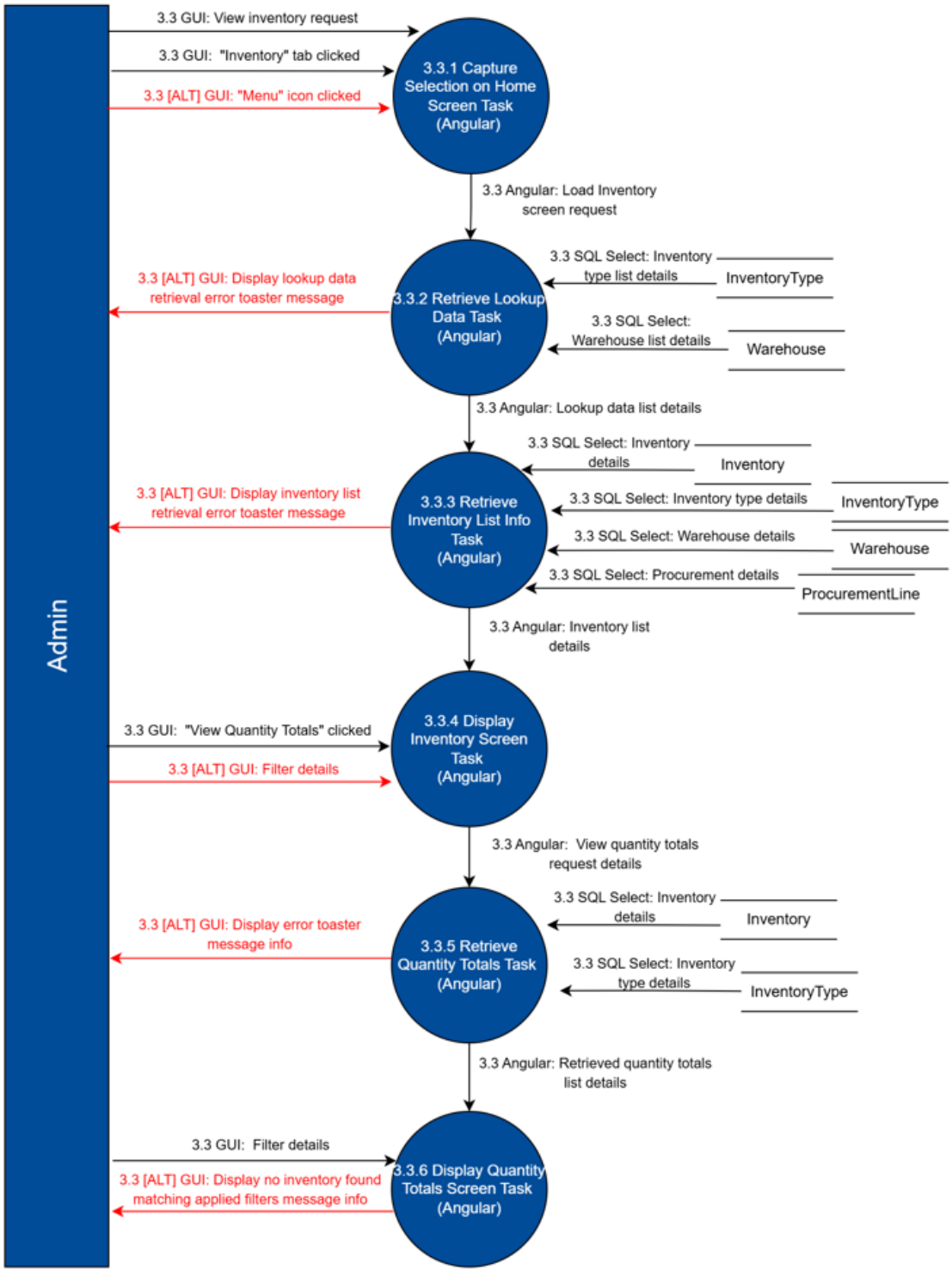


Figure 131: Technical Primitive Level Diagram 3.3 View Inventory

2.6.3.4 Write-Off Inventory: Use Case 3.4

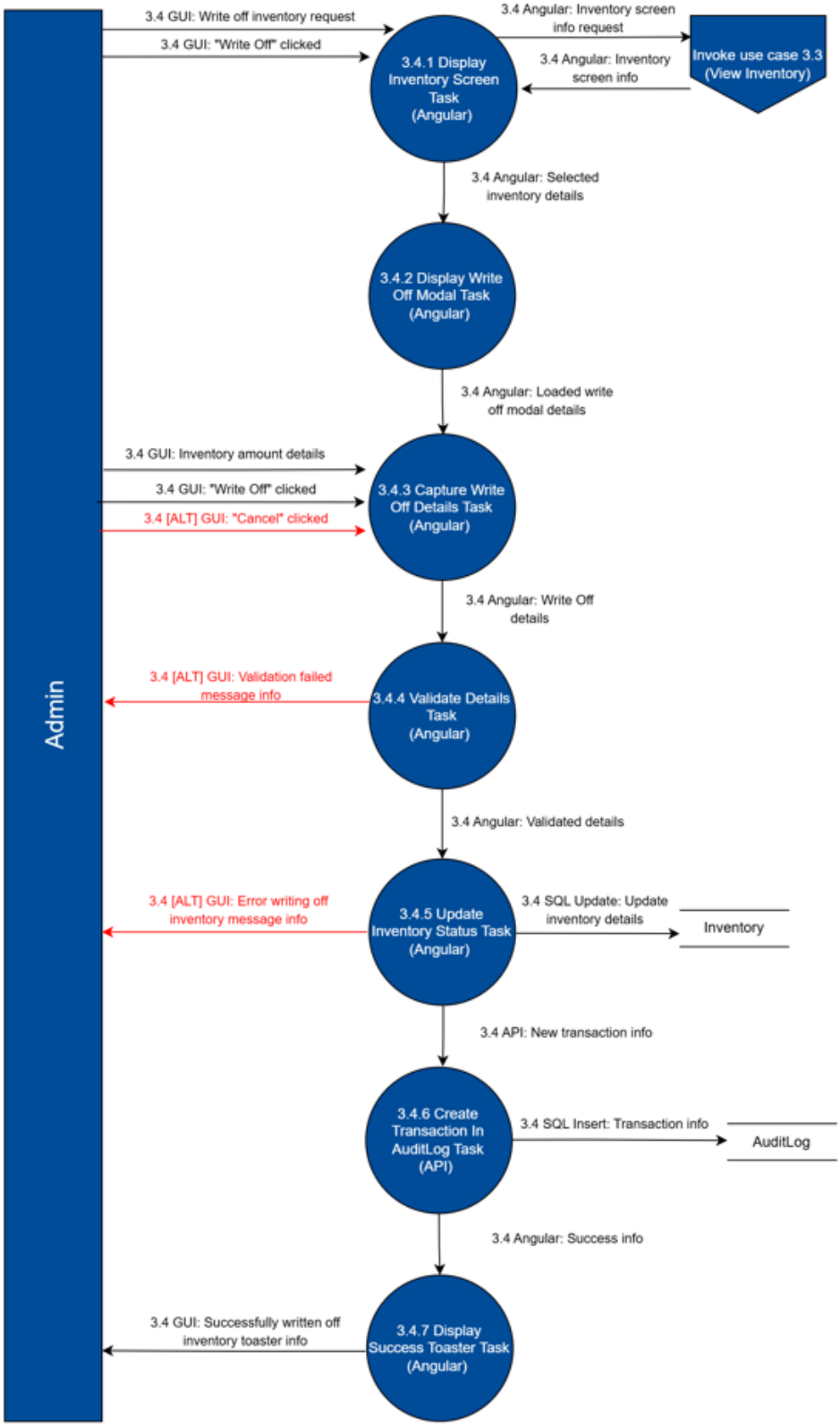


Figure 132: Technical Primitive Level Diagram 3.4 Write-Off Inventory

2.6.3.5 Allocate Inventory: Use Case 3.5

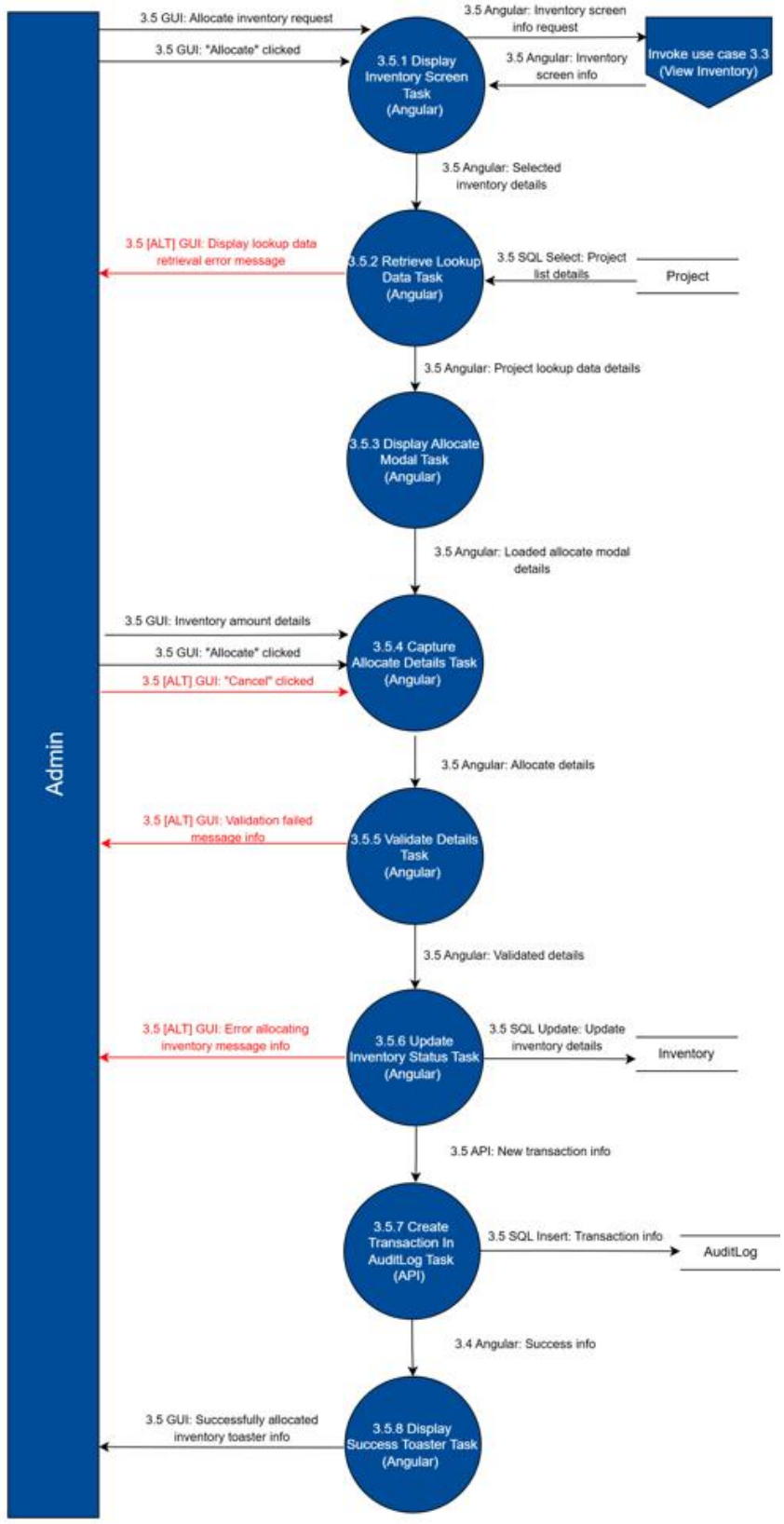


Figure 133: Technical Primitive Level Diagram 3.5 Allocate Inventory

2.6.3.6 Create Inventory Item Type: Use Case 3.6

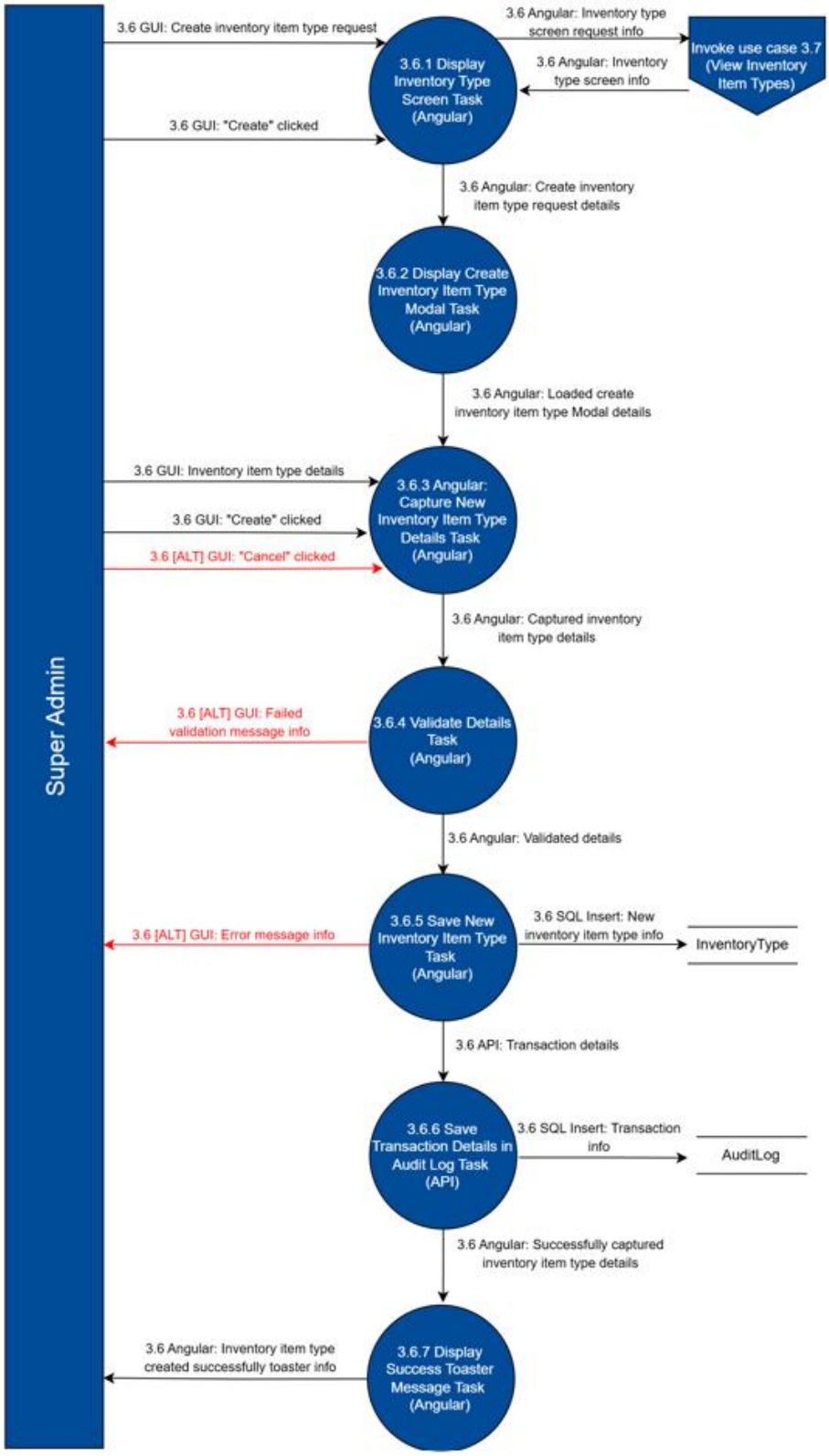


Figure 134: Technical Primitive Level Diagram 3.6 Create Inventory Item Type

2.6.3.7 View Inventory Item Types: Use Case 3.7

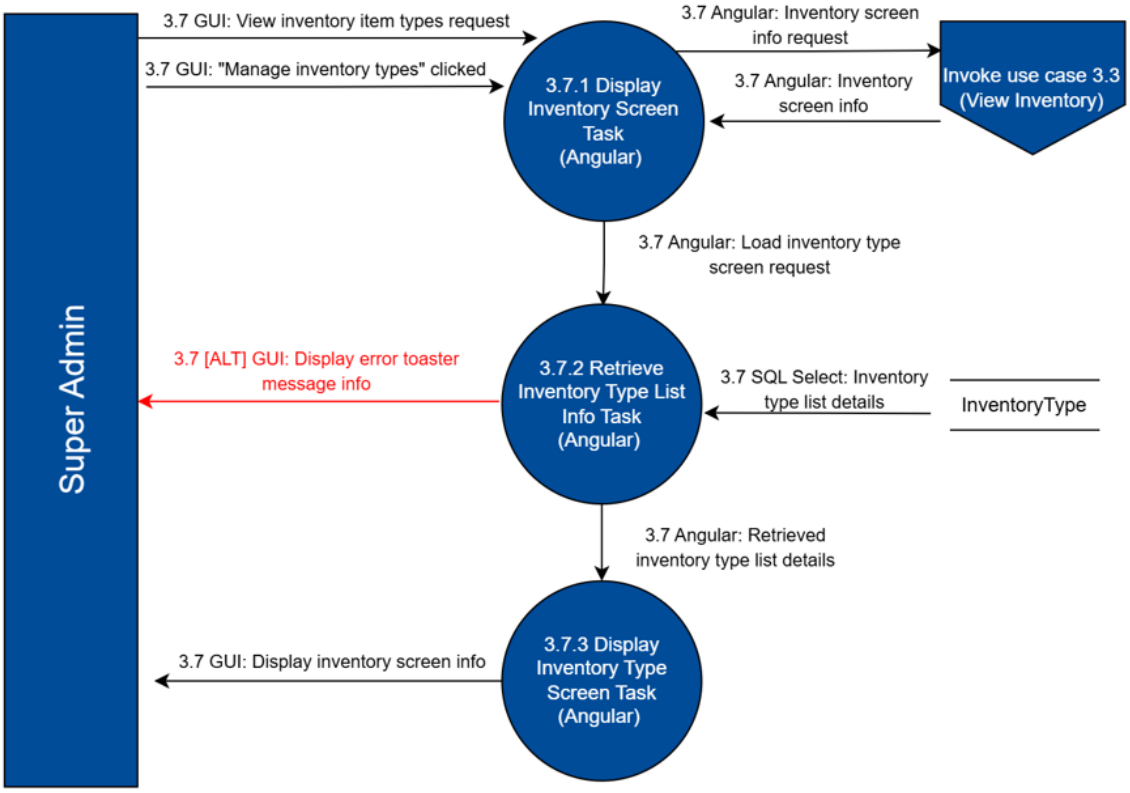


Figure 135: Technical Primitive Level Diagram 3.7 View Inventory Item Types

2.6.3.8 Update Inventory Item Type: Use Case 3.8

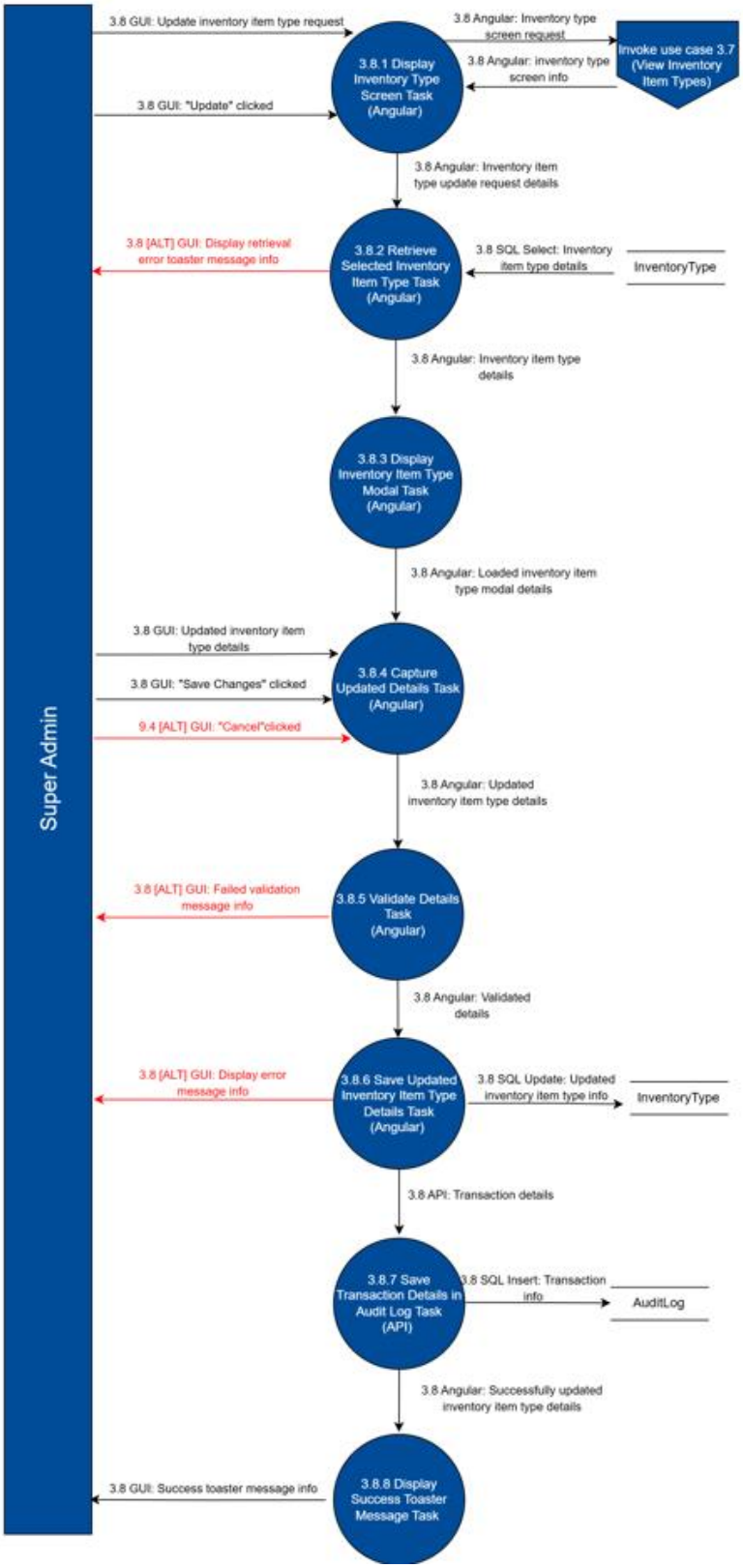


Figure 136: Technical Primitive Level Diagram 3.8 Update Inventory Item Type

2.6.3.9 Deactivate Inventory Item Type: Use Case 3.9

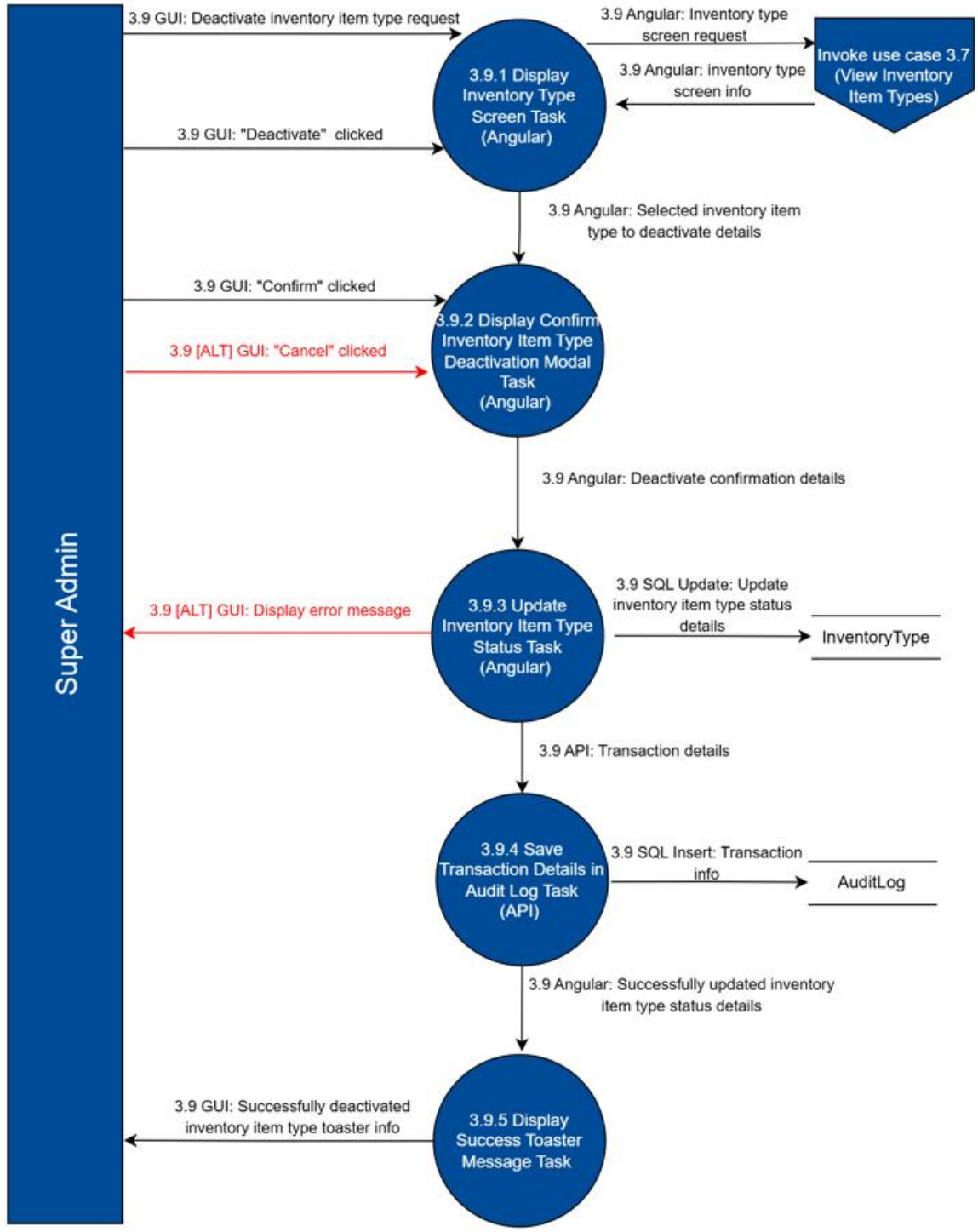


Figure 137: Technical Primitive Level Diagram 3.9 Deactivate Inventory Item Type

2.6.3.10 Reactivate Inventory Item Type: Use Case 3.10

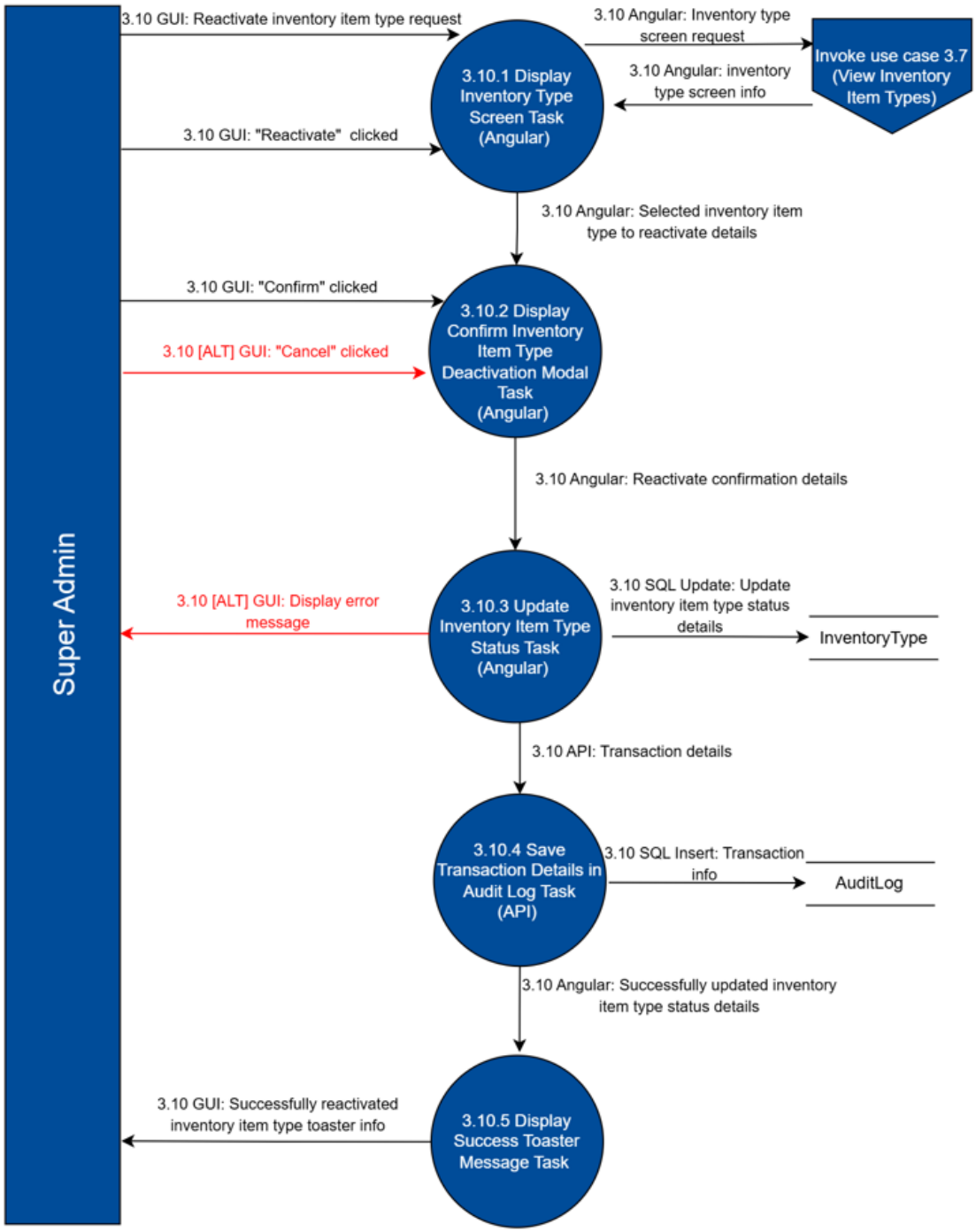


Figure 138: Technical Primitive Level Diagram 3.10 Reactivate Inventory Item Type

2.6.4 Warehouse Subsystem Primitive Diagrams

2.6.4.1 Create Warehouse: Use Case 5.1

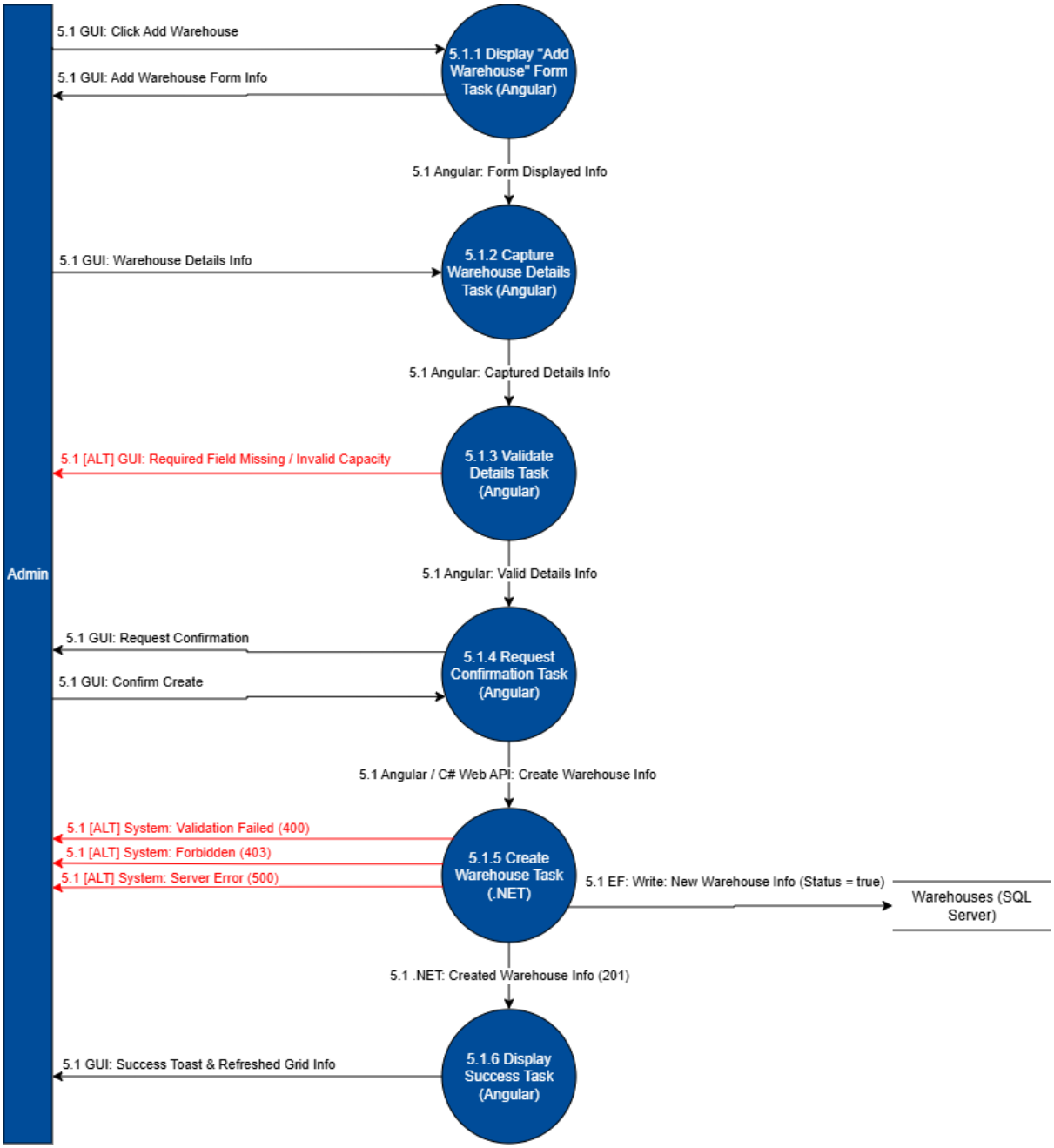


Figure 139: Technical Primitive Level Diagram 5.1 Create Warehouse

2.6.4.2 View Warehouse: Use Case 5.2

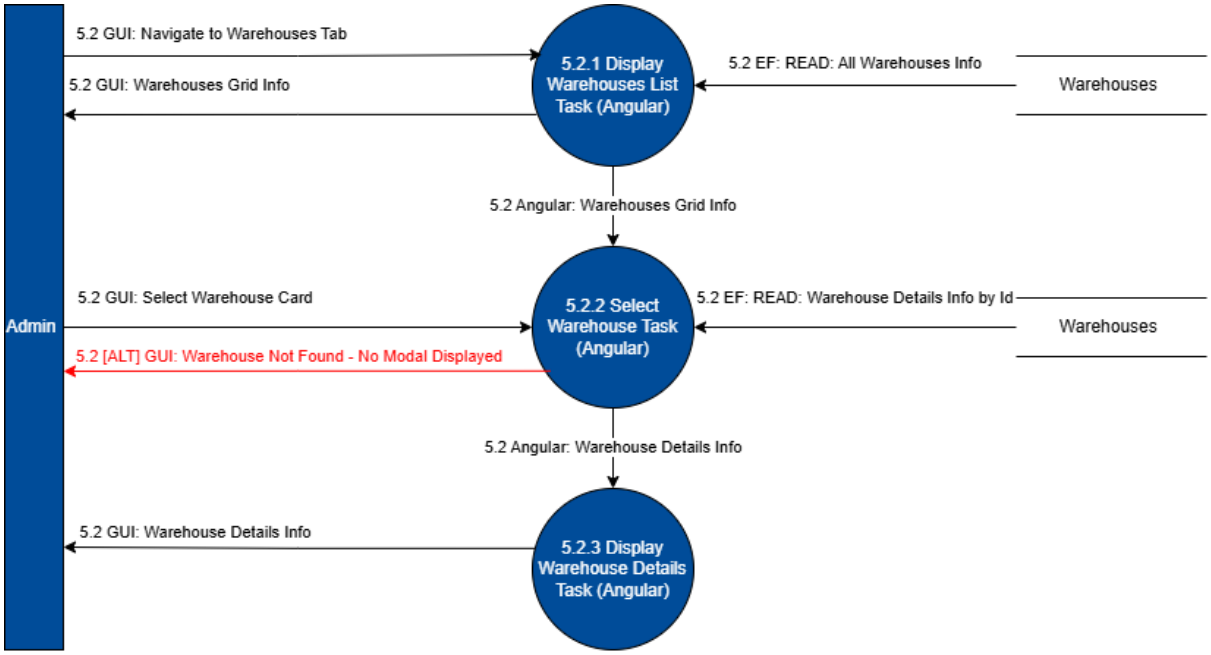


Figure 140: Technical Primitive Level Diagram 5.2 View Warehouse

2.6.4.3 Update Warehouse: Use Case 5.3

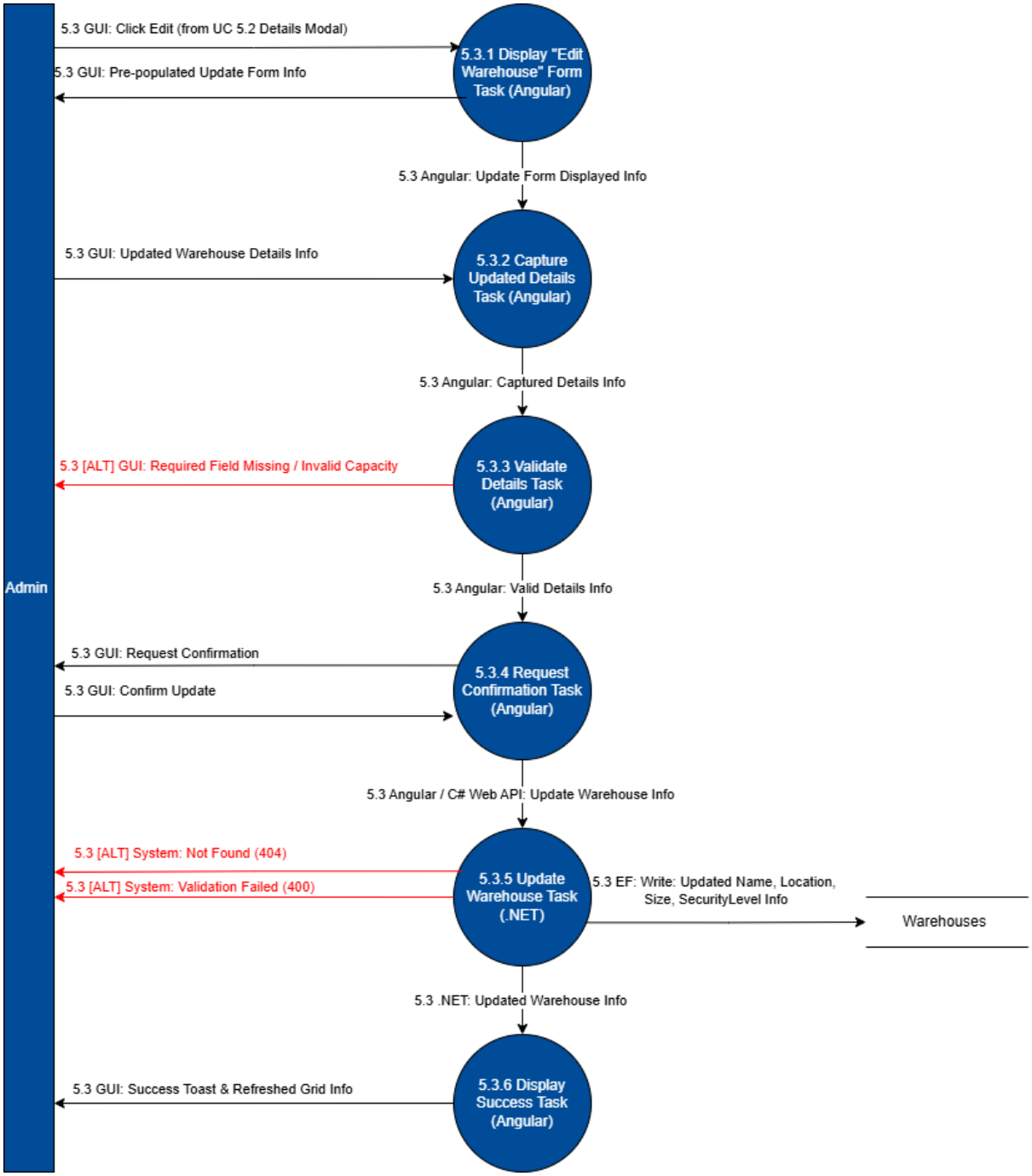


Figure 141: Technical Primitive Level Diagram 5.3 Update Warehouse

2.6.4.4 Reactivate Warehouse: Use Case 5.4

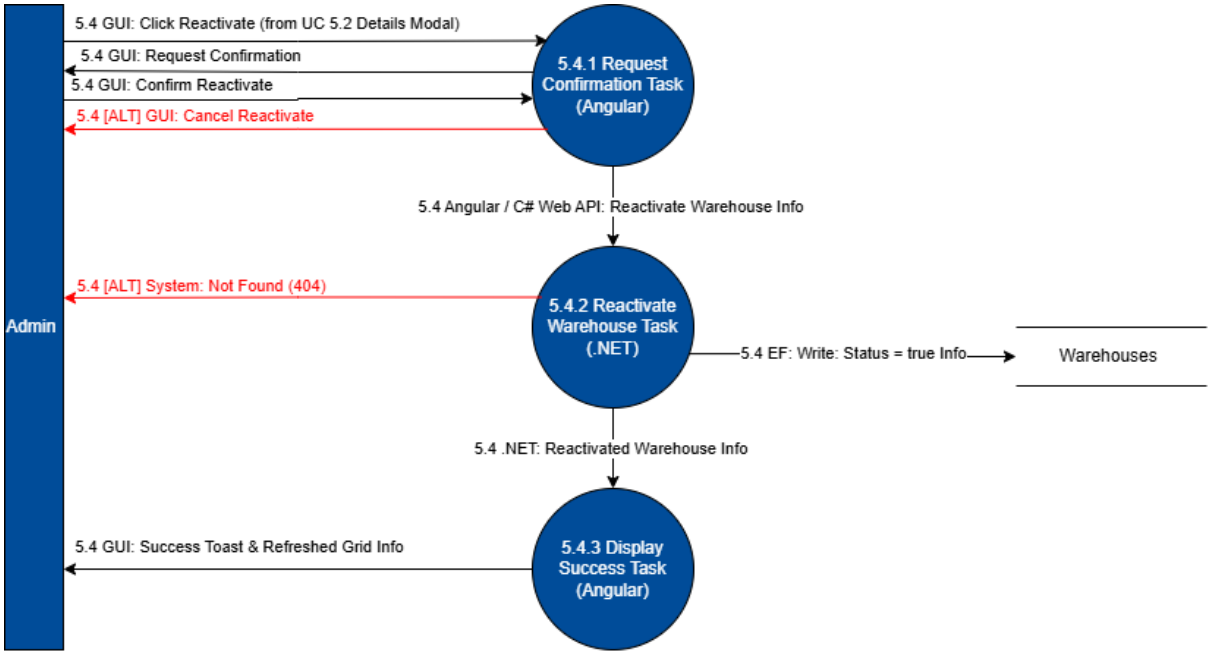


Figure 142: Technical Primitive Level Diagram 5.4 Reactivate Warehouse

2.6.4.5 Deactivate Warehouse: Use Case 5.5

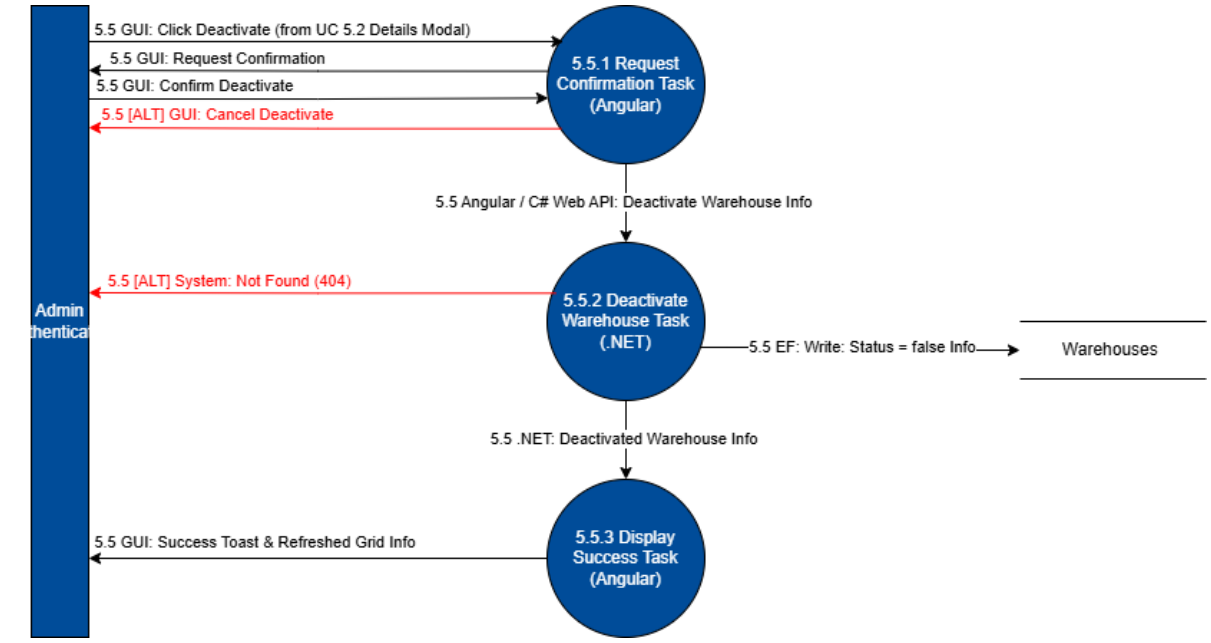


Figure 143: Technical Primitive Level Diagram 5.5 Deactivate Warehouse

2.6.4.6 Inventory Stock Take: Use Case 5.6

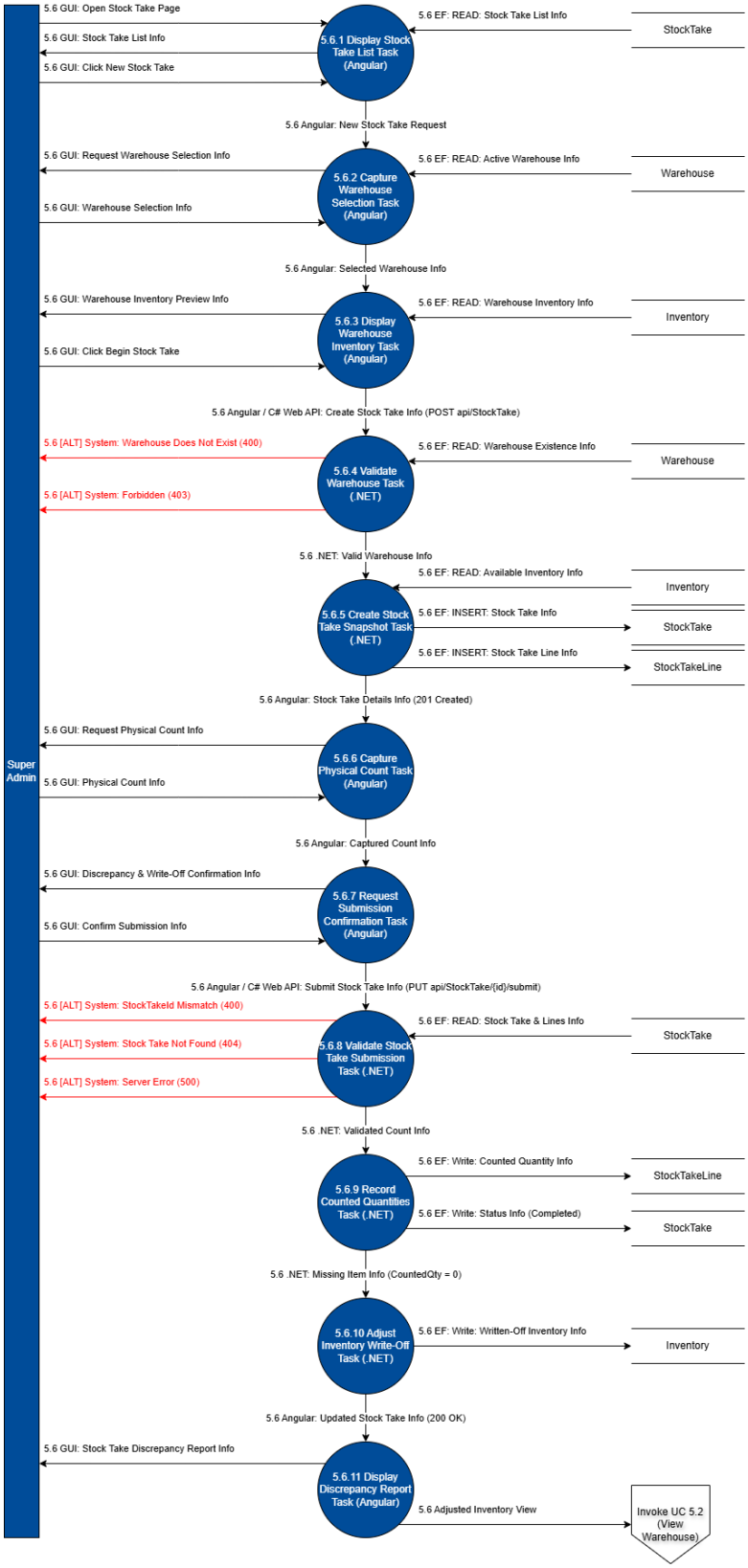


Figure 144: Technical Primitive Level Diagram 5.6 Inventory Stock Take

2.6.5 Vendor Subsystem Primitive Diagrams

2.6.5.1 Create Vendor: Use Case 6.1

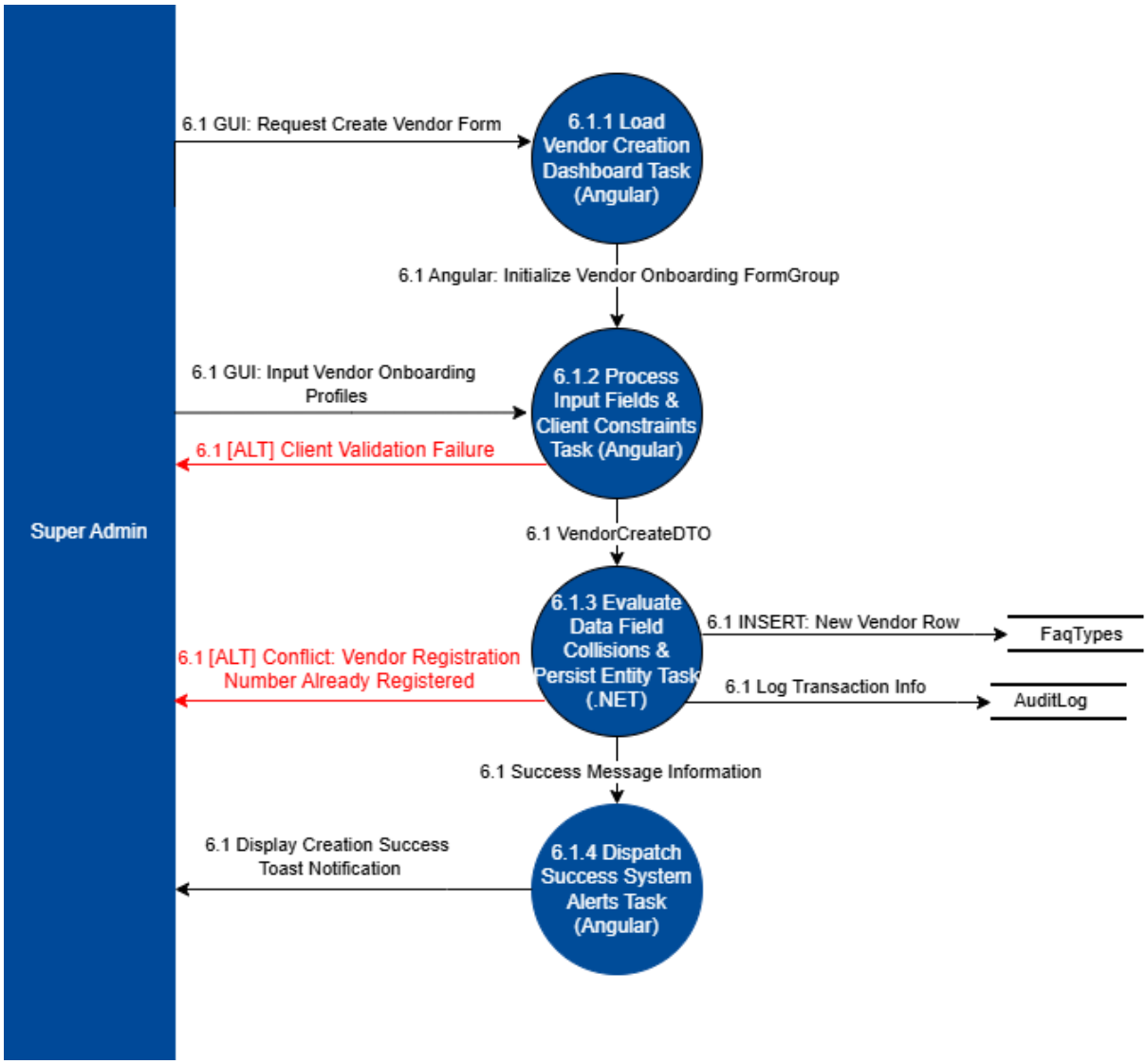


Figure 145: Technical Primitive Level Diagram 6.1 Create Vendor

2.6.5.2 View Vendor: Use Case 6.2

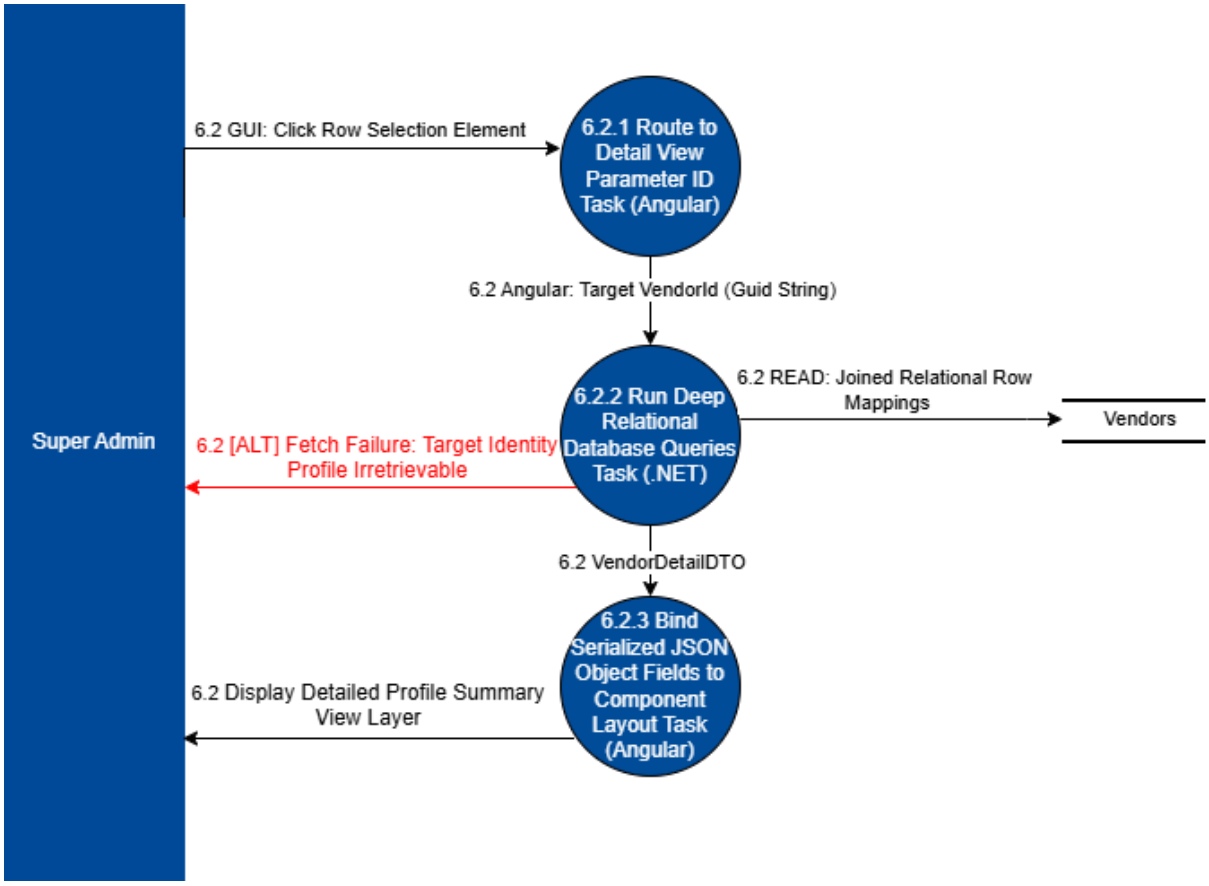


Figure 146: Technical Primitive Level Diagram 6.2 View Vendor

2.6.5.3 View Vendors: Use Case 6.3

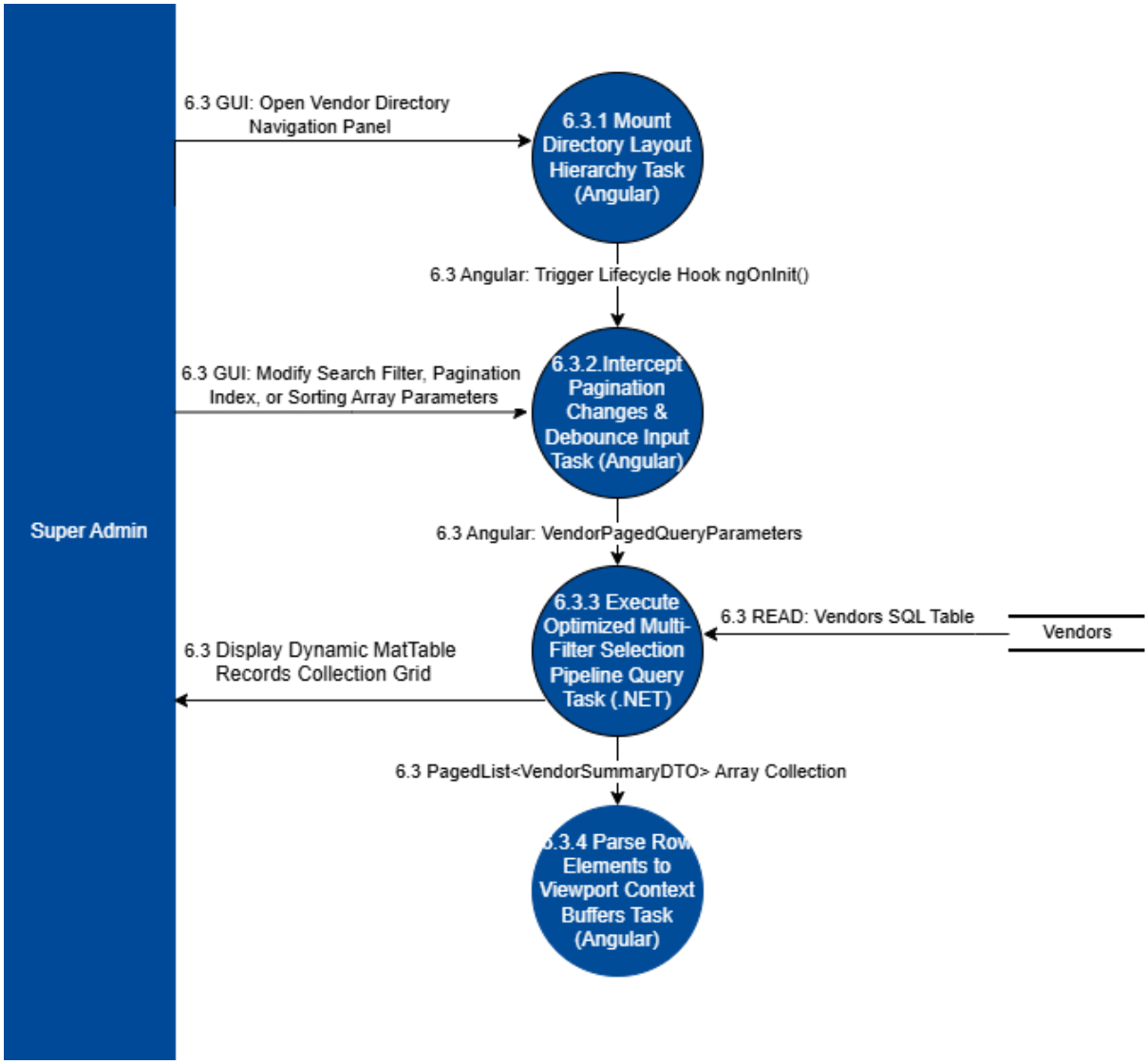


Figure 147: Technical Primitive Level Diagram 6.3 View Vendors

2.6.5.4 Edit Vendor: Use Case 6.4

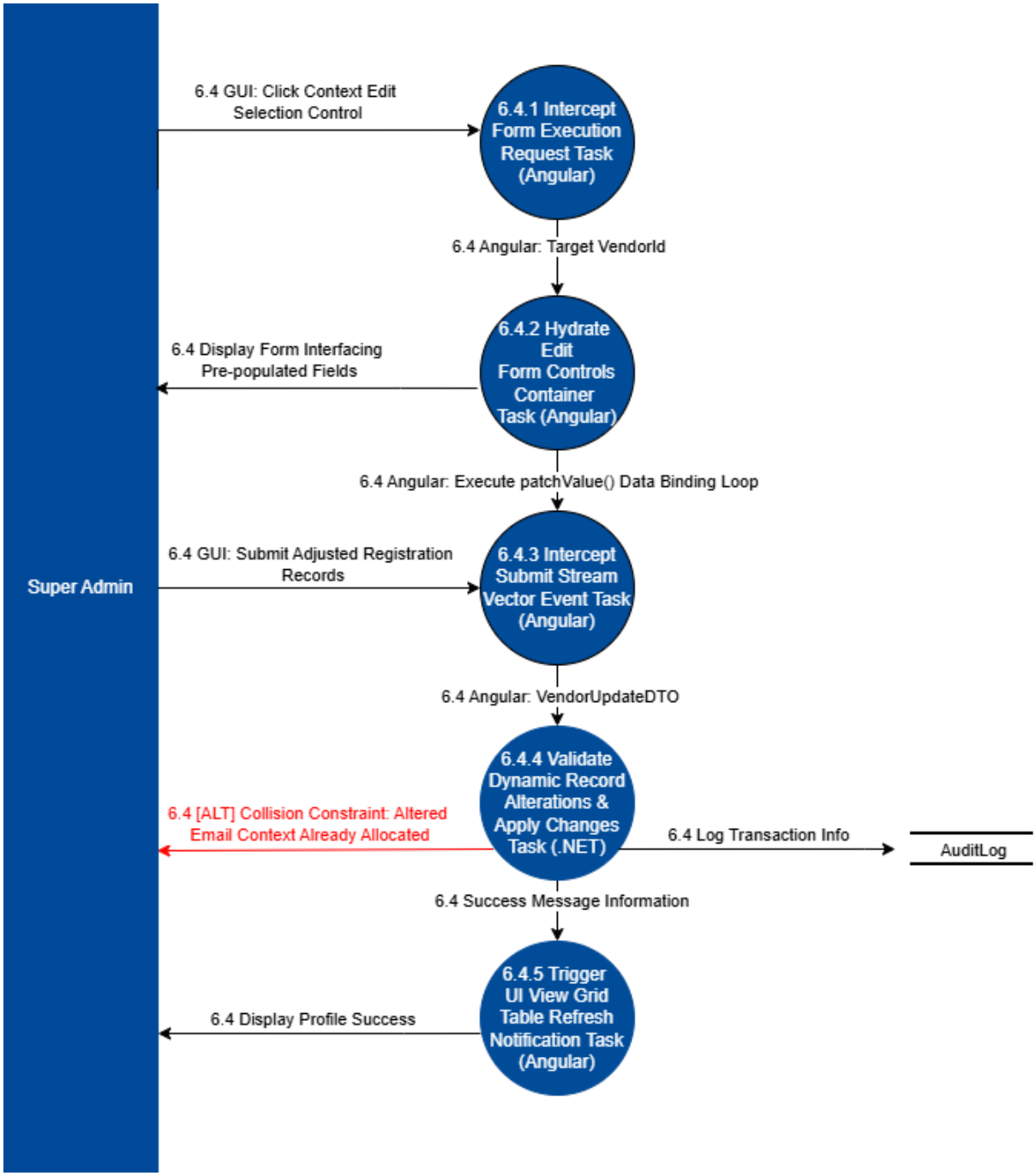


Figure 148: Technical Primitive Level Diagram 6.4 Edit Vendor

2.6.5.5 Reactivate Vendor: Use Case 6.5

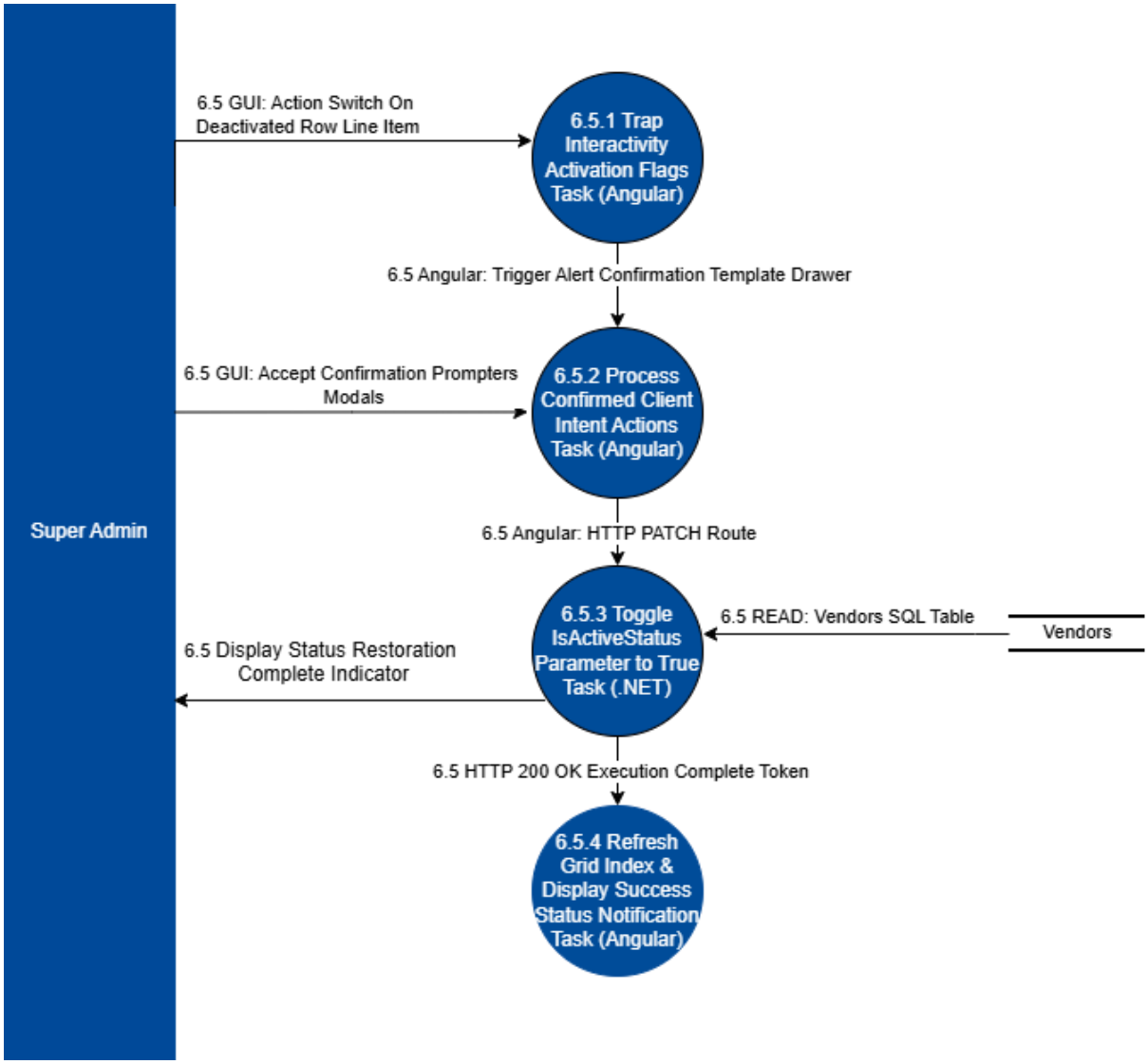


Figure 149: Technical Primitive Level Diagram 6.5 Reactivate Vendor

2.6.5.6 Deactivate Vendor: Use Case 6.6

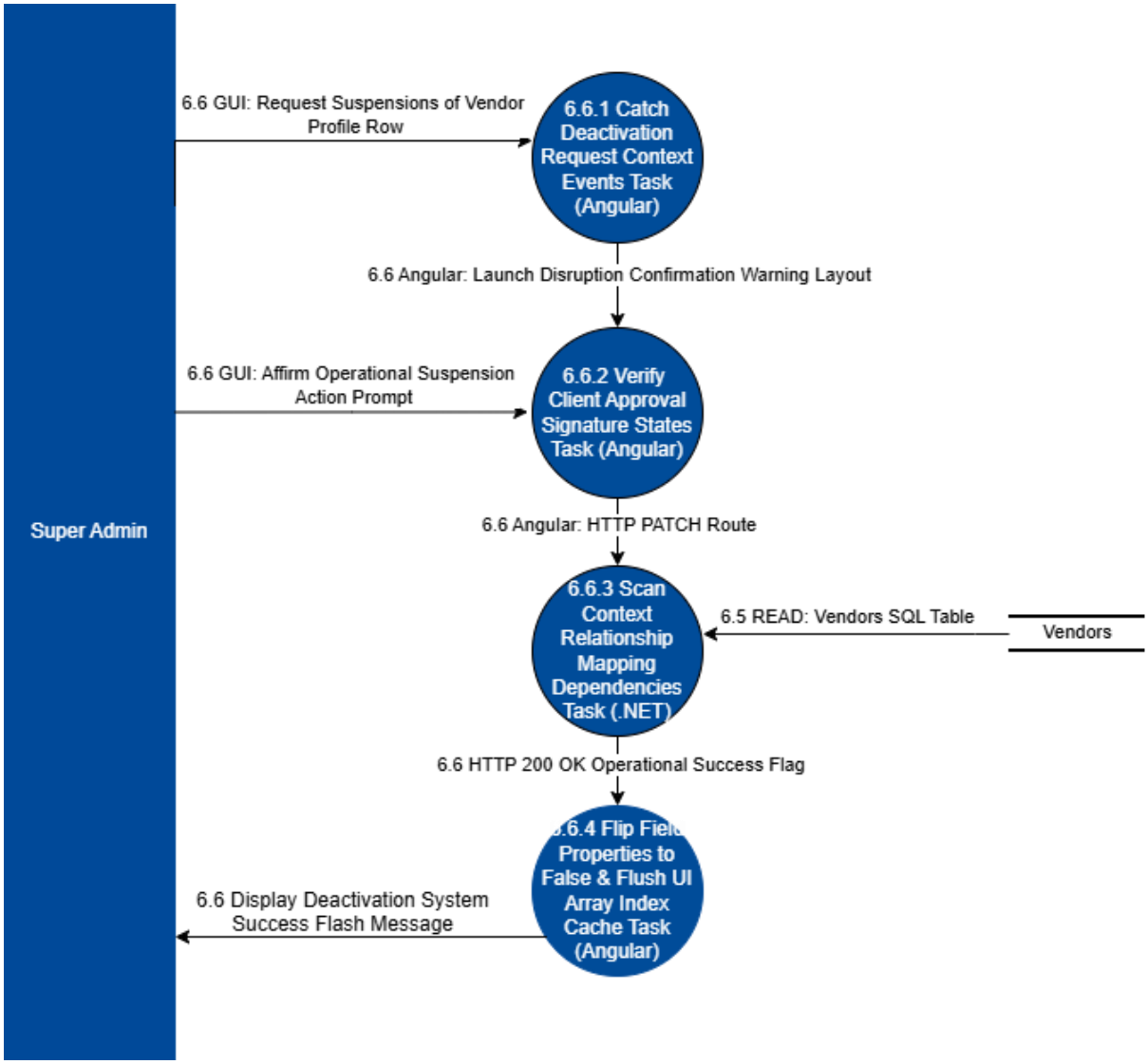


Figure 150: Technical Primitive Level Diagram 6.6 Deactivate Vendor

2.6.5.7 Create Vendor Type: Use Case 6.7

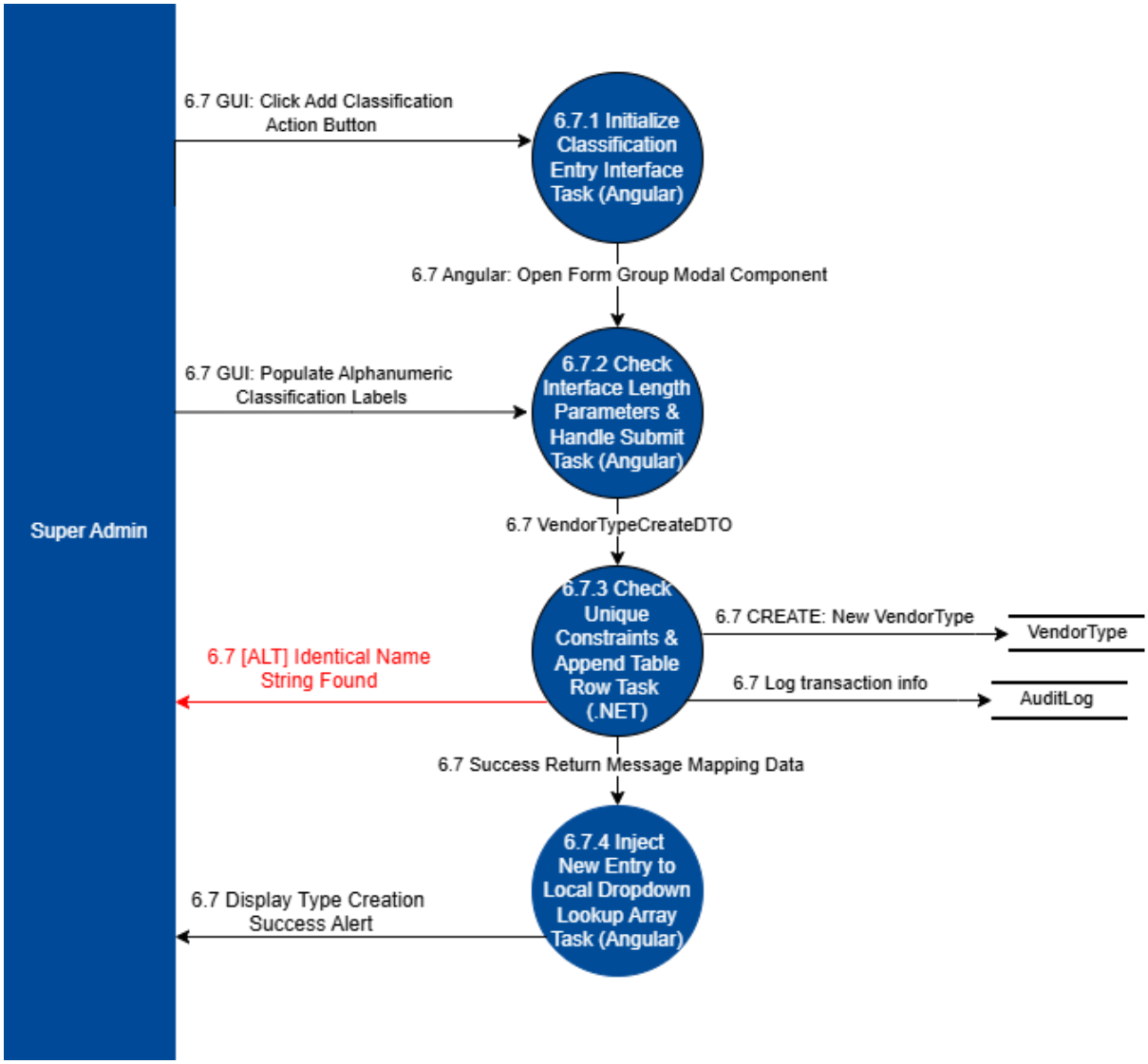


Figure 151: Technical Primitive Level Diagram 6.7 Create Vendor Type

2.6.5.8 Update Vendor Type: Use Case 6.8

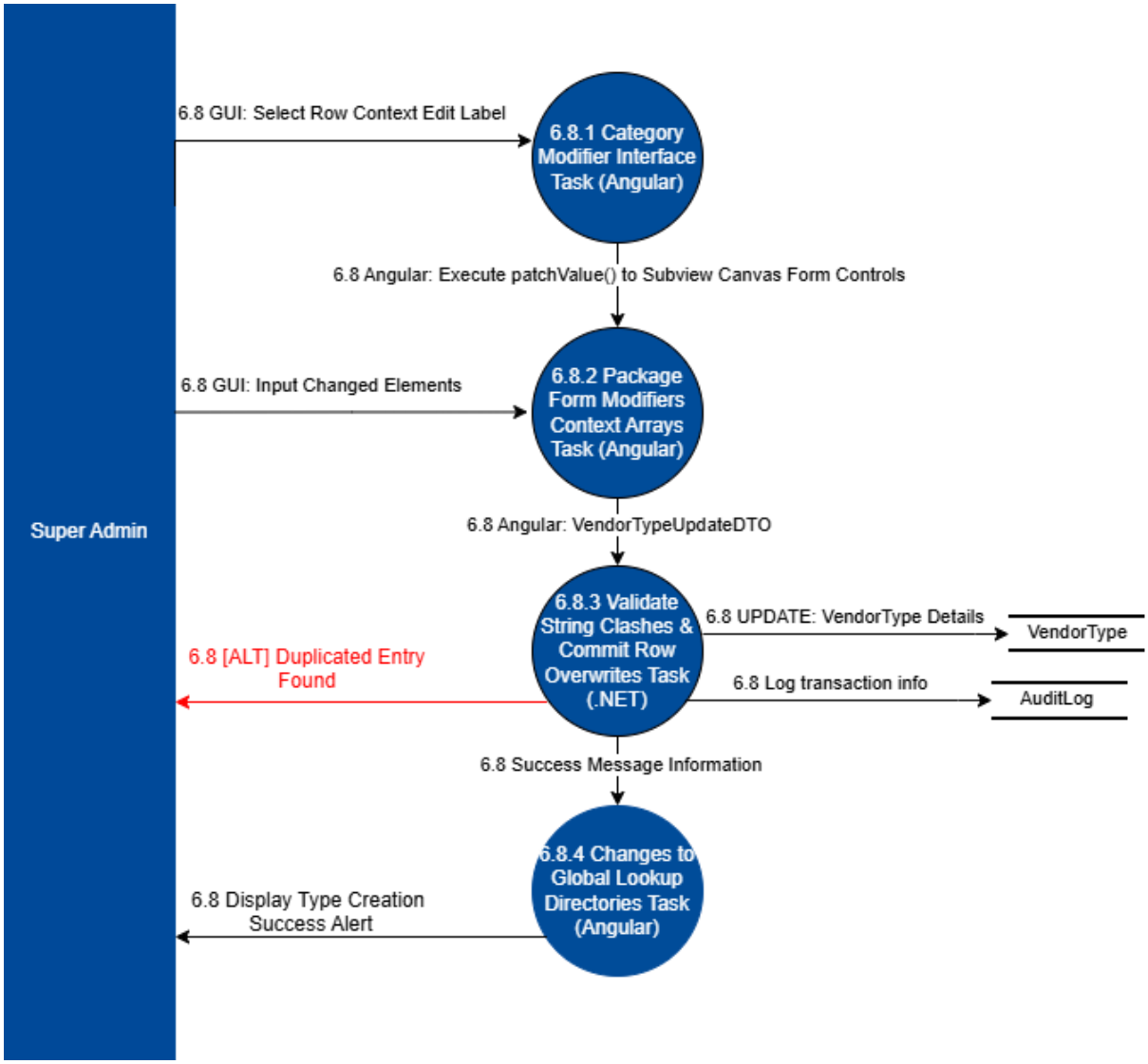


Figure 152: Technical Primitive Level Diagram 6.8 Update Vendor Type

2.6.5.9 View Vendor Types: Use Case 6.9

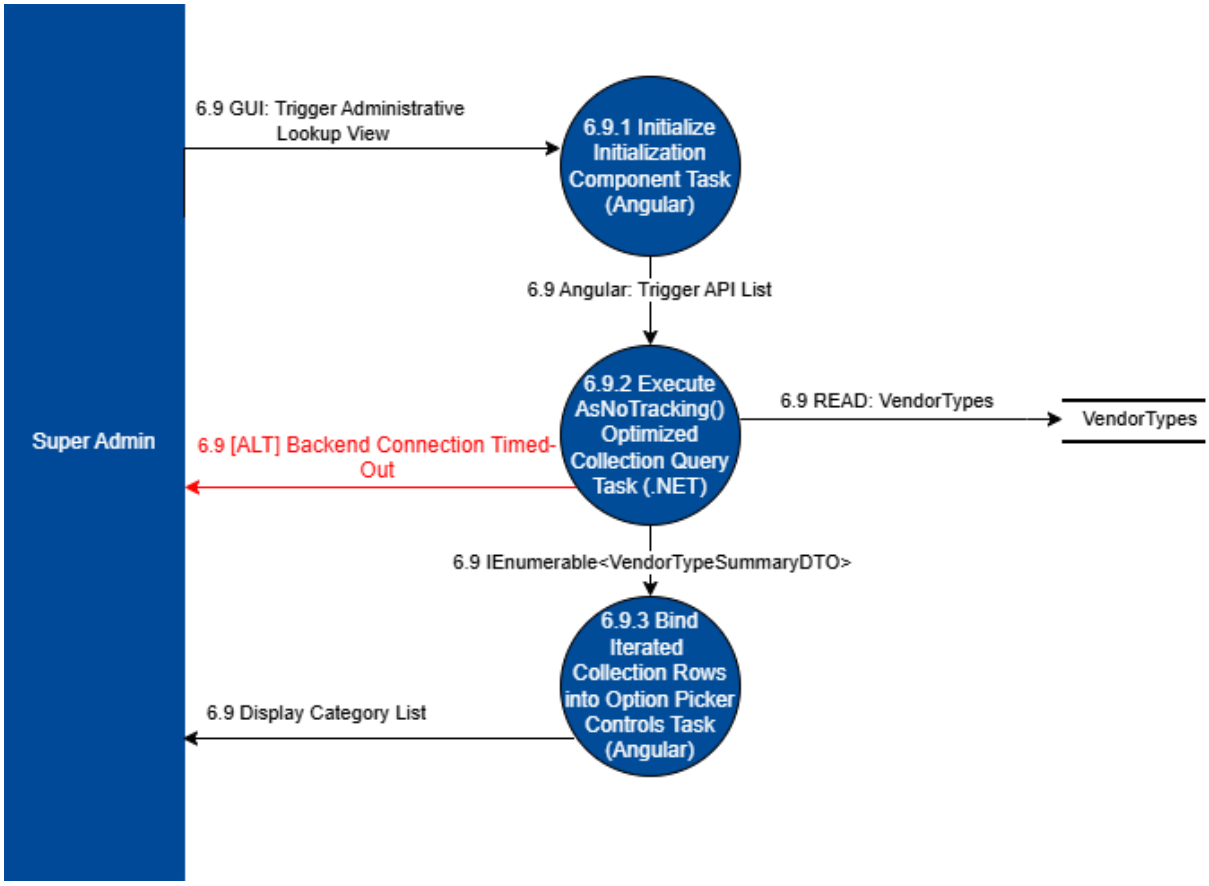


Figure 153: Technical Primitive Level Diagram 6.9 View Vendor Types

2.6.5.10 Reactivate Vendor Type: Use Case 6.10

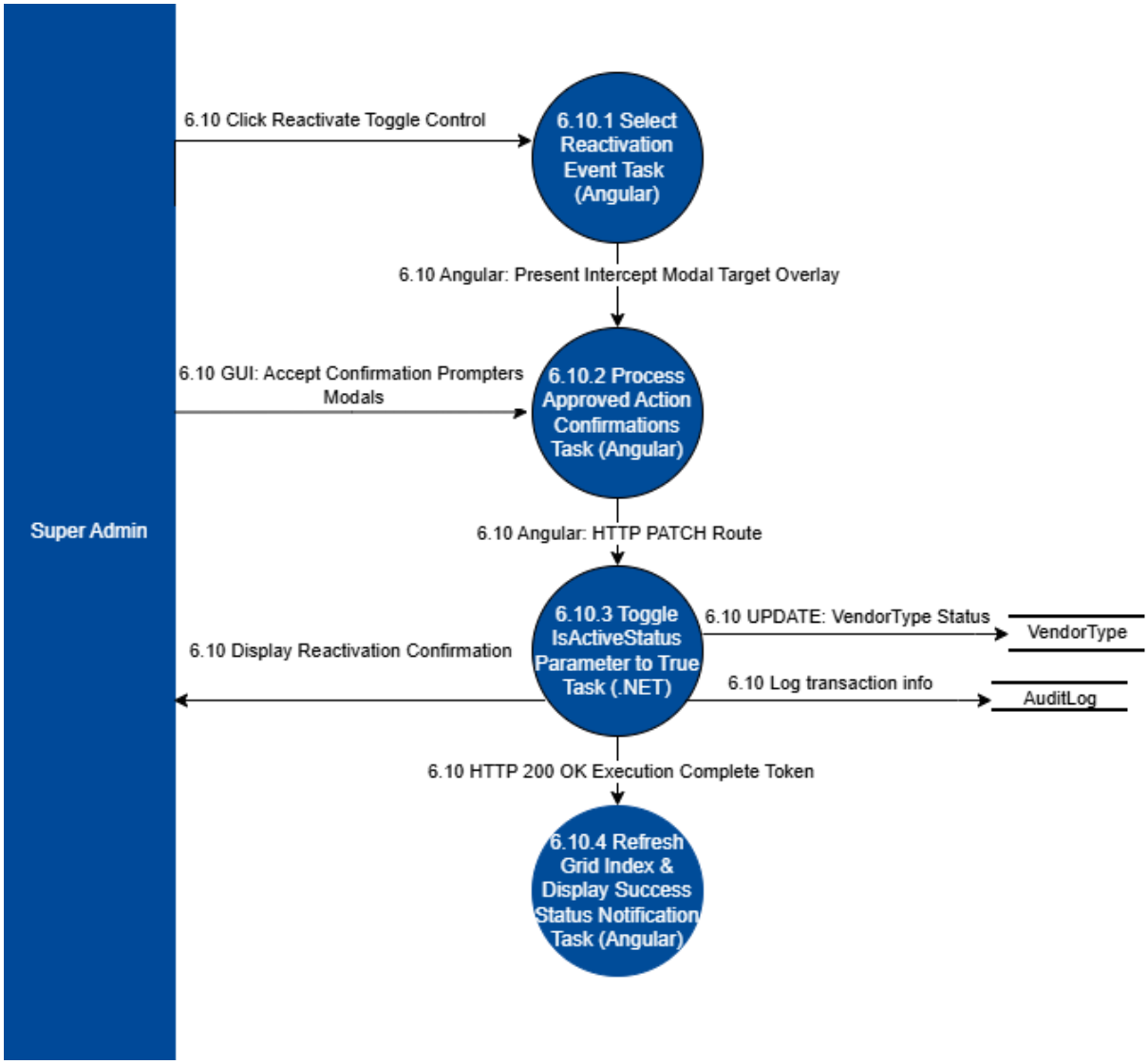


Figure 154: Technical Primitive Level Diagram 6.10 Reactivate Vendor Type

2.6.5.11 Deactivate Vendor Type: Use Case 6.11

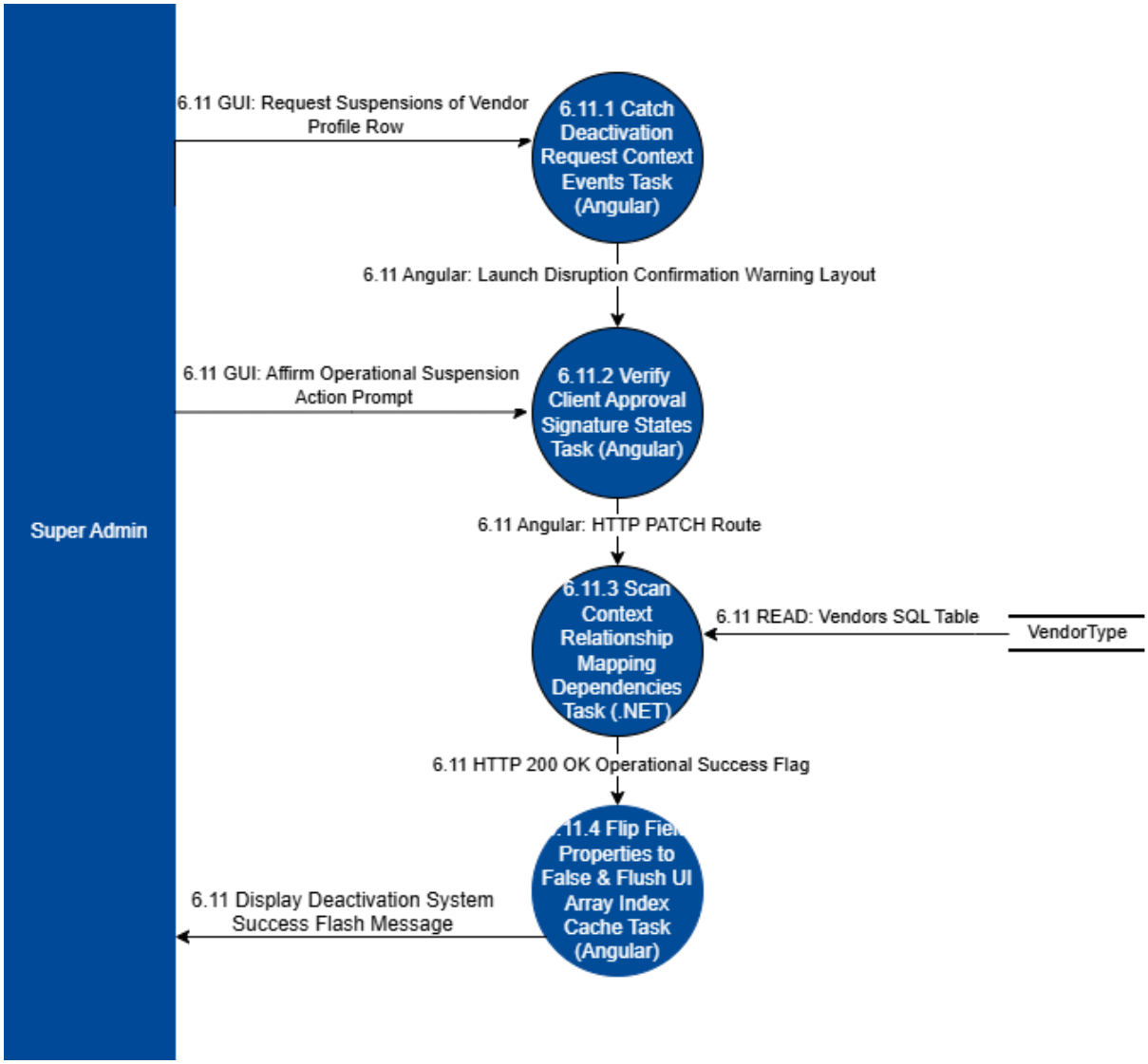


Figure 155: Technical Primitive Level Diagram 6.11 Deactivate Vendor Type

2.6.6 Project Management Subsystem Primitive Diagrams

2.6.6.1 Capture Project Feedback: Use Case 7.14

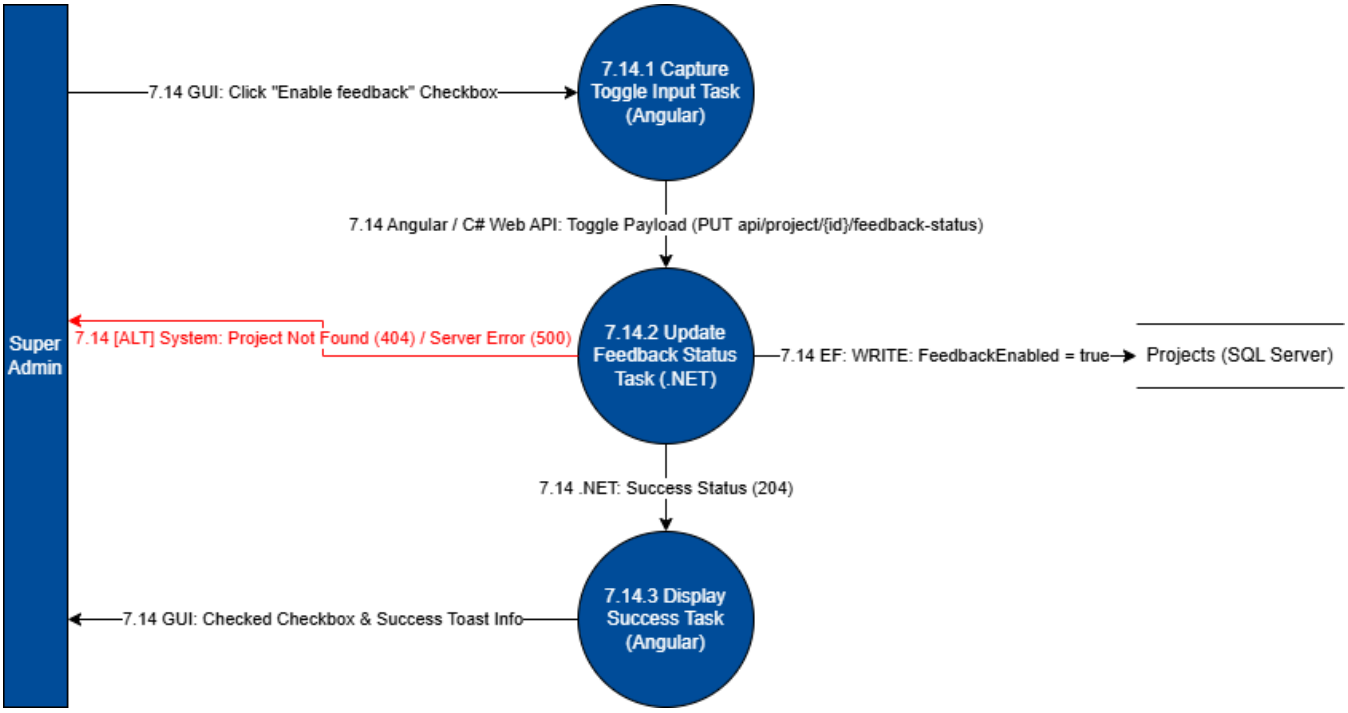


Figure 156: Technical Primitive Level Diagram 7.14 Capture Project Feedback

2.6.6.2 View Project Feedback: Use Case 7.15

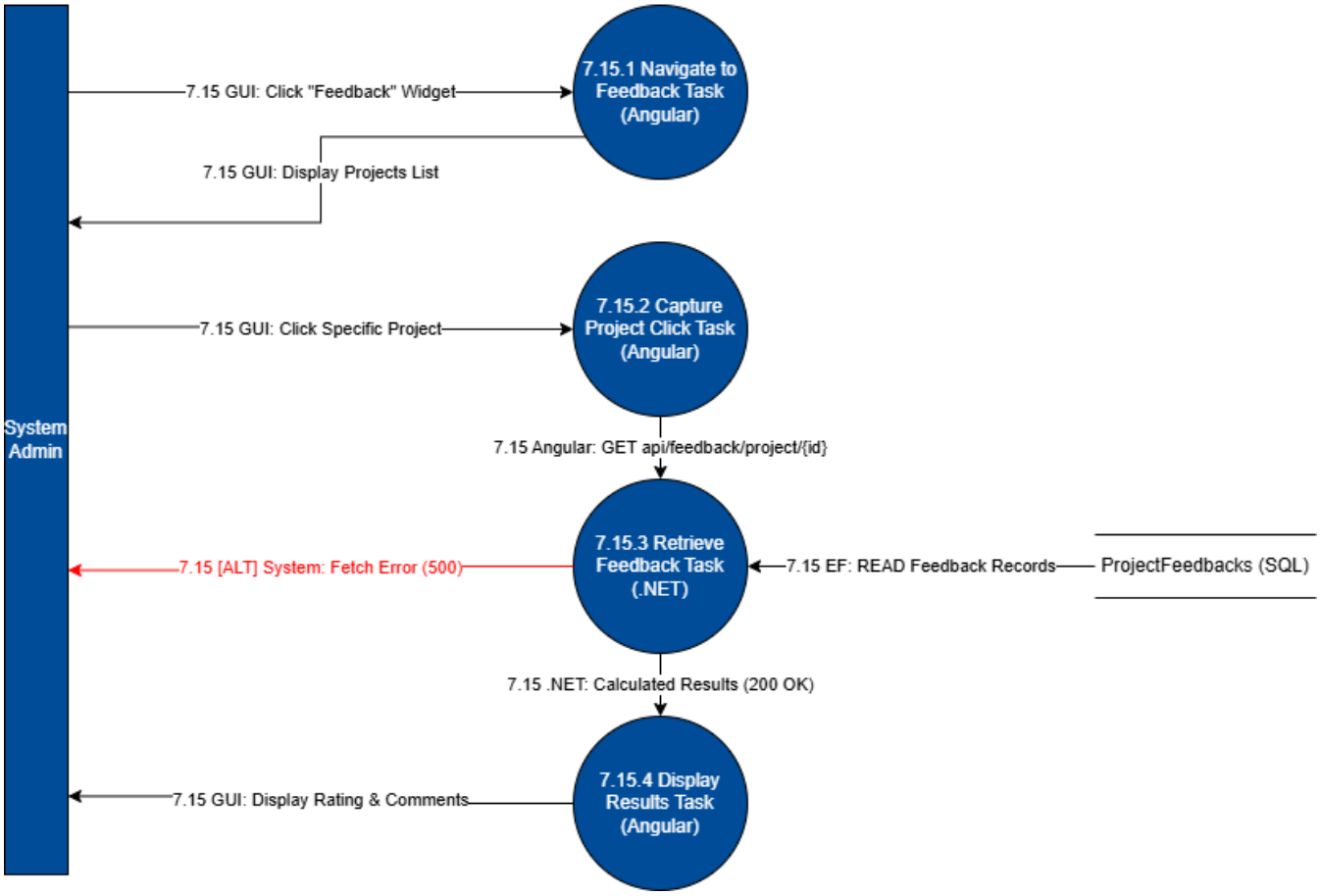


Figure 157: : Technical Primitive Level Diagram 7.15 View Project Feedback

2.6.6.3 Create Impact Feedback form: Use Case 7.16

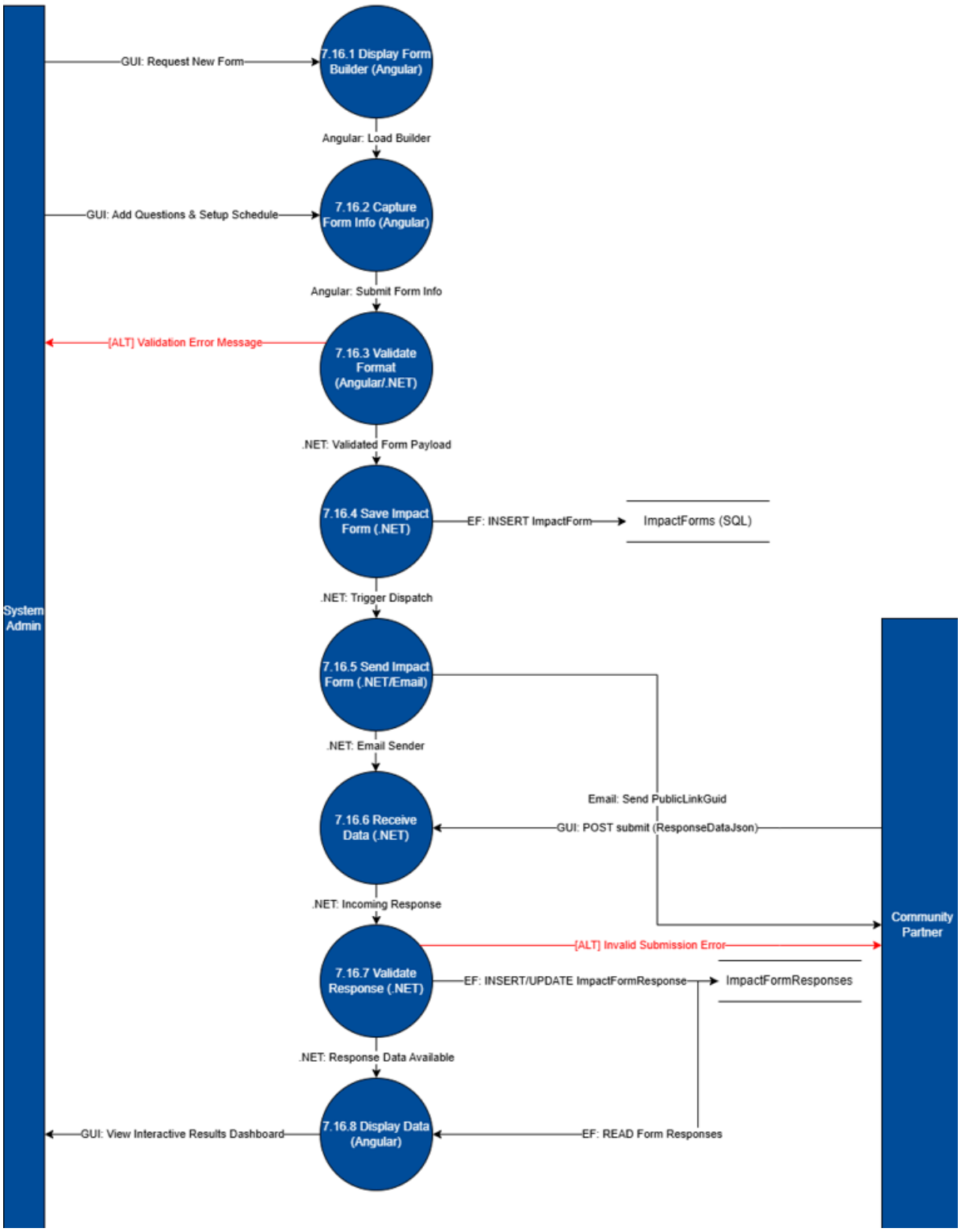


Figure 158: Technical Primitive Level Diagram 7.16 Create Impact Feedback Form

2.6.6.4 View Impact Forms: Use Case 7.17

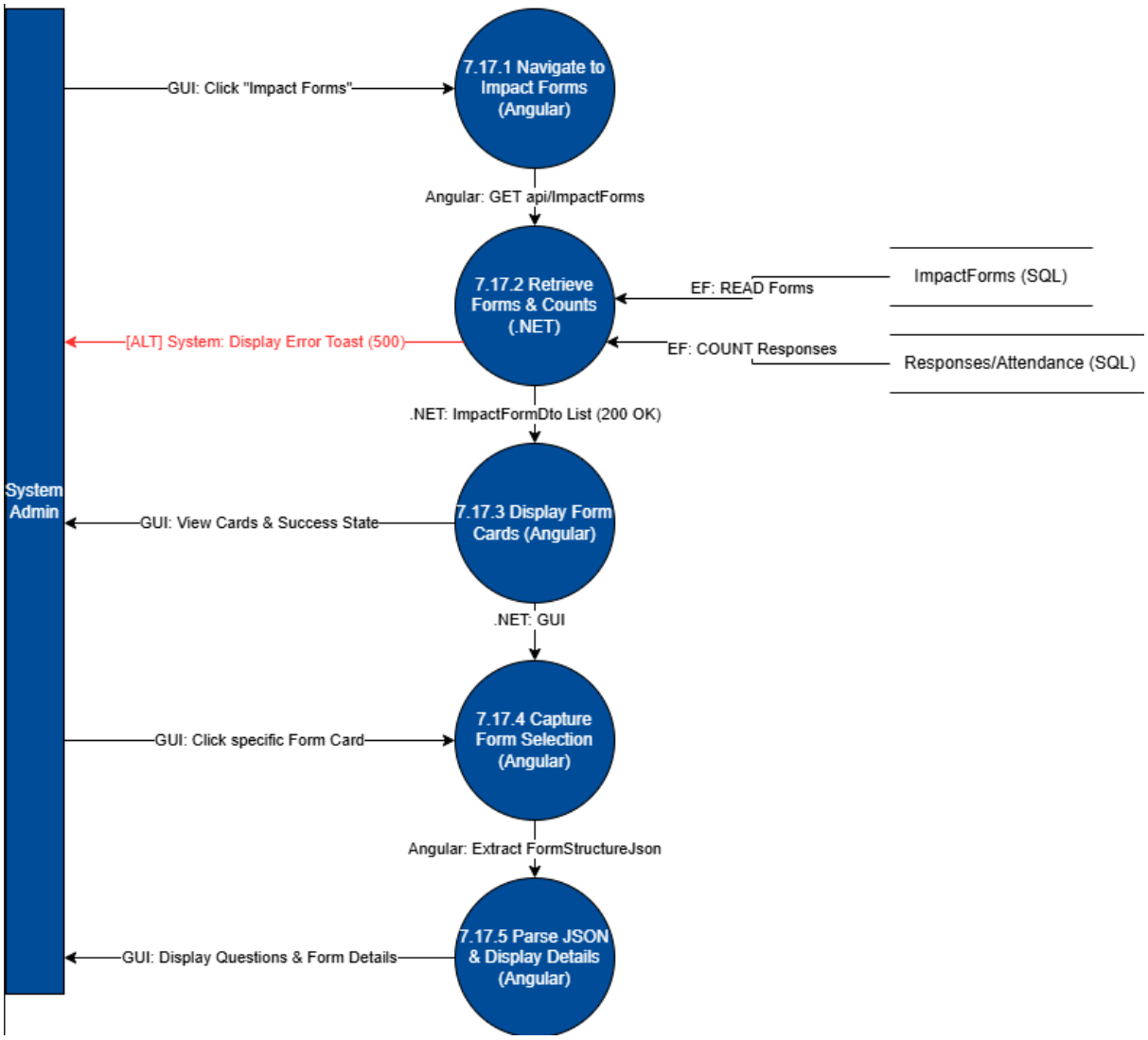


Figure 159: Technical Primitive Level Diagram 7.17 View Impact Forms

2.6.6.5 Update Impact Feedback Form: Use Case 7.18

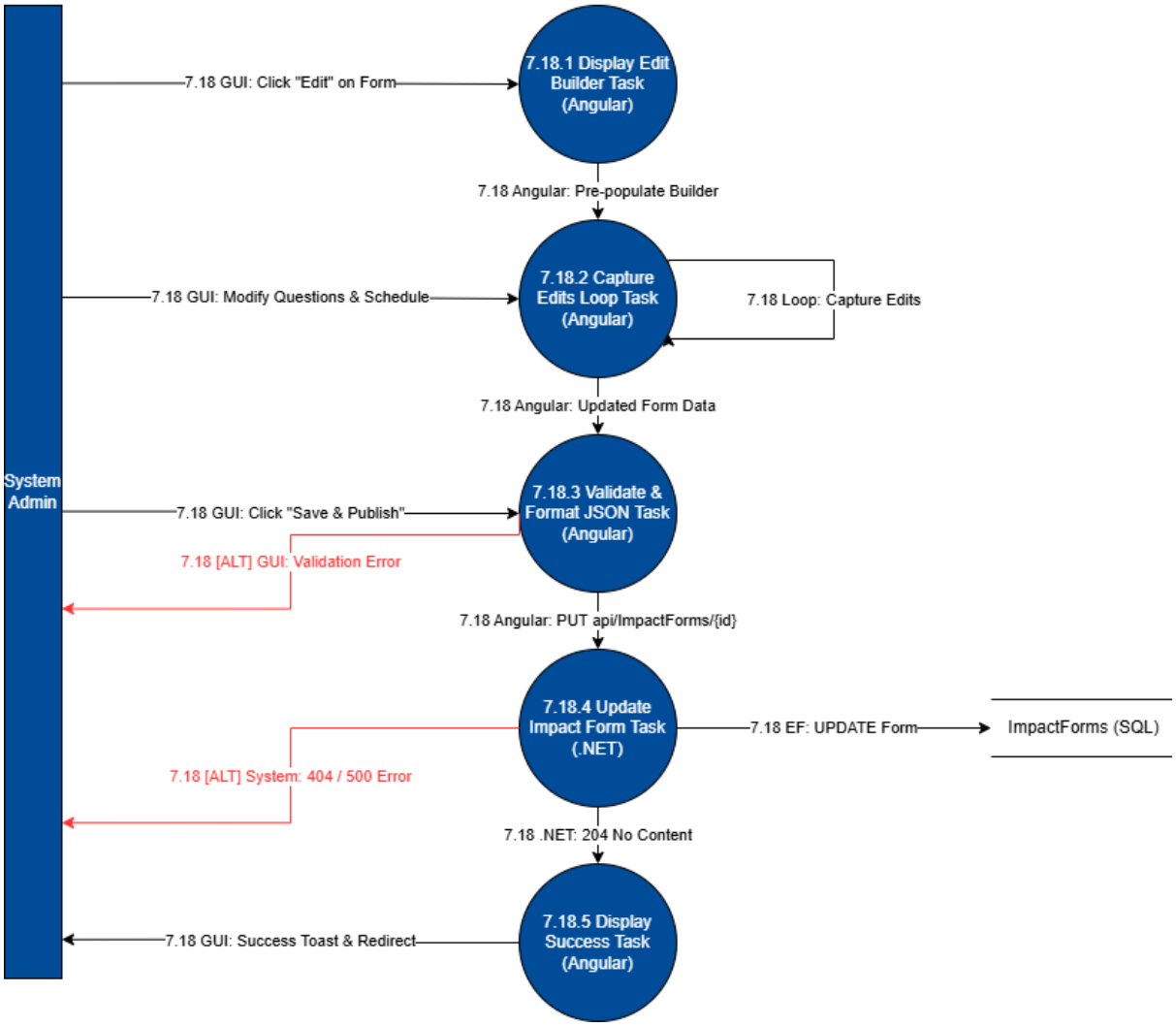


Figure 160: Technical Primitive Level Diagram 7.18 Update Impact Feedback Form

2.6.6.6 Deactivate Impact Feedback Form: Use Case 7.19

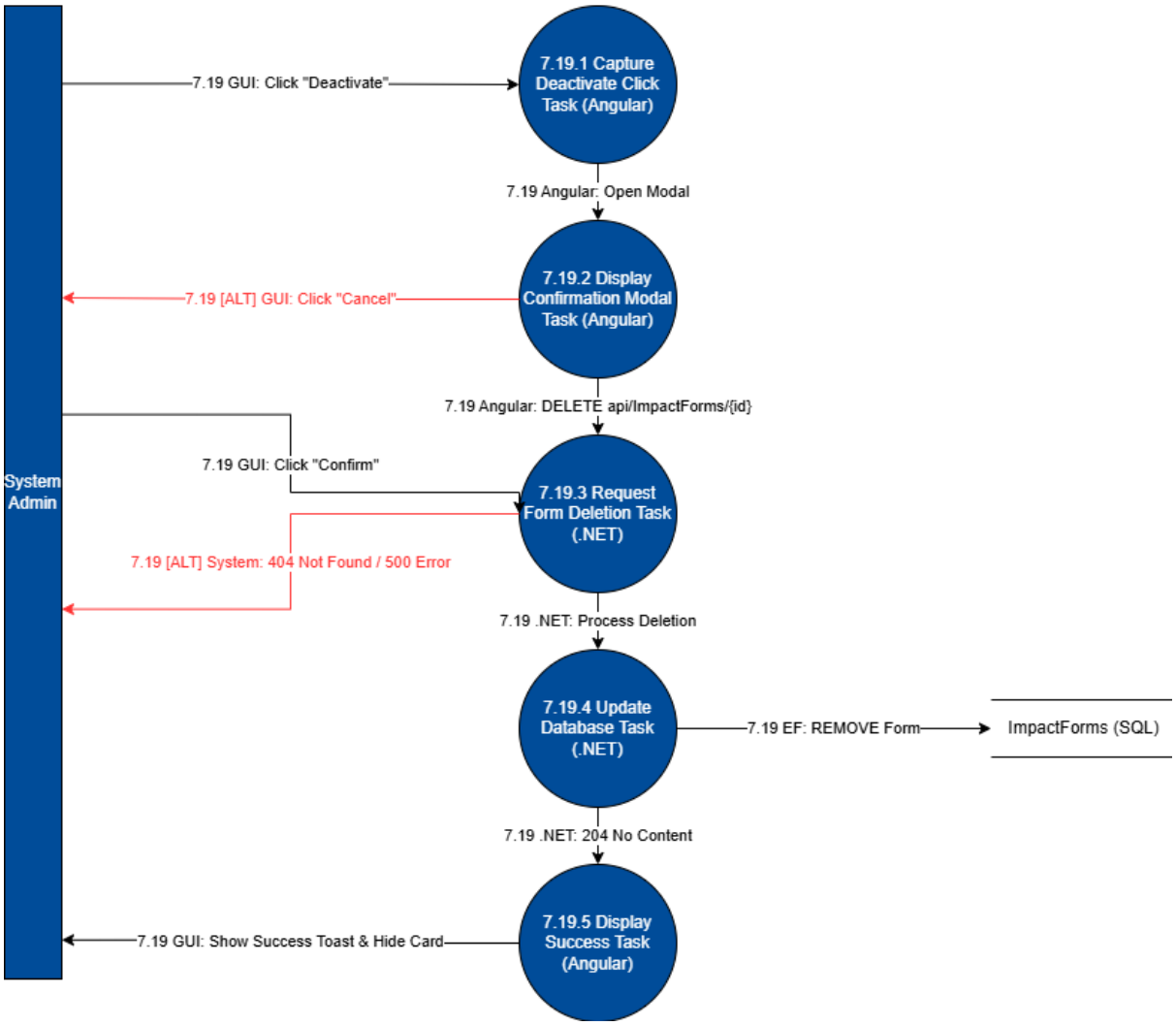


Figure 161: Technical Primitive Level Diagram 7.19 Deactivate Impact Feedback Form

2.6.6.7 Capture Impact Feedback: Use Case 7.20

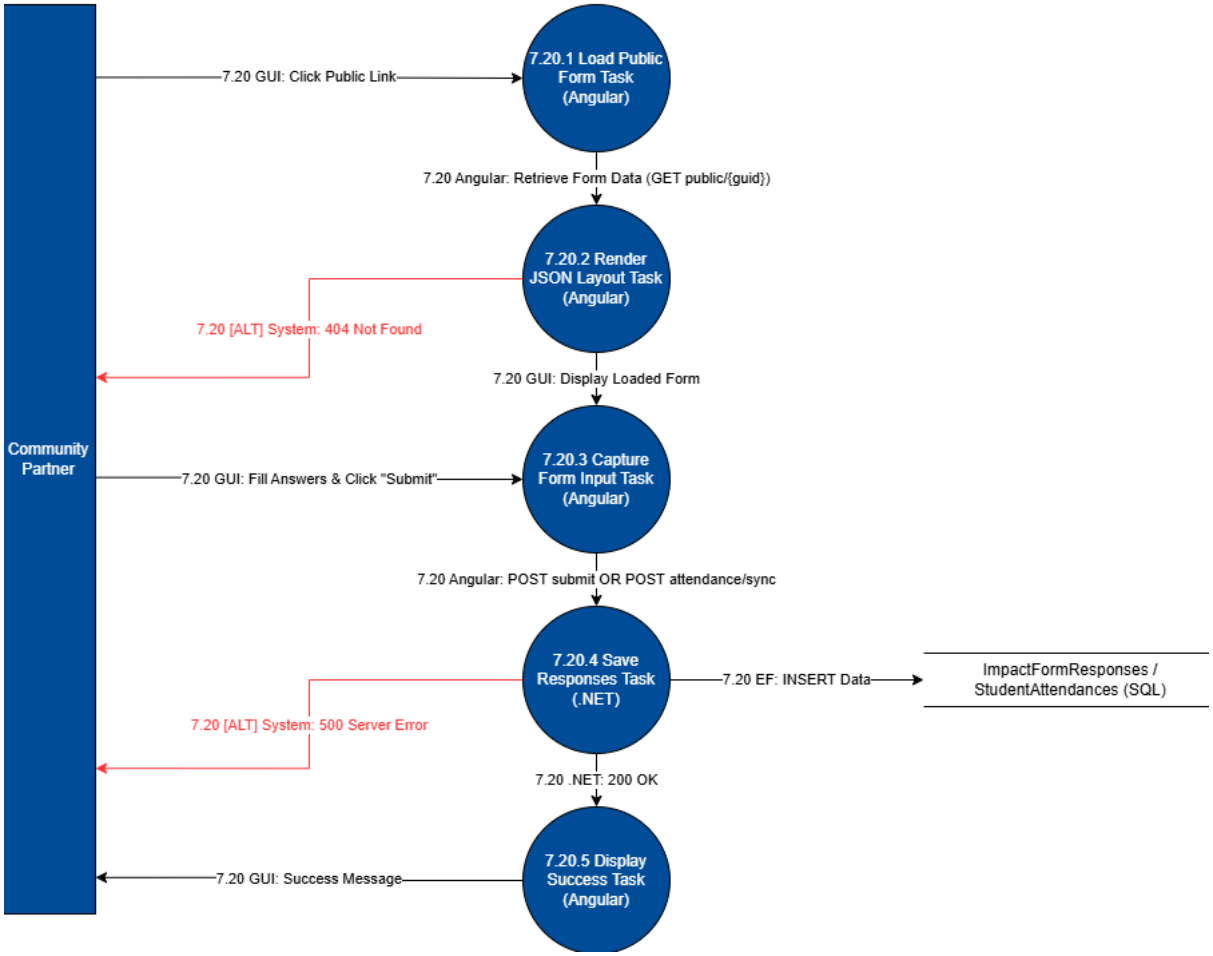


Figure 162: Technical Primitive Level Diagram 7.20 Capture Impact Feedback

2.6.6.8 View Impact Feedback: Use Case 7.21

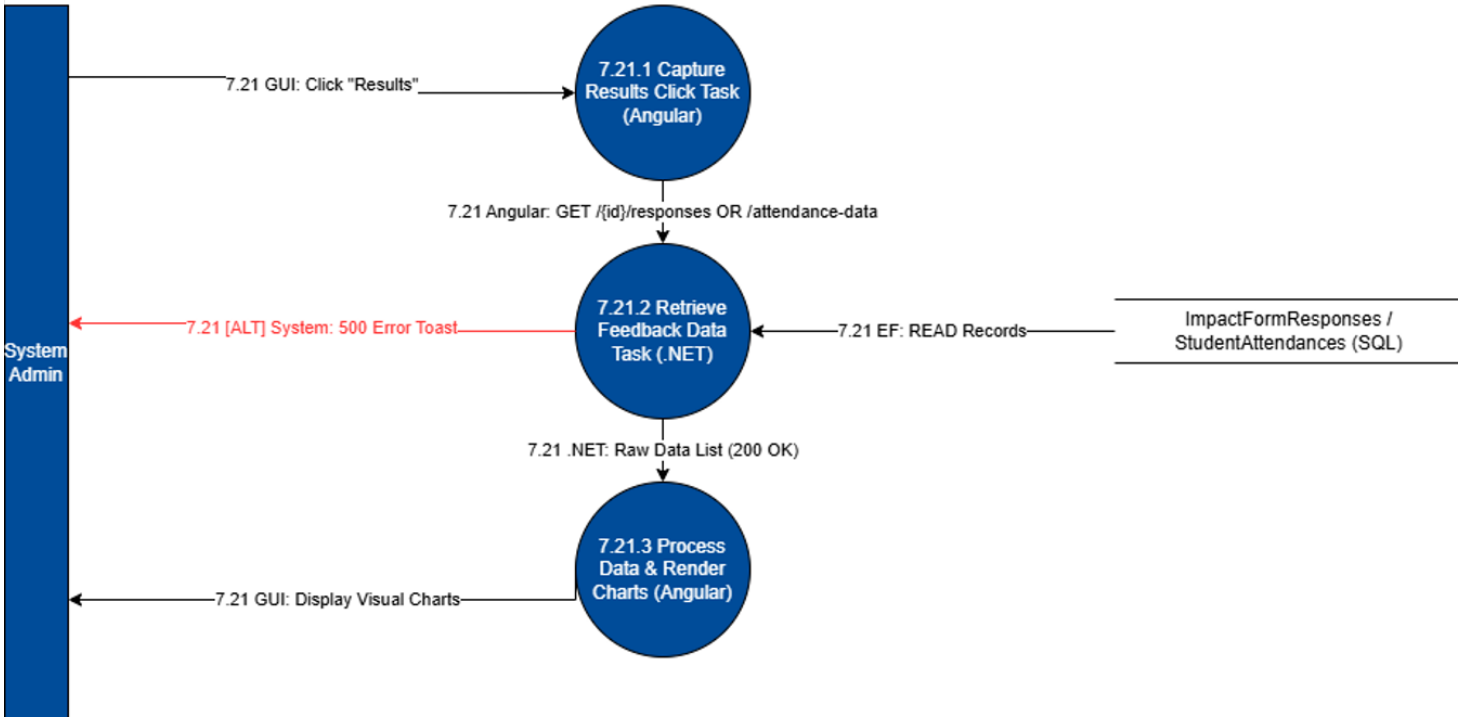


Figure 163: Technical Primitive Level Diagram 7.21 View Impact Feedback

2.6.6.9 Add Photo: Use Case 7.22

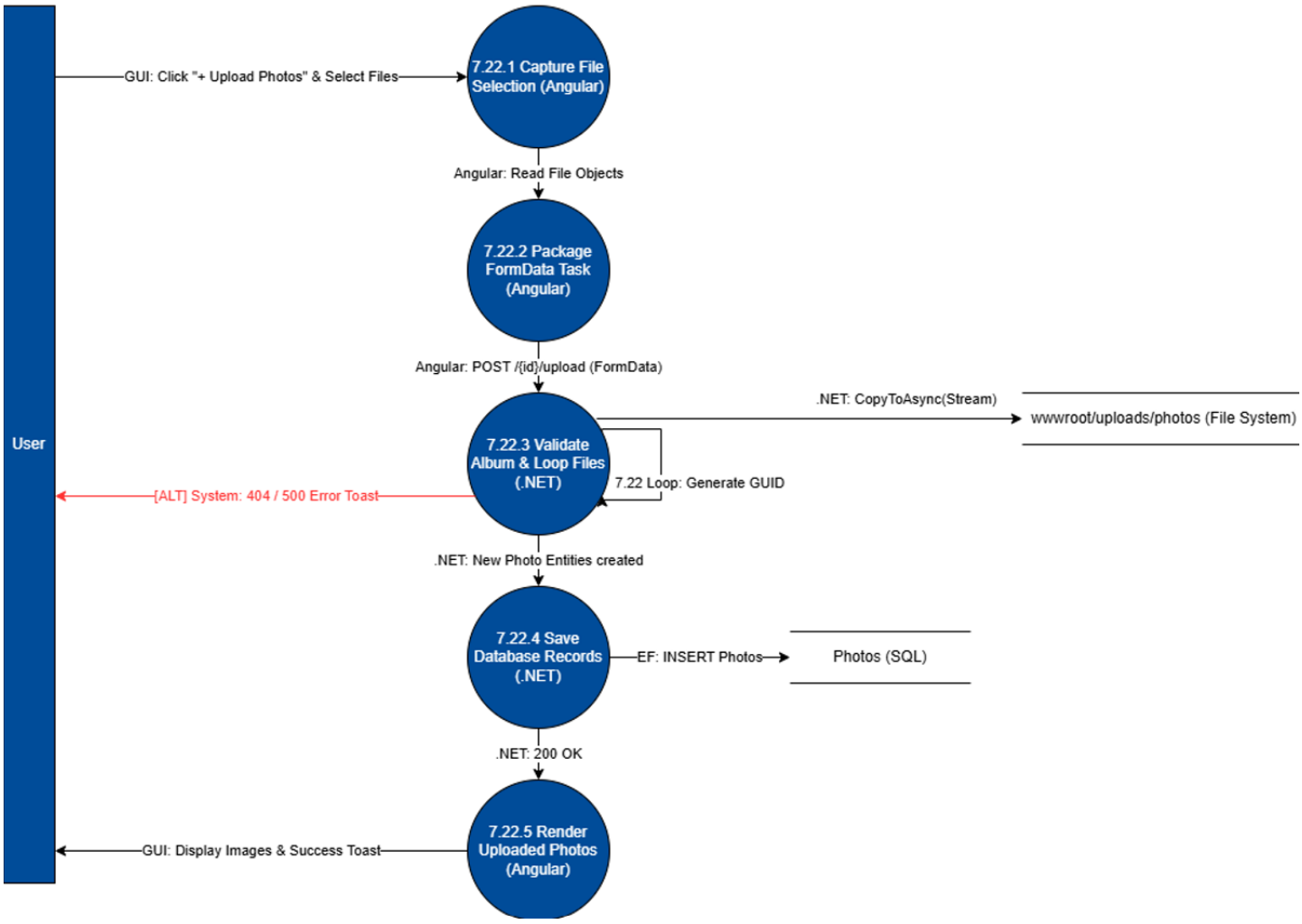


Figure 164: Technical Primitive Level Diagram 7.22 Add Photo

2.6.6.10 View Photo: Use Case 7.23

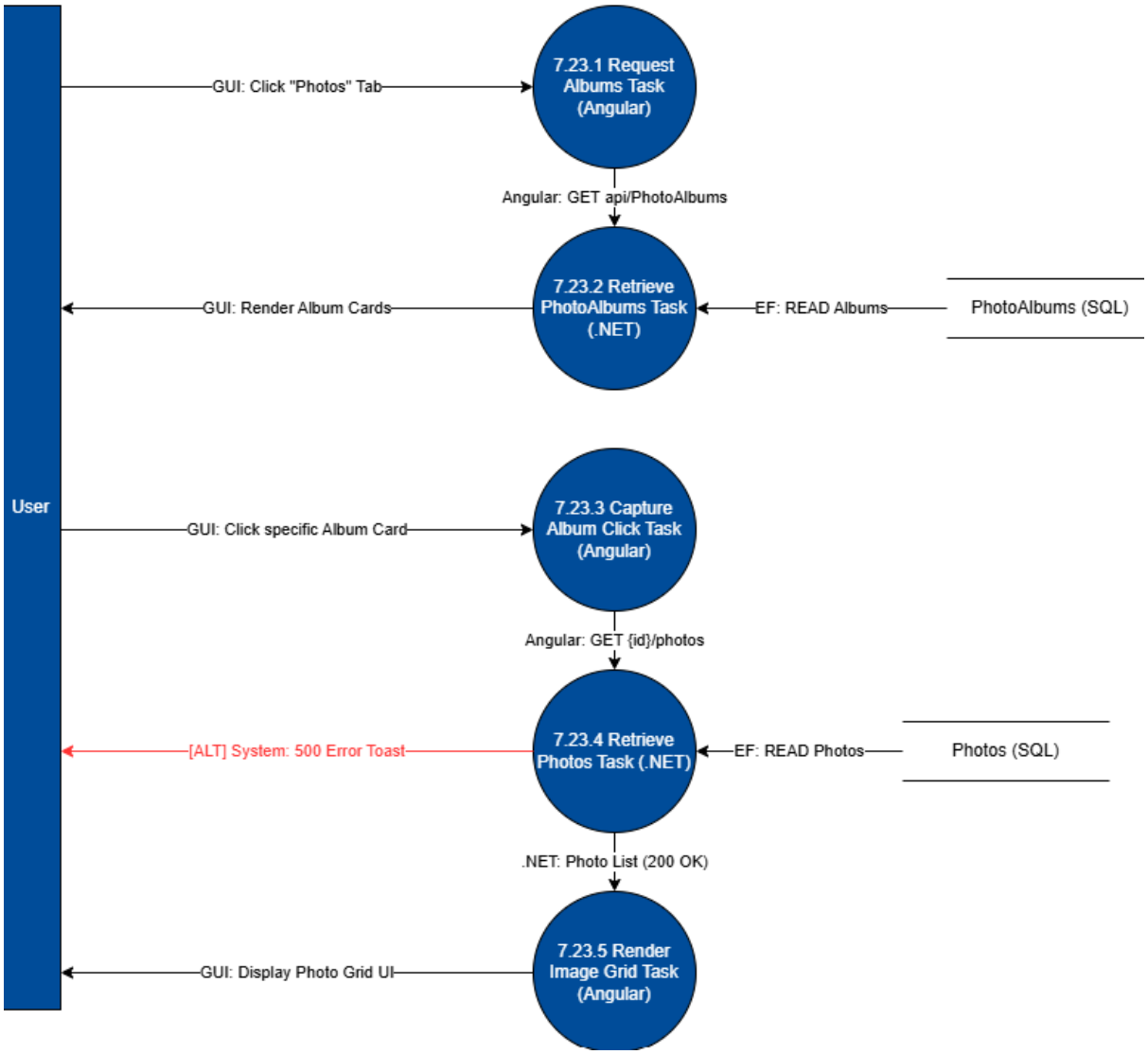


Figure 165: Technical Primitive Level Diagram 7.23 View Photo

2.6.6.11 Delete Photo: Use Case 7.24

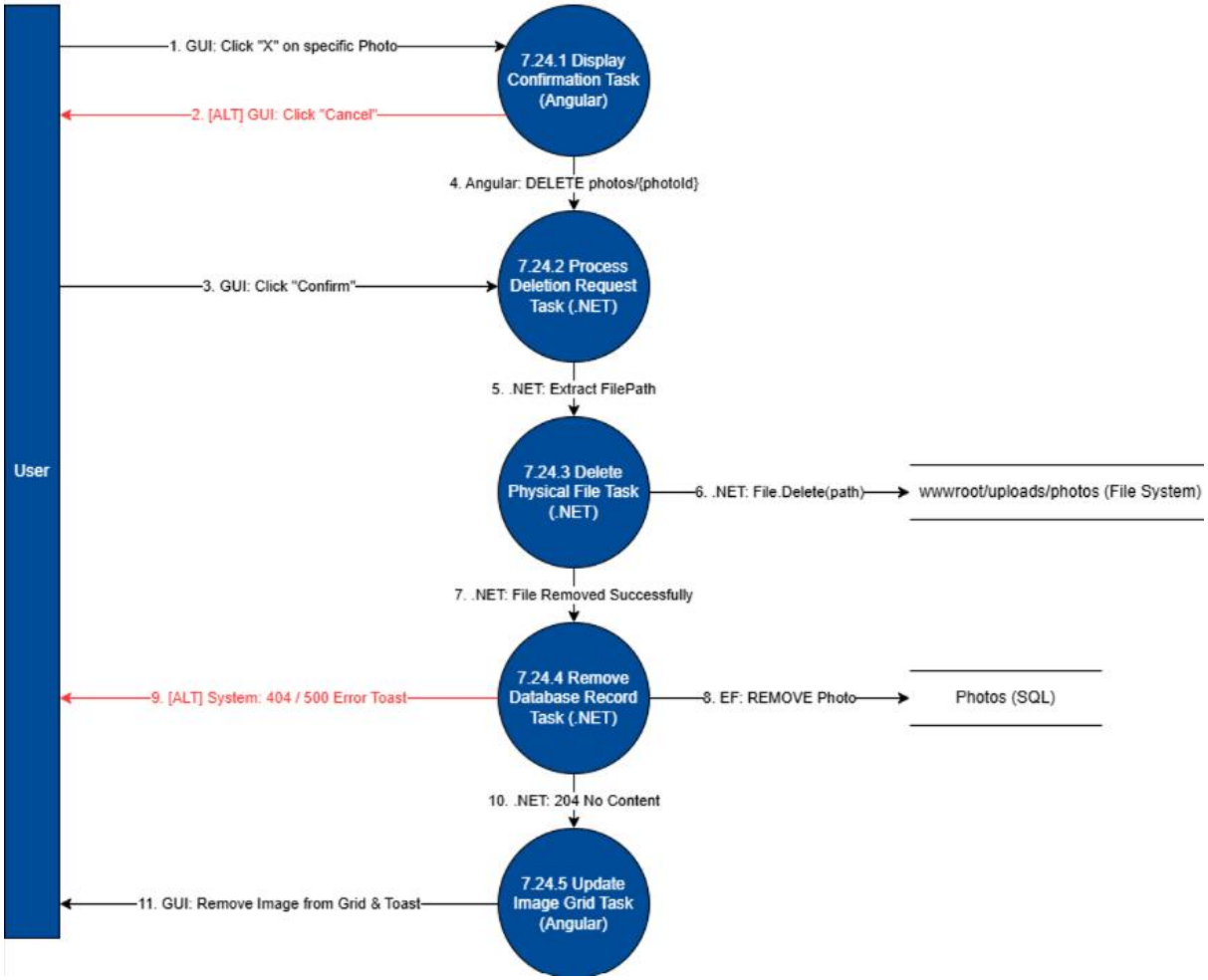


Figure 166: Technical Primitive Level Diagram 7.24 Delete Photo

2.6.7 K9 Subsystem Primitive Diagrams

2.6.7.1 Add Dog: Use Case 8.1

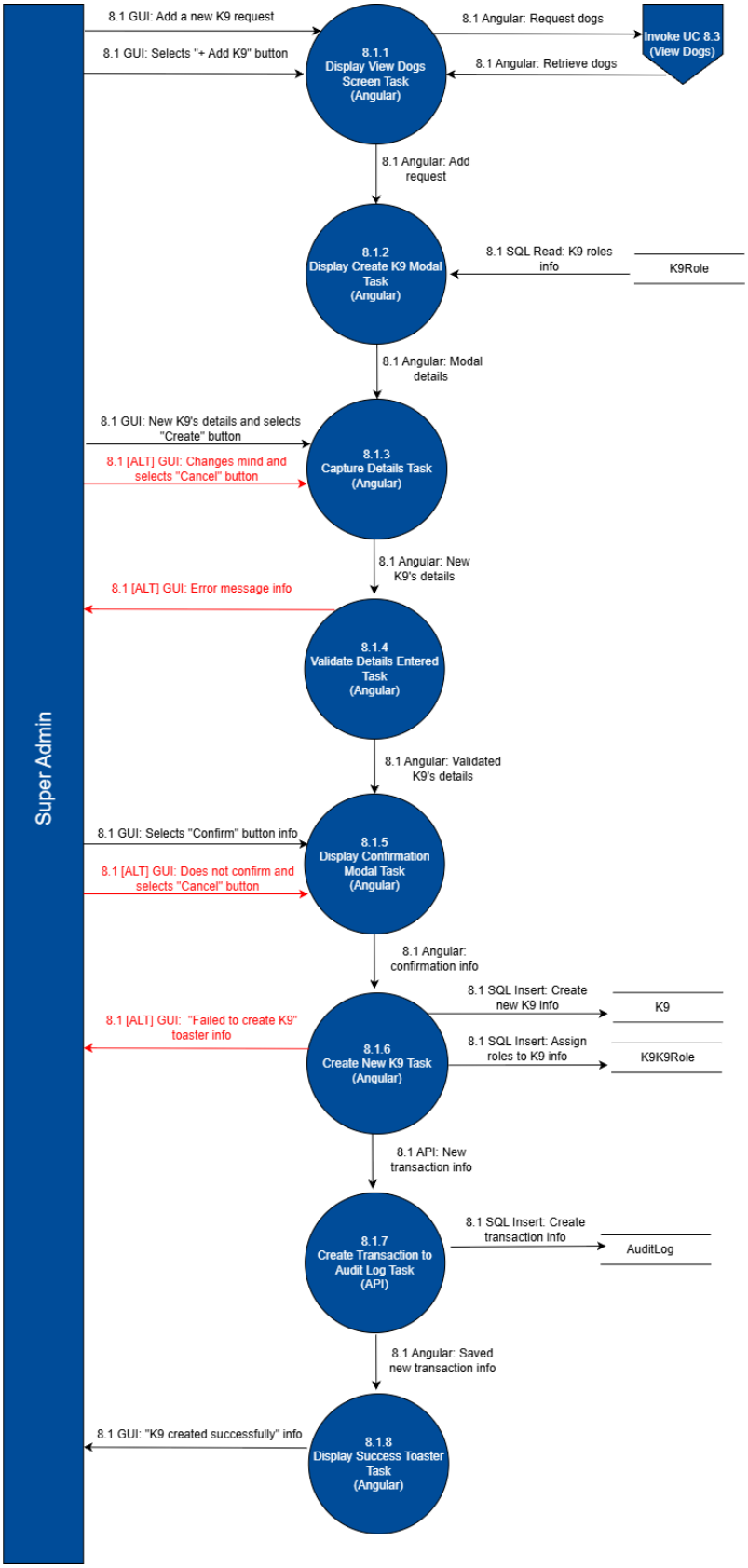


Figure 167: Technical Primitive Level Diagram 8.1 Add Dog

2.6.7.2 View Dog: Use Case 8.2

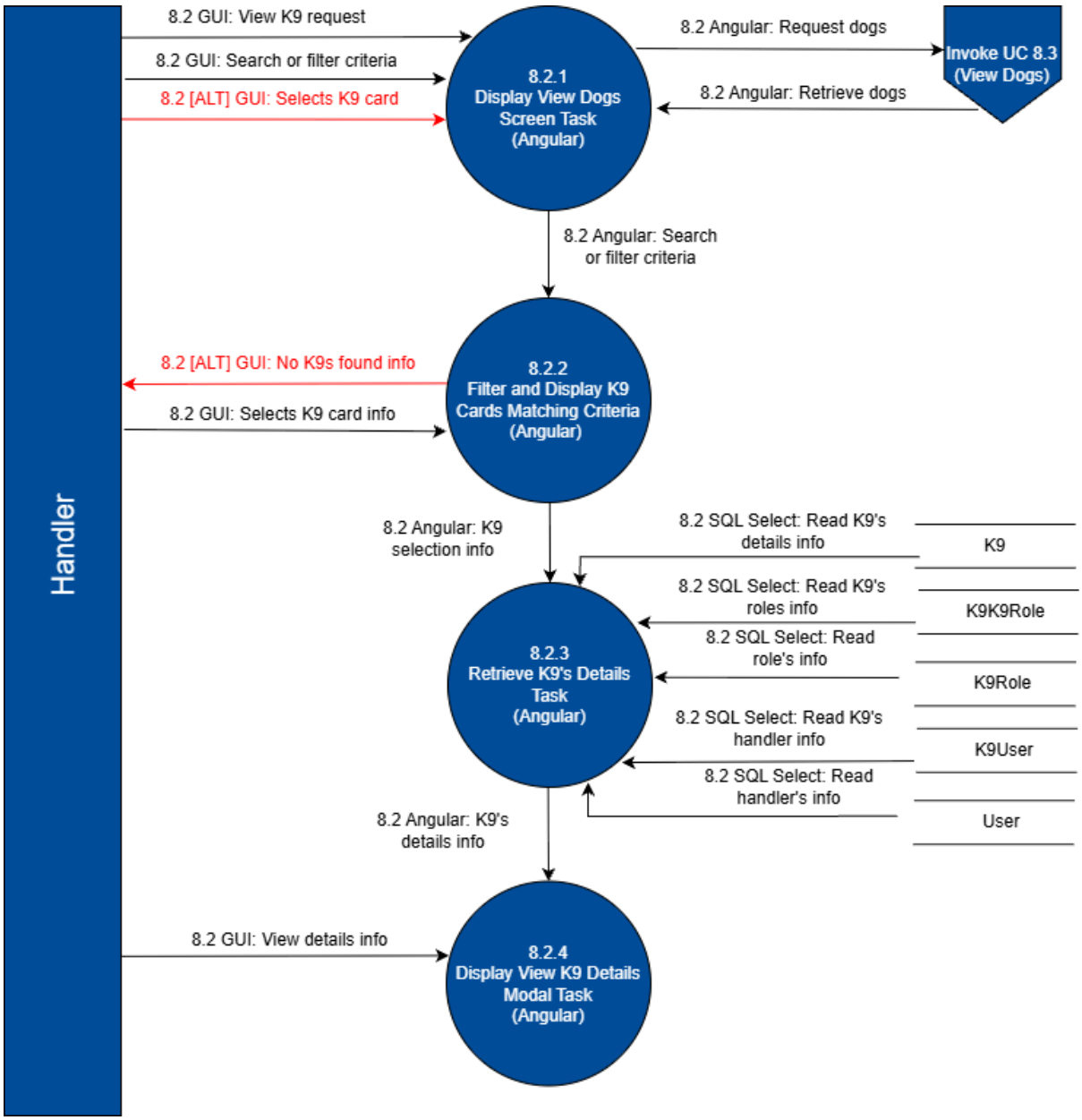


Figure 168: Technical Primitive Level Diagram 8.2 View Dog

2.6.7.3 View Dogs: Use Case 8.3

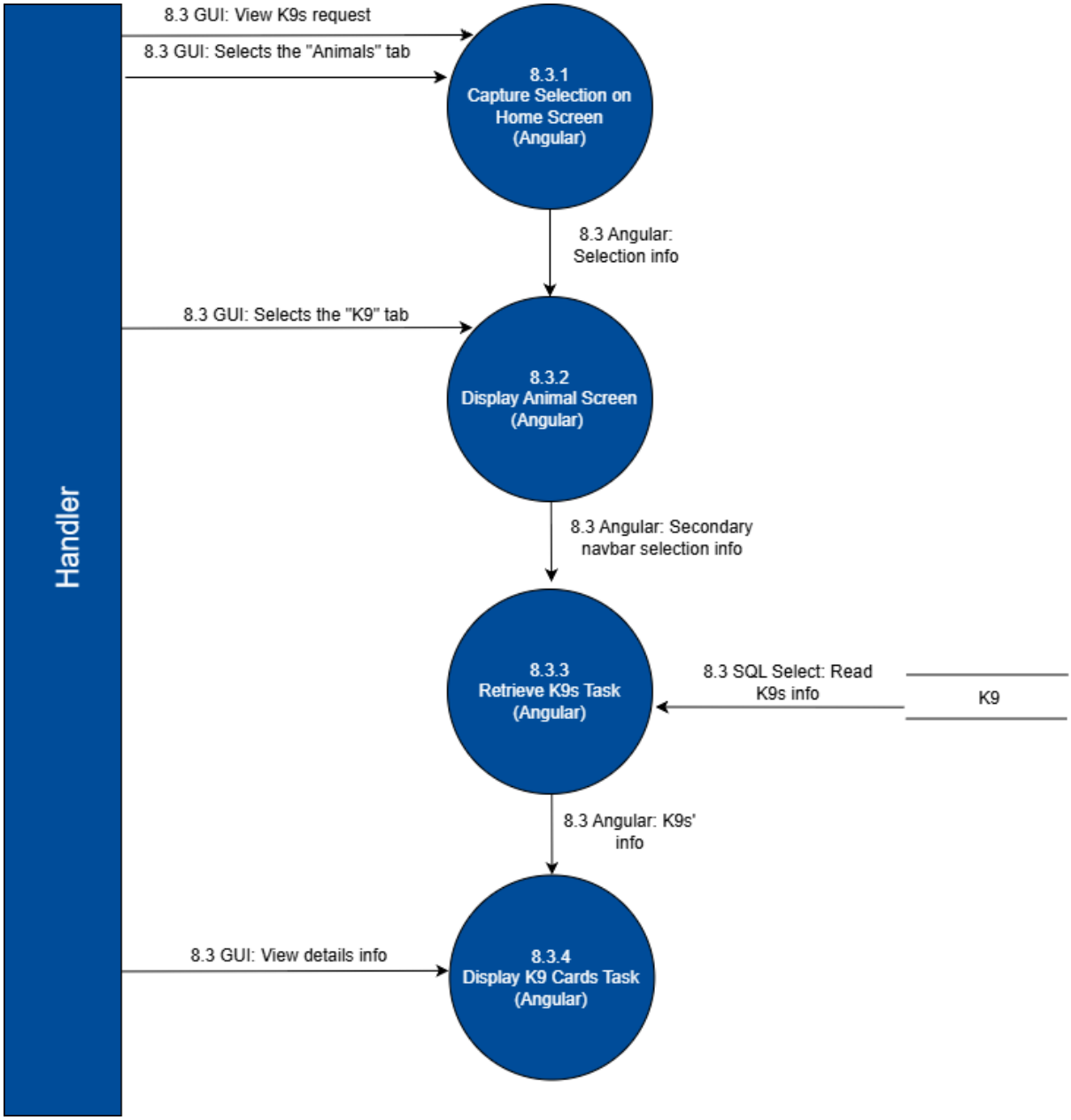


Figure 169: Technical Primitive Level Diagram 8.3 View Dogs

2.6.7.4 Update Dog: Use Case 8.4

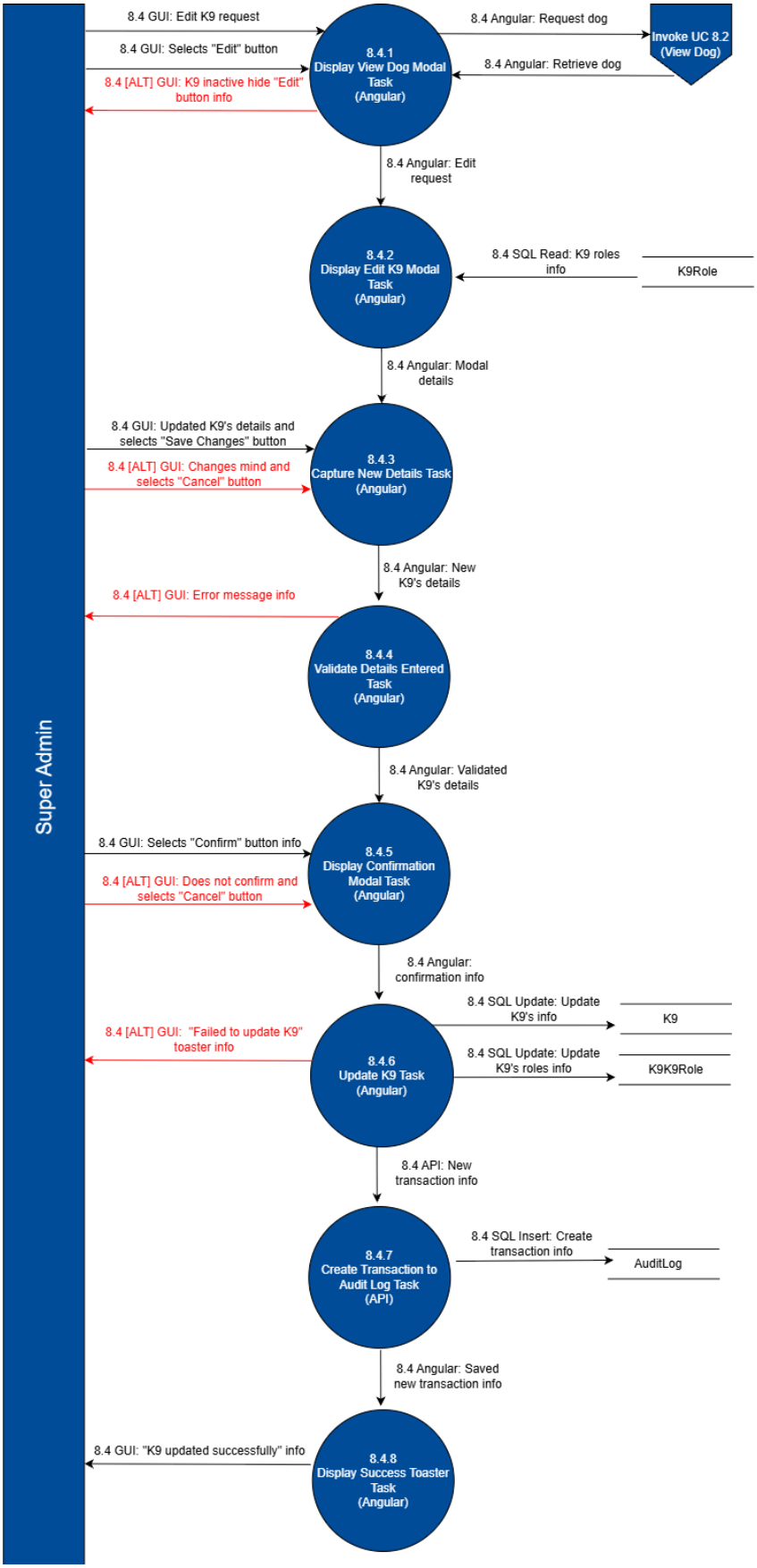


Figure 170: Technical Primitive Level Diagram 8.4 Update Dog

2.6.7.5 Deactivate Dog: Use Case 8.5

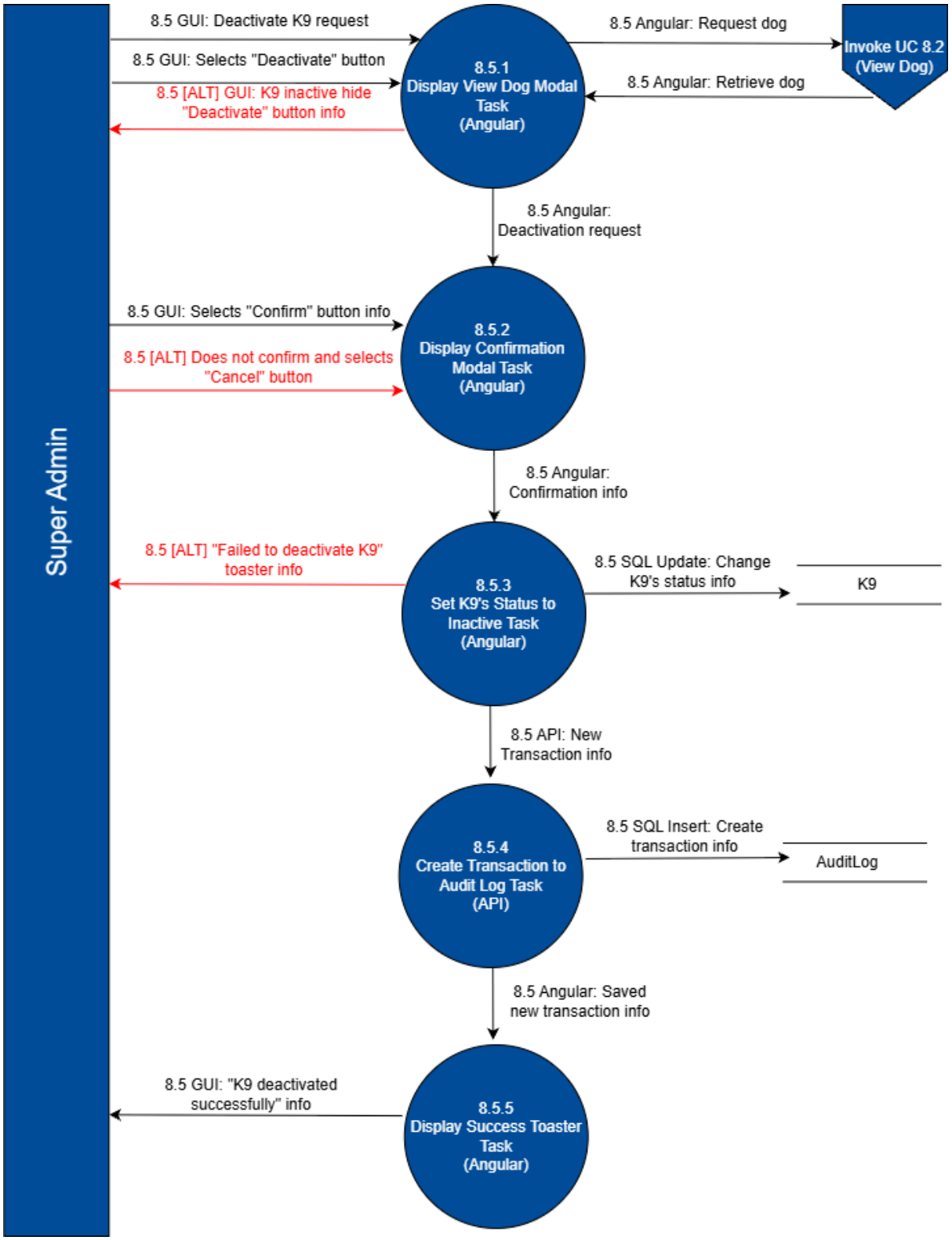


Figure 171: Technical Primitive Level Diagram 8.5 Deactivate Dog

2.6.7.6 Assign User to Dog: Use Case 8.6

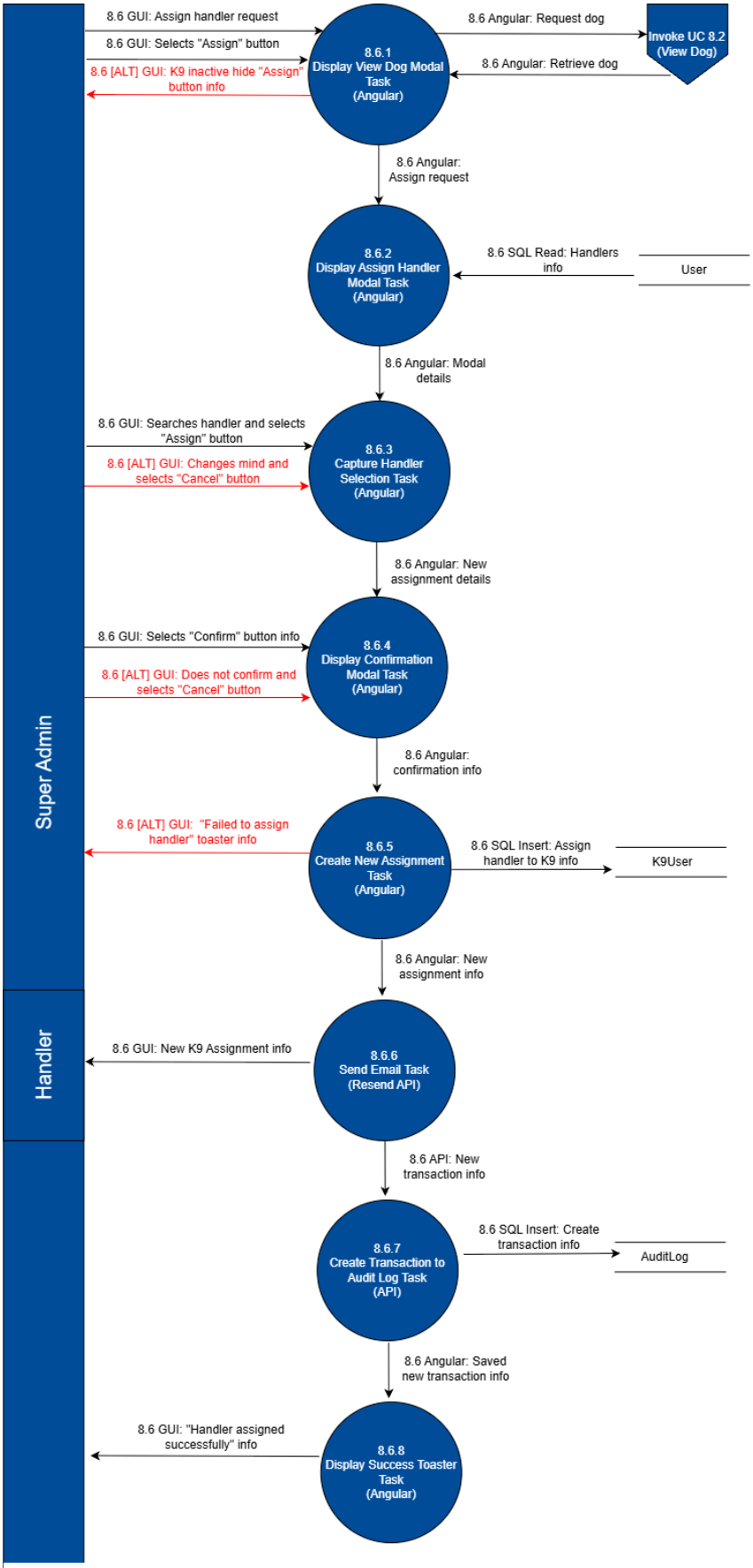


Figure 172: Technical Primitive Level Diagram 8.6 Assign User to Dog

2.6.7.7 View User-Dog Assignment: Use Case 8.7

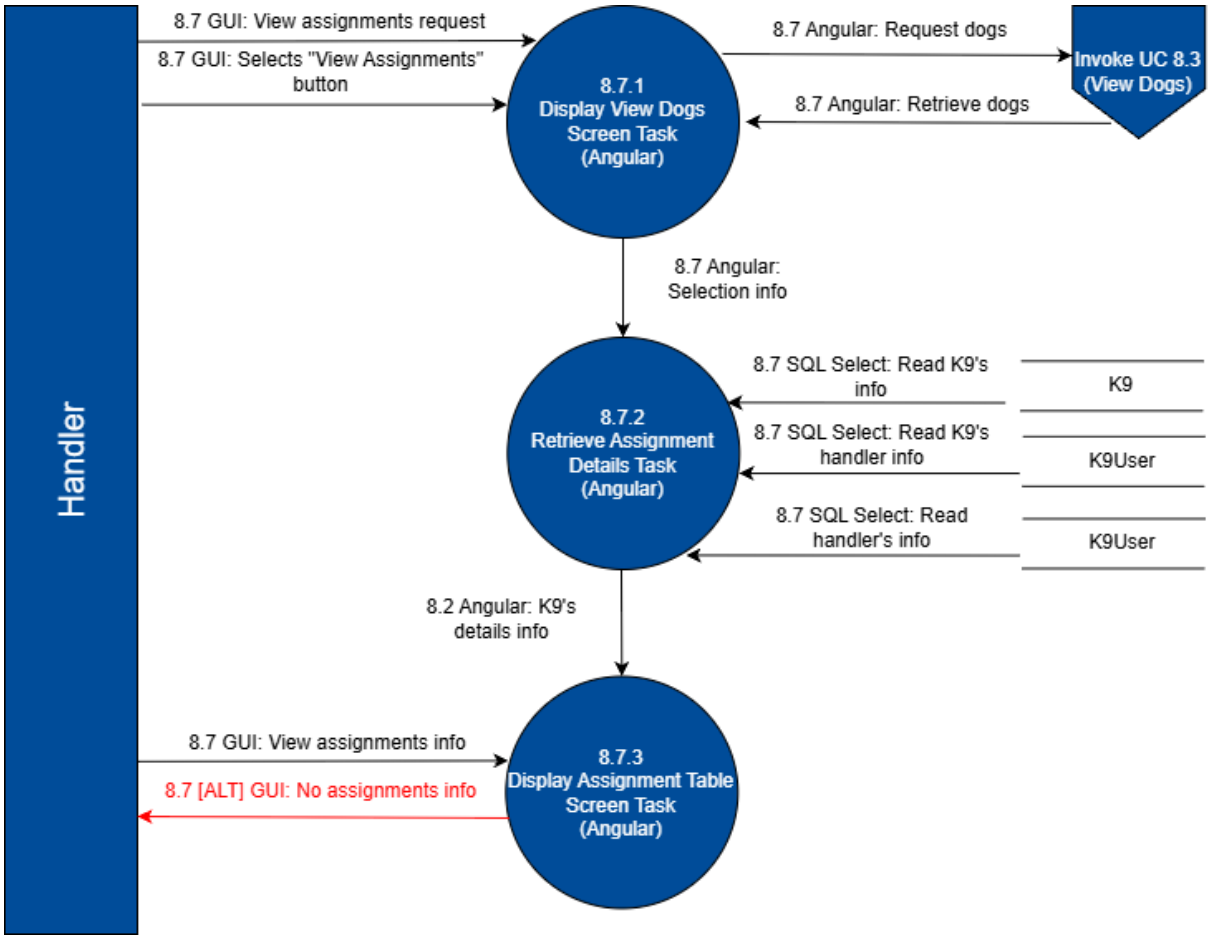


Figure 173: Technical Primitive Level Diagram 8.7 View User-Dog Assignment

2.6.7.8 Update User-Dog Assignment: Use Case 8.8

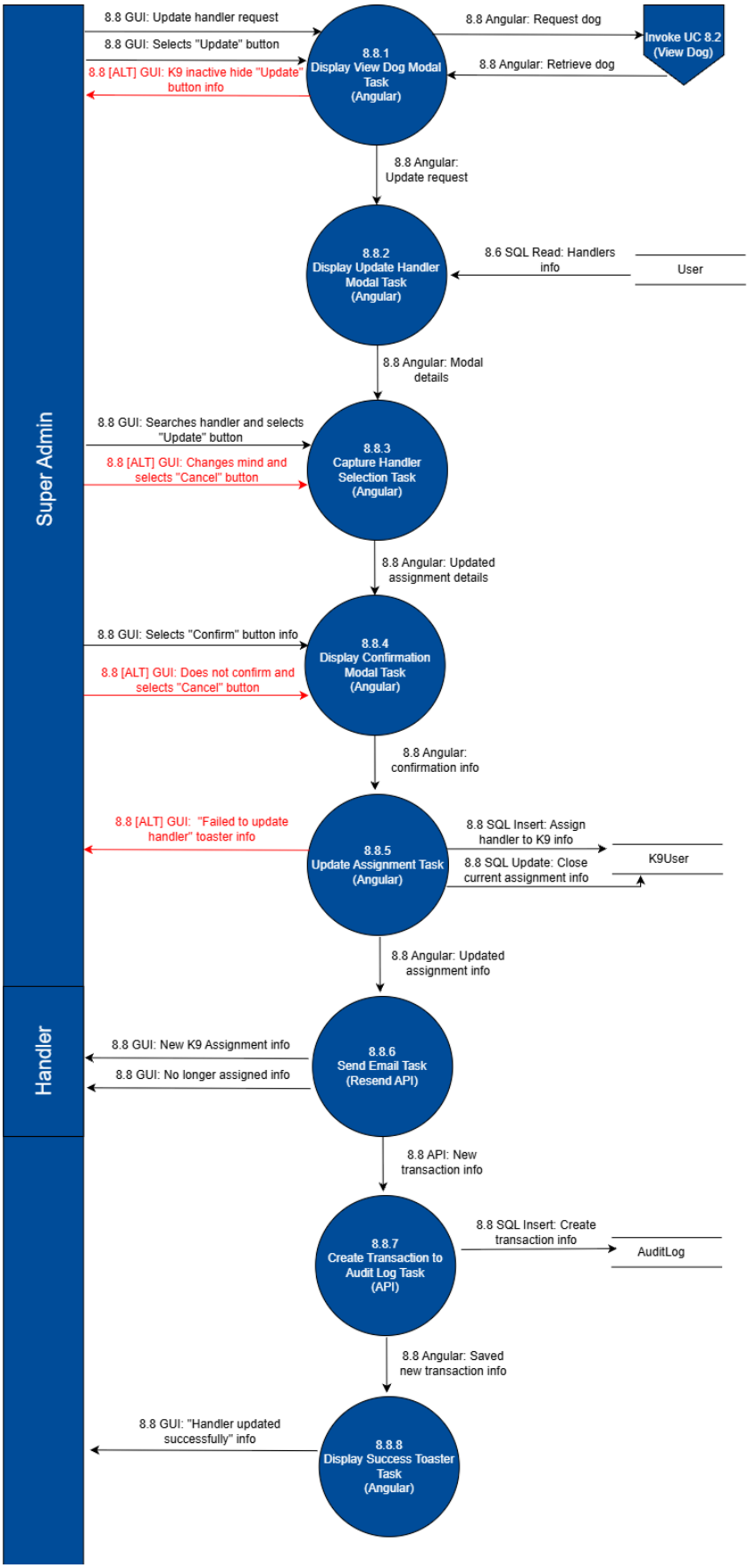


Figure 174: Technical Primitive Level Diagram 8.8 Update User-Dog Assignment

2.6.7.9 Reactivate User-Dog Assignment: Use Case 8.9

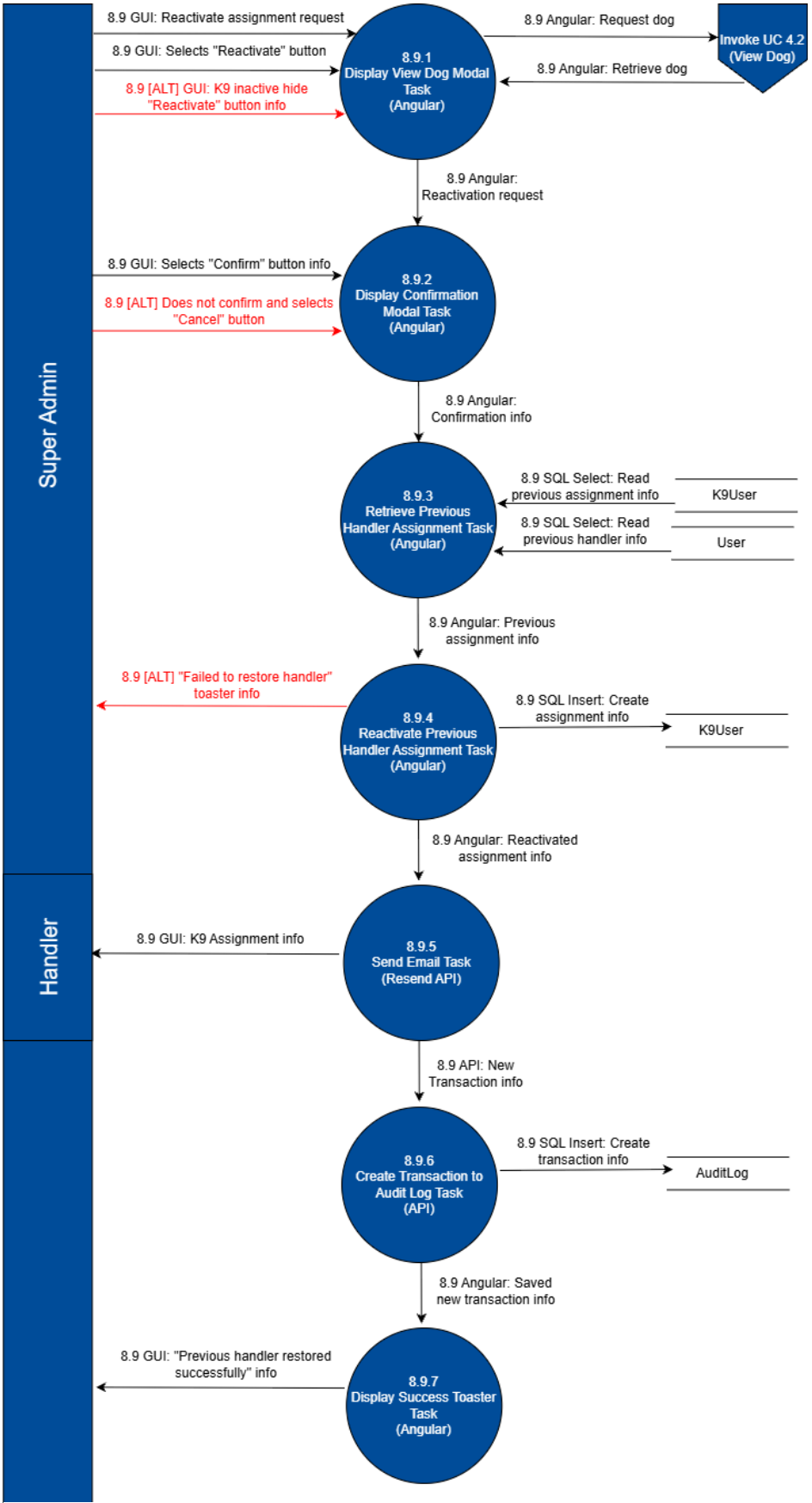


Figure 175: Technical Primitive Level Diagram 8.9 Reactivate User-Dog Assignment

2.6.7.10 Deactivate User-Dog Assignment: Use Case 8.10

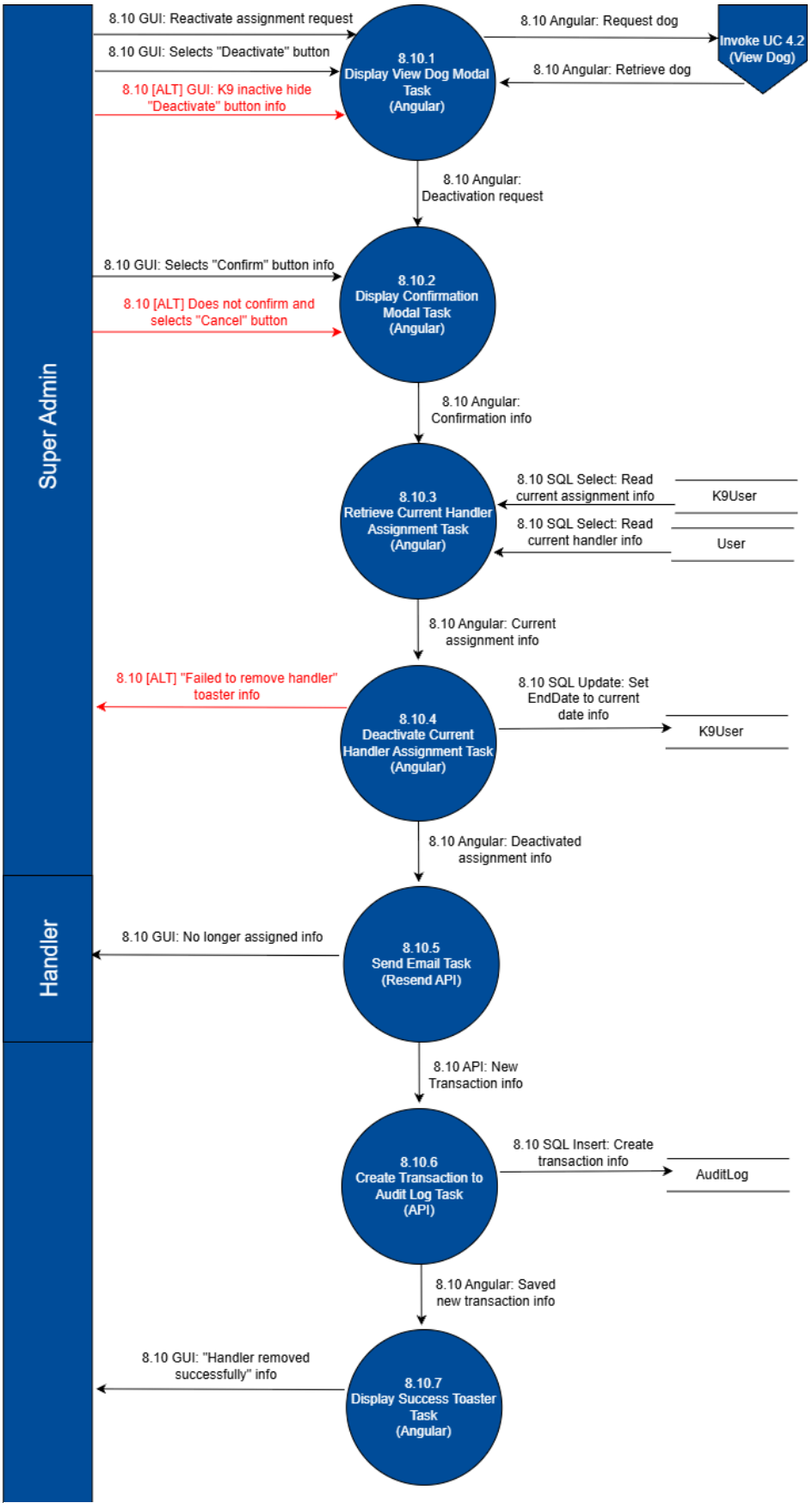


Figure 176: Technical Primitive Level Diagram 8.10 Deactivate User-Dog Assignment

2.6.7.11 Add K9 Role: Use Case 8.11

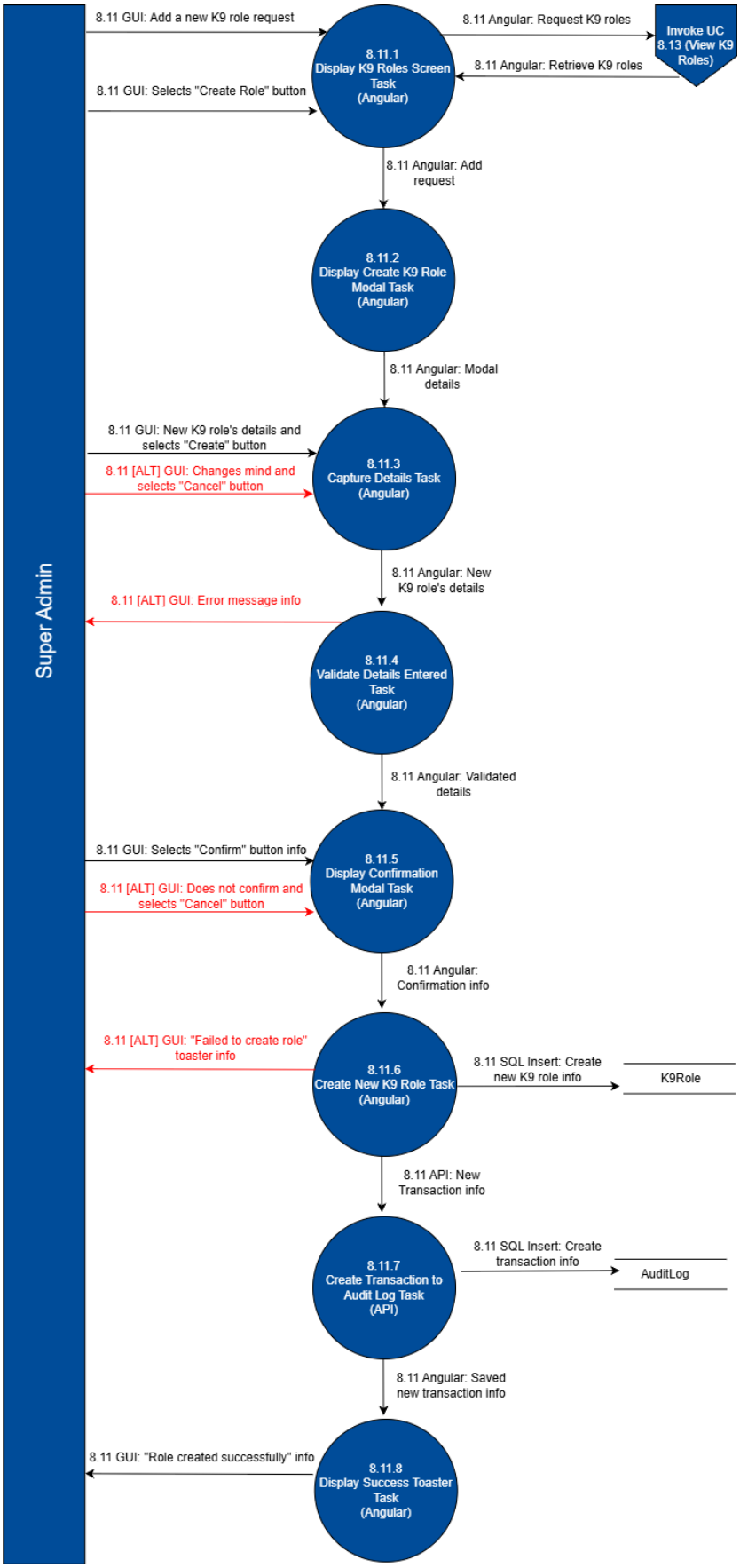


Figure 177: Technical Primitive Level Diagram 8.11 Add K9 Role

2.6.7.12 Update K9 Role: Use Case 8.12

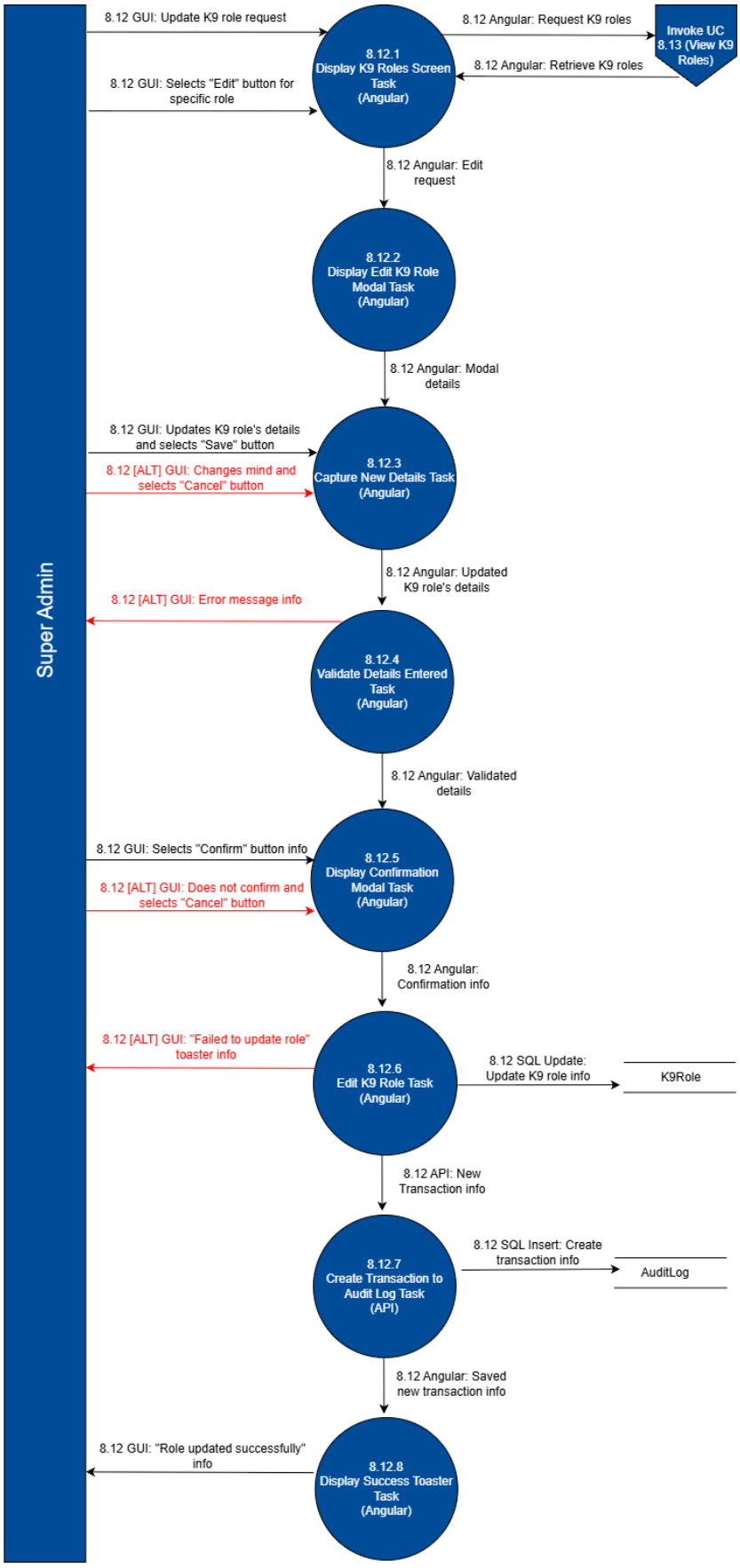


Figure 178: Technical Primitive Level Diagram 8.12 Update K9 Role

2.6.7.13 View K9 Roles: Use Case 8.13

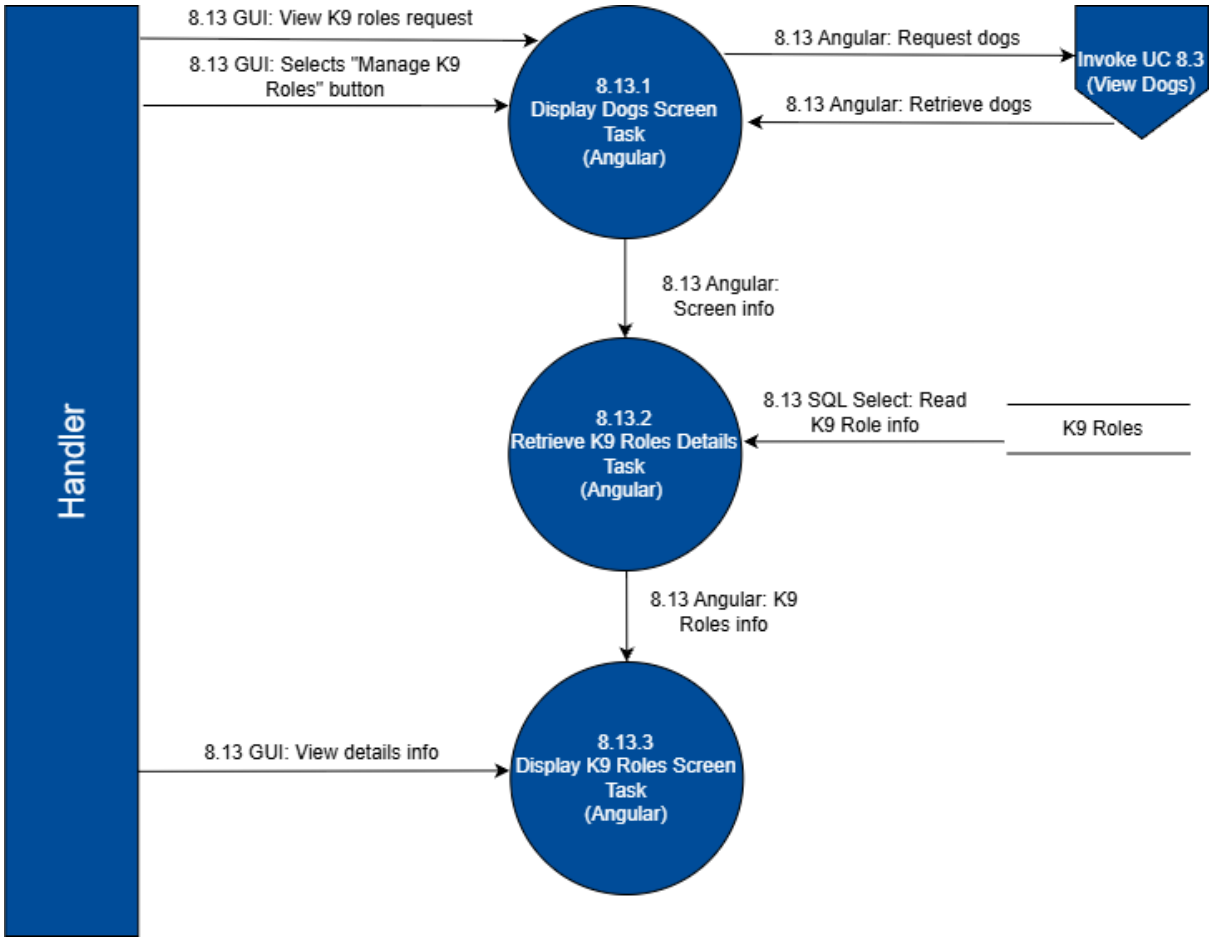


Figure 179: Technical Primitive Level Diagram 8.13 View K9 Roles

2.6.7.14 Reactivate K9 Role: Use Case 8.14

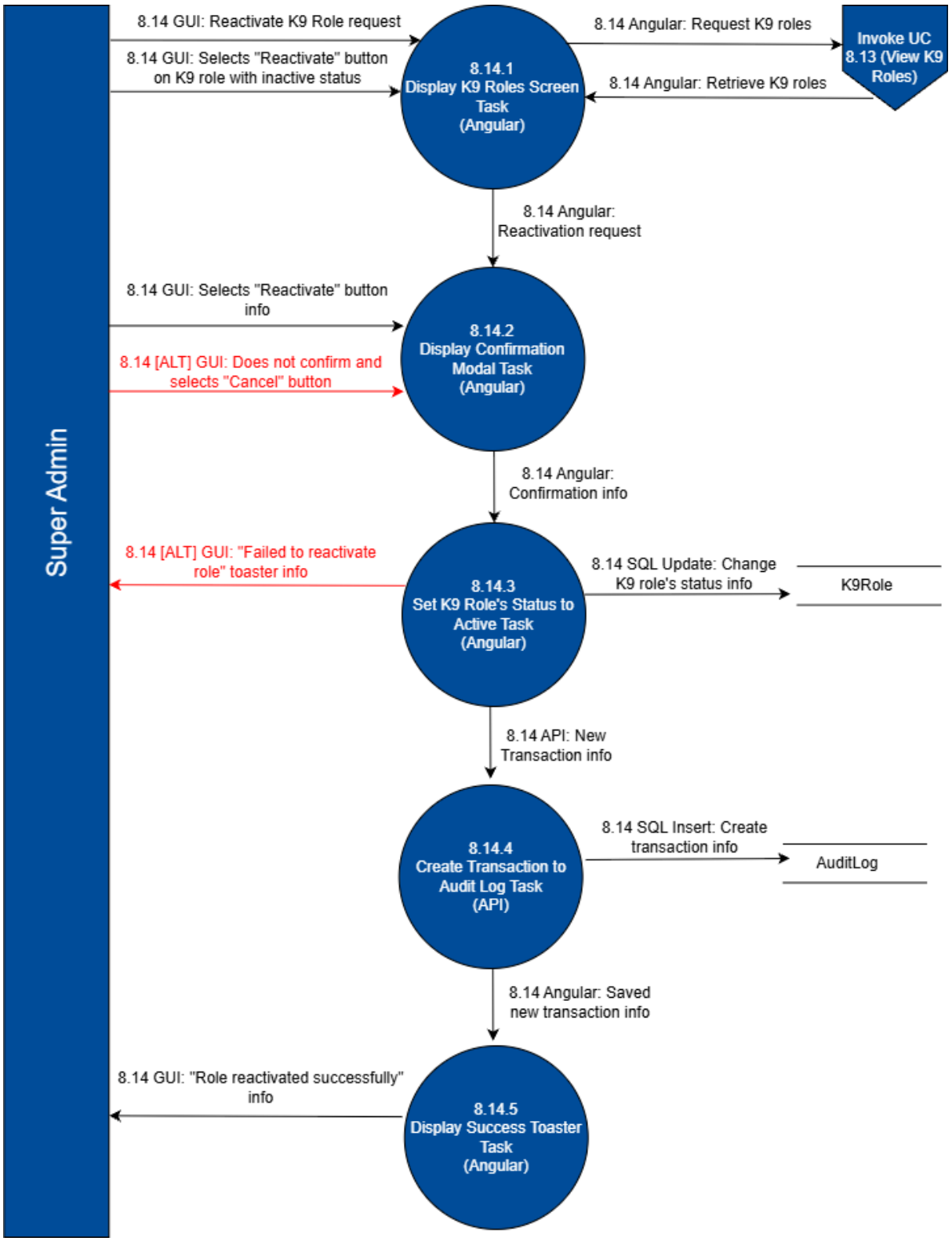


Figure 180: Technical Primitive Level Diagram 8.14 Reactivate K9 Role

2.6.7.15 Deactivate K9 Role: Use Case 8.15

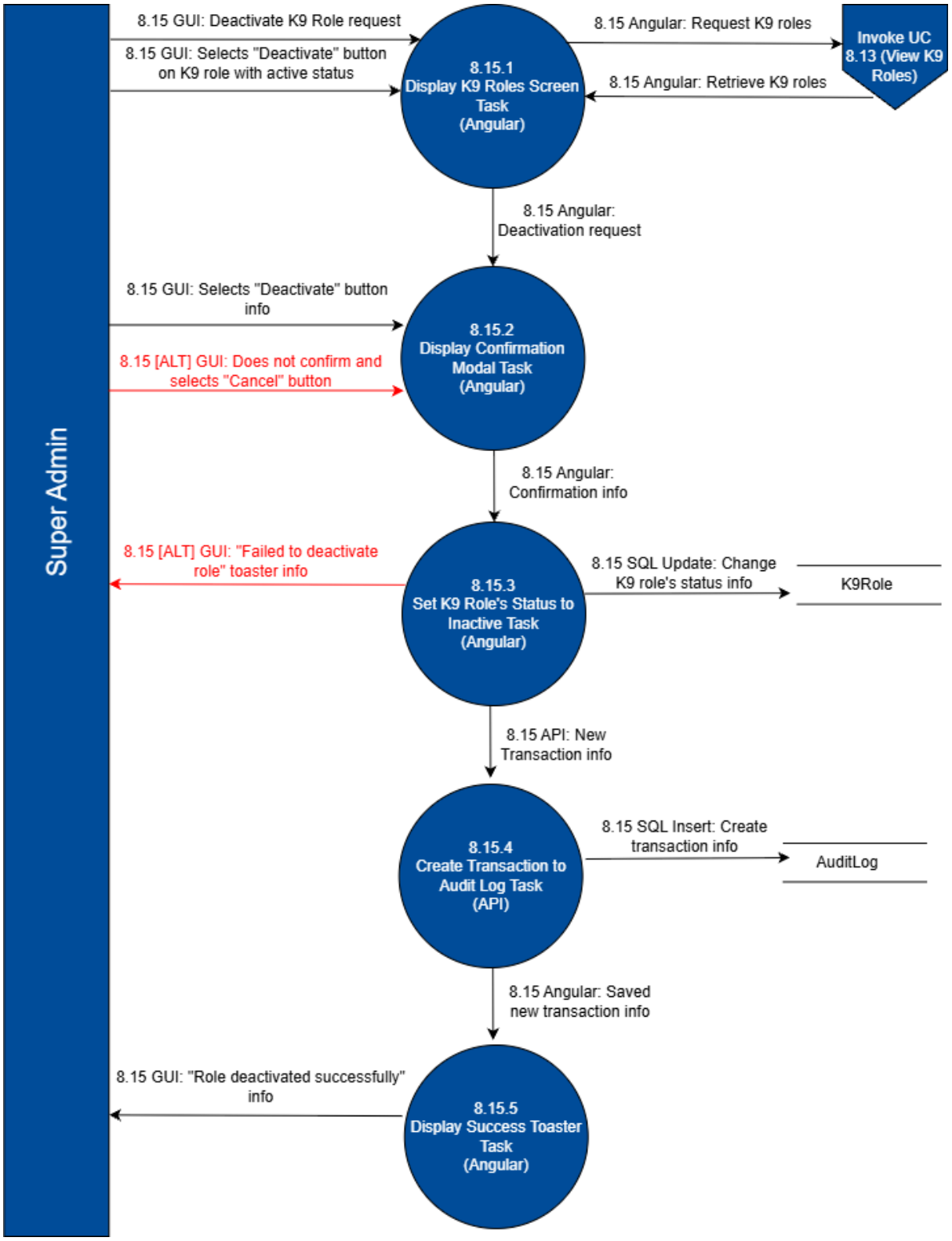


Figure 181: Technical Primitive Level Diagram 8.15 Deactivate K9 Role

2.6.7.16 View K9 Ancestry: Use Case 8.16

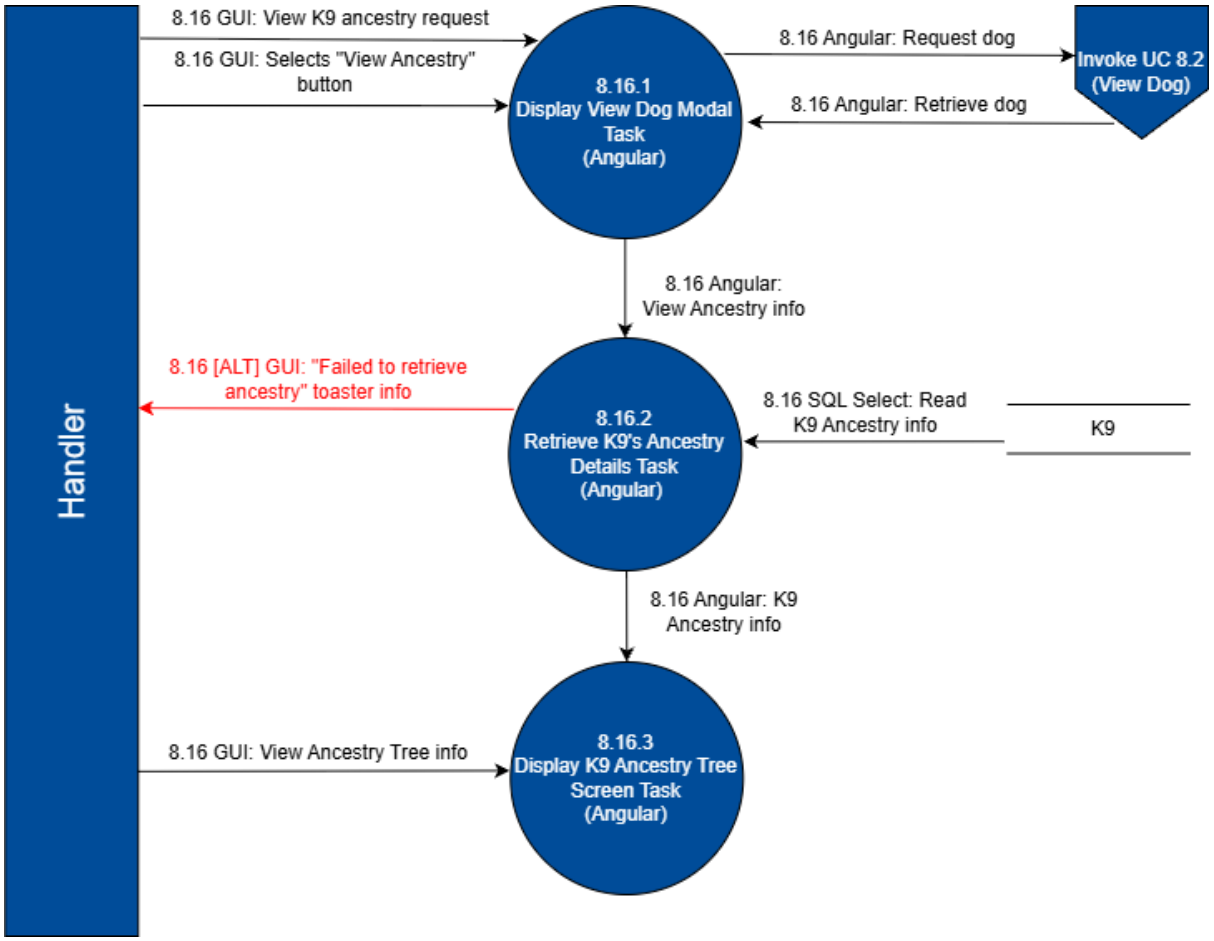


Figure 182: Technical Primitive Level Diagram 8.16 View K9 Ancestry

2.6.8 Rhino Subsystem Primitive Diagrams

2.6.8.1 Add Rhino: Use Case 9.1

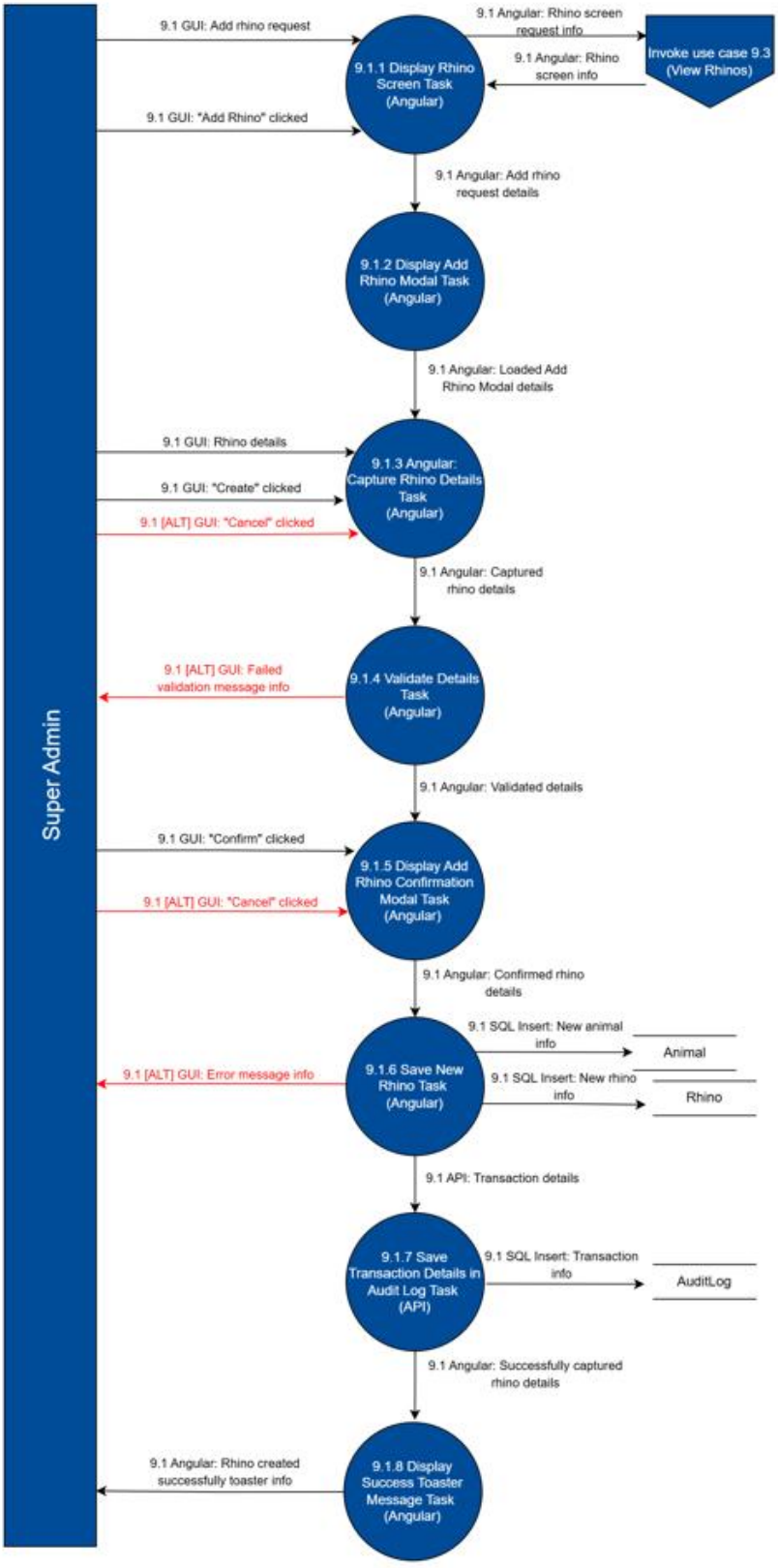


Figure 183: Technical Primitive Level Diagram 9.1 Add Rhino

2.6.8.2 View Rhino: Use Case 9.2

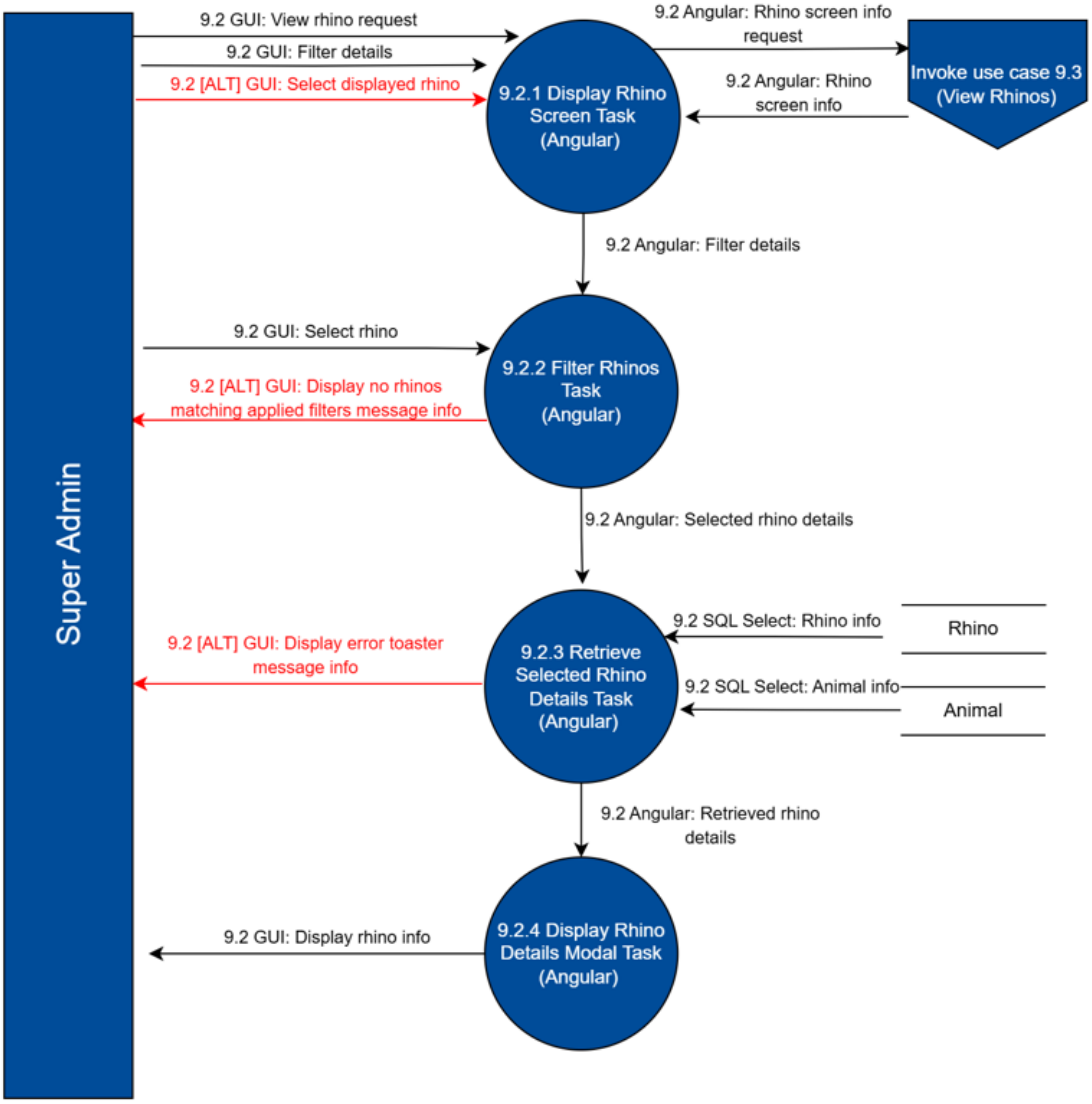


Figure 184: Technical Primitive Level Diagram 9.2 View Rhino

2.6.8.3 View Rhinos: Use Case 9.3

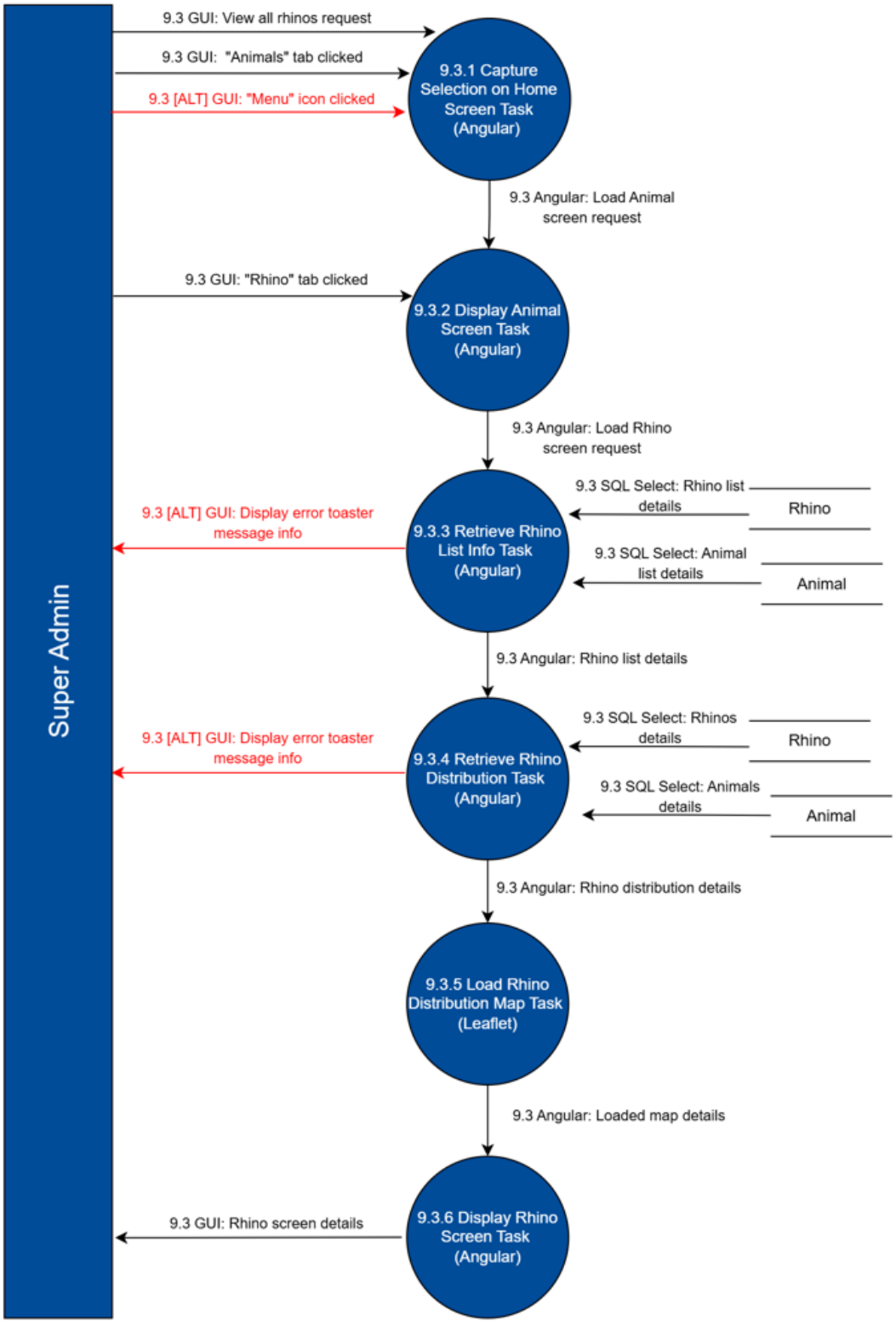


Figure 185: Technical Primitive Level Diagram 9.3 View Rhinos

2.6.8.4 Update Rhino: Use Case 9.4

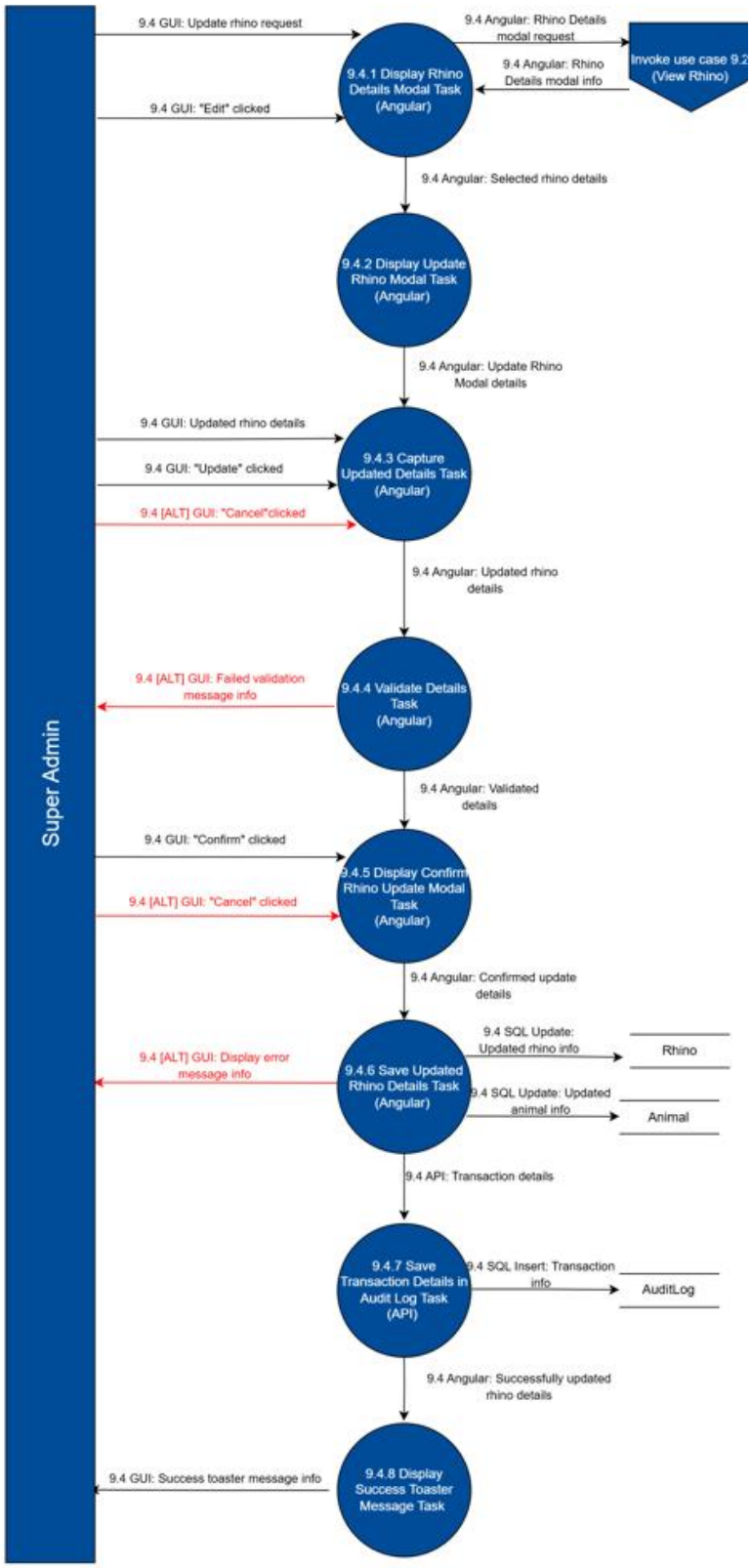


Figure 186: Technical Primitive Level Diagram 9.4 Update Rhino

2.6.8.5 Deactivate Rhino: Use Case 9.5

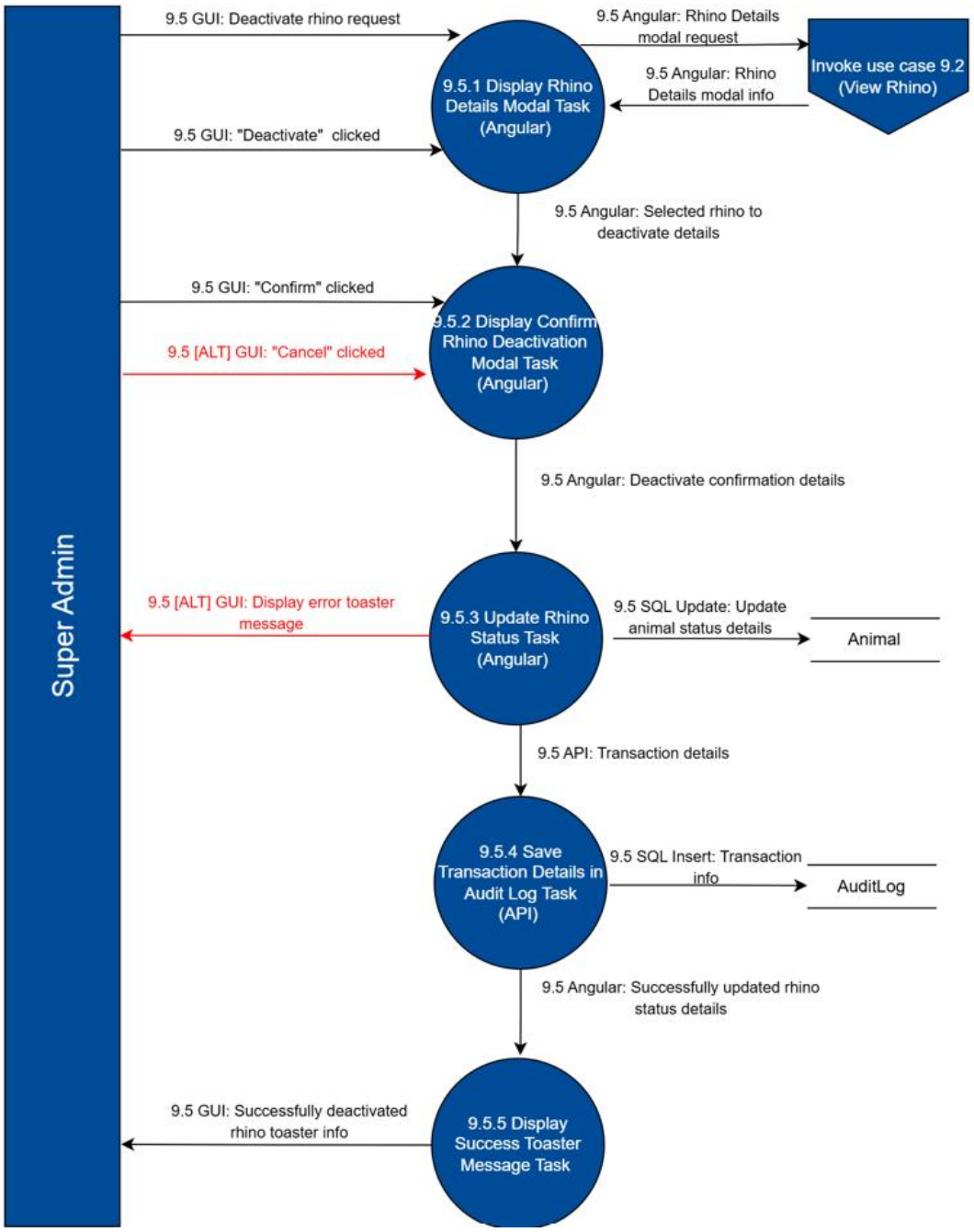


Figure 187: Technical Primitive Level Diagram 9.5 Deactivate Rhino

2.6.9 Elephant Subsystem Primitive Diagrams

2.6.9.1 Add Elephant: Use Case 10.1

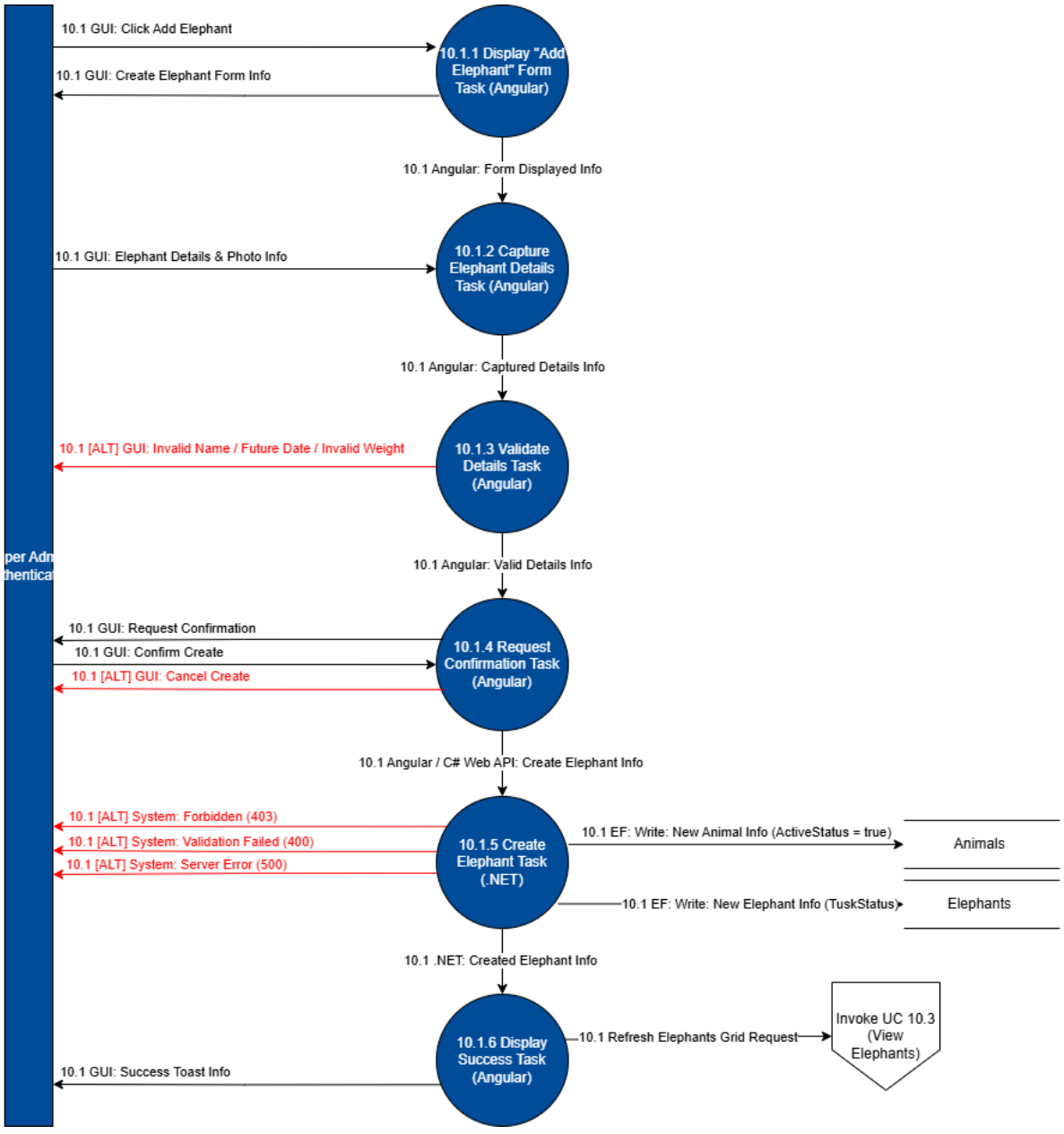


Figure 188: Technical Primitive Level Diagram 10.1 Add Elephant

2.6.9.2 View Elephant: Use Case 10.2

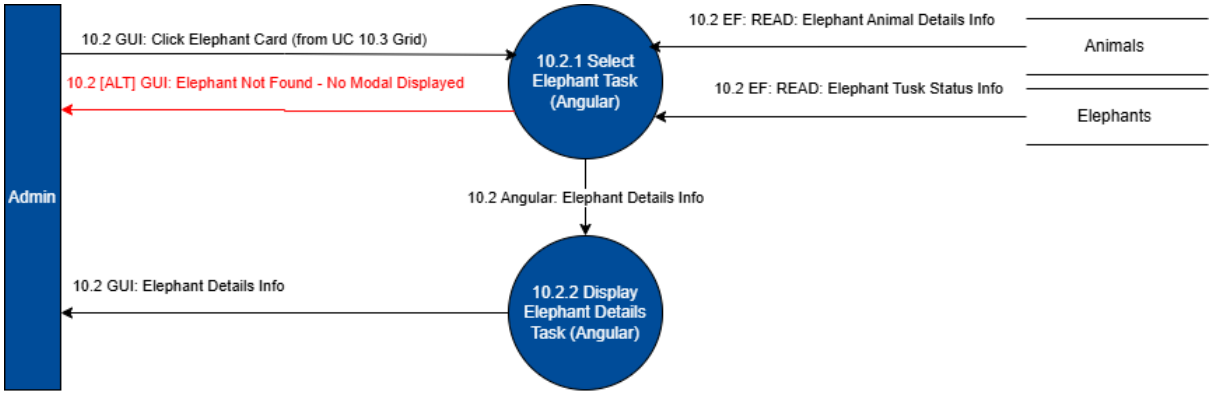


Figure 189: Technical Primitive Level Diagram 10.2 View Elephant

2.6.9.3 View Elephants: Use Case 10.3

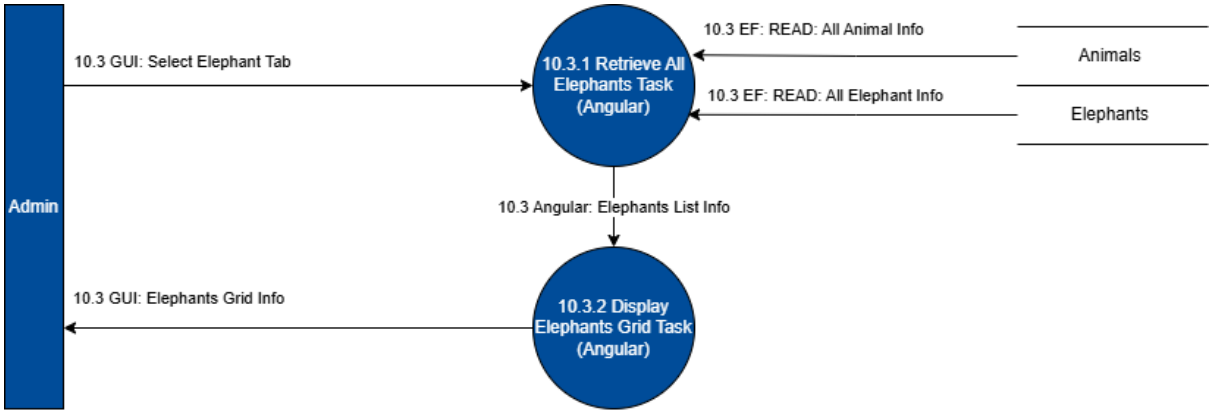


Figure 190: Technical Primitive Level Diagram 10.3 View Elephants

2.6.9.4 Update Elephant: Use Case 10.4

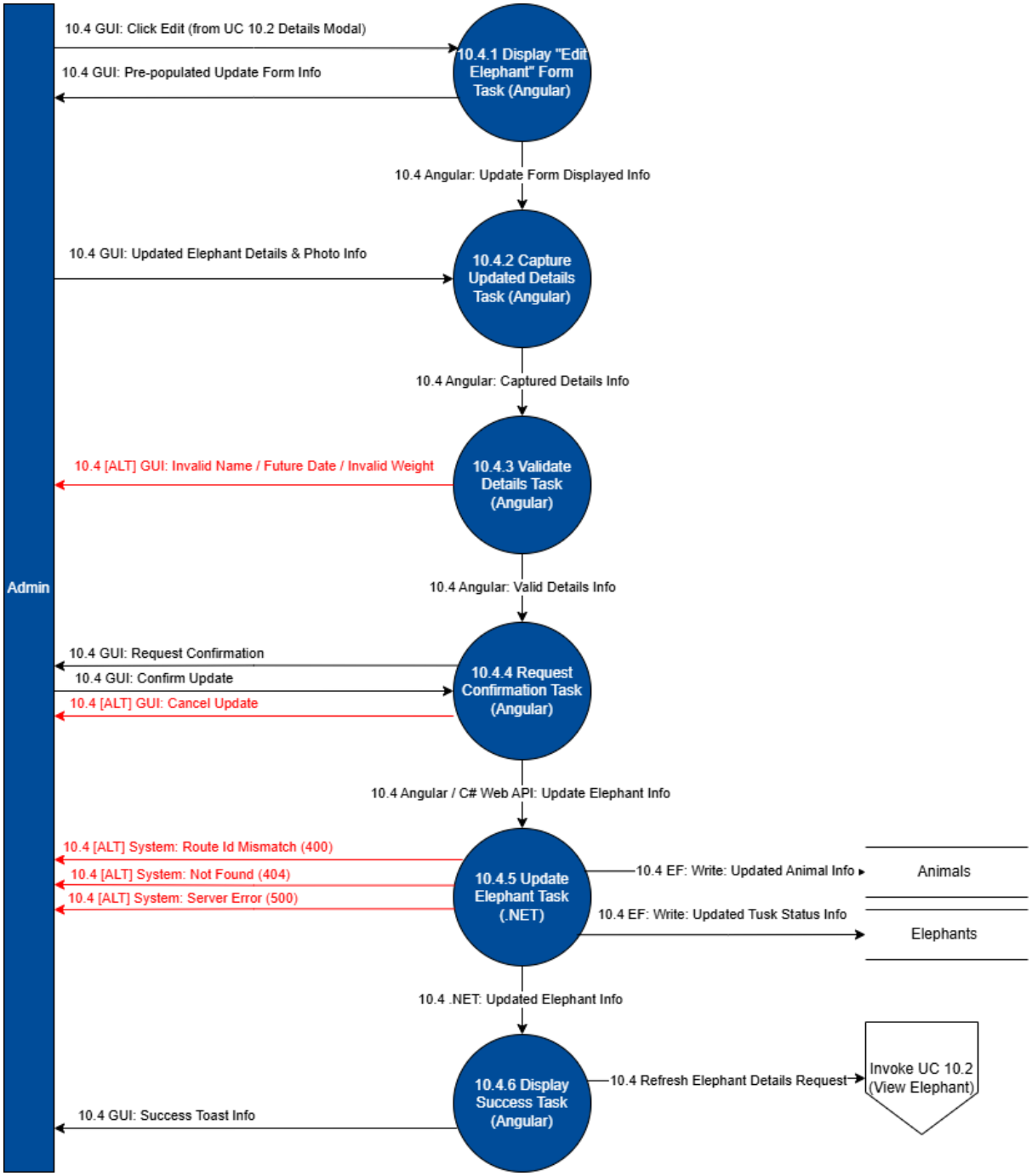


Figure 191: Technical Primitive Level Diagram 10.4 Update Elephant

2.6.9.5 Deactivate Elephant: Use Case 10.5

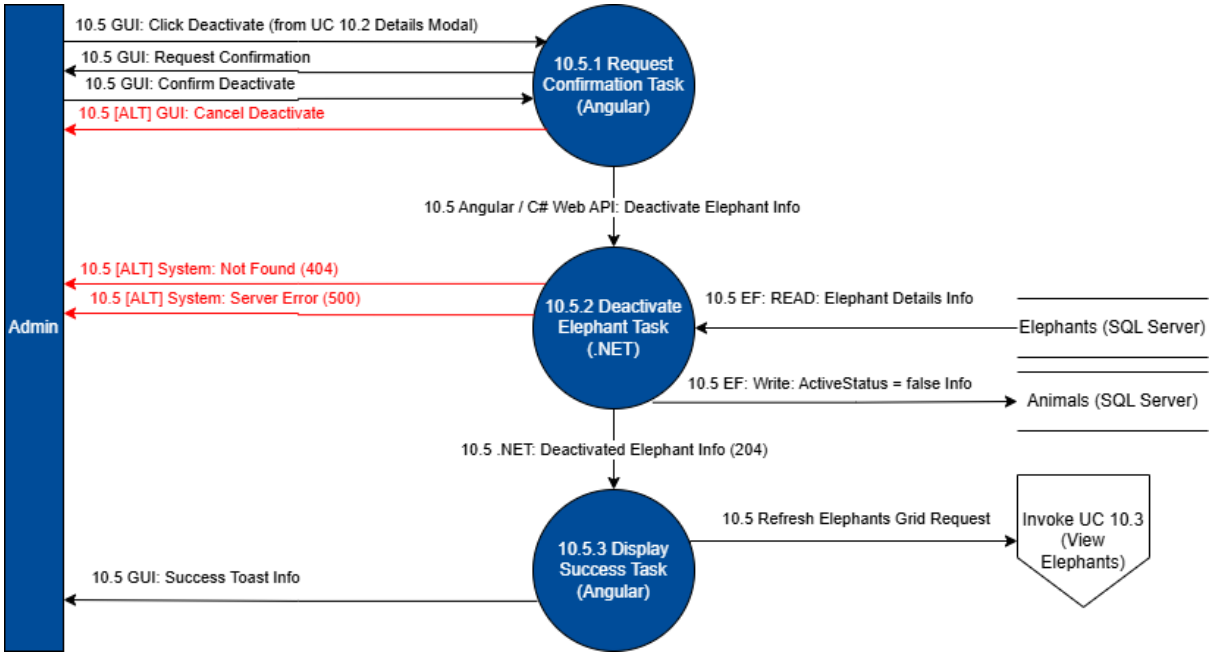


Figure 192: Technical Primitive Level Diagram 10.5 Deactivate Elephant

2.6.10 Report Subsystem Primitive Diagrams

2.6.10.1 Generate Custom Report: Use Case 12.1

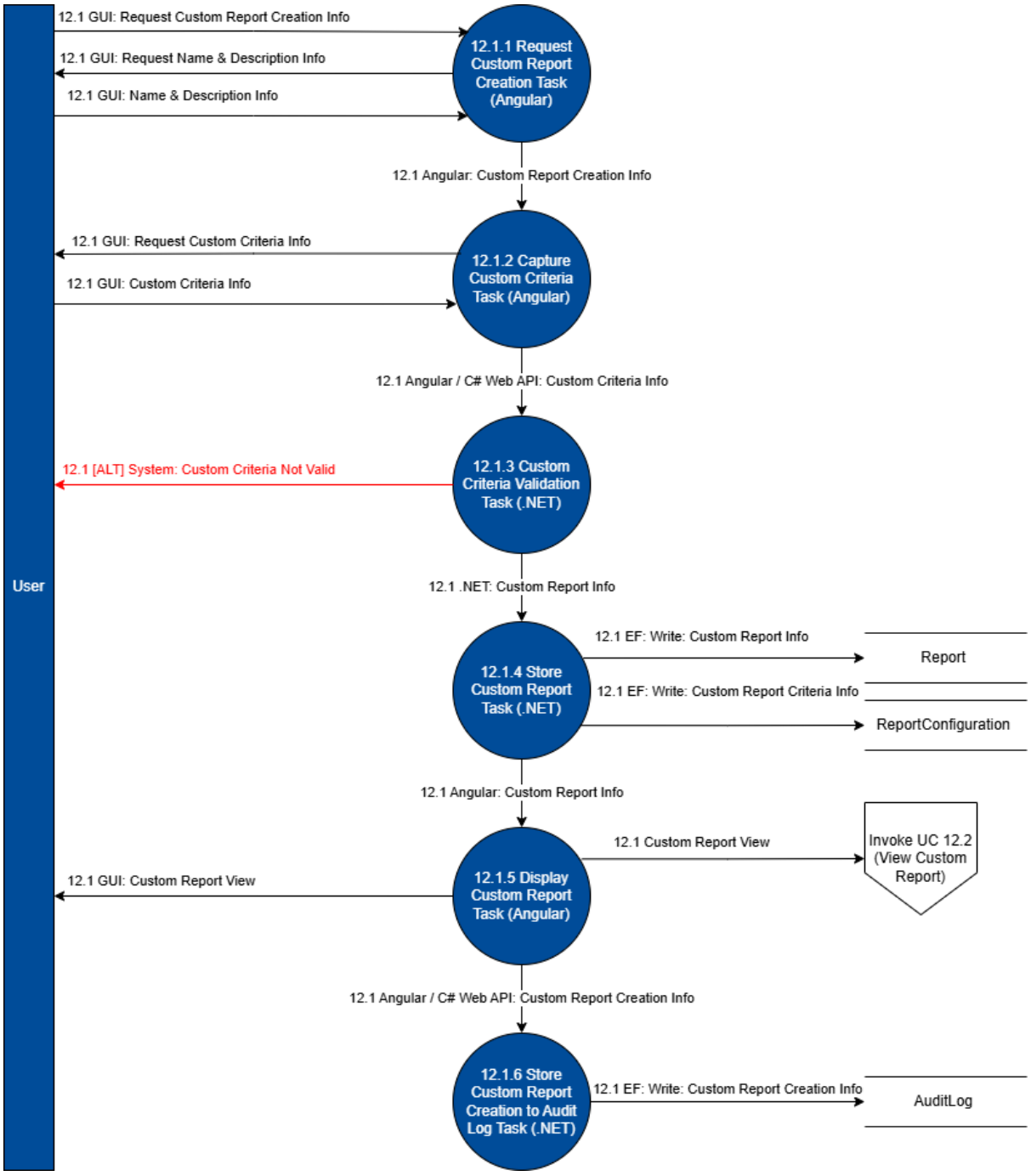


Figure 193: Technical Primitive Level Diagram 12.1 Generate Custom Report

2.6.10.2 View Custom Report: Use Case 12.2

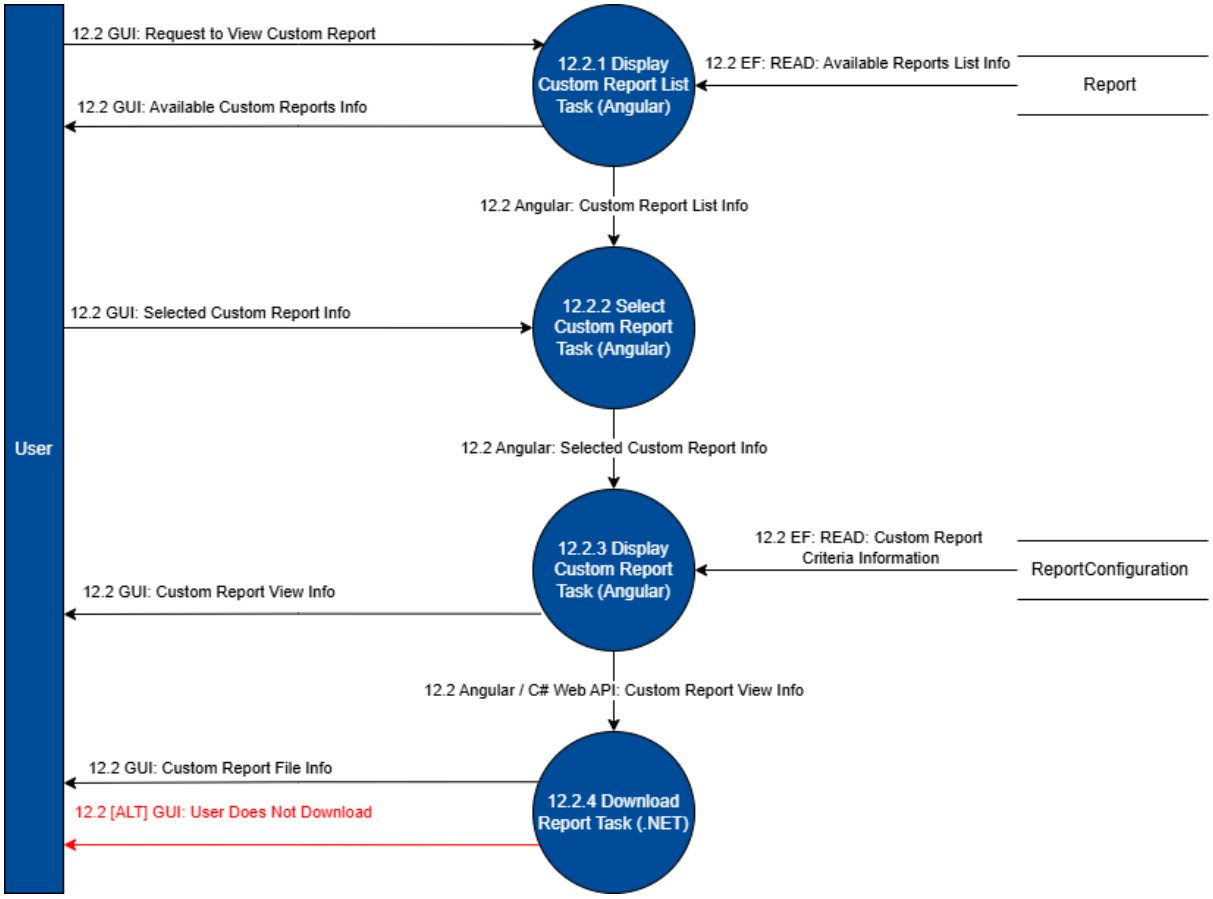


Figure 194: Technical Primitive Level Diagram 12.2 View Custom Report

2.6.10.3 Update Custom Report: Use Case 12.3

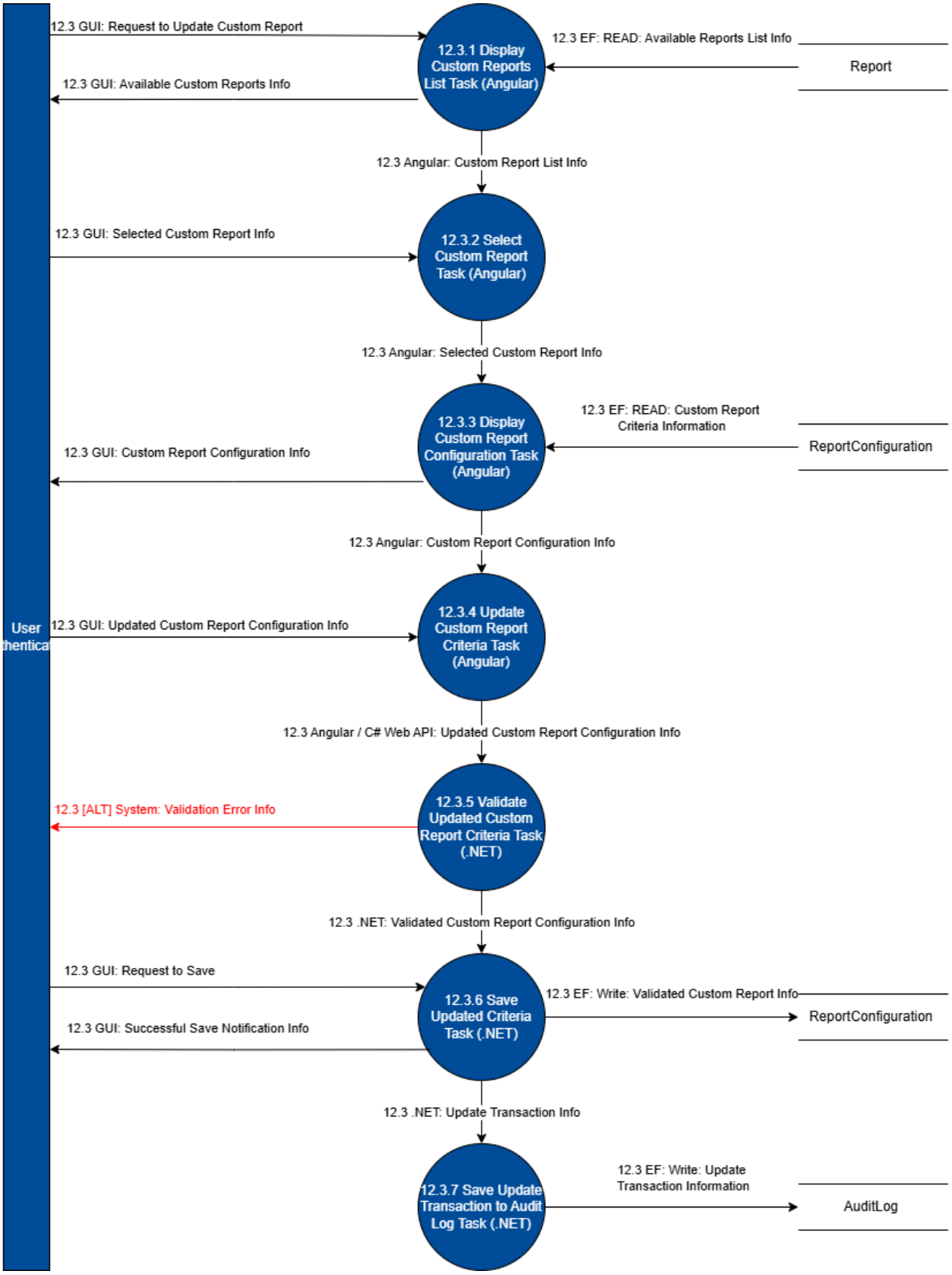


Figure 195: Technical Primitive Level Diagram 12.3 Update Custom Report

2.6.10.4 Remove Custom Report: Use Case 12.4

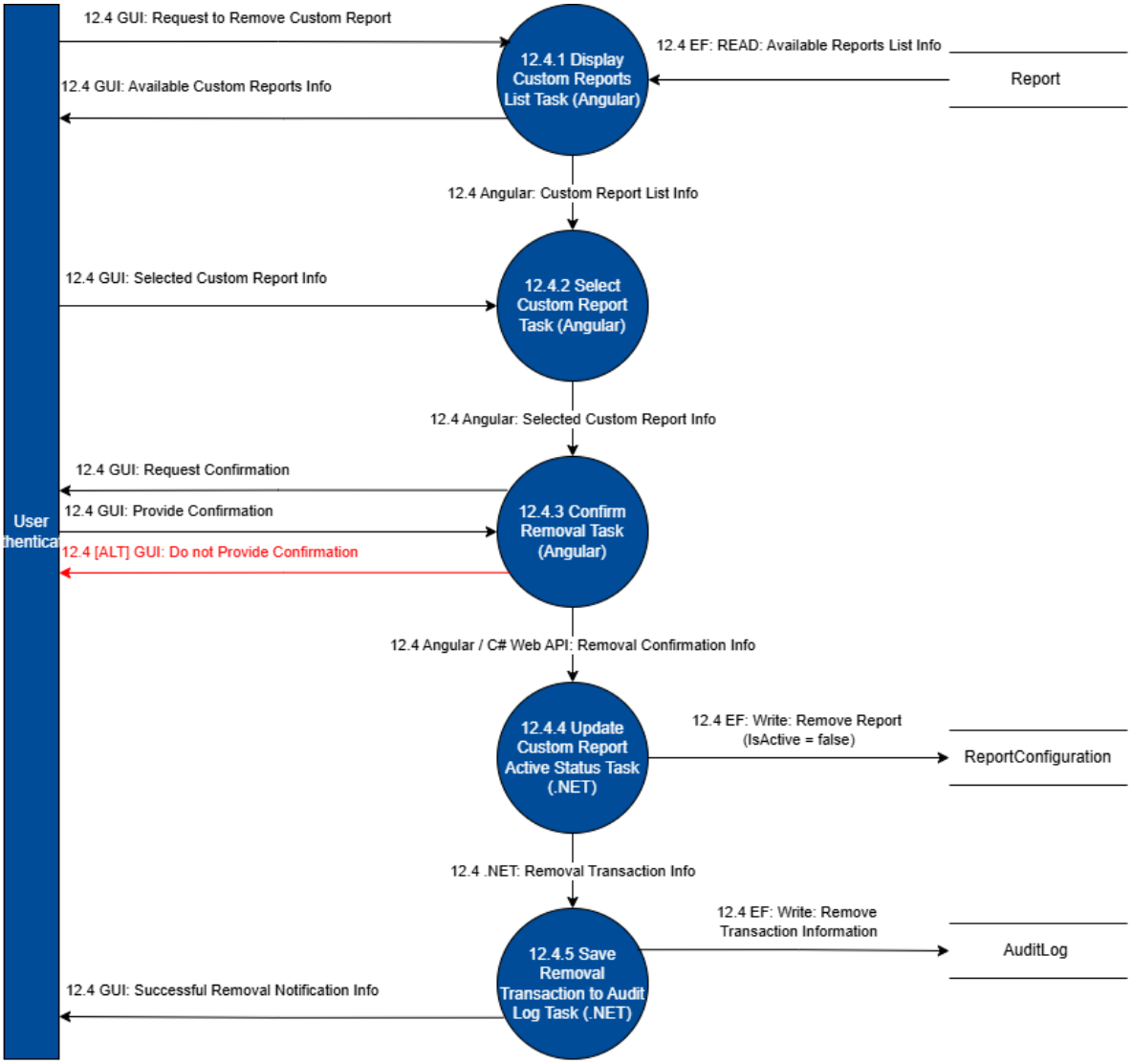


Figure 196: Technical Primitive Level Diagram 12.4 Remove Custom Report

2.6.11 Budget Subsystem Primitive Diagrams

2.6.11.1 View Project Budget: Use Case 14.1

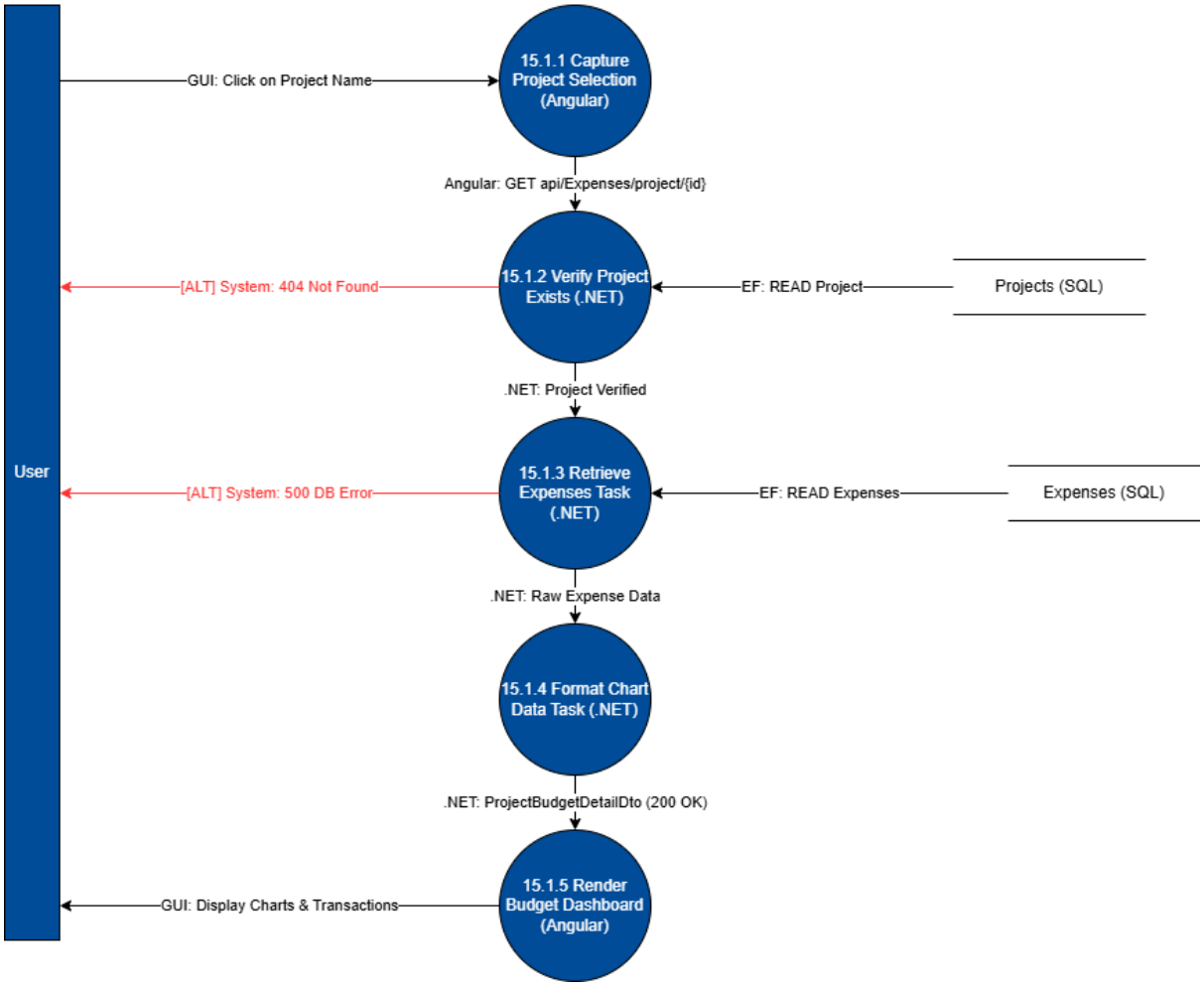


Figure 197: Technical Primitive Level Diagram 14.1 View Project Budget

2.6.11.2 View Project Budgets: Use Case 14.2

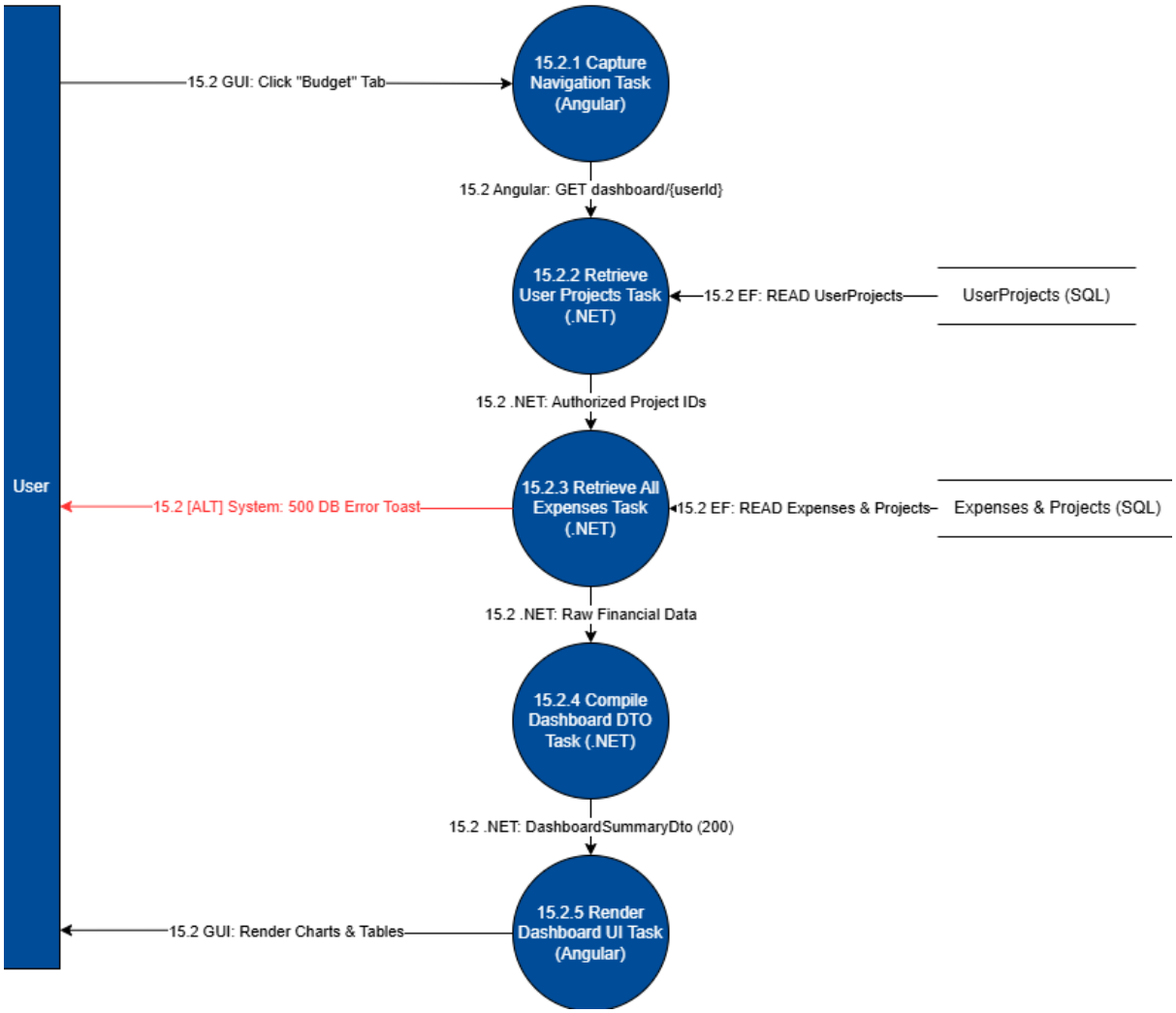


Figure 198: Technical Primitive Level Diagram 14.2 View Project Budgets

2.6.11.3 Log An Expense: Use Case 14.3

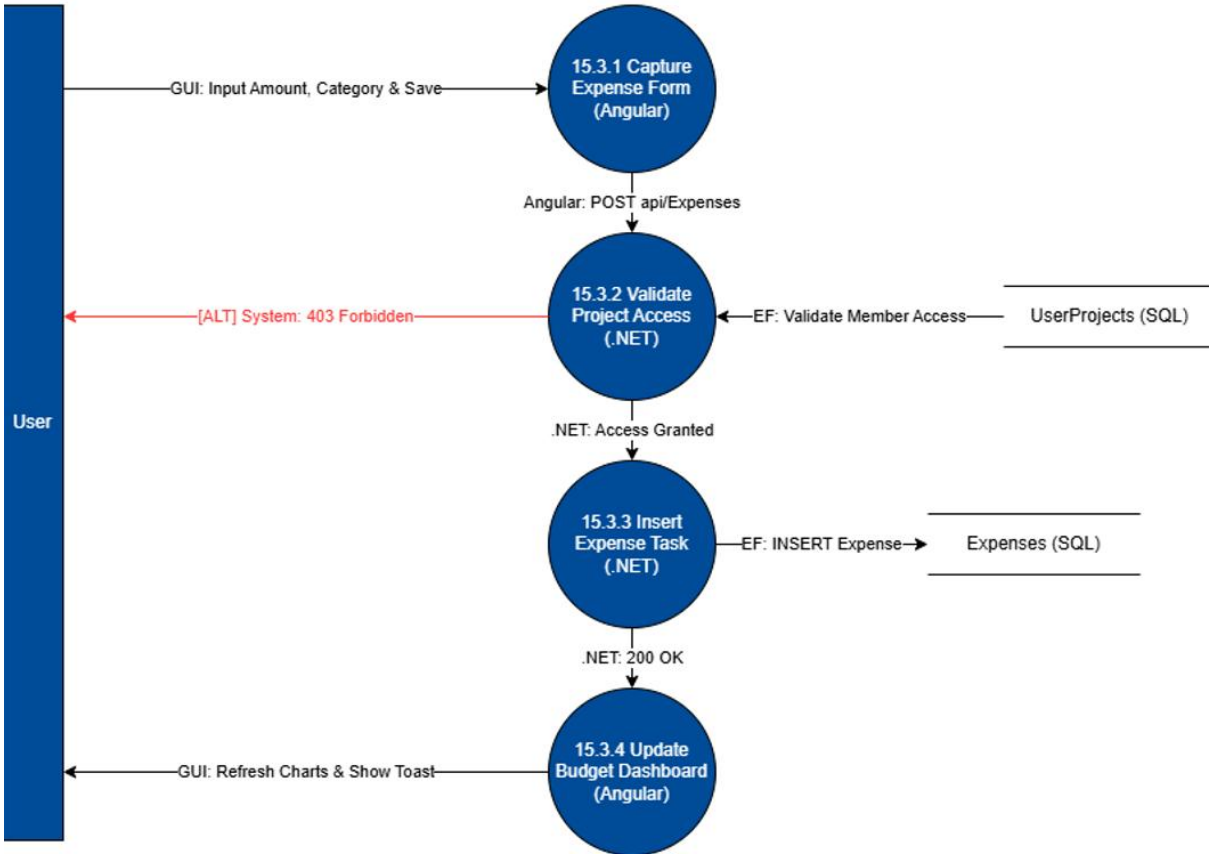


Figure 199: Technical Primitive Level Diagram 14.3 Log An Expense

2.6.11.4 Update An Expense: Use Case 14.4

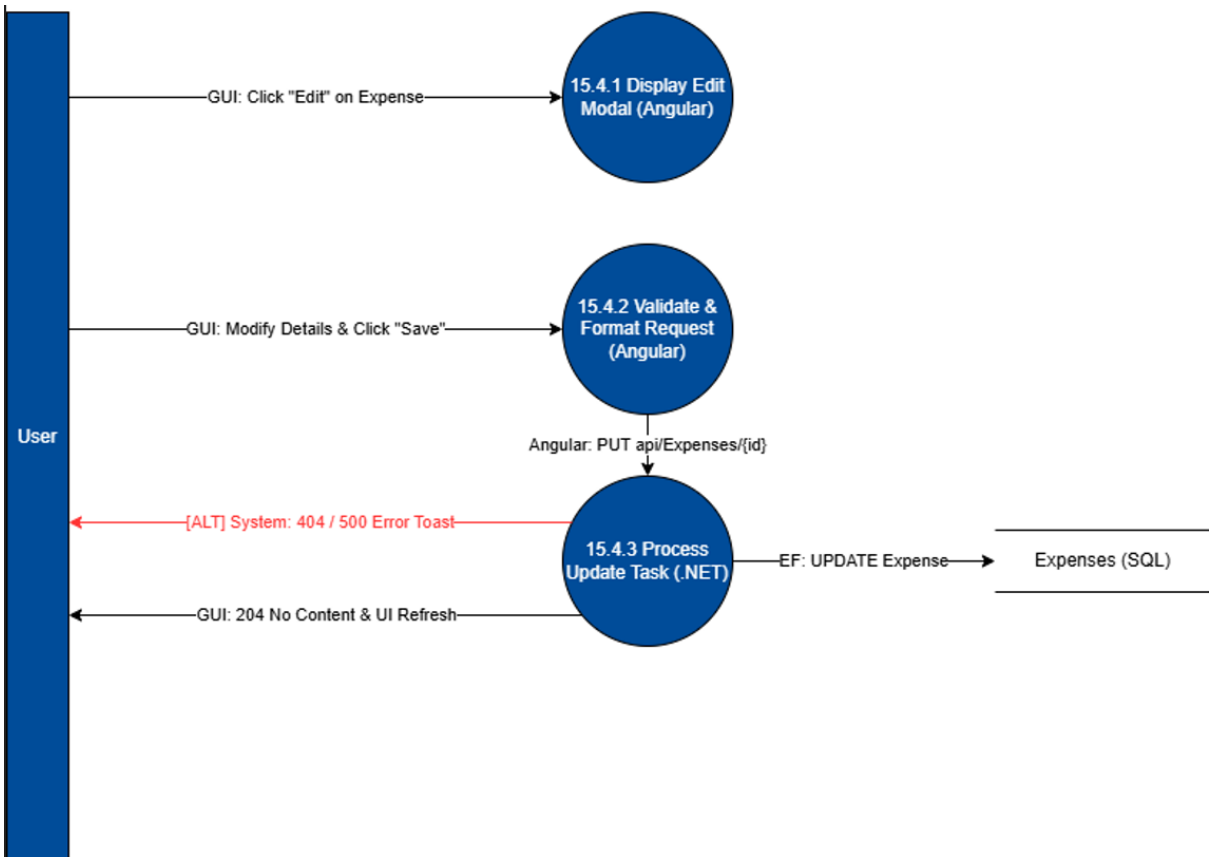


Figure 200: Technical Primitive Level Diagram 14.4 Update An Expense

2.6.11.5 View Expense: Use Case 14.5

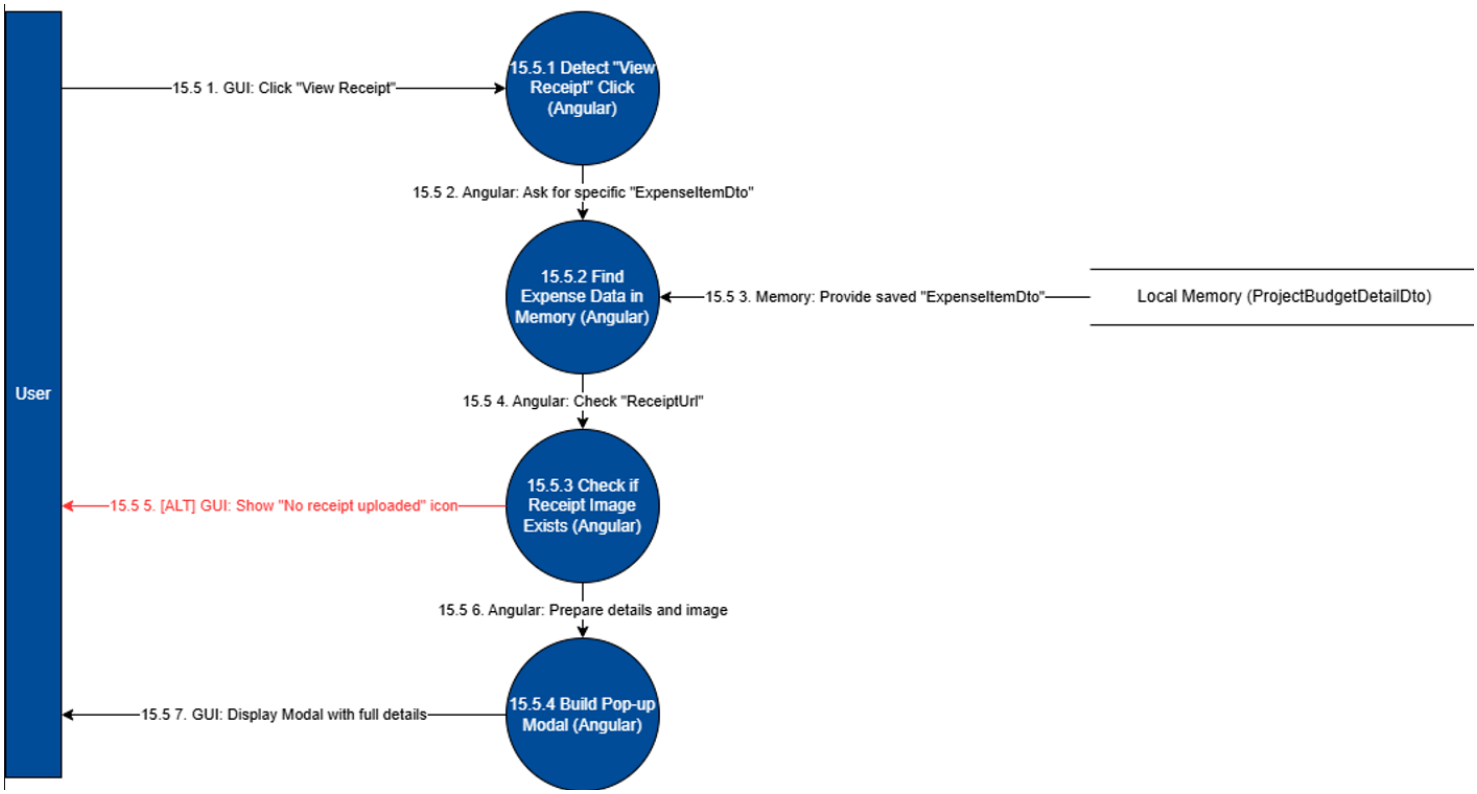


Figure 201: Technical Primitive Level Diagram 14.5 View Expense

2.6.11.6 Remove Expense: Use Case 14.6

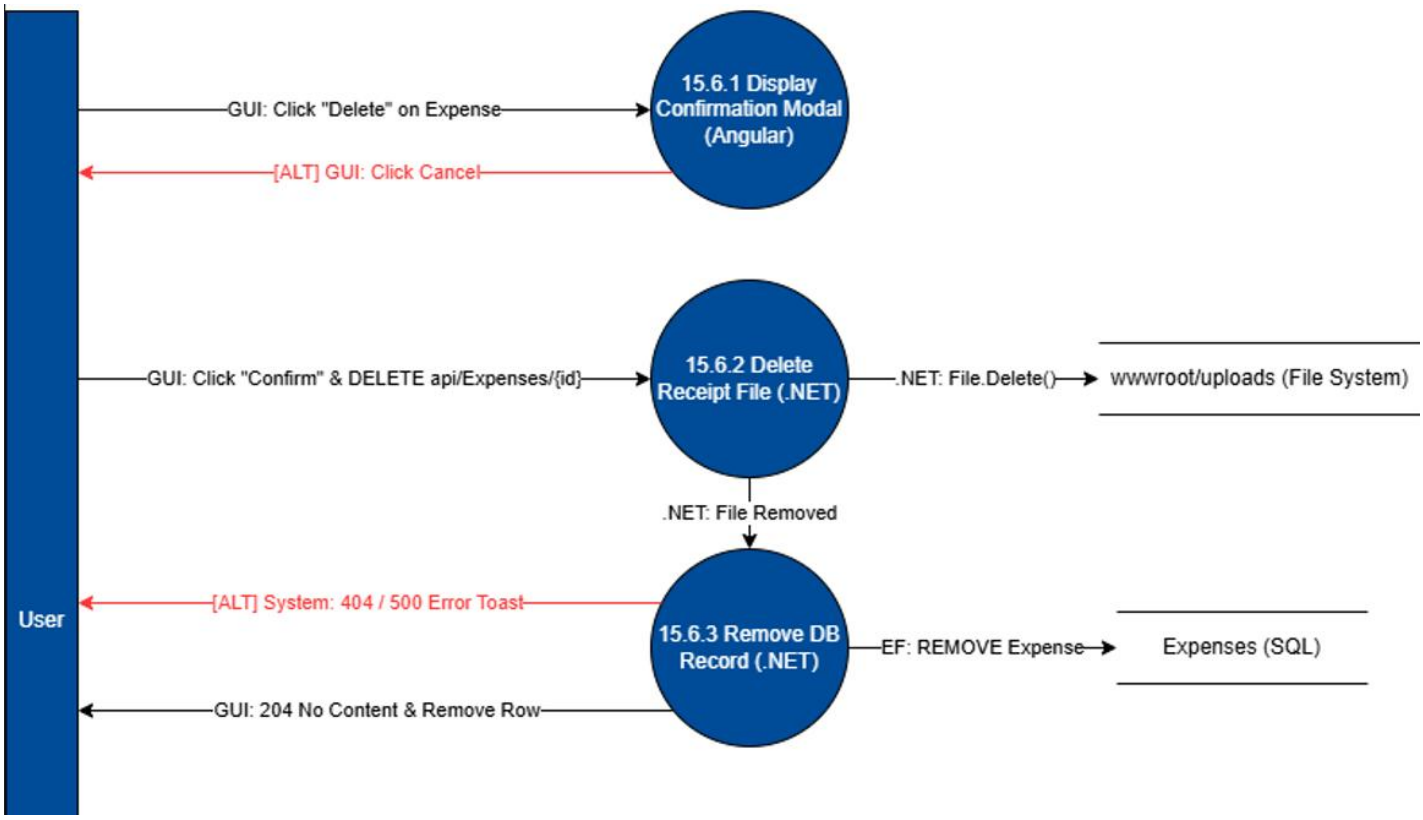


Figure 202: Technical Primitive Level Diagram 14.6 Remove Expense

2.7 Sequence Diagram

2.7.1 User Subsystem Sequence Diagrams

2.7.1.1 View FAQ: Use Case 1.9

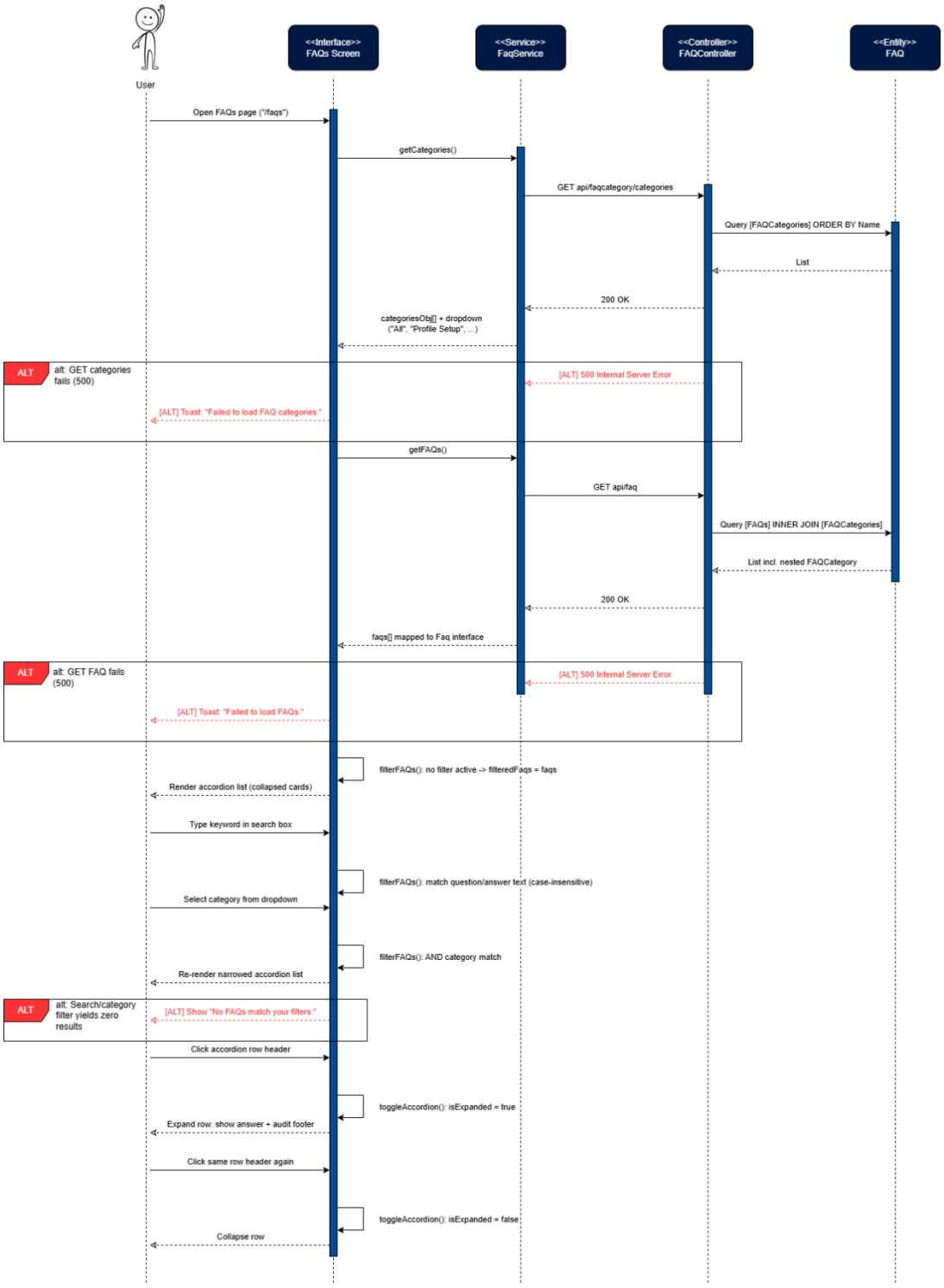


Figure 203: Sequence Diagram 1.9 View FAQ

2.7.2 Admin Subsystem Sequence Diagrams

2.7.2.1 Add FAQ: Use Case 2.5

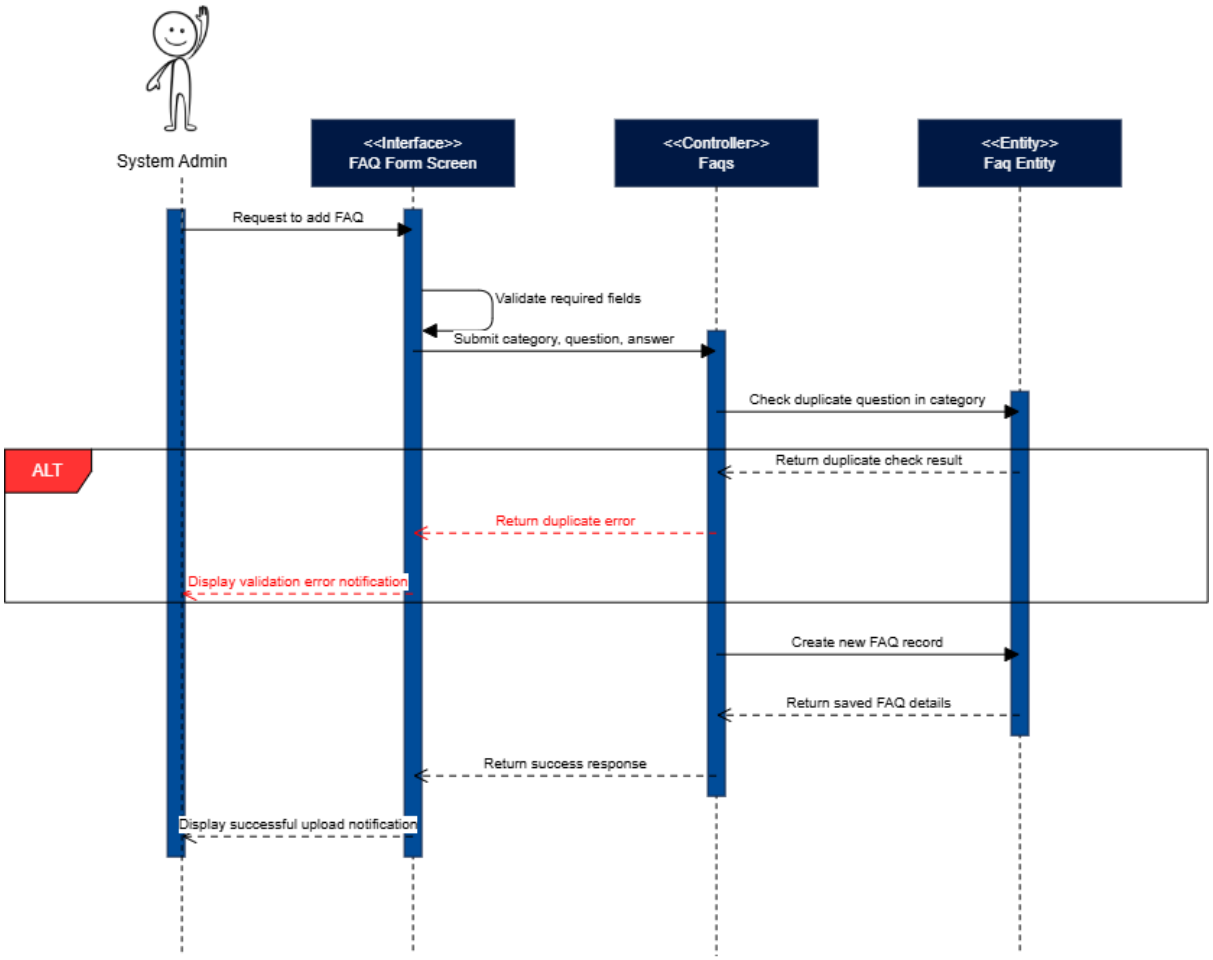


Figure 204: Sequence Diagram 2.5 Add FAQ

2.7.2.2 Update FAQ: Use Case 2.6

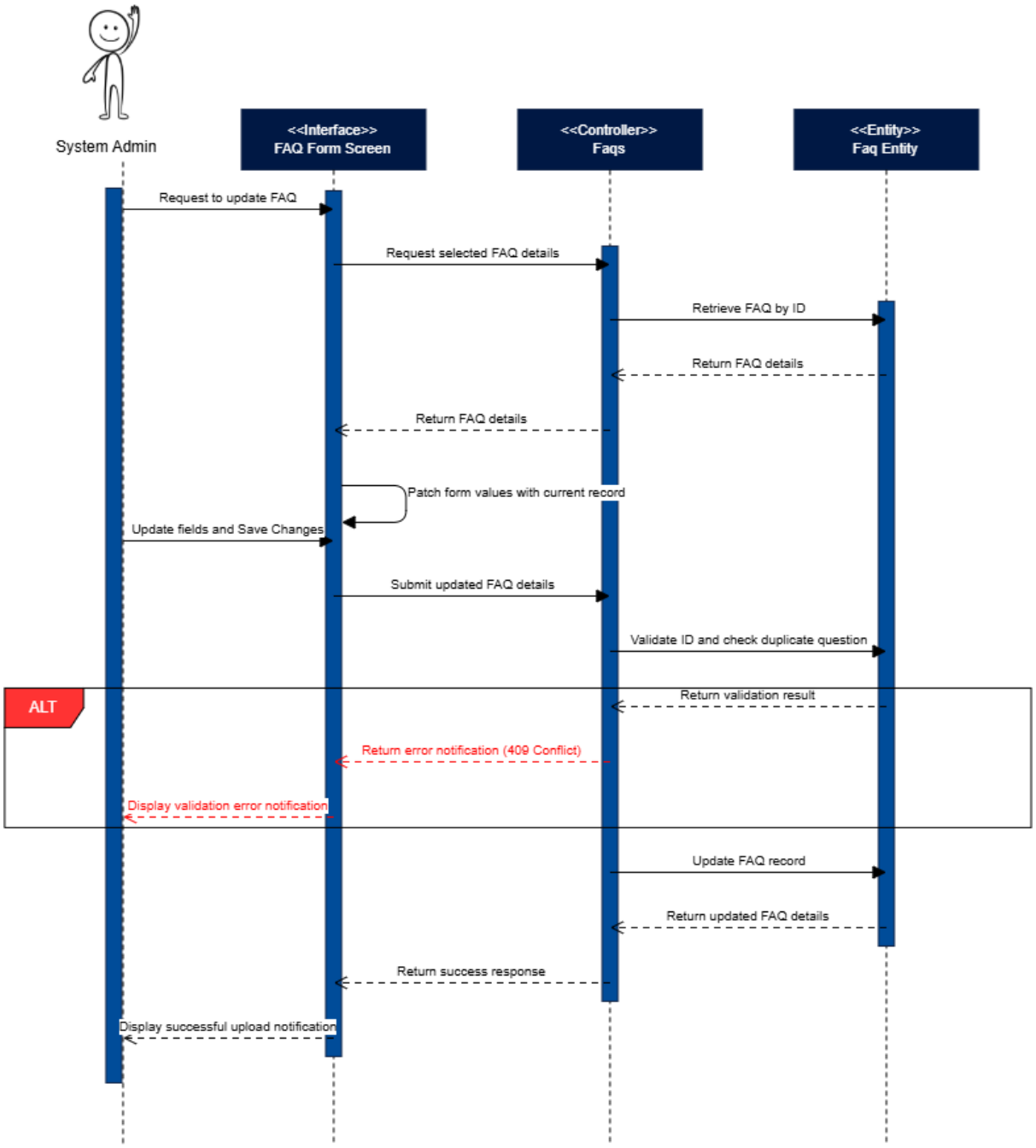


Figure 205: Sequence Diagram 2.6 Update FAQ

2.7.2.3 Remove FAQ: Use Case 2.7

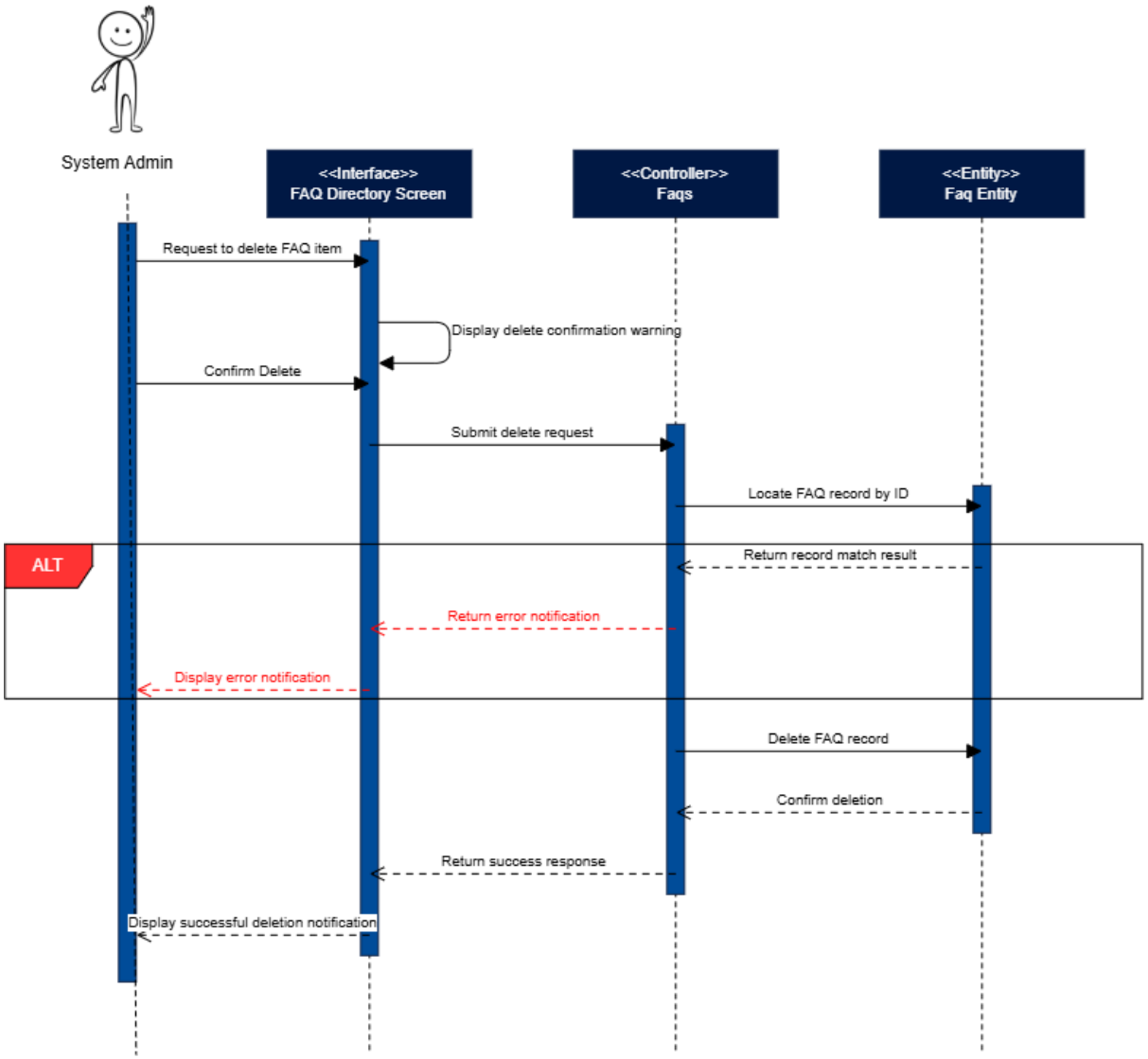


Figure 206: Sequence Diagram 2.7 Remove FAQ

2.7.2.4 Add FAQ Type: Use Case 2.8

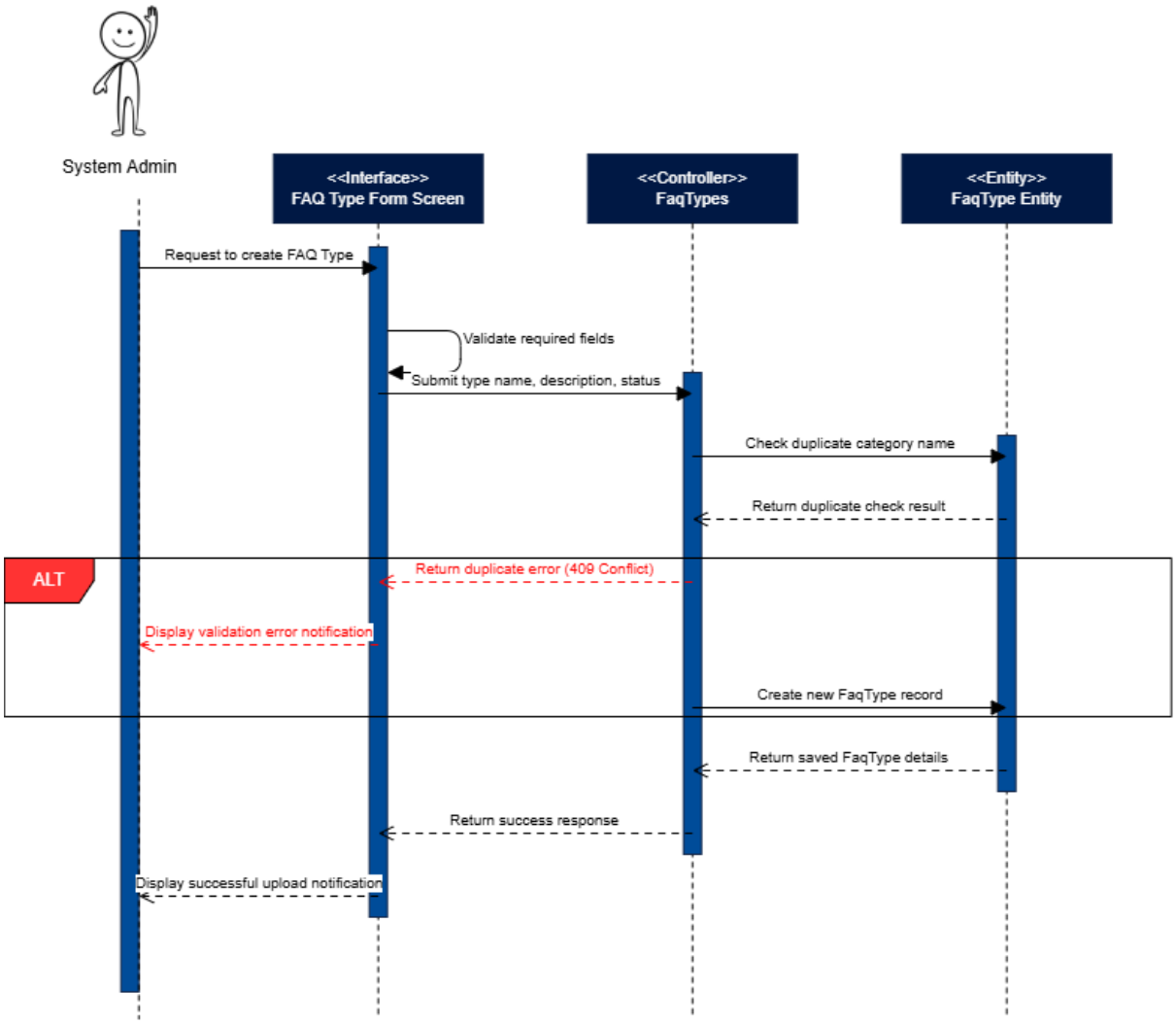


Figure 207: Sequence Diagram 2.8 Add FAQ Type

2.7.2.5 Update FAQ Type: Use Case 2.9

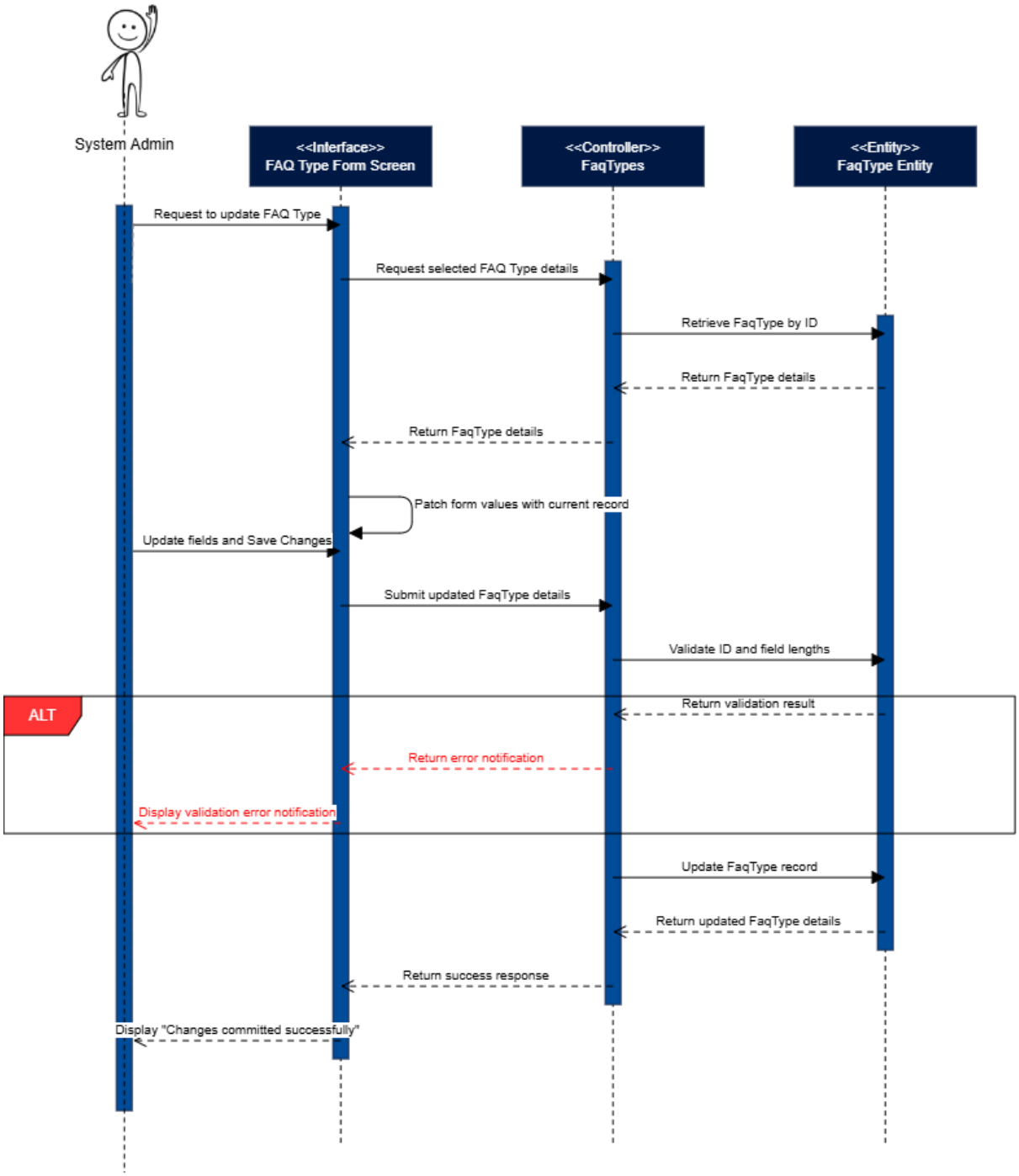


Figure 208: Sequence Diagram 2.9 Update FAQ Type

2.7.2.6 Remove FAQ Type: Use Case 2.10

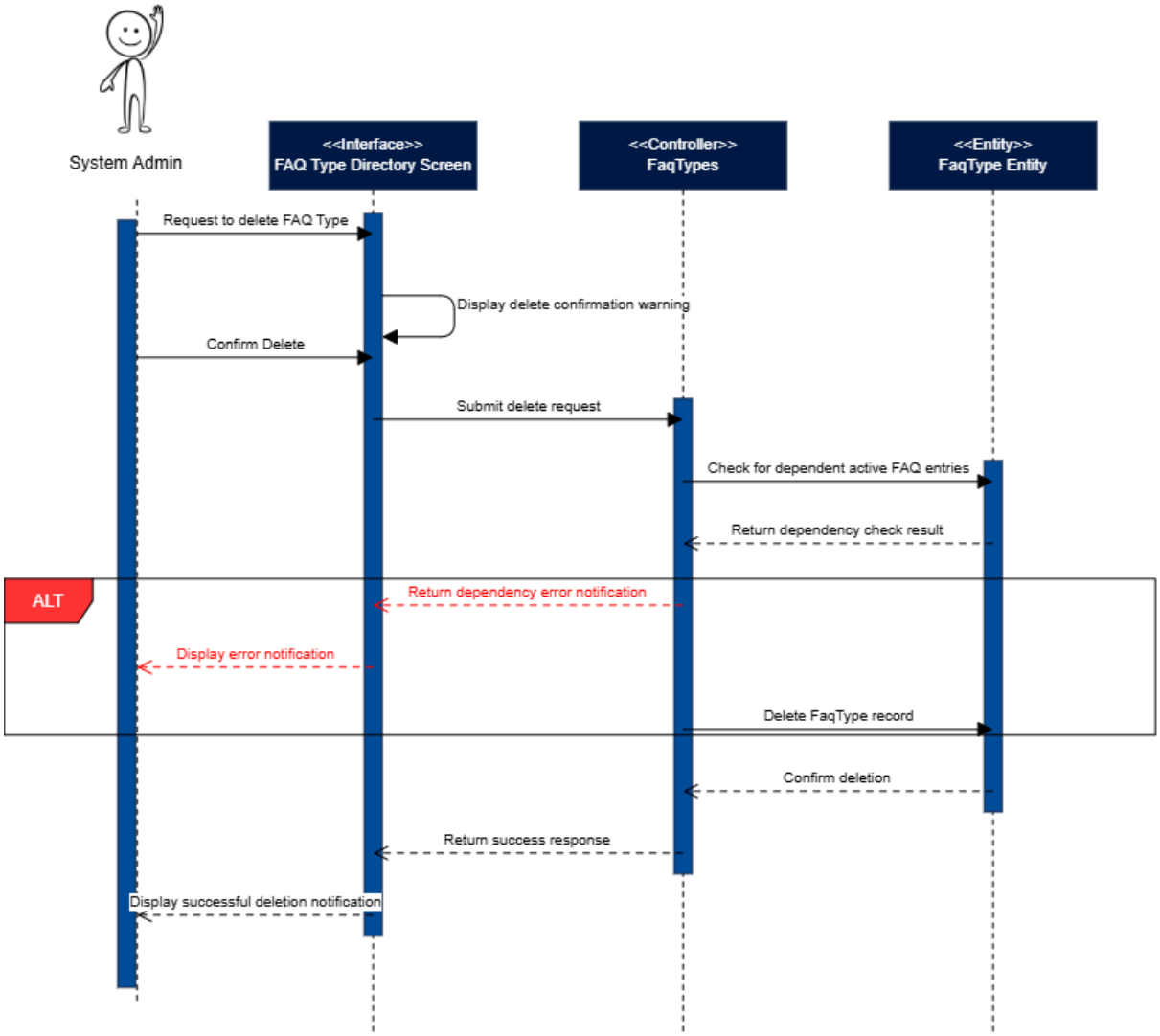


Figure 209: Sequence Diagram 2.10 Remove FAQ Type

2.7.2.7 Update Role Permission: Use Case 2.16

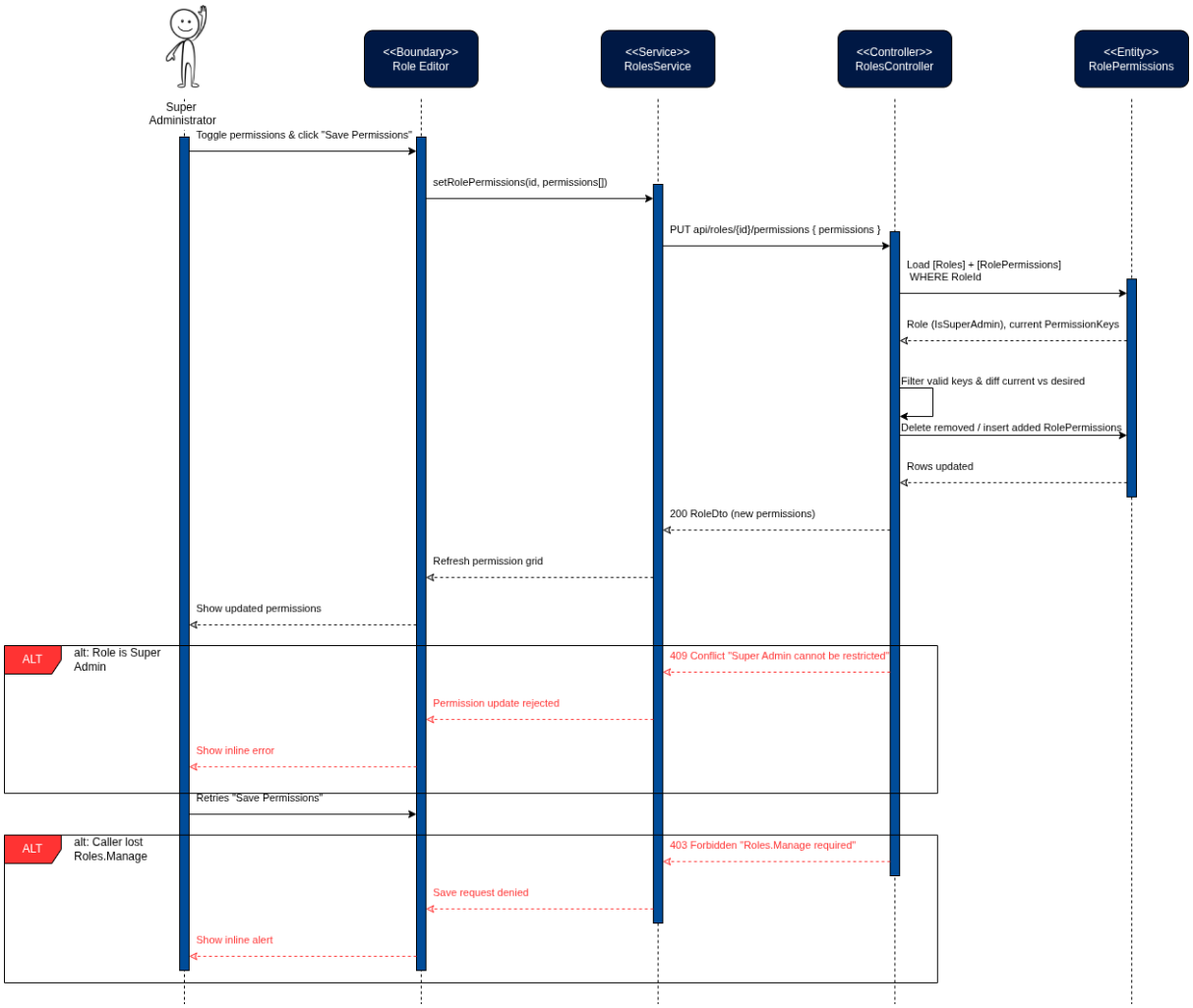


Figure 210: Sequence Diagram 2.16 Update Role Permission

2.7.2.8 View Role Permission: Use Case 2.17

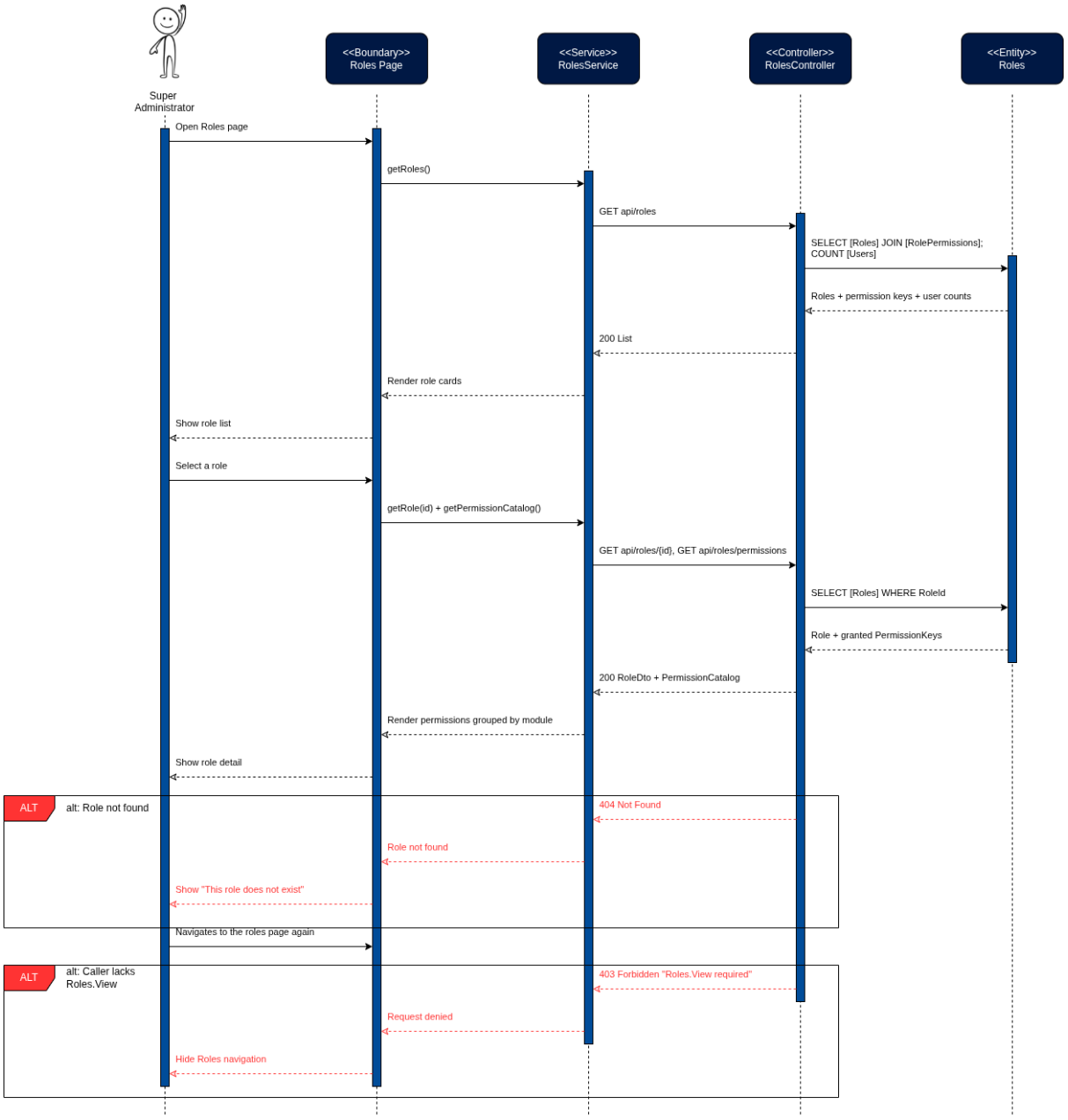


Figure 211: Sequence Diagram 2.17 View Role Permission

2.7.3 Inventory Subsystem Sequence Diagrams

2.7.3.1 Create Procurement: Use Case 3.1

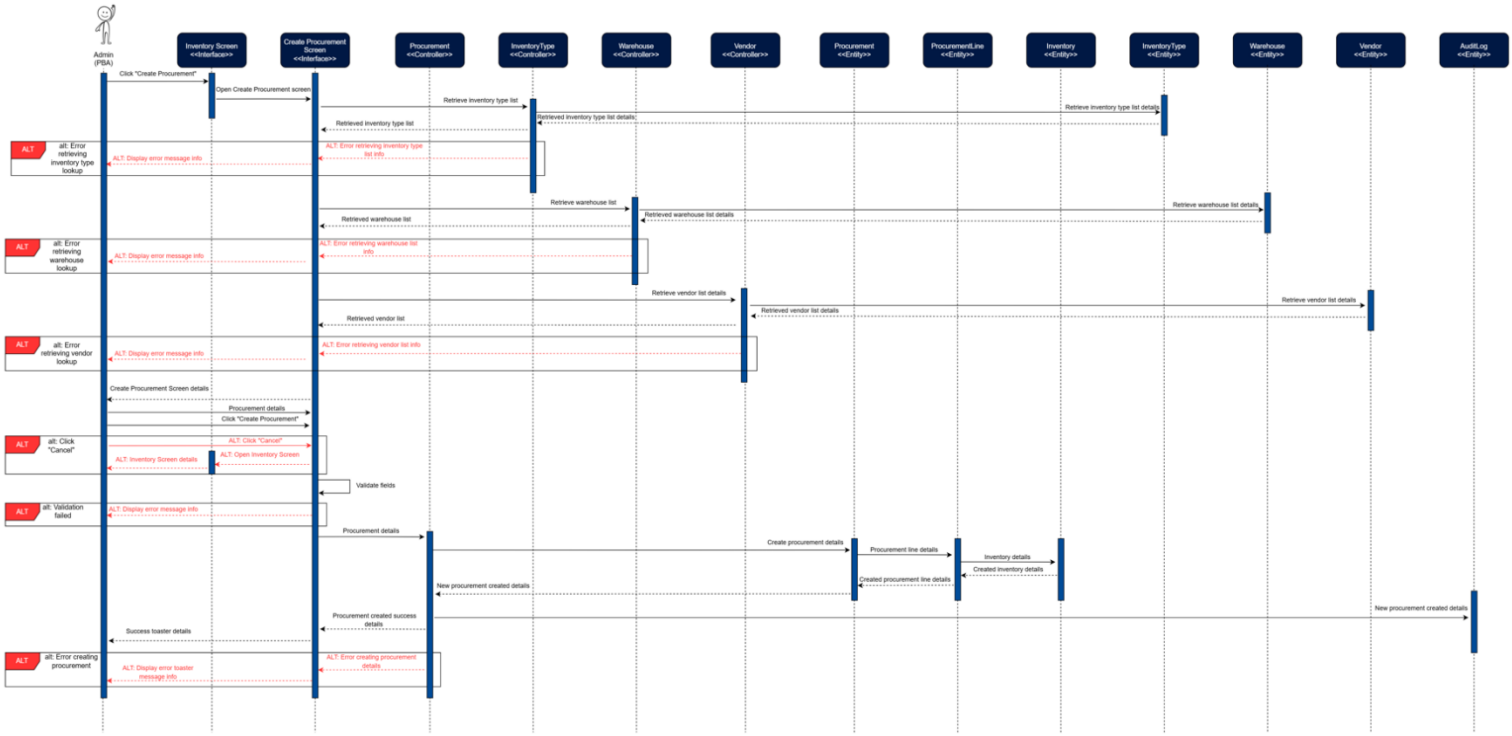


Figure 212: Sequence Diagram 3.1 Create Procurement

2.7.3.2 View Procurement: Use Case 3.2

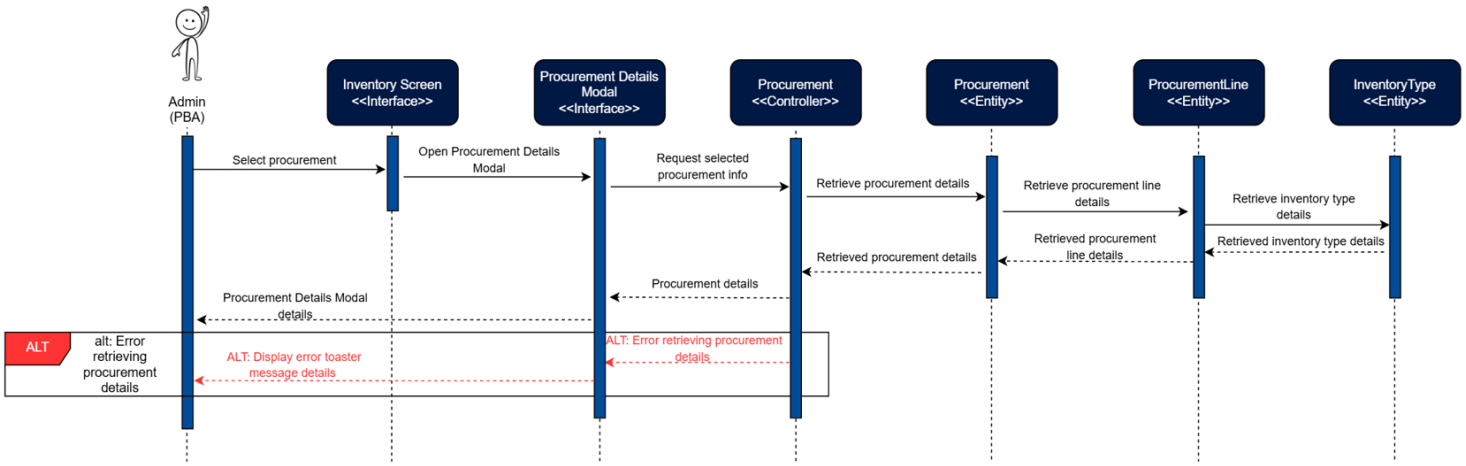


Figure 213: Sequence Diagram 3.2 View Procurement

2.7.3.3 View Inventory: Use Case 3.3

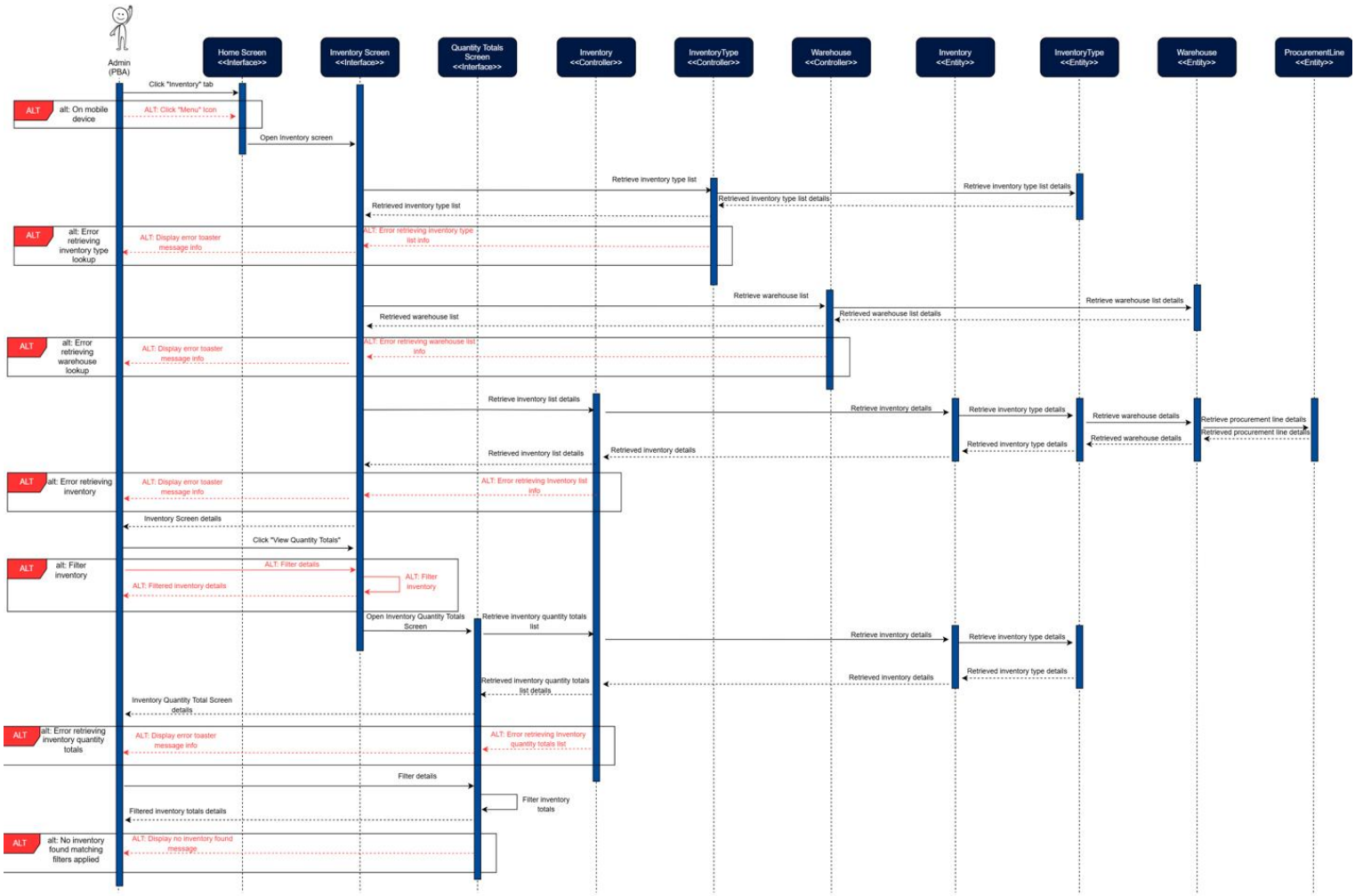


Figure 214: Sequence Diagram 3.3 View Inventory

2.7.3.4 Write-Off Inventory: Use Case 3.4

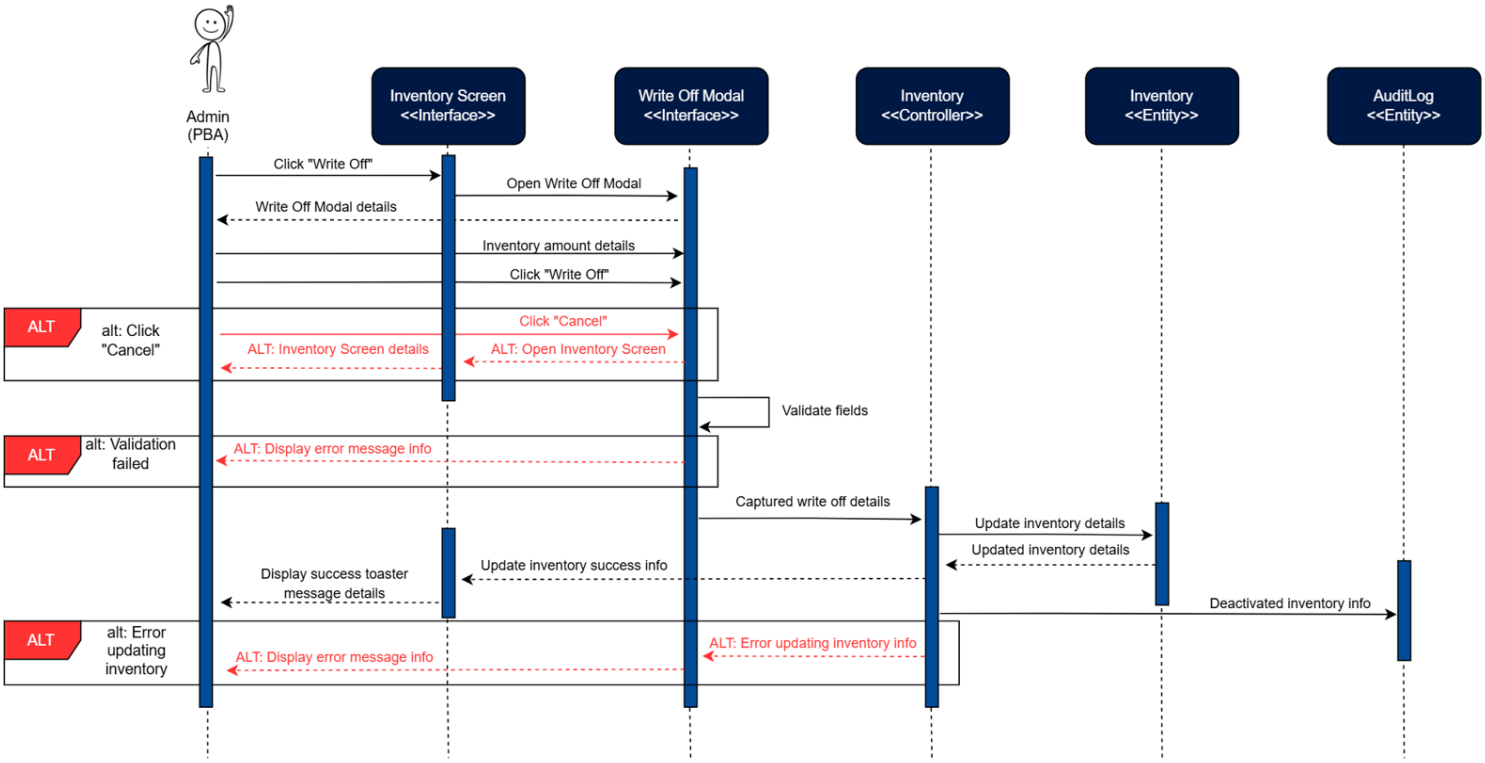


Figure 215: Sequence Diagram 3.4 Write-Off Inventory

2.7.3.5 Allocate Inventory: Use Case 3.5

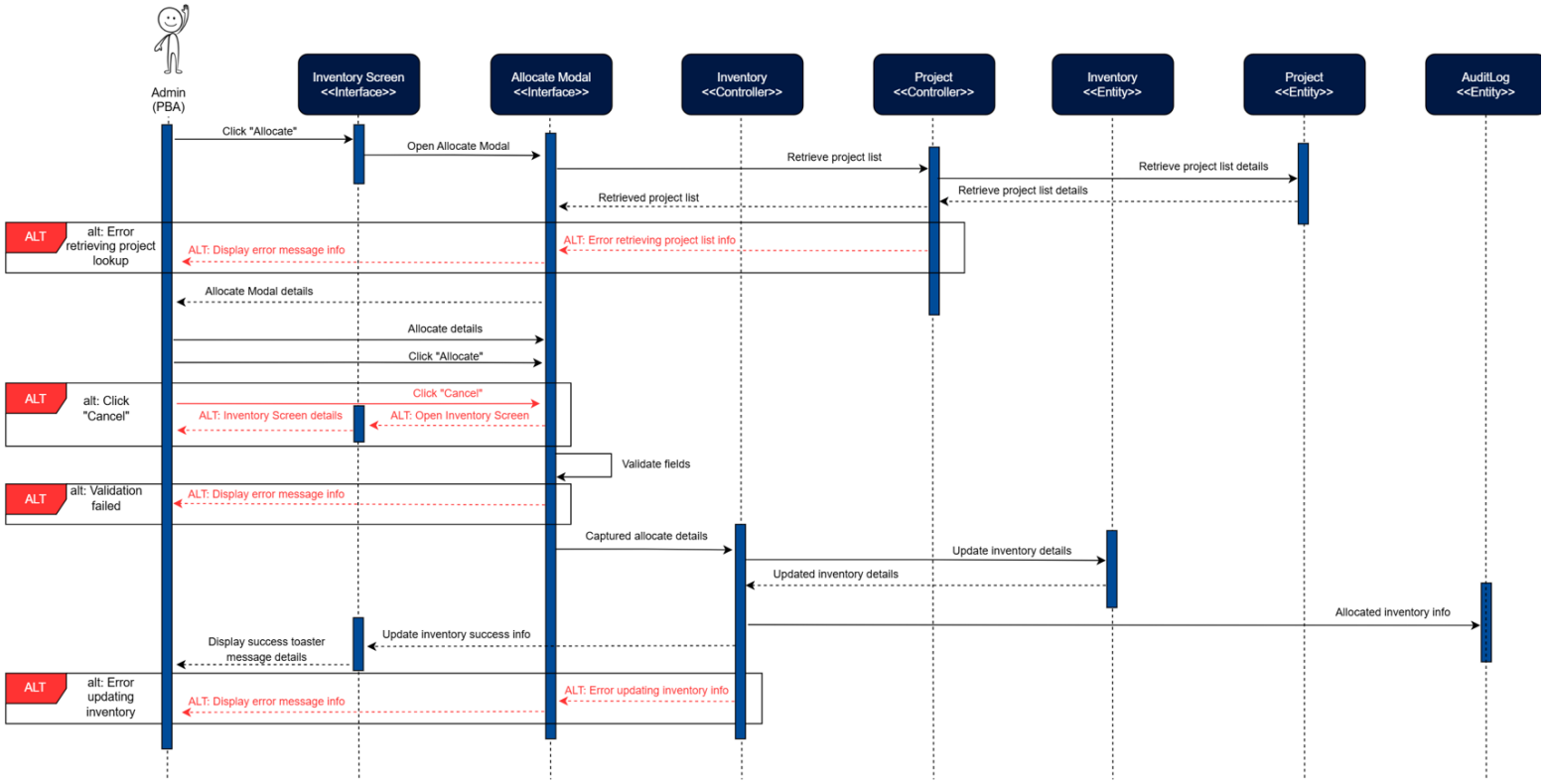


Figure 216: Sequence Diagram 3.5 Allocate Inventory

2.7.3.6 Create Inventory Item Type: Use Case 3.6

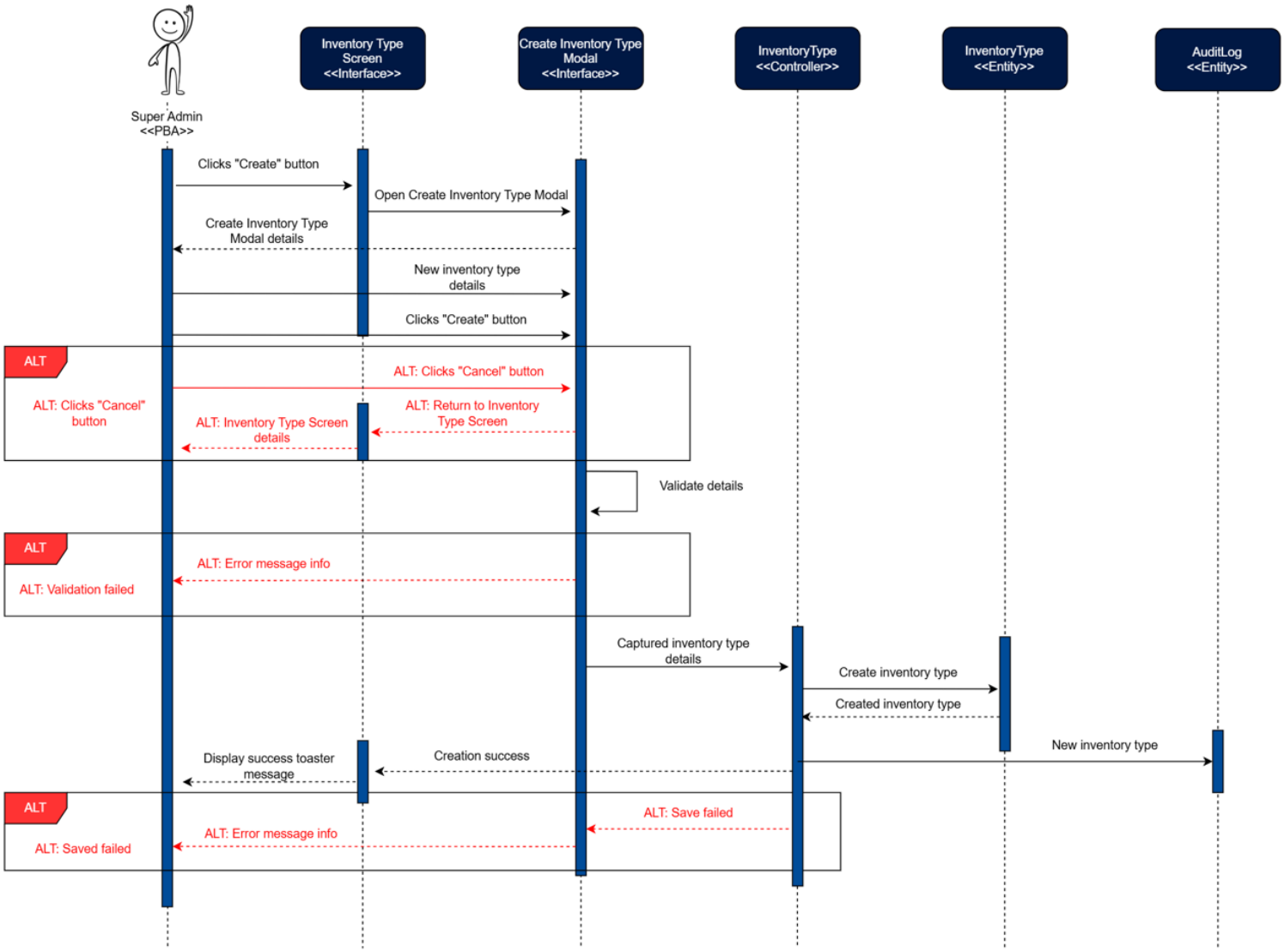


Figure 217: Sequence Diagram 3.6 Create Inventory Item Type

2.7.3.7 View Inventory Item Types: Use Case 3.7

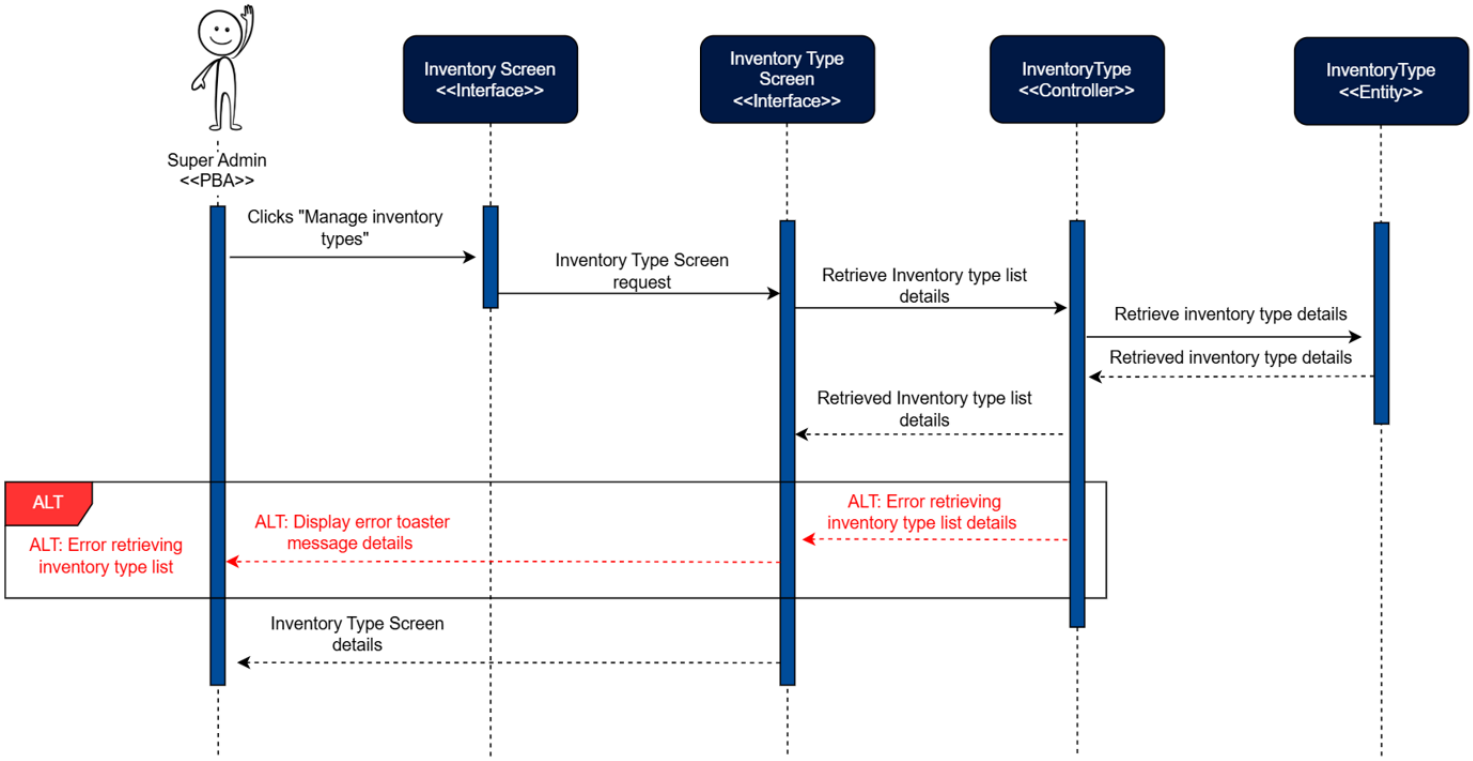


Figure 218: Sequence Diagram 3.7 View Inventory Item Types

2.7.3.8 Update Inventory Item Type: Use Case 3.8

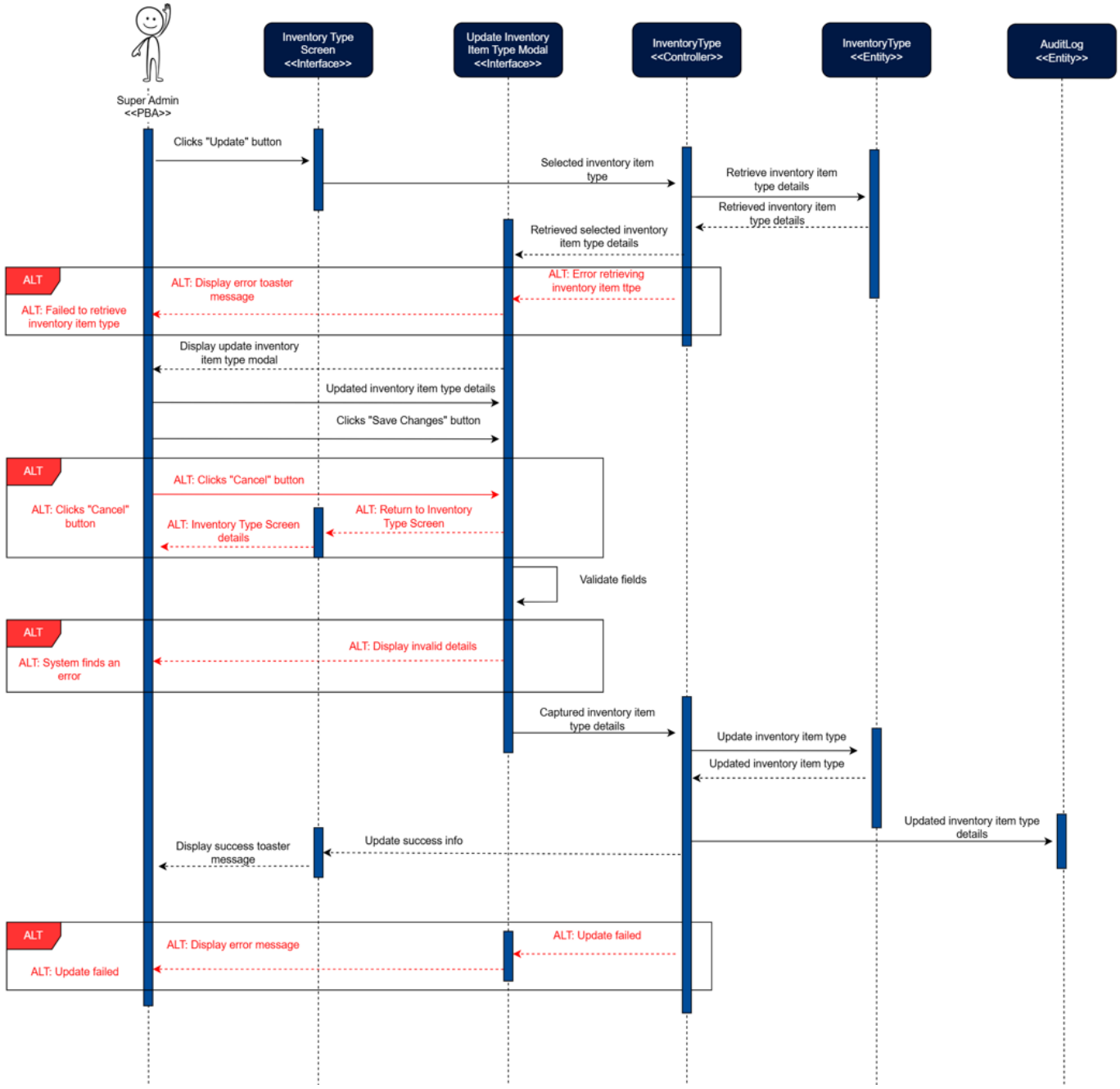


Figure 219: Sequence Diagram 3.8 Update Inventory Item Type

2.7.3.9 Deactivate Inventory Item Type: Use Case 3.9

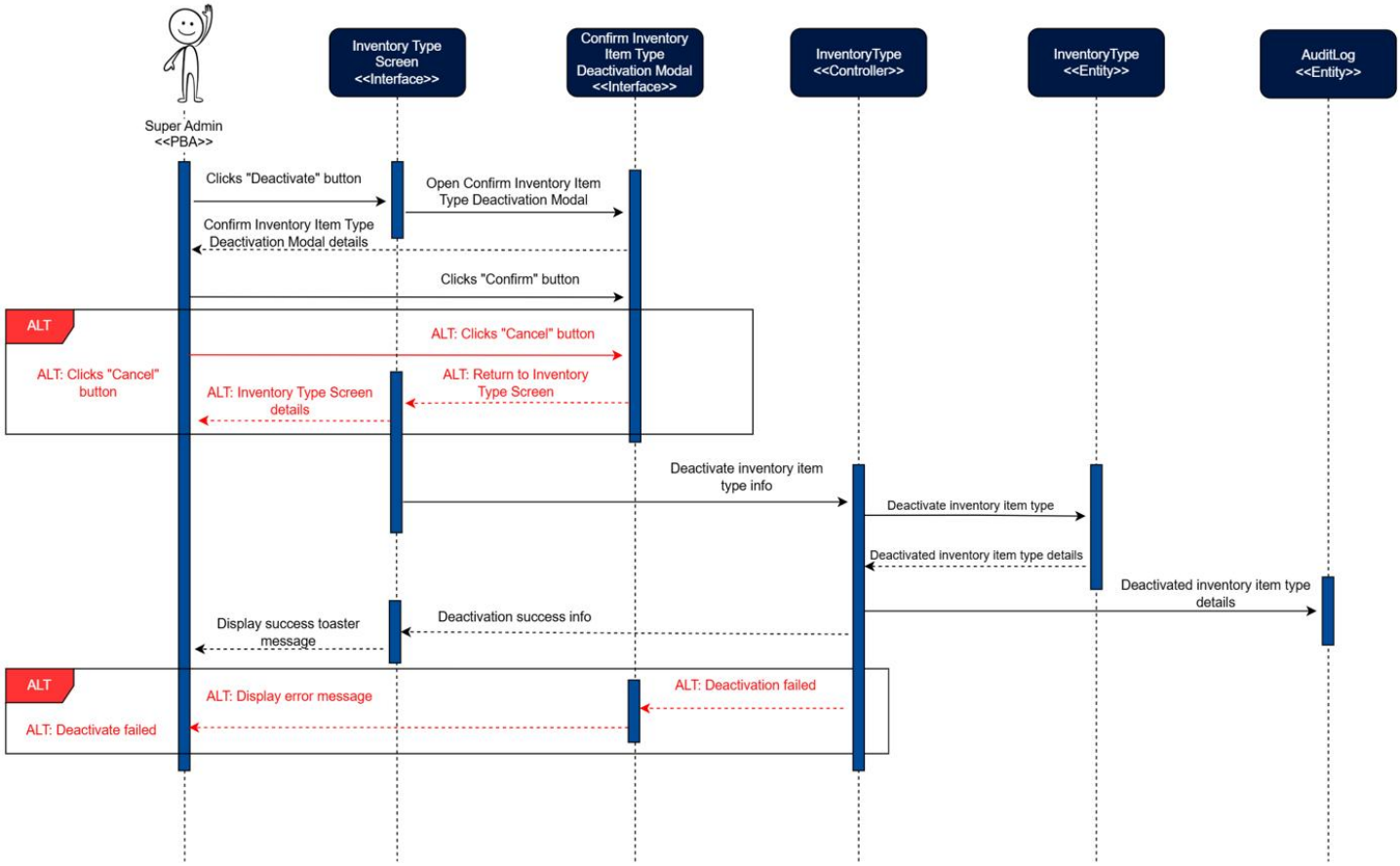


Figure 220: Sequence Diagram 3.9 Deactivate Inventory Item Type

2.7.3.10 Reactivate Inventory Item Type: Use Case 3.1

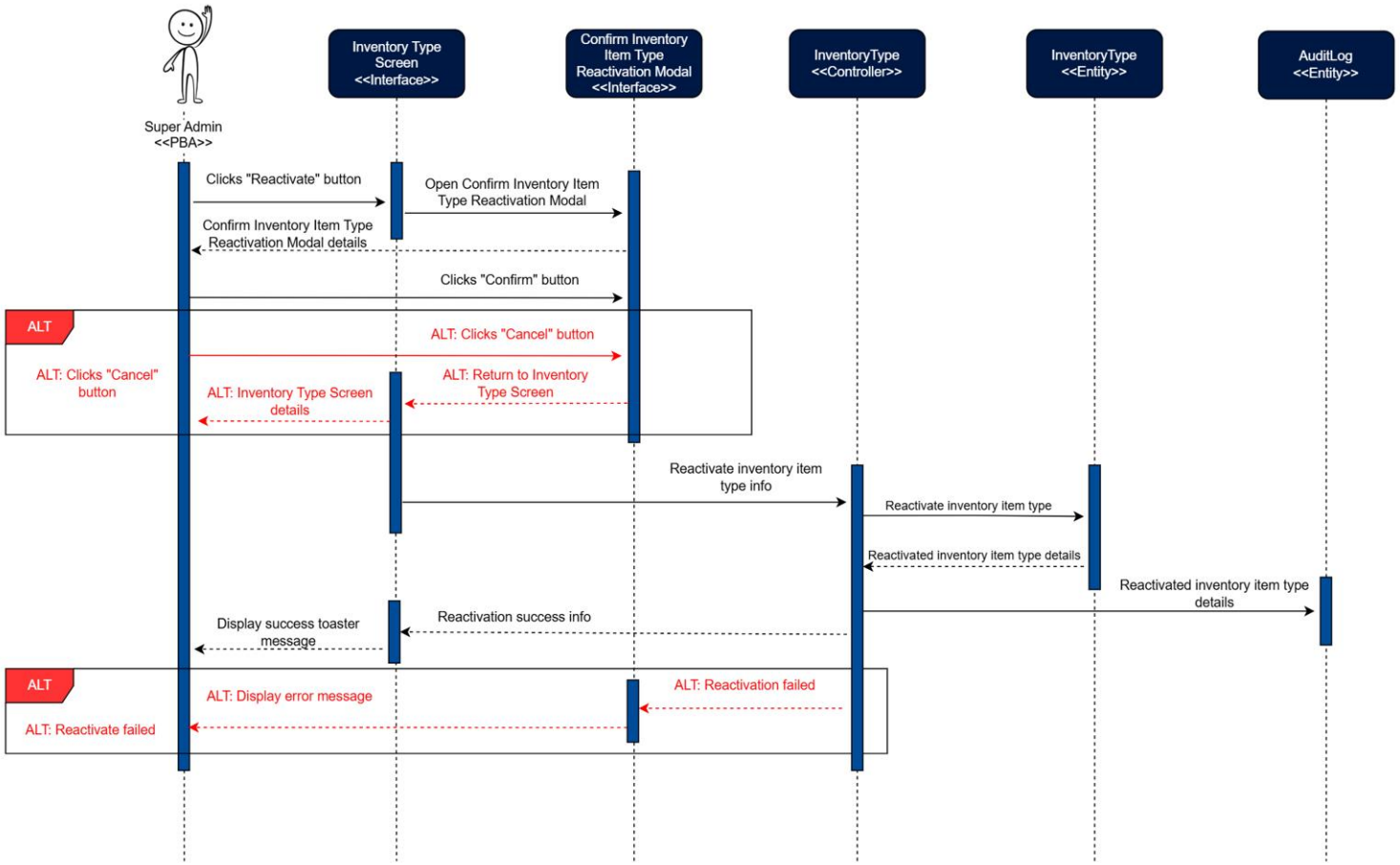


Figure 221: Sequence Diagram 3.10 Reactivate Inventory Item Type

2.7.4 *Warehouse Subsystem Sequence Diagrams*

2.7.4.1 Create Warehouse: Use Case 5.1

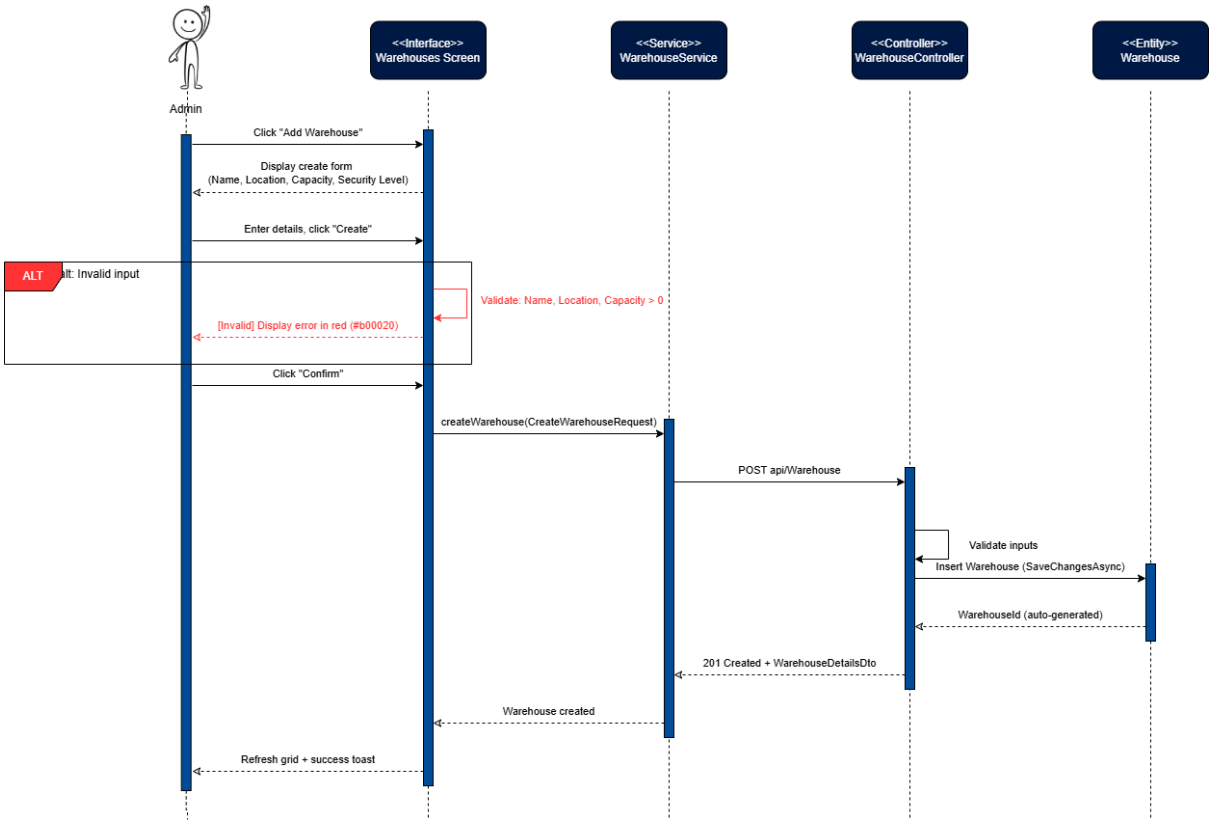


Figure 222: Sequence Diagram 5.1 Create Warehouse

2.7.4.2 View Warehouse: Use Case 5.2

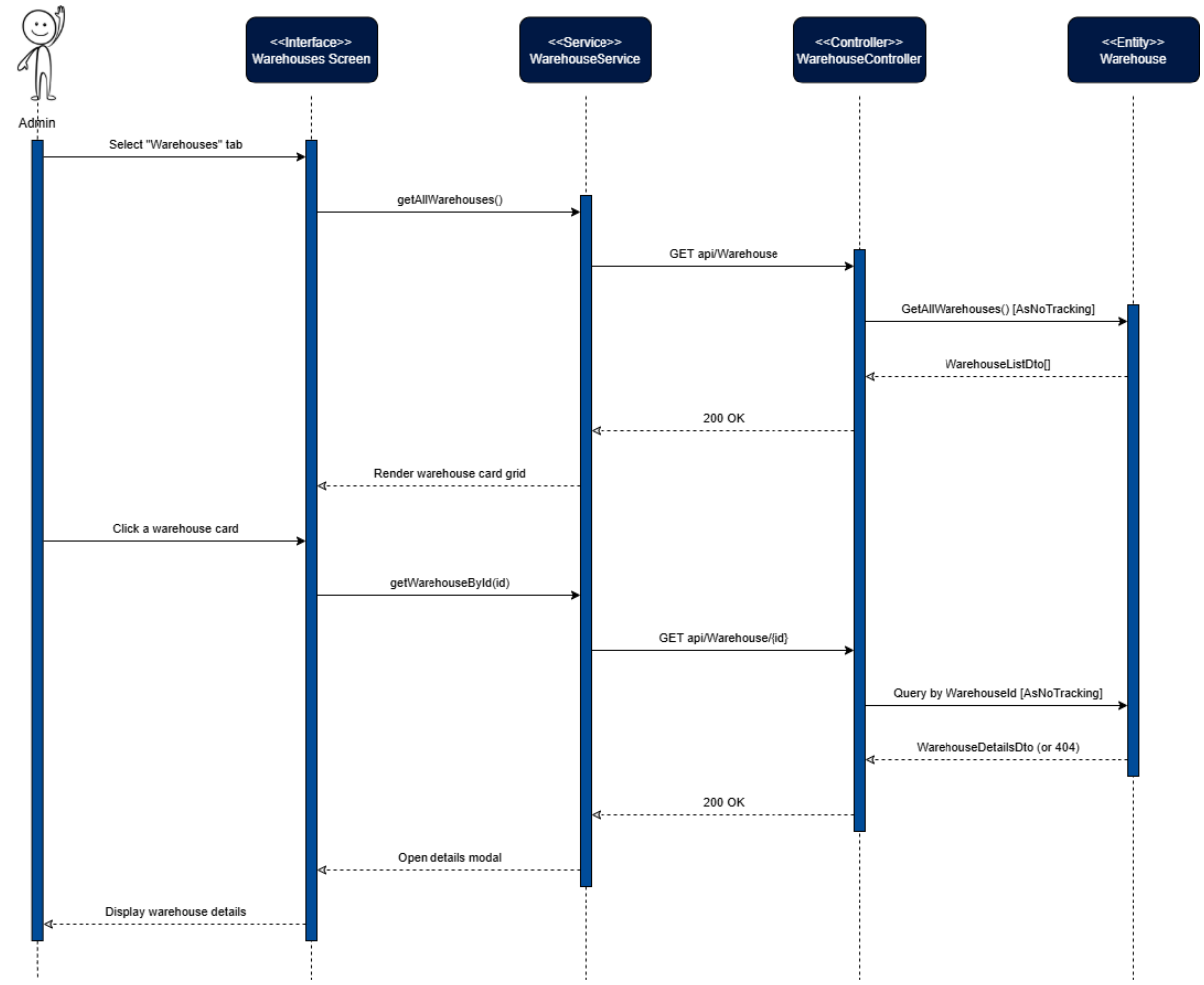


Figure 223: Sequence Diagram 5.2 View Warehouse

2.7.4.3 Update Warehouse: Use Case 5.3

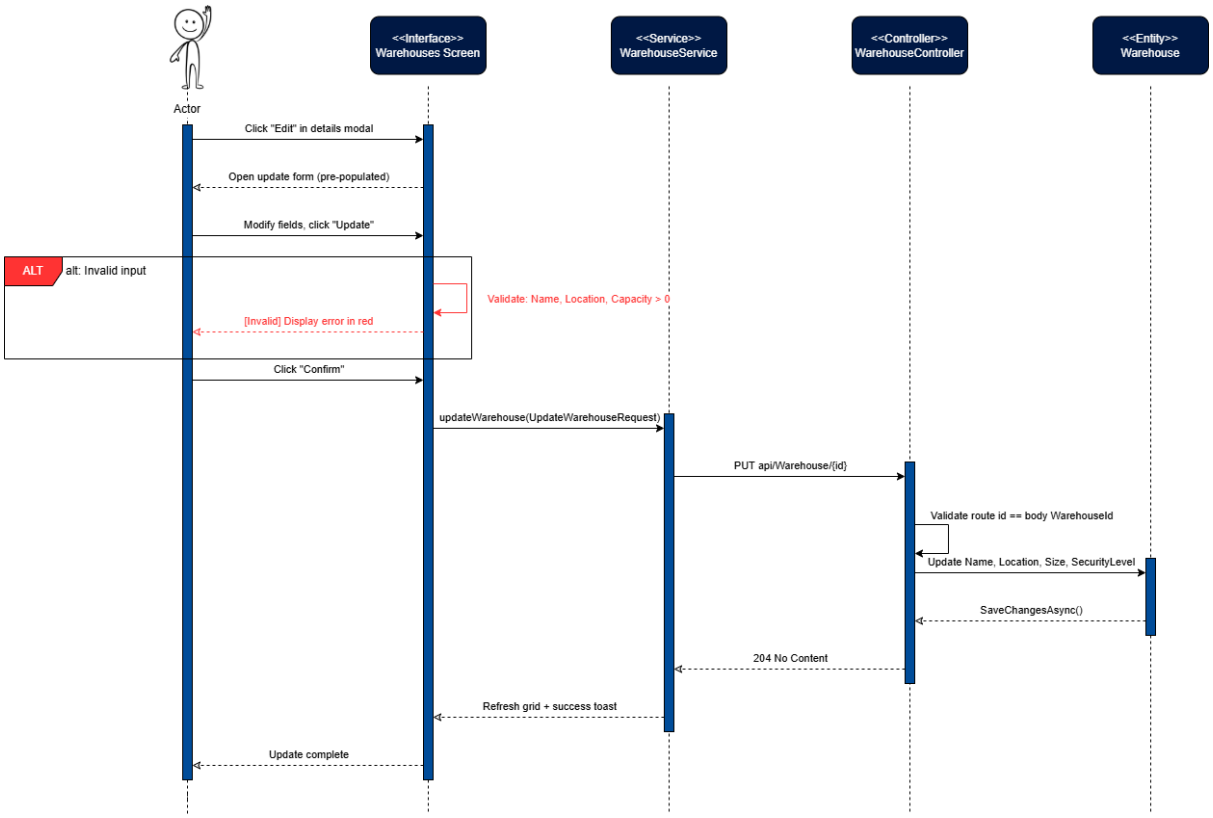


Figure 224: Sequence Diagram 5.3 Update Warehouse

2.7.4.4 Reactivate Warehouse: Use Case 5.4

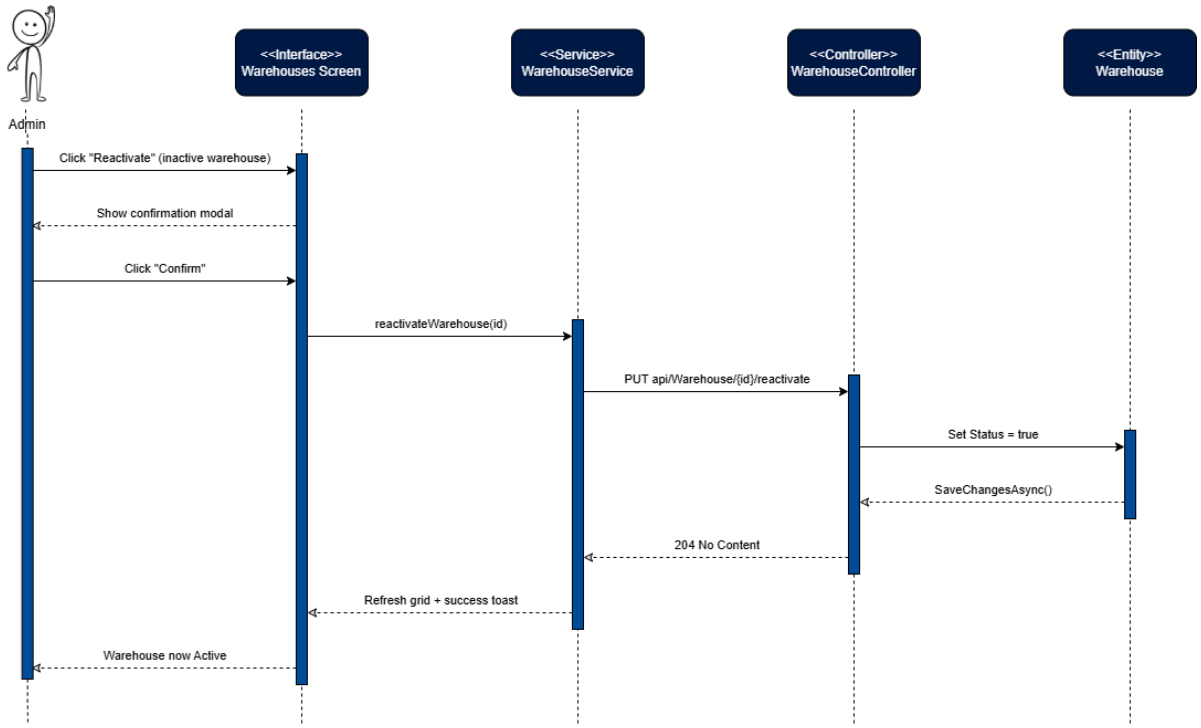


Figure 225: Sequence Diagram 5.4 Reactivate Warehouse

2.7.4.5 Deactivate Warehouse: Use Case 5.5

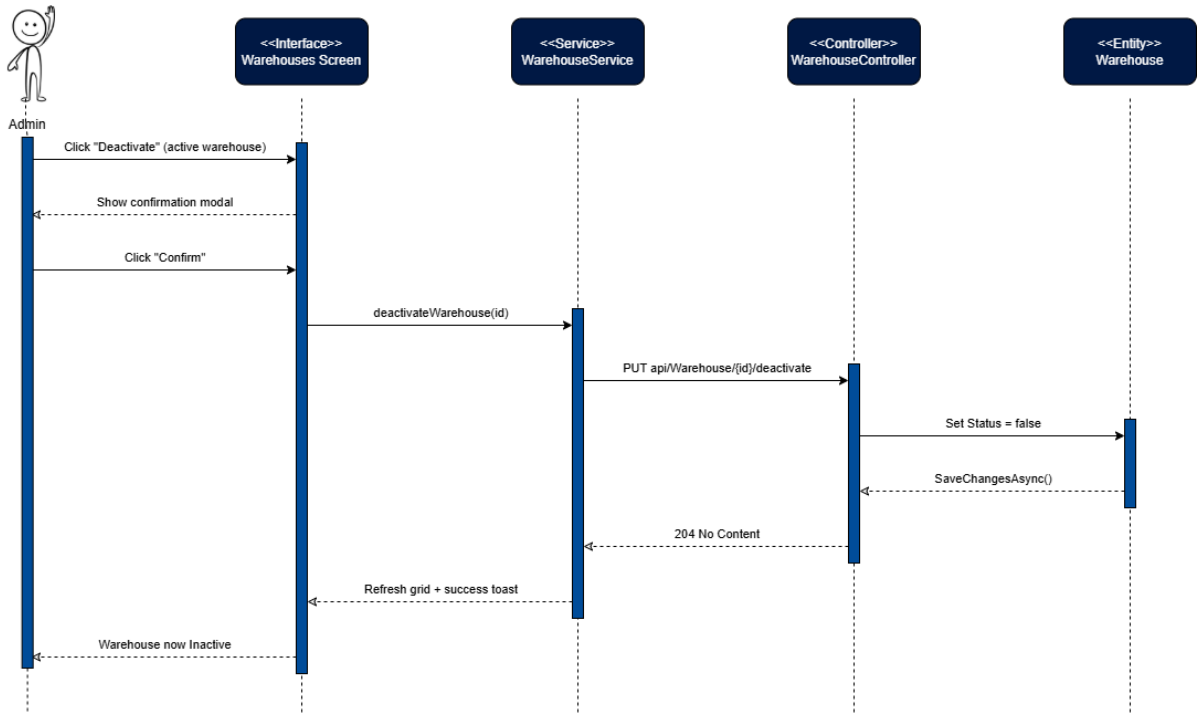


Figure 226: Sequence Diagram 5.5 Deactivate Warehouse

2.7.4.6 Inventory Stock Take: Use Case 5.6

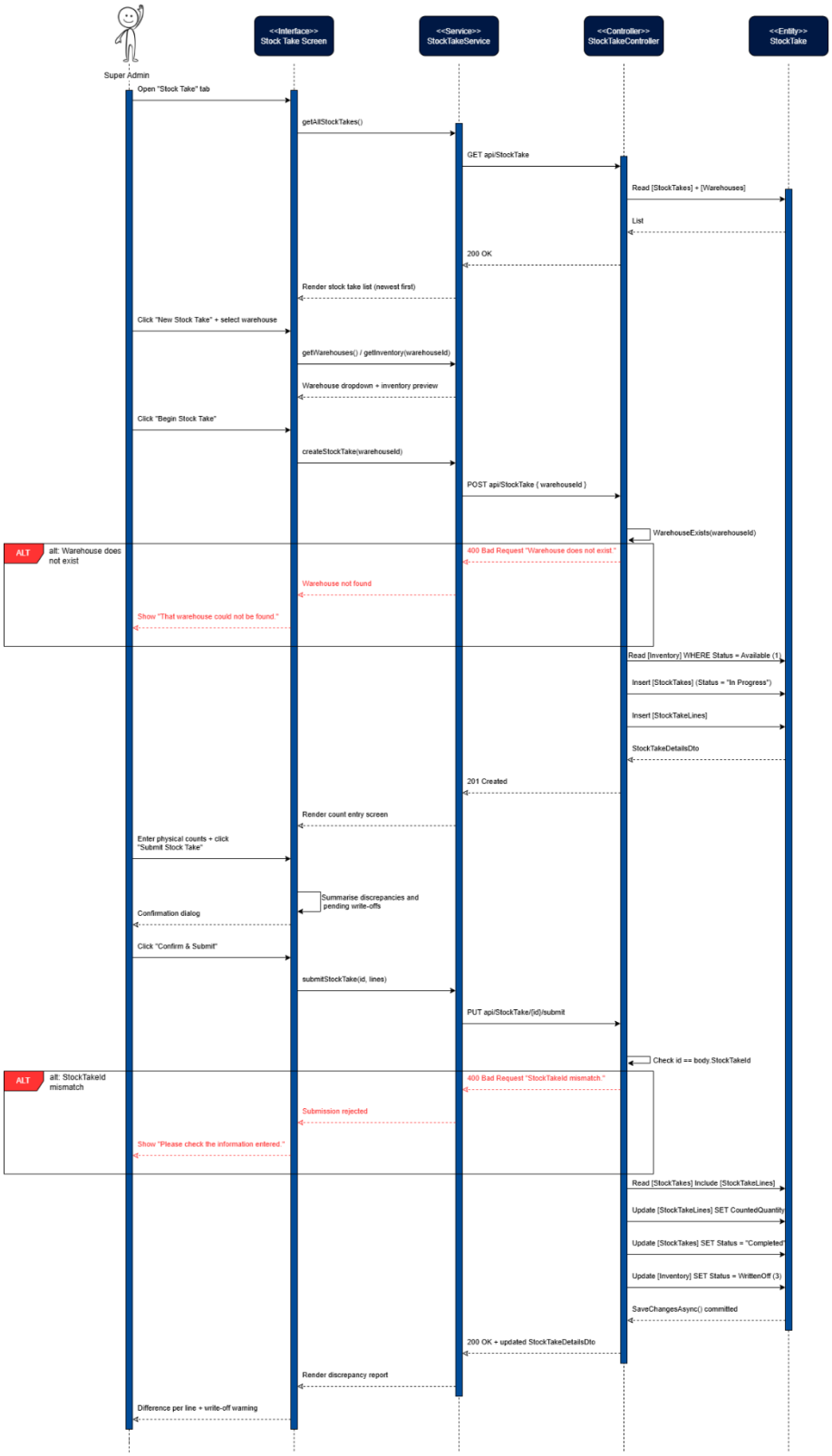


Figure 227: Sequence Diagram 5.6 Inventory Stock Take

2.7.5 *Vendor Subsystem Sequence Diagrams*

2.7.5.1 Create Vendor: Use Case 6.1

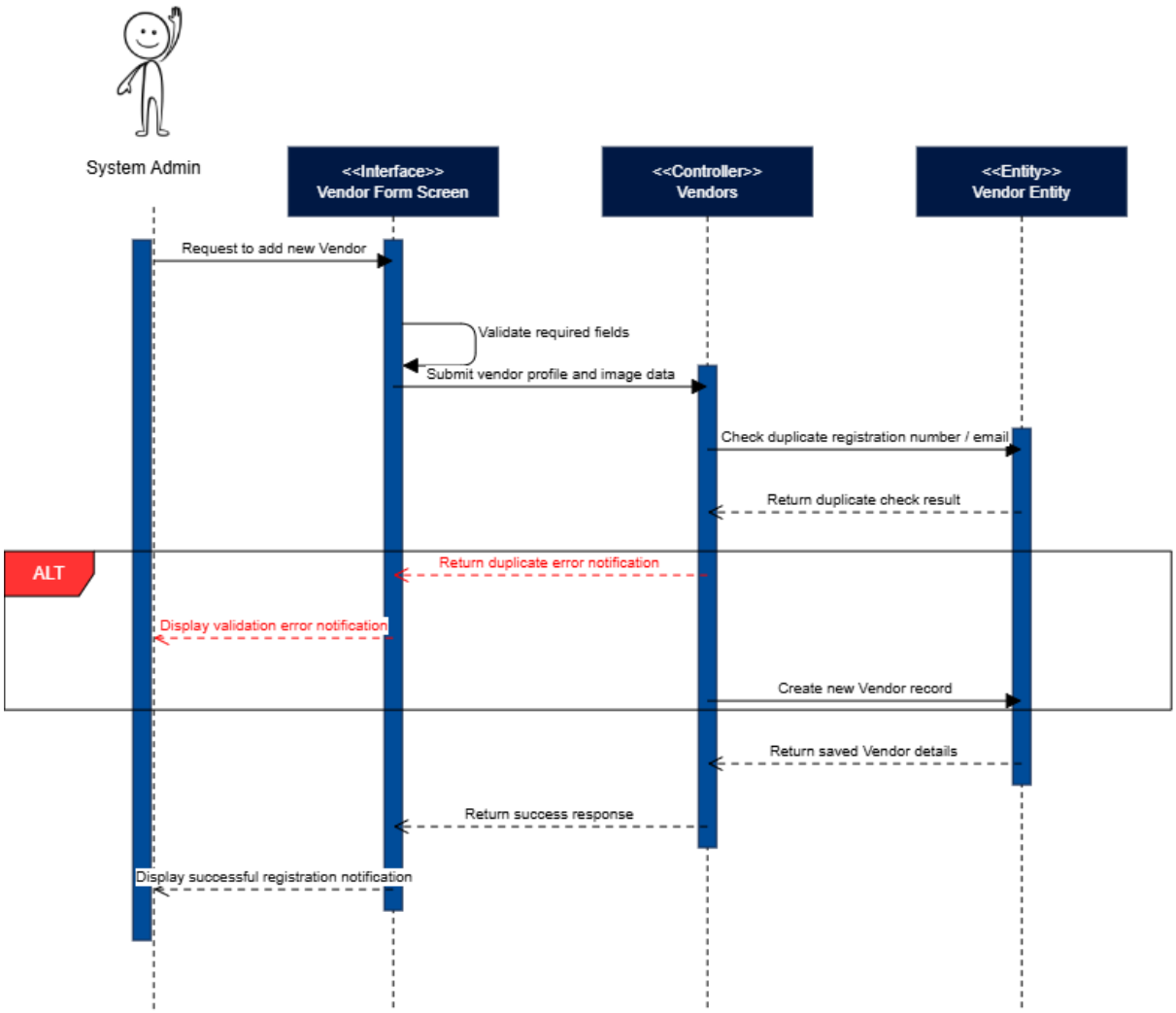


Figure 228: Sequence Diagram 6.1 Create Vendor

2.7.5.2 View Vendor: Use Case 6.2

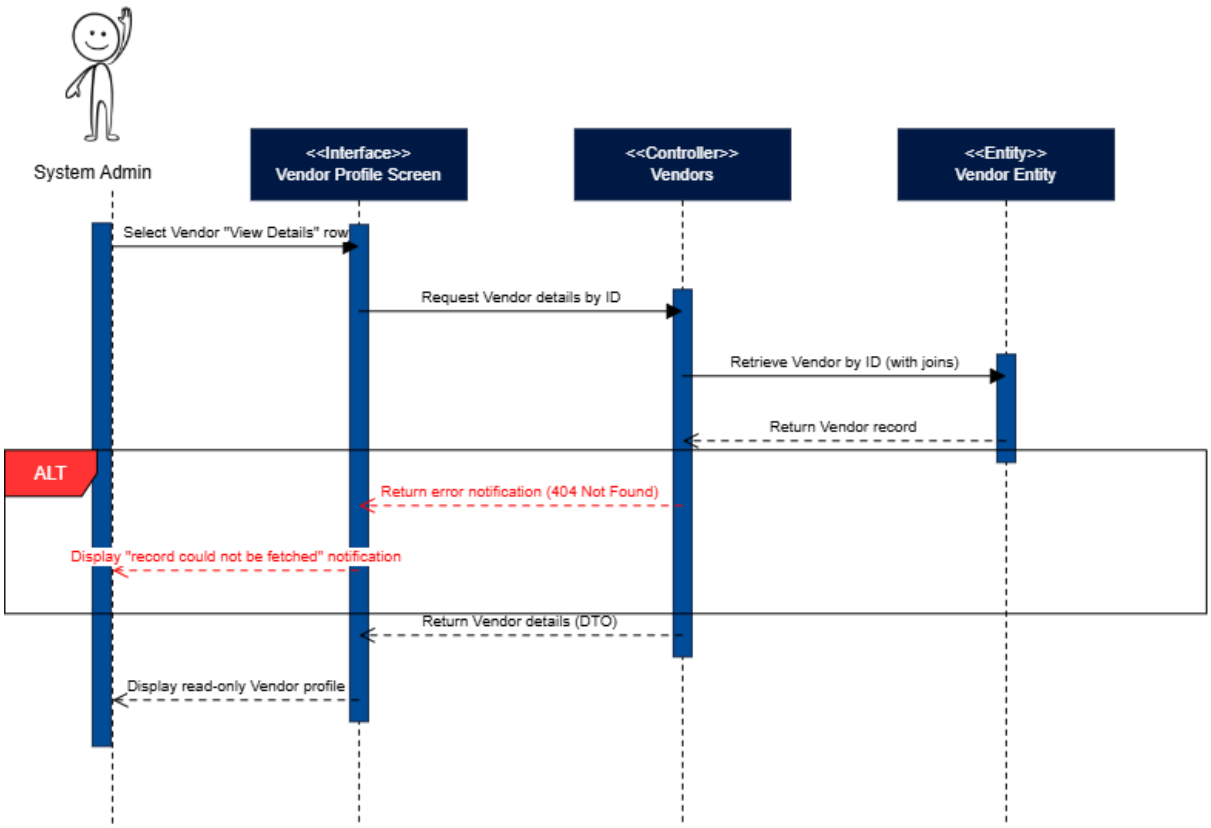


Figure 229: Sequence Diagram 6.2 View Vendor

2.7.5.3 View Vendors: Use Case 6.3

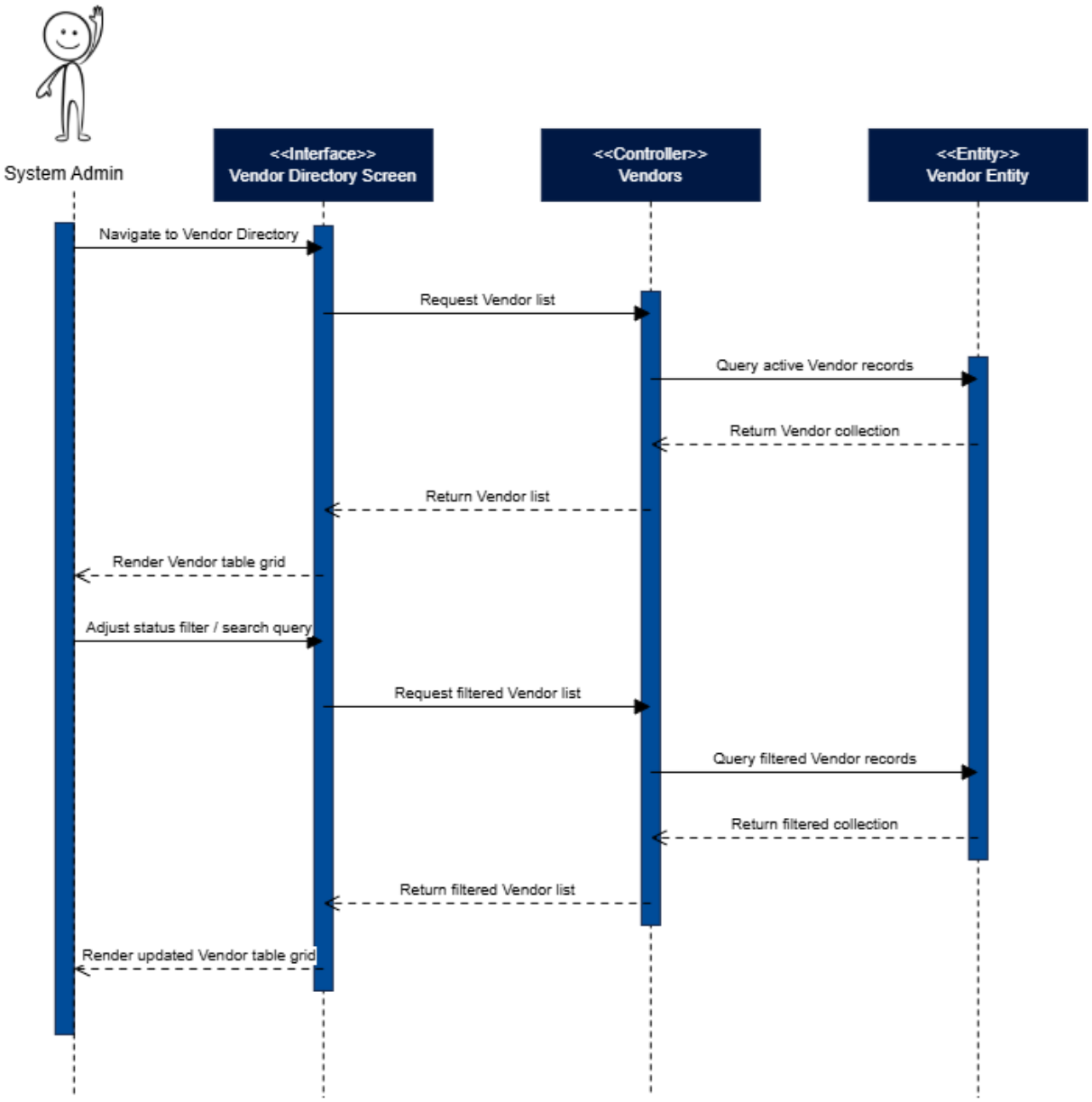


Figure 230: Sequence Diagram 6.3 View Vendors

2.7.5.4 Edit Vendor: Use Case 6.4

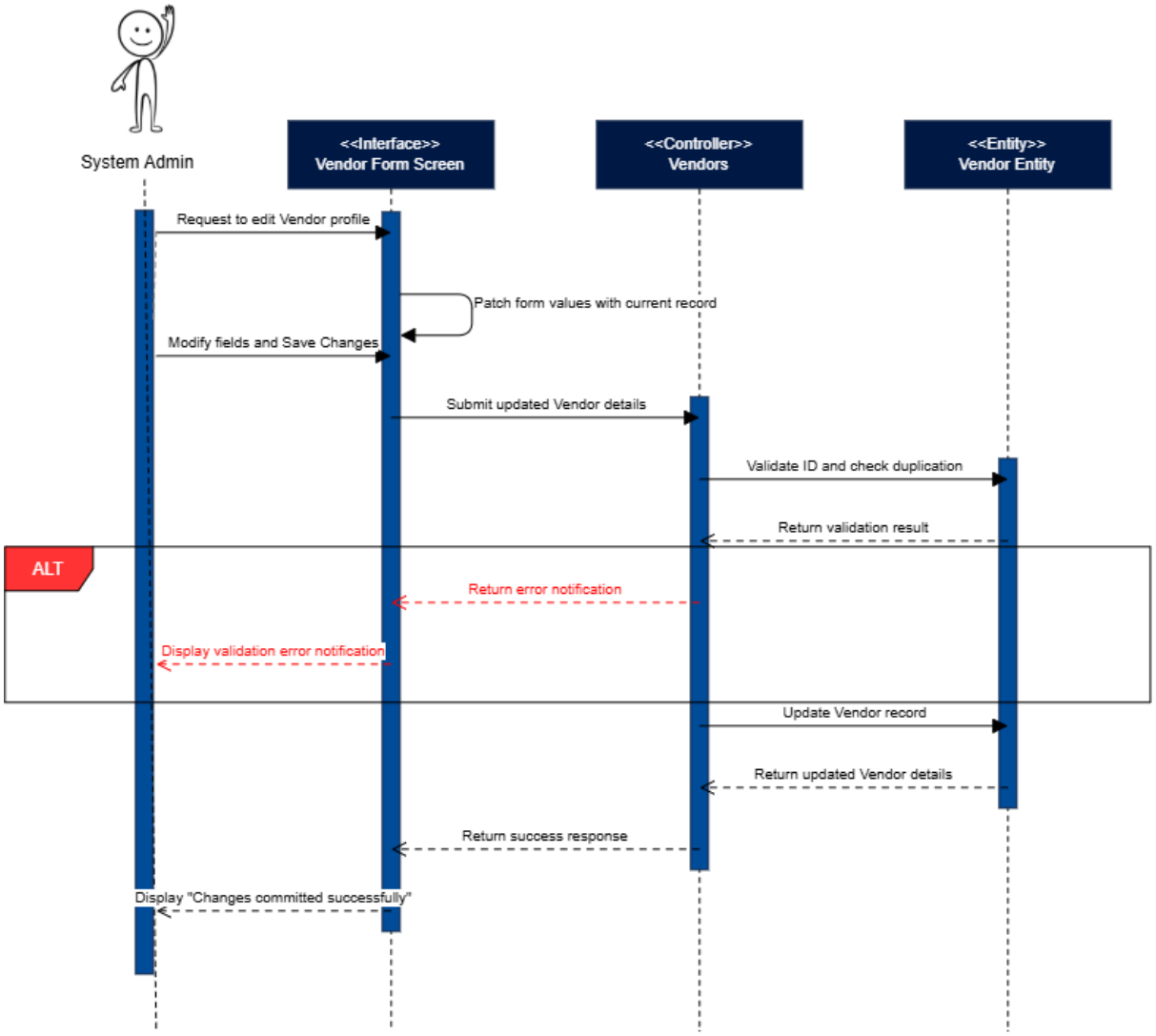


Figure 231: Sequence Diagram 6.4 Edit Vendor

2.7.5.5 Reactivate Vendor: Use Case 6.5

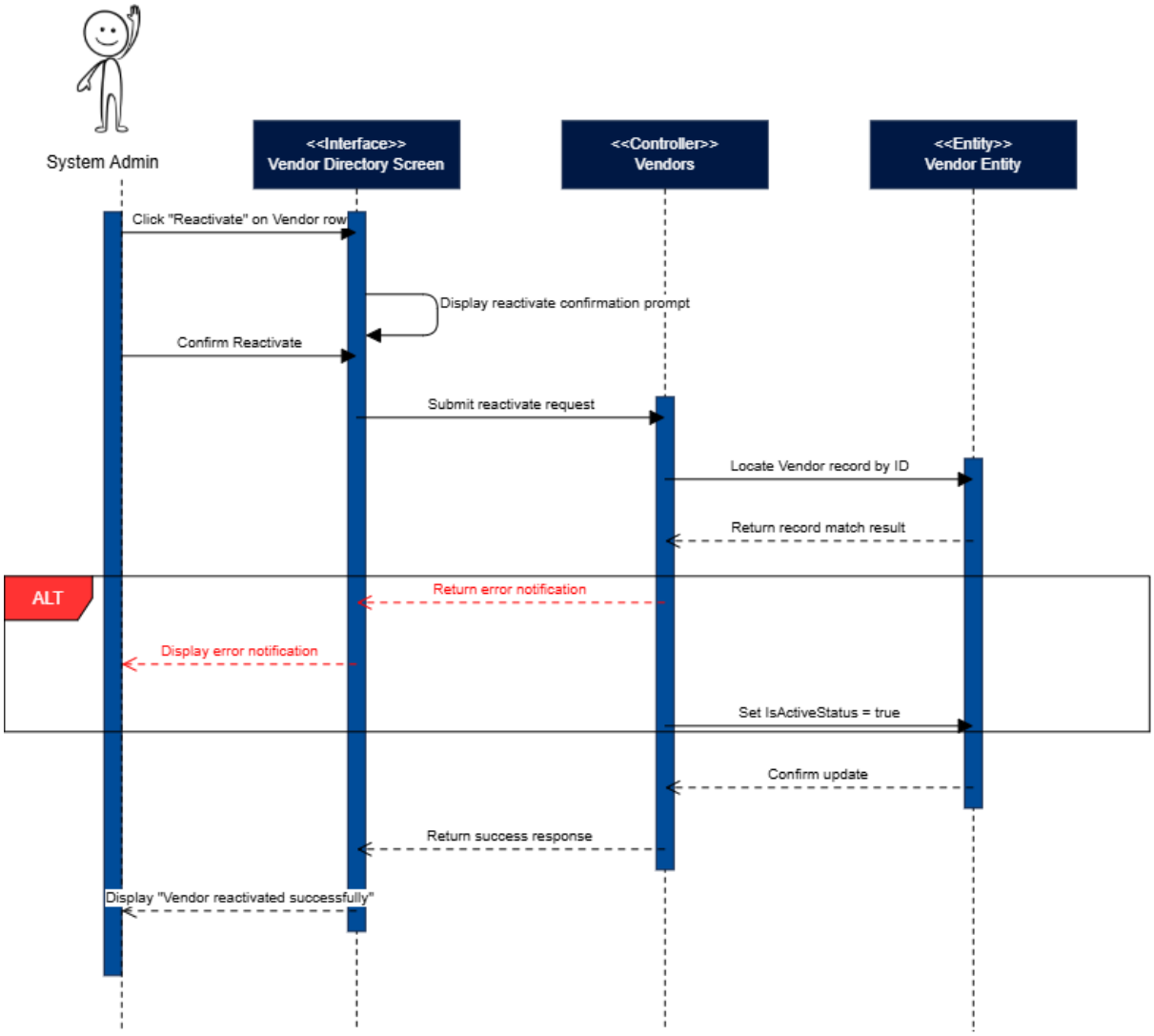


Figure 232: Sequence Diagram 6.5 Reactivate Vendor

2.7.5.6 Deactivate Vendor: Use Case 6.6

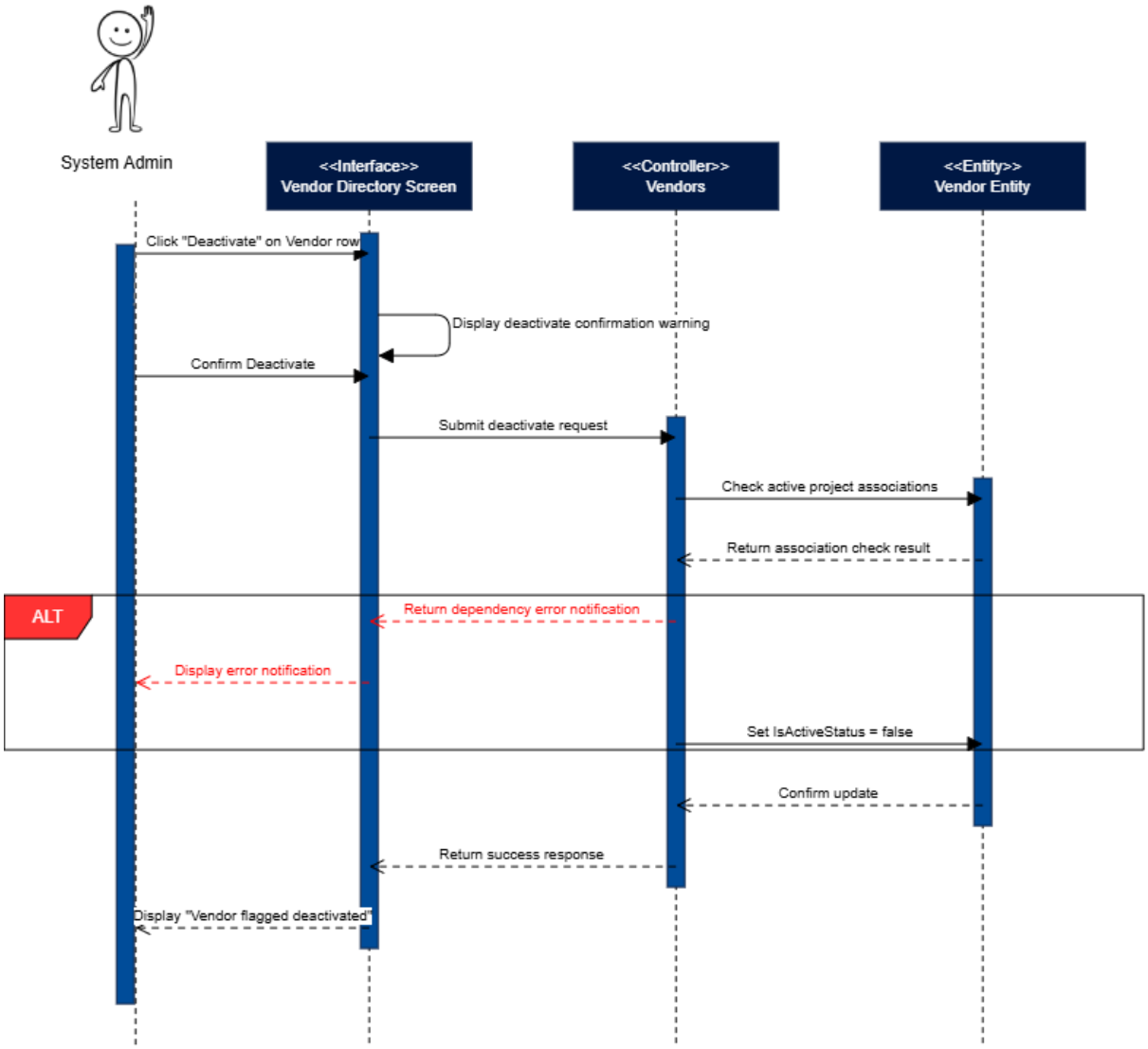


Figure 233: Sequence Diagram 6.6 Deactivate Vendor

2.7.5.7 Create Vendor Type: Use Case 6.7

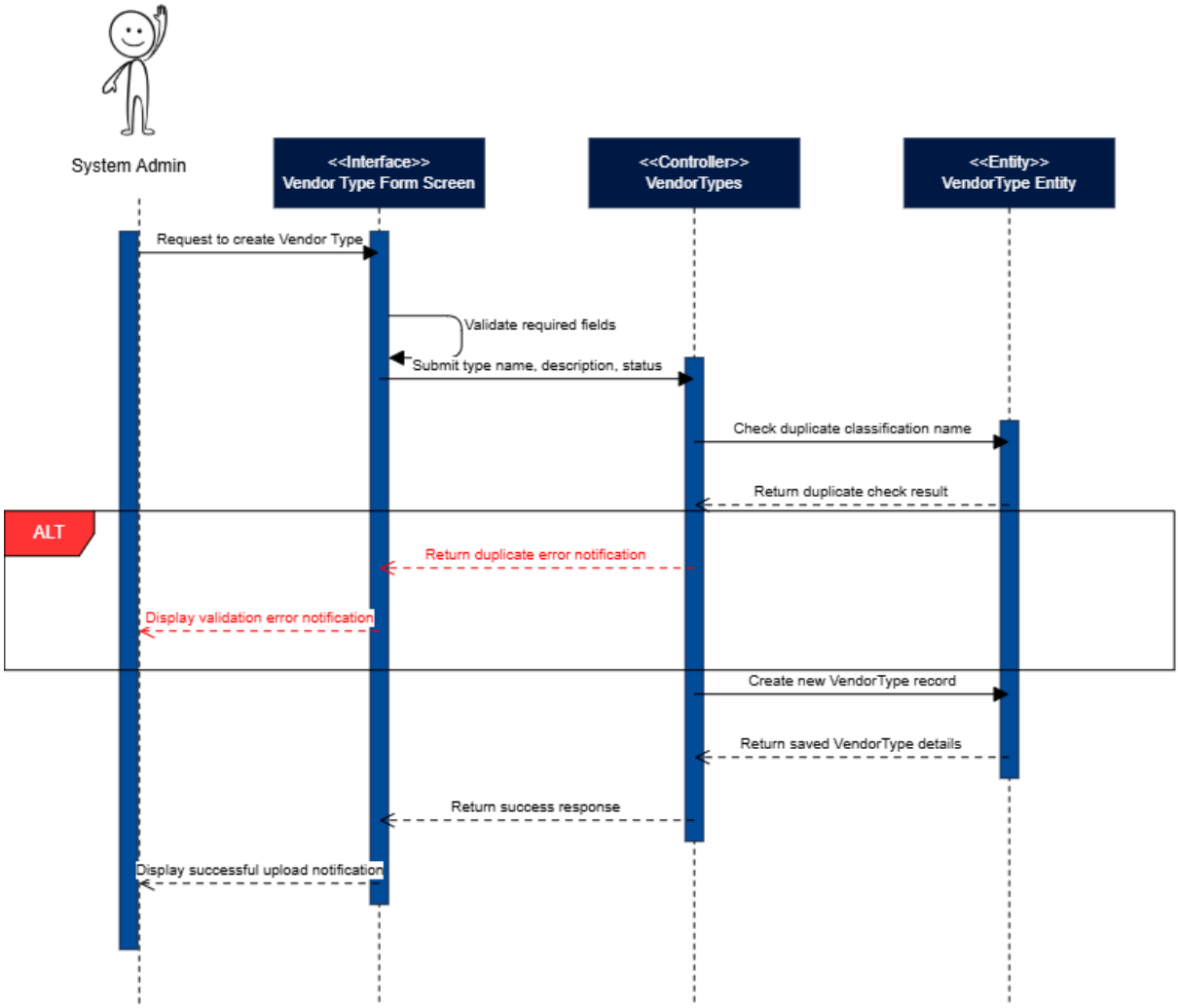


Figure 234: Sequence Diagram 6.7 Create Vendor Type

2.7.5.8 Update Vendor Type: Use Case 6.8

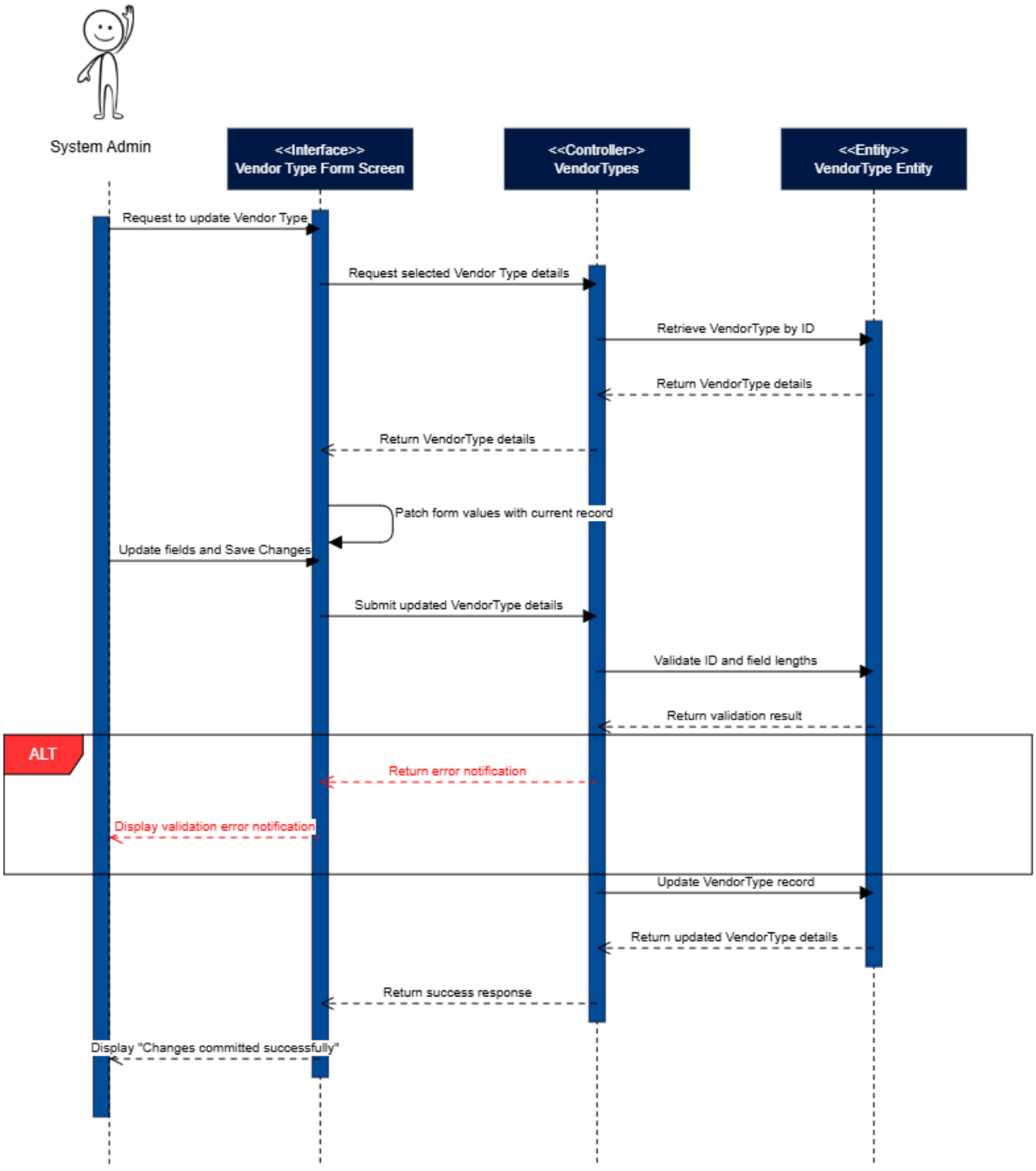


Figure 235: Sequence Diagram 6.8 Update Vendor Type

2.7.5.9 View Vendor Types: Use Case 6.9

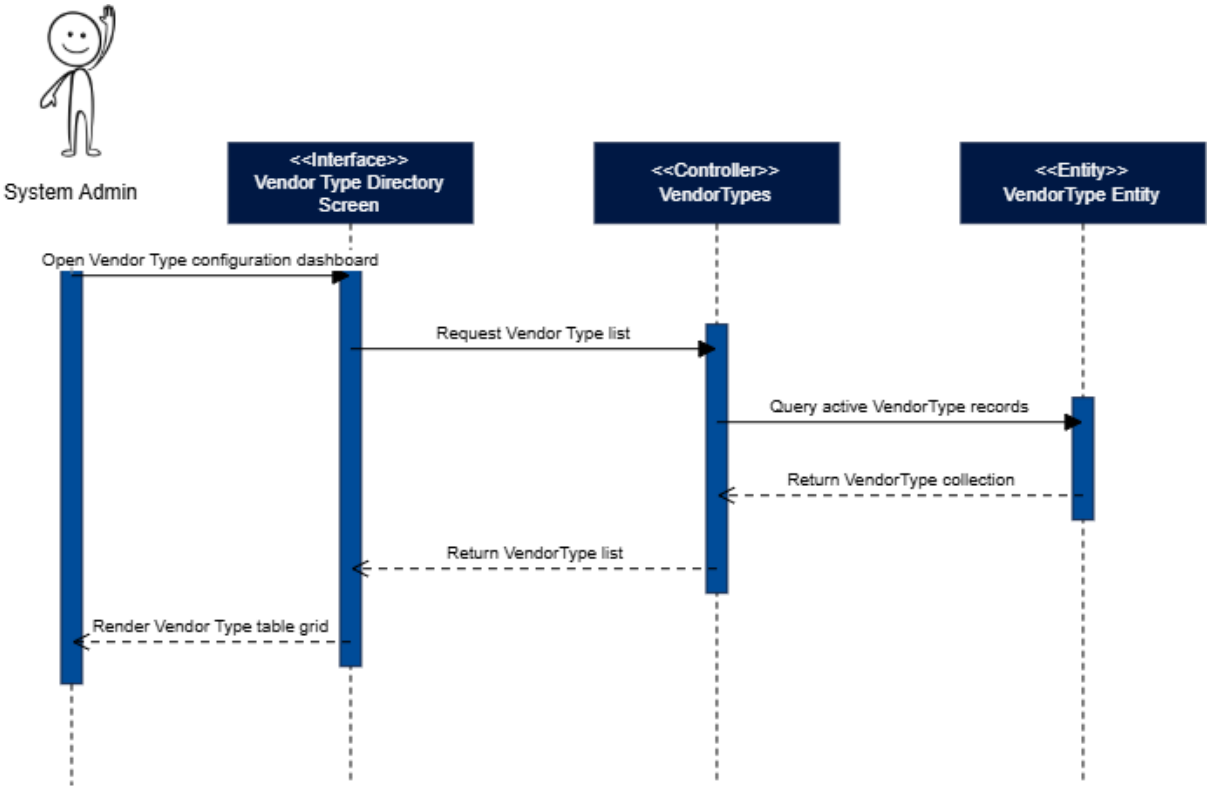


Figure 236: Sequence Diagram 6.9 View Vendor Types

2.7.5.10 Reactivate Vendor Type: Use Case 6.10

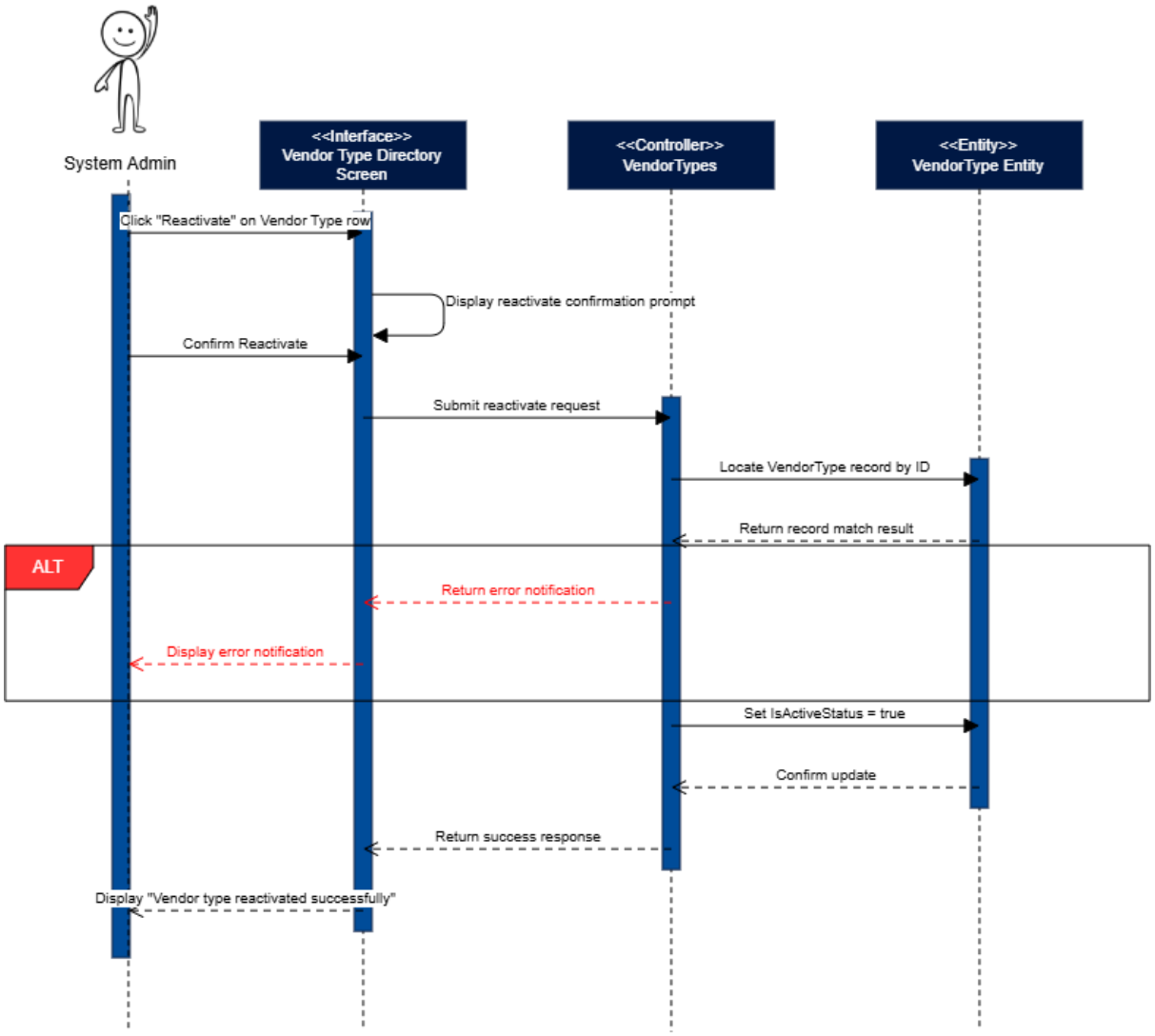


Figure 237: Sequence Diagram 6.10 Reactivate Vendor Type

2.7.5.11 Deactivate Vendor Type: Use Case 6.11

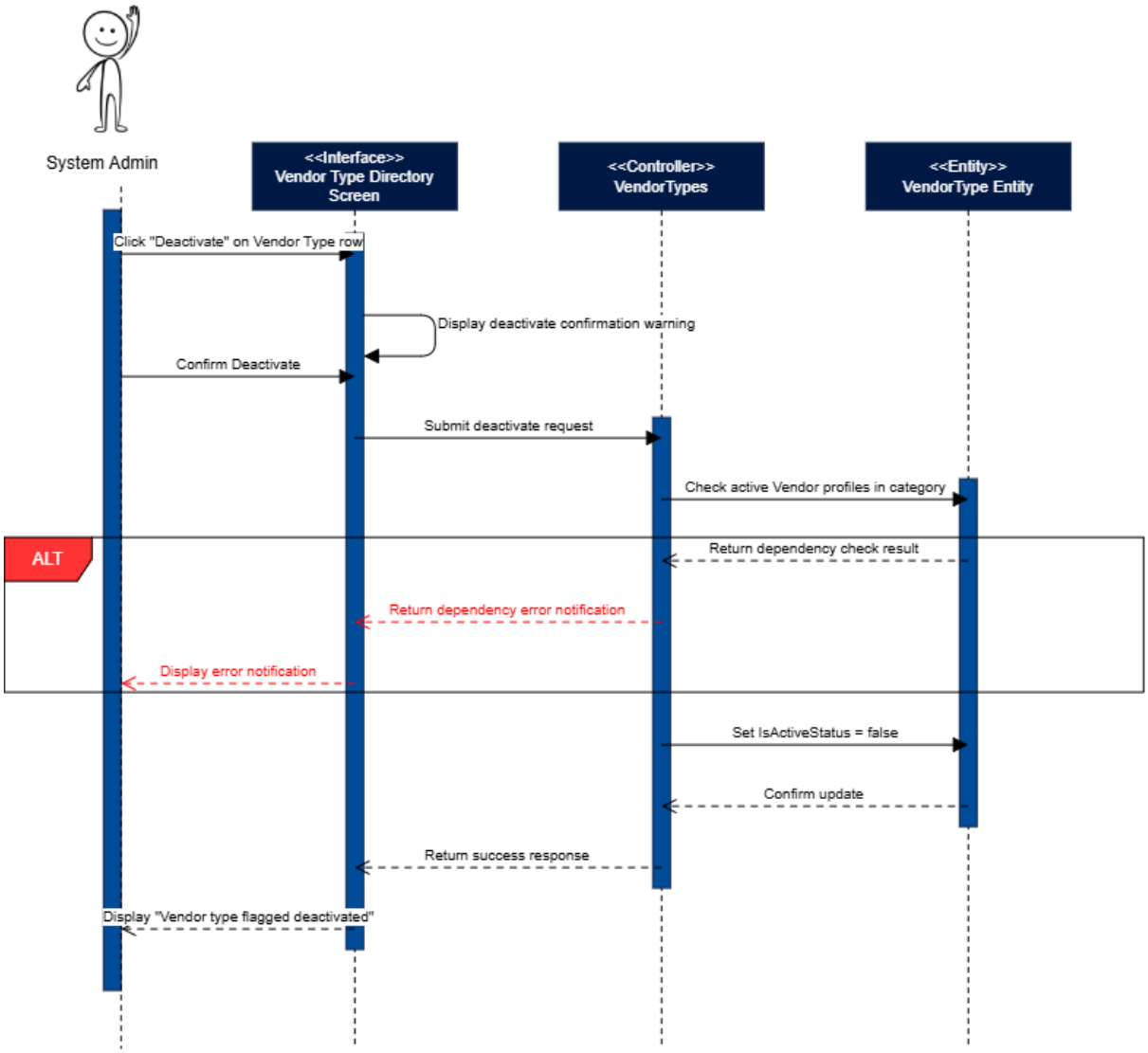


Figure 238: Sequence Diagram 6.11 Deactivate Vendor Type

2.7.6 *Project Management Subsystem Sequence Diagrams*

2.7.6.1 Create Project Feedback: Use Case 7.14

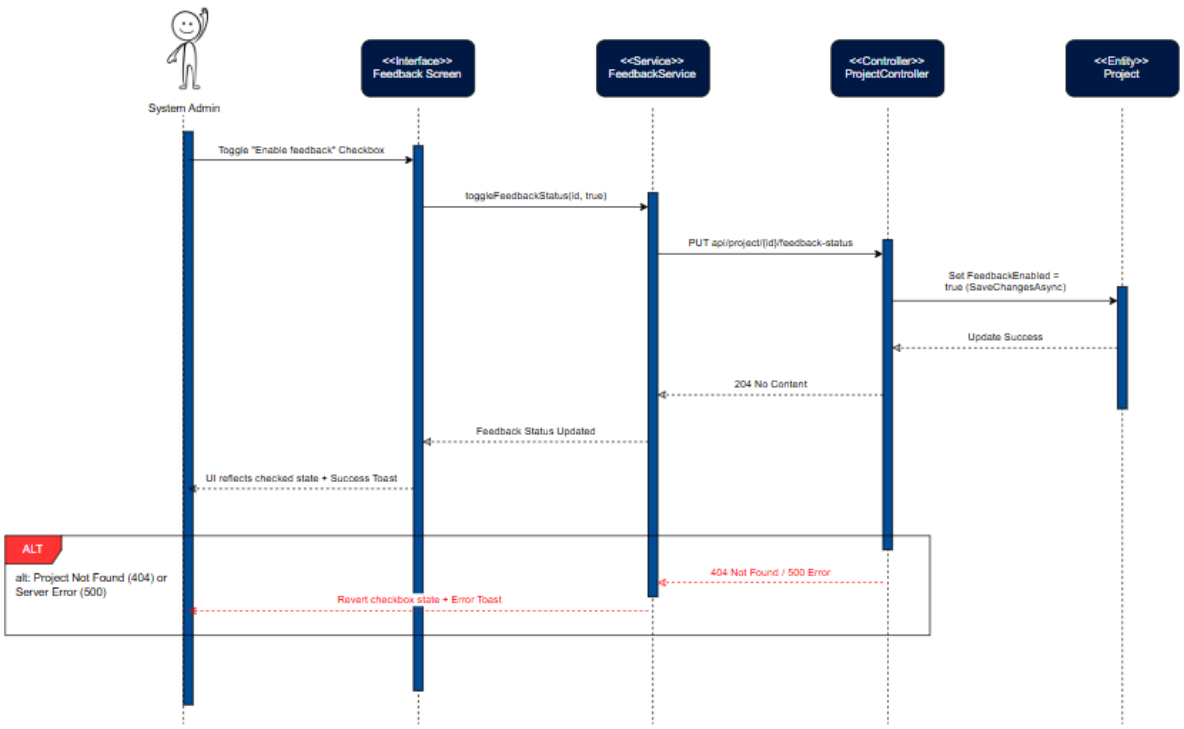


Figure 239: Sequence Diagram 7.14 Create Project Feedback

2.7.6.2 View Project Feedback: Use Case 7.15

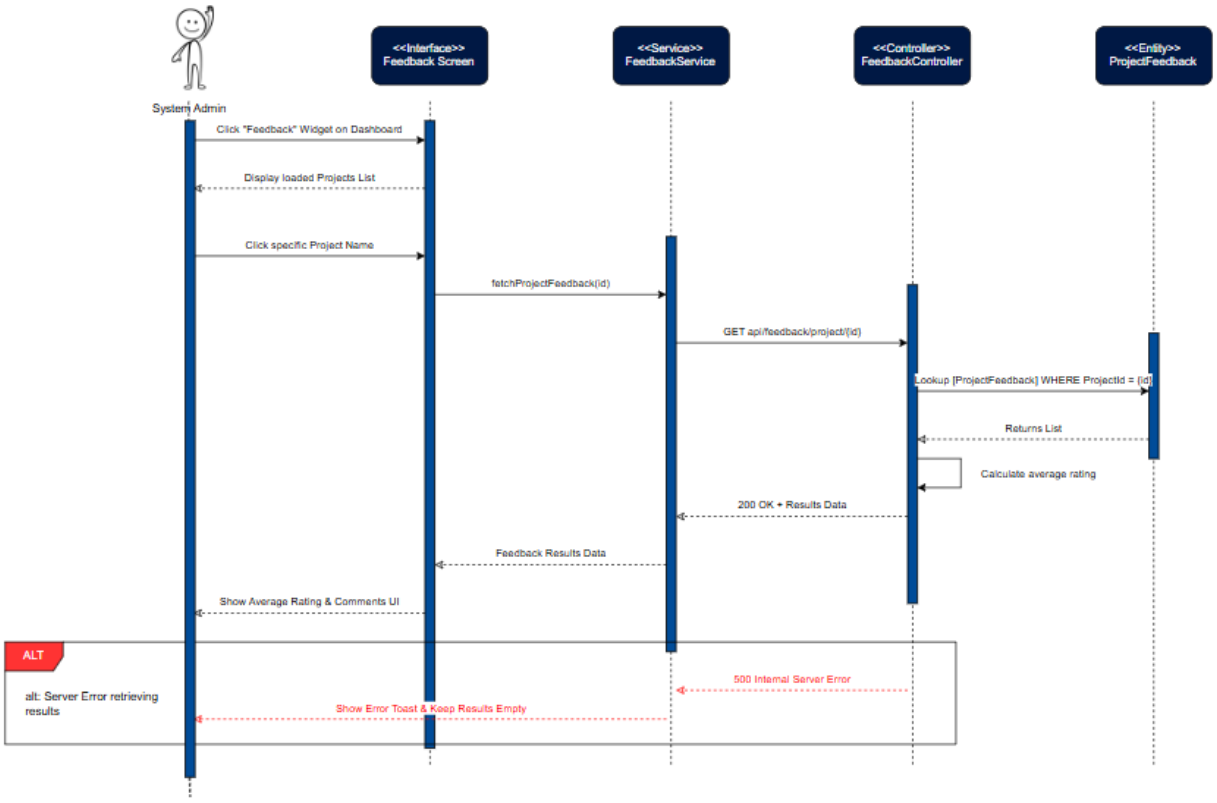


Figure 240: Sequence Diagram 7.15 View Project Feedback

2.7.6.3 Create Impact Feedback Form: Use Case 7.16

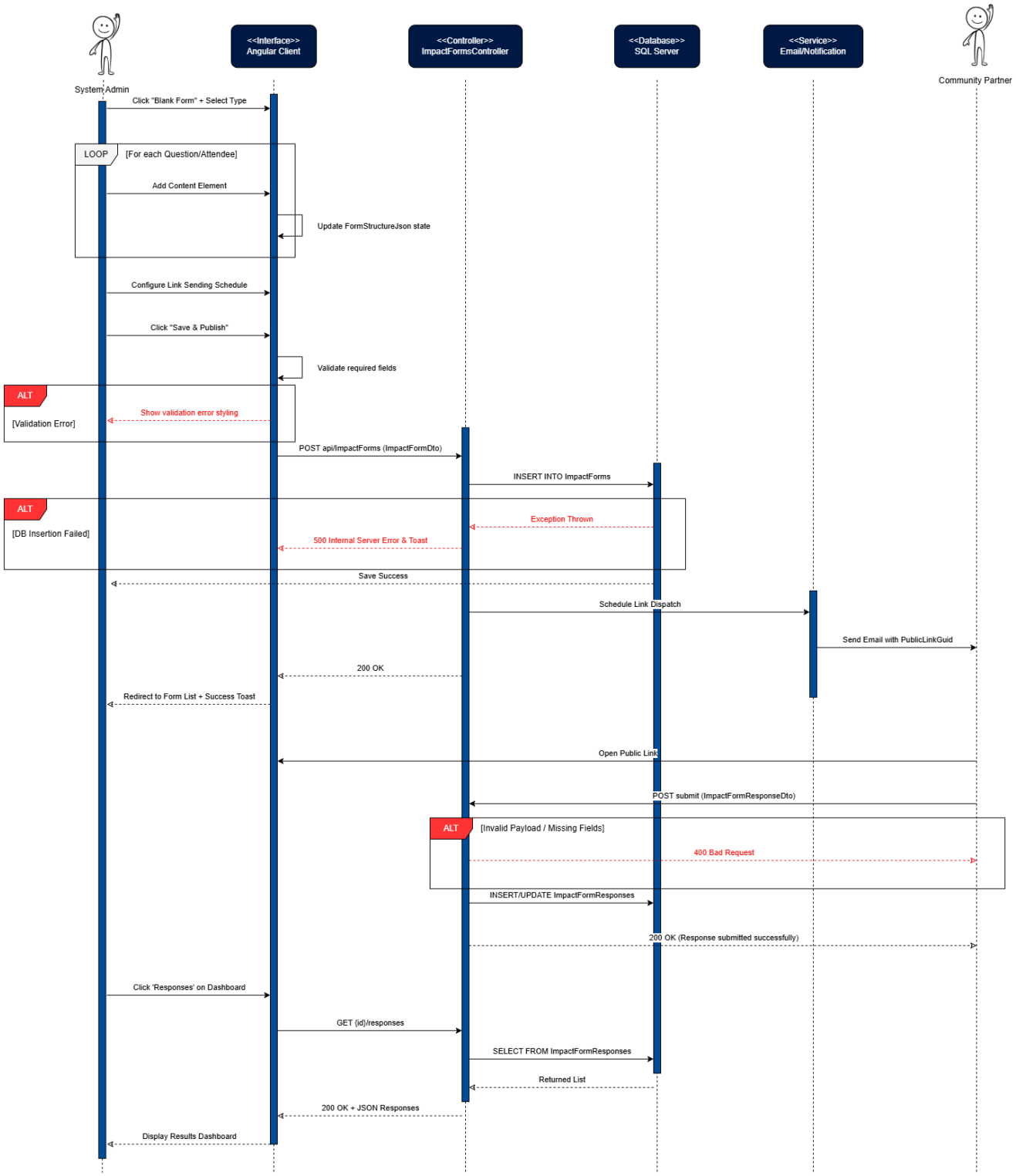


Figure 241: Sequence Diagram 7.16 Create Impact Feedback Form

2.7.6.4 View Impact Feedback Form: Use Case 7.17

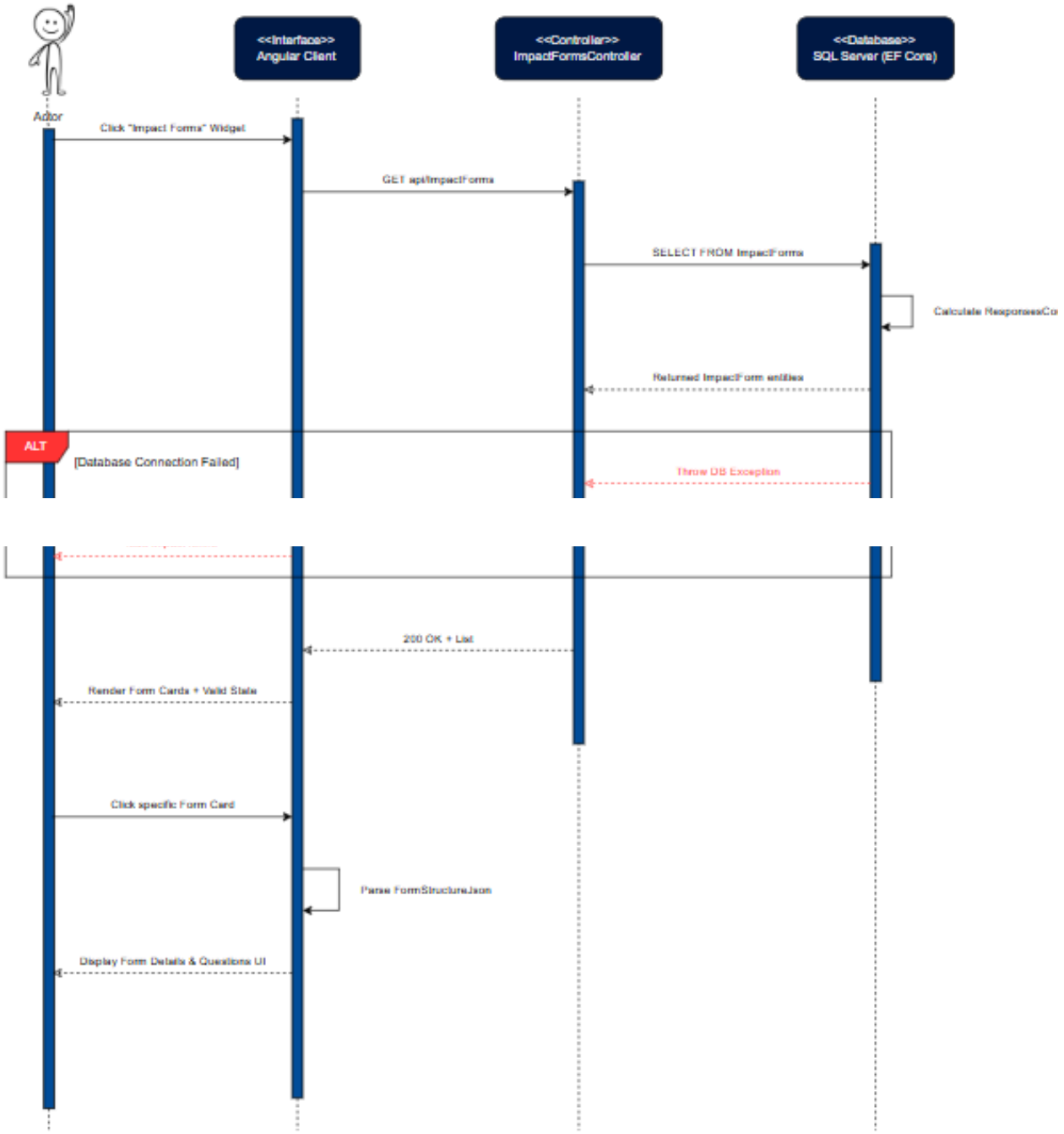


Figure 242: Sequence Diagram 7.17 View Impact

2.7.6.5 Update Impact Feedback Form: Use Case 7.18

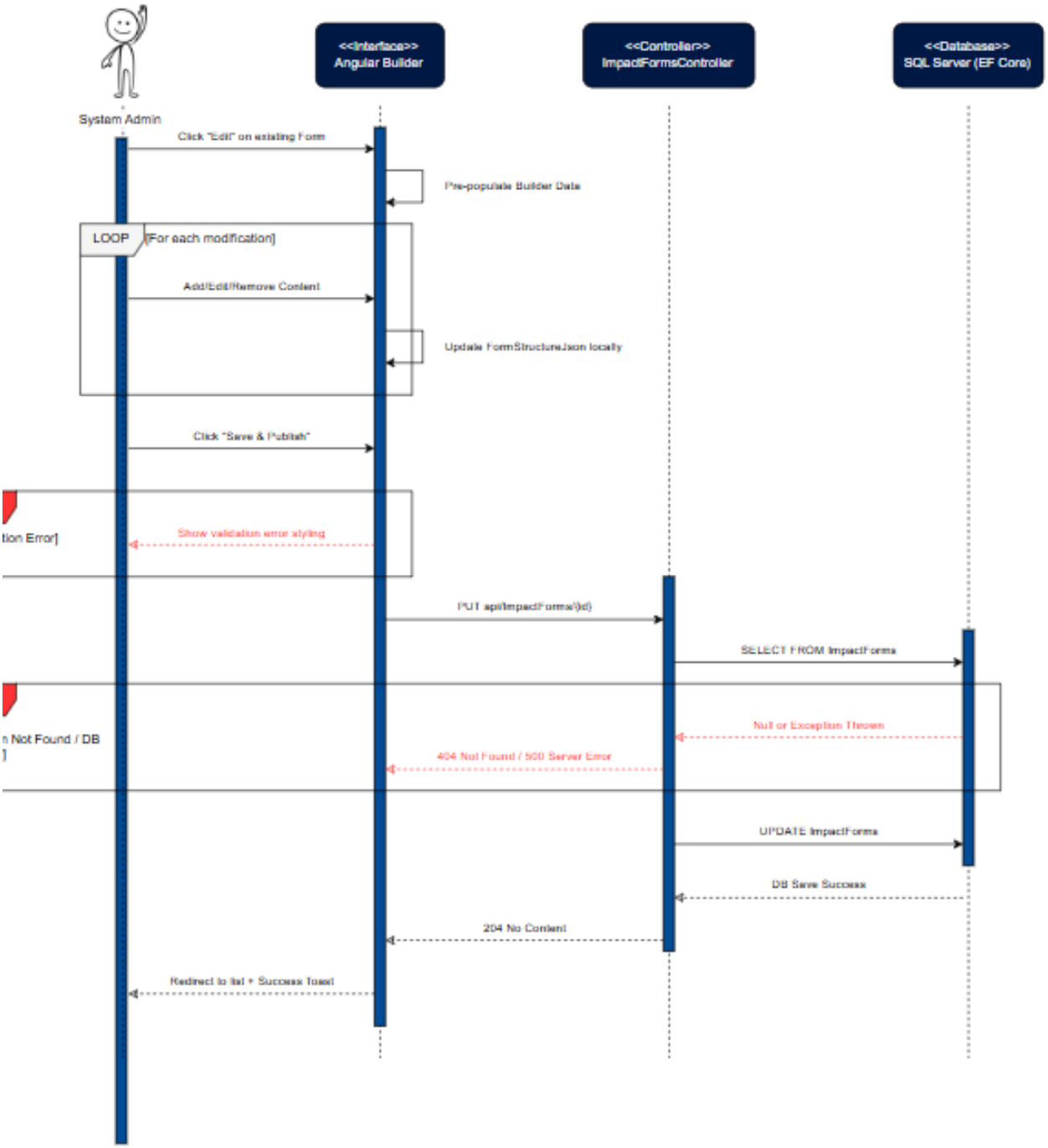


Figure 243: Sequence Diagram 7.18 Update Impact Feedback Form

2.7.6.6 Deactivate Impact Feedback Form: Use Case 7.19

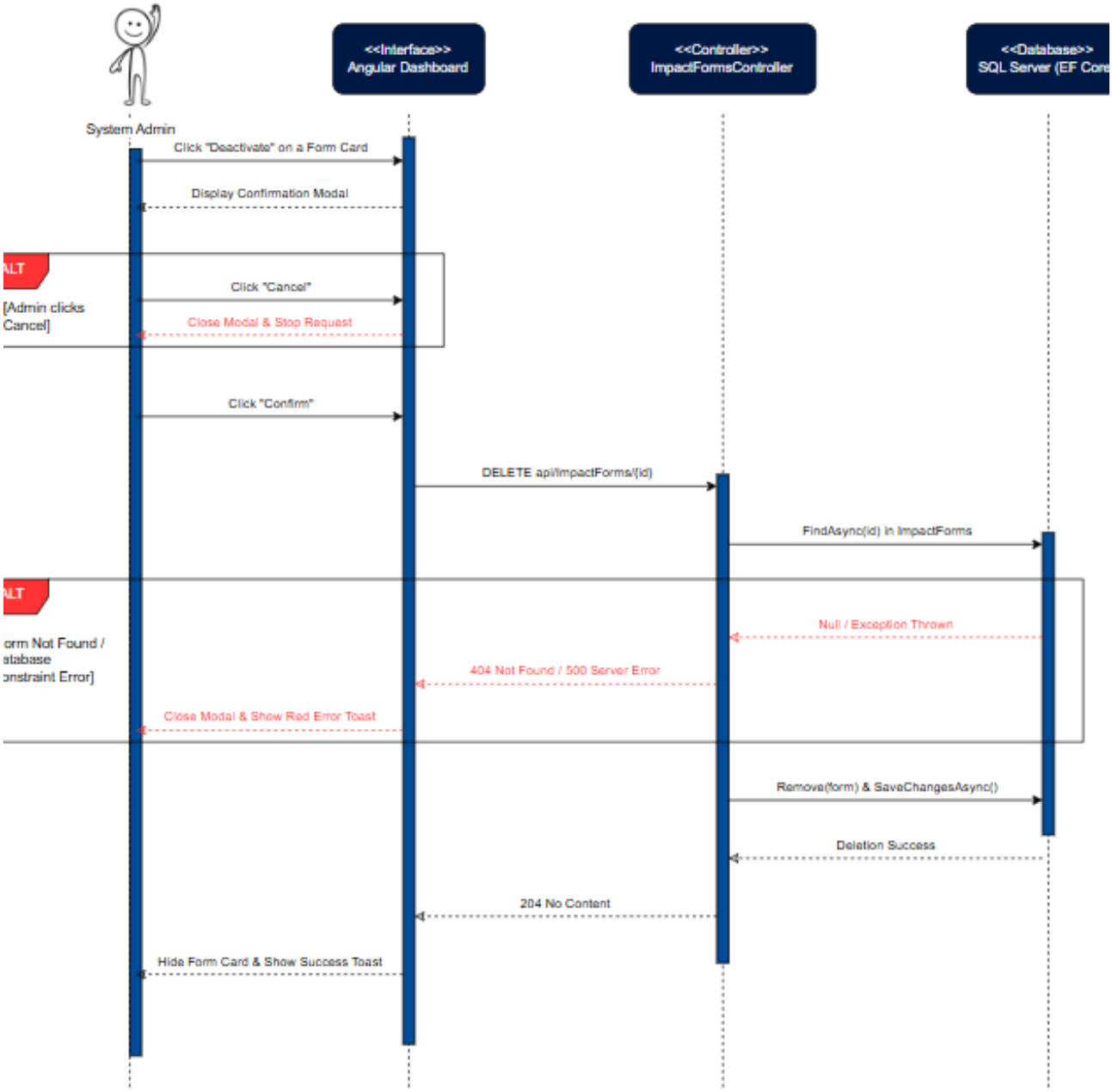


Figure 244: Sequence Diagram 7.19 Deactivate Impact Feedback Form

2.7.6.7 Capture Impact Feedback: Use Case 7.20

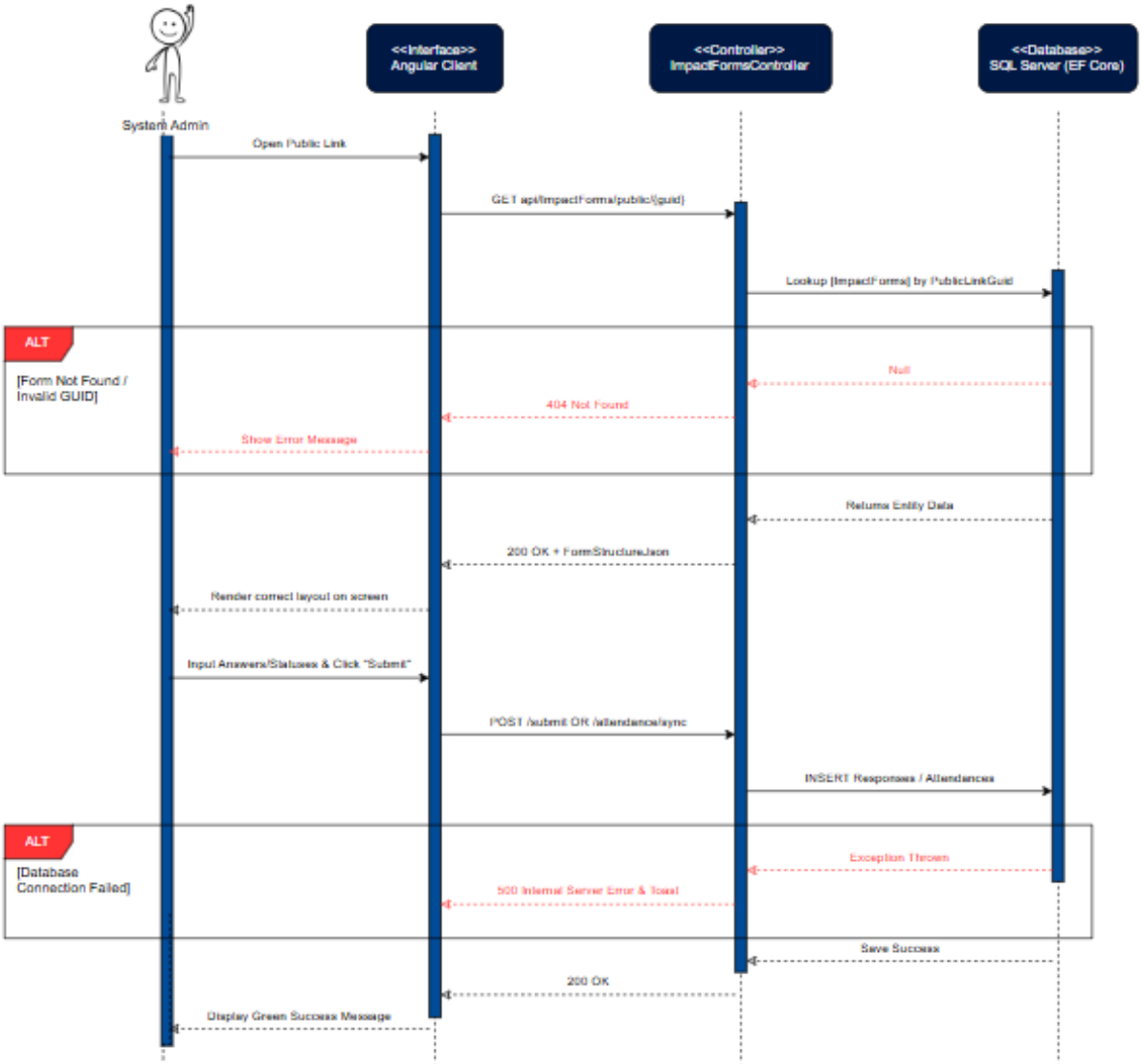


Figure 245: Sequence Diagram 7.20 Capture Impact Feedback Form

2.7.6.8 View Impact Form: Use Case 7.21

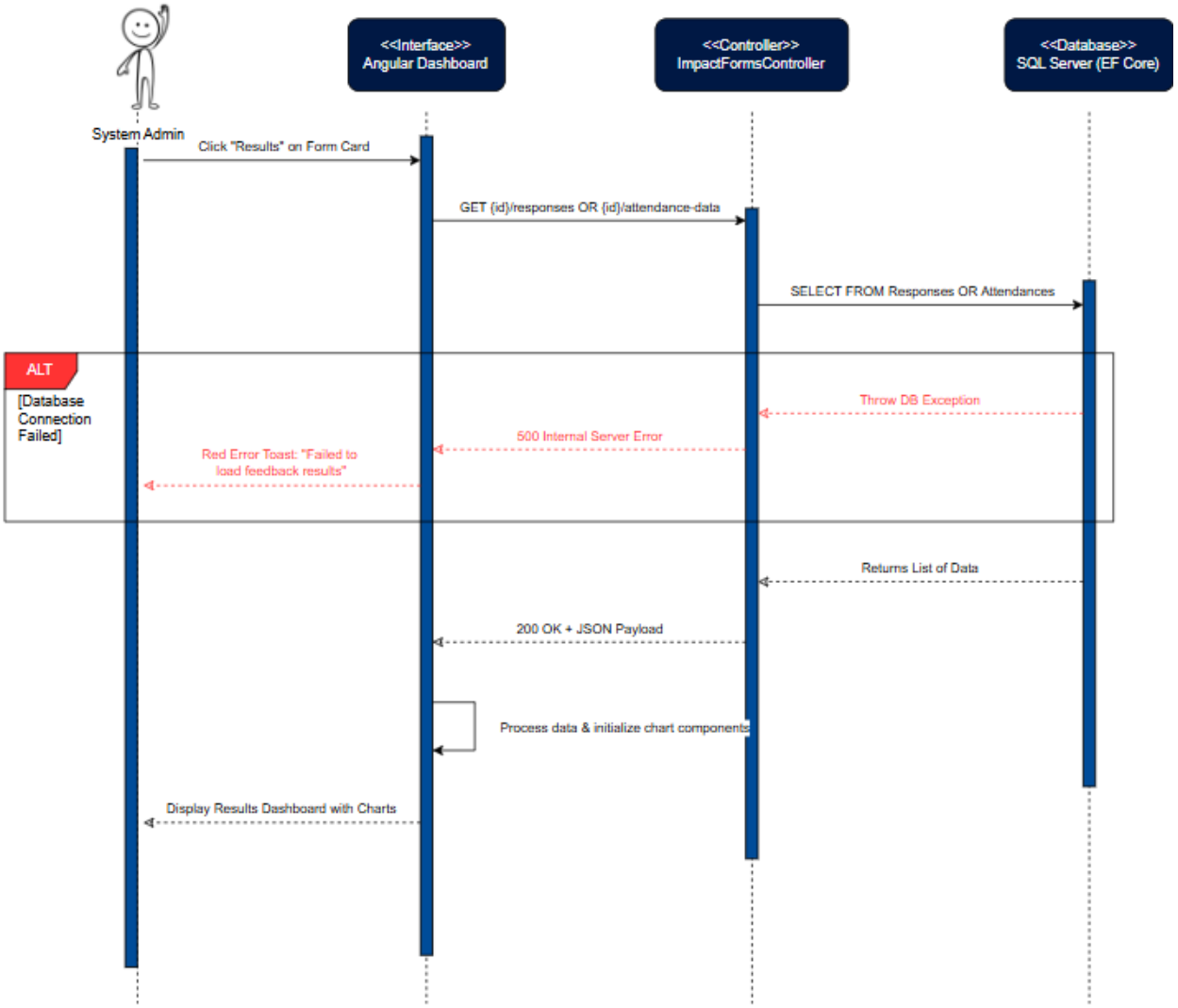


Figure 246: Sequence Diagram 7.21 View Impact Form

2.7.6.9 Add Photo: Use Case 7.22

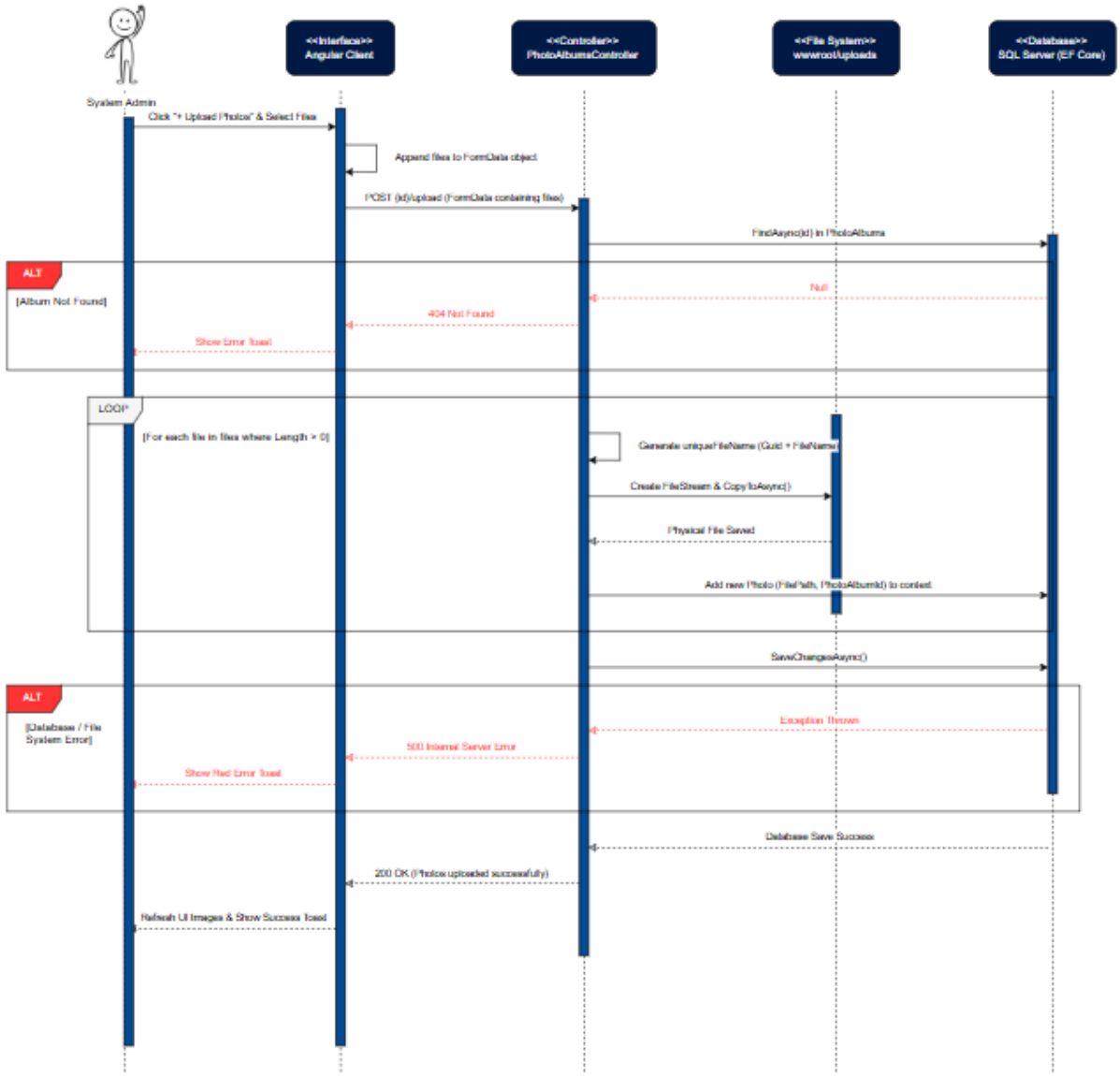


Figure 247: Sequence Diagram 7.22 Add Photo

2.7.6.10 View Photo: Use Case 7.23

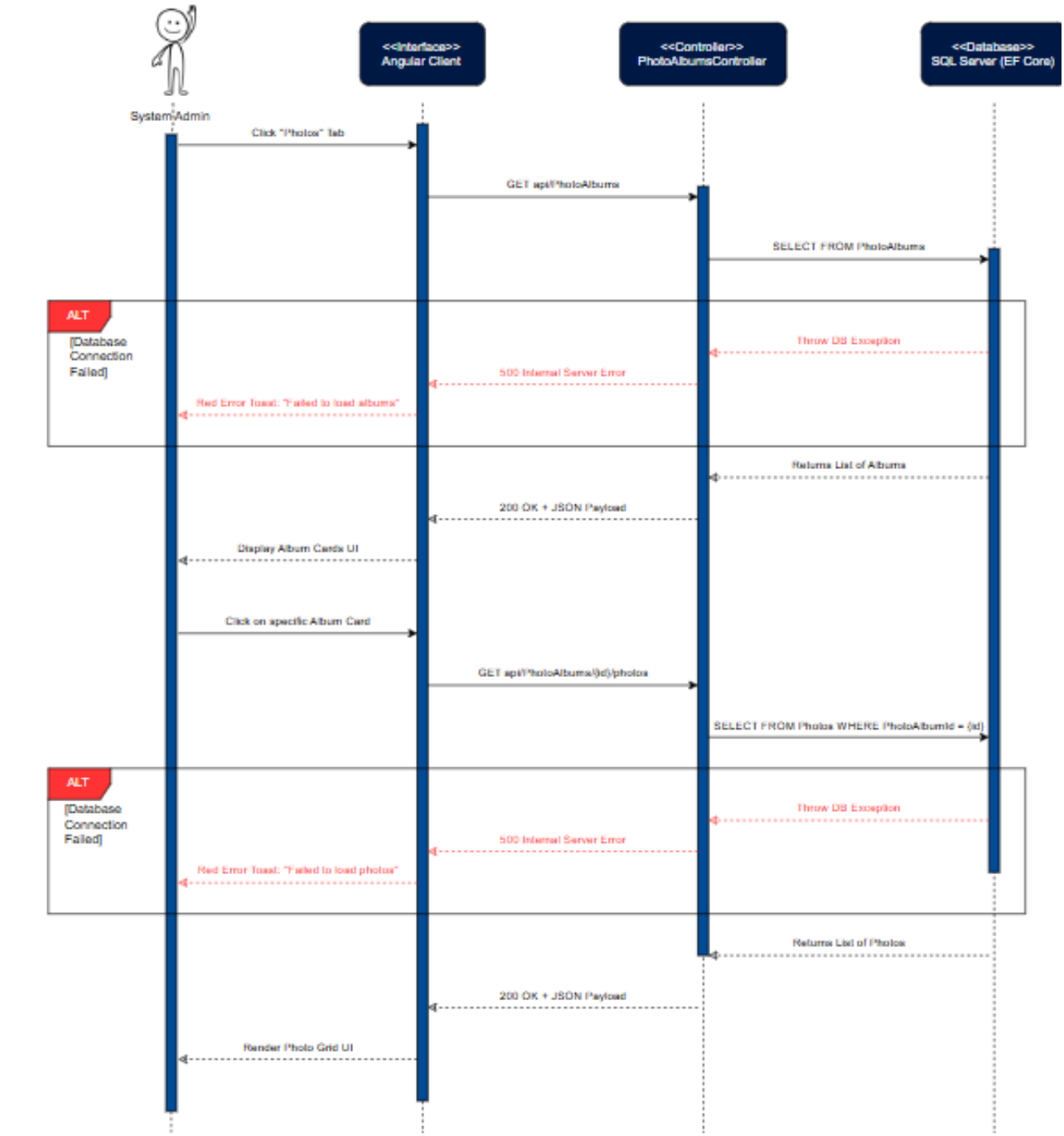


Figure 248: Sequence Diagram 7.23 View Photo

2.7.6.11 Delete Photo: Use Case 7.24

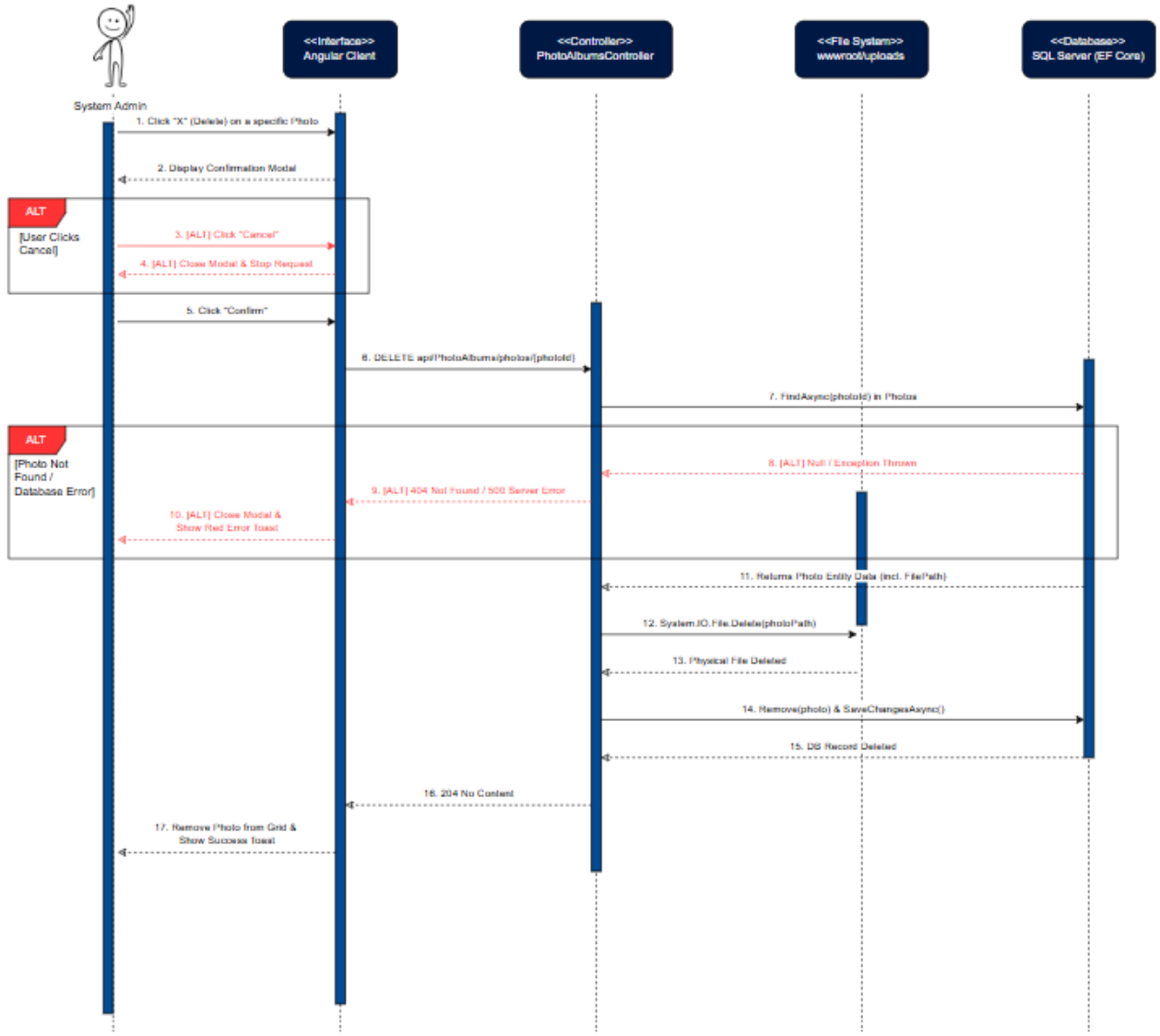


Figure 249: Sequence Diagram 7.24 Delete Photo

2.7.7 *K9 Subsystem Sequence Diagrams*

2.7.7.1 Add Dog: Use Case 8.*1*

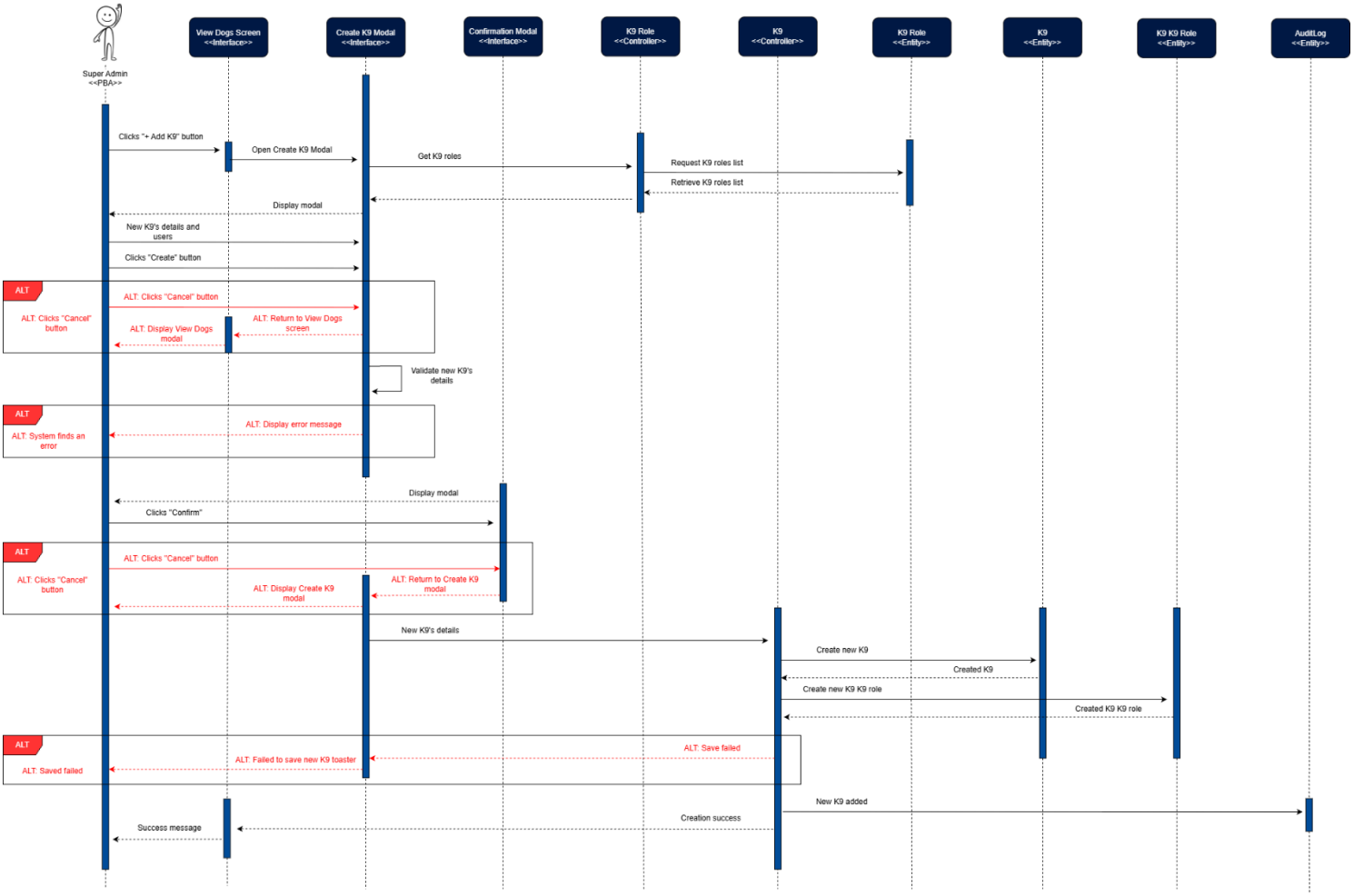


Figure 250: Sequence Diagram 8.1 Add Dog

2.7.7.2 View Dog: Use Case 8.2

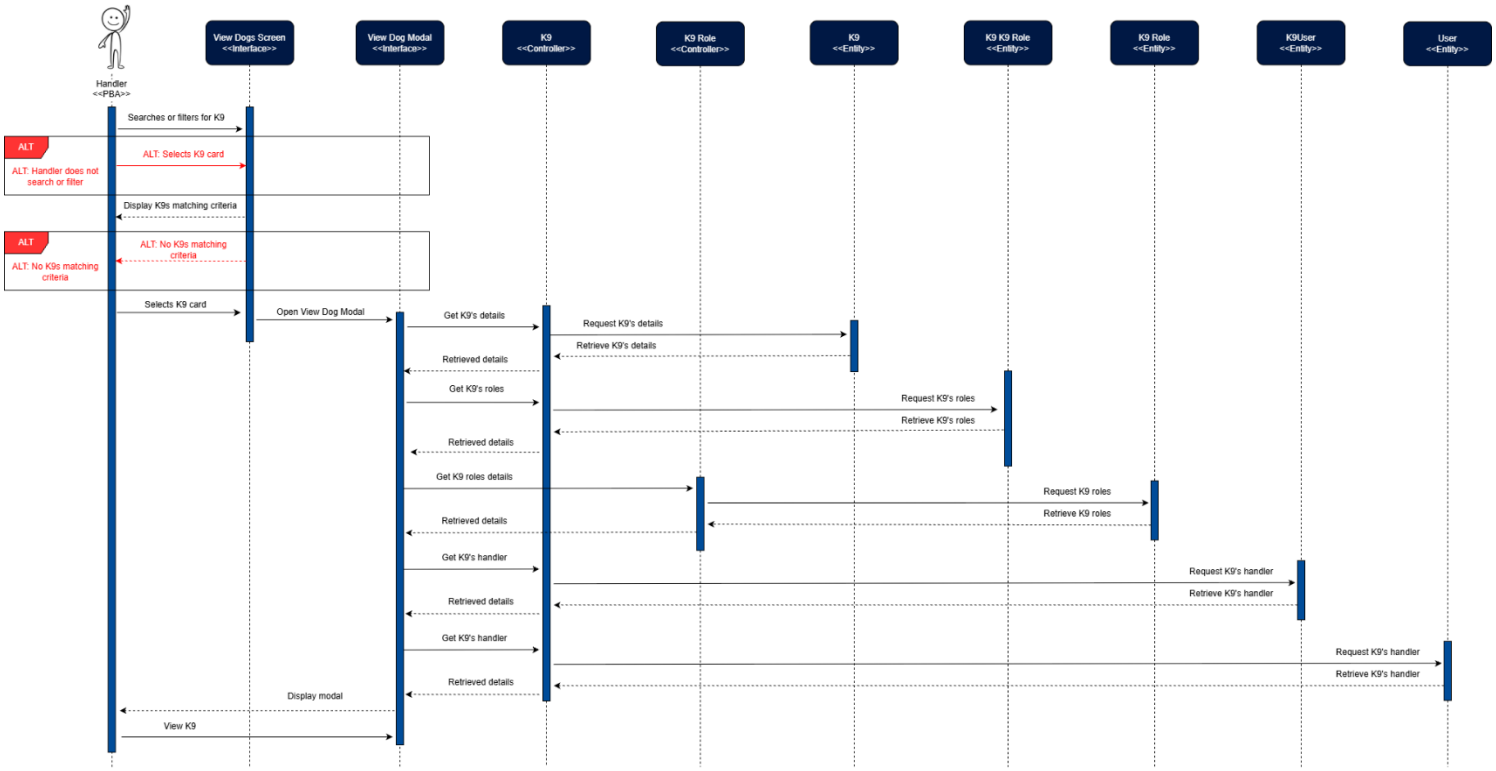


Figure 251: Sequence Diagram 8.2 View Dog

2.7.7.3 View Dogs: Use Case 8.3

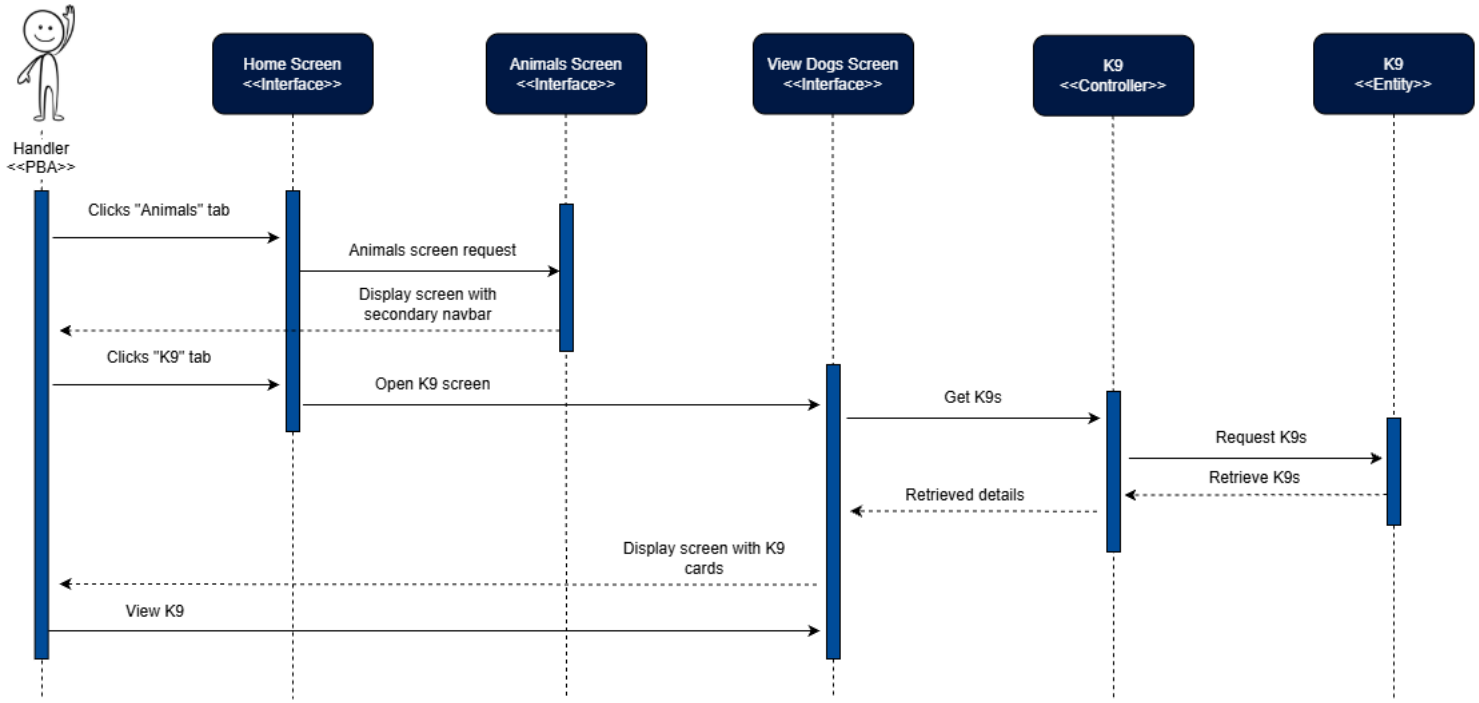


Figure 252: Sequence Diagram 8.3 View Dogs

2.7.7.4 Update Dog: Use Case 8.4

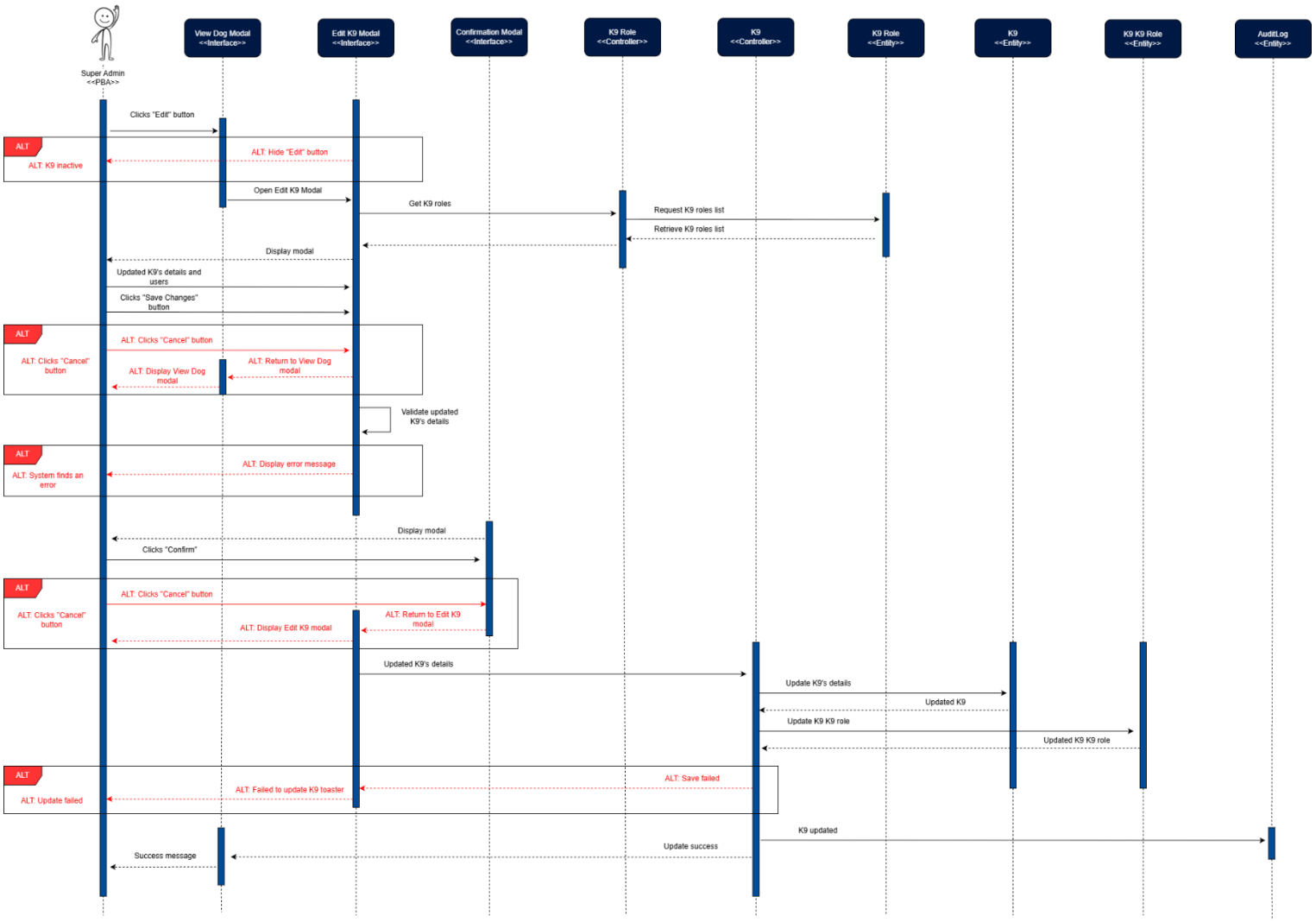


Figure 253: Sequence Diagram 8.4 Update Dog

2.7.7.5 Deactivate Dog: Use Case 8.5

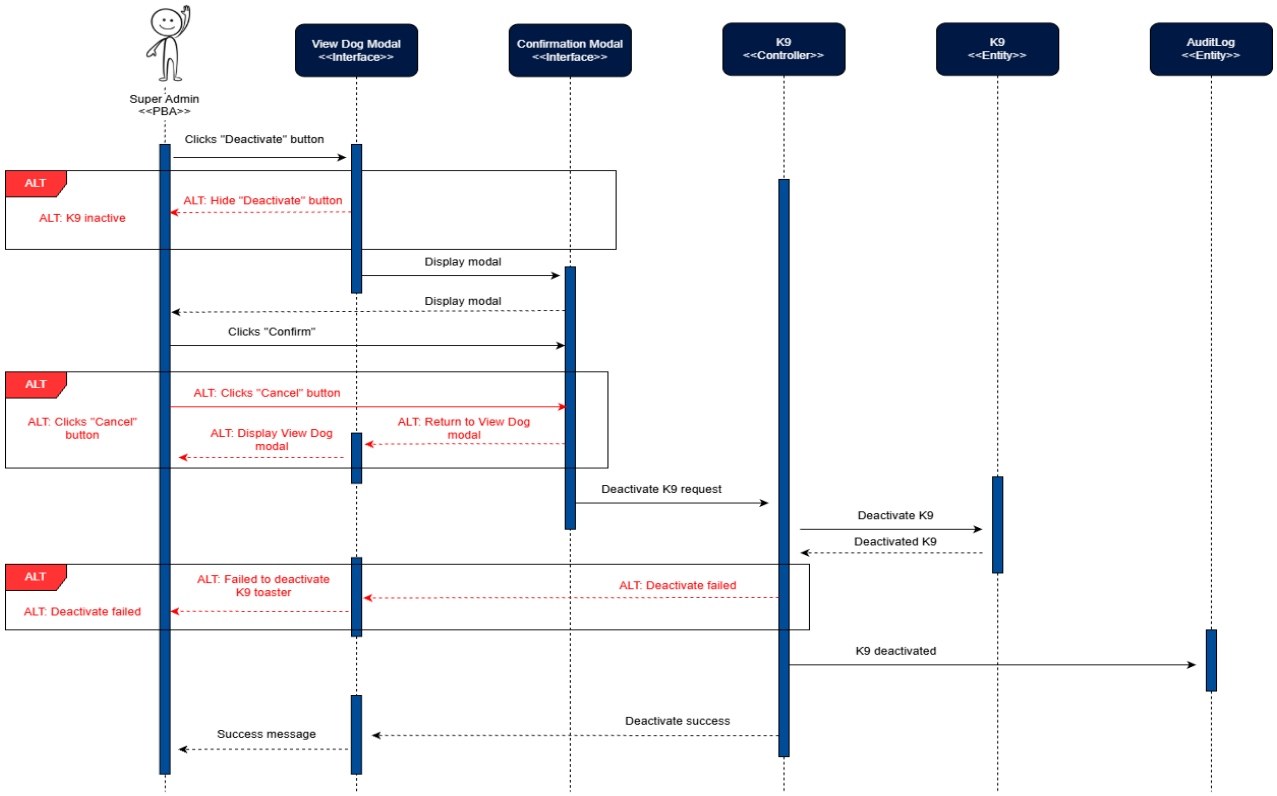


Figure 254: Sequence Diagram 8.5 Deactivate Dog

2.7.7.6 Assign User to Dog: Use Case 8.6

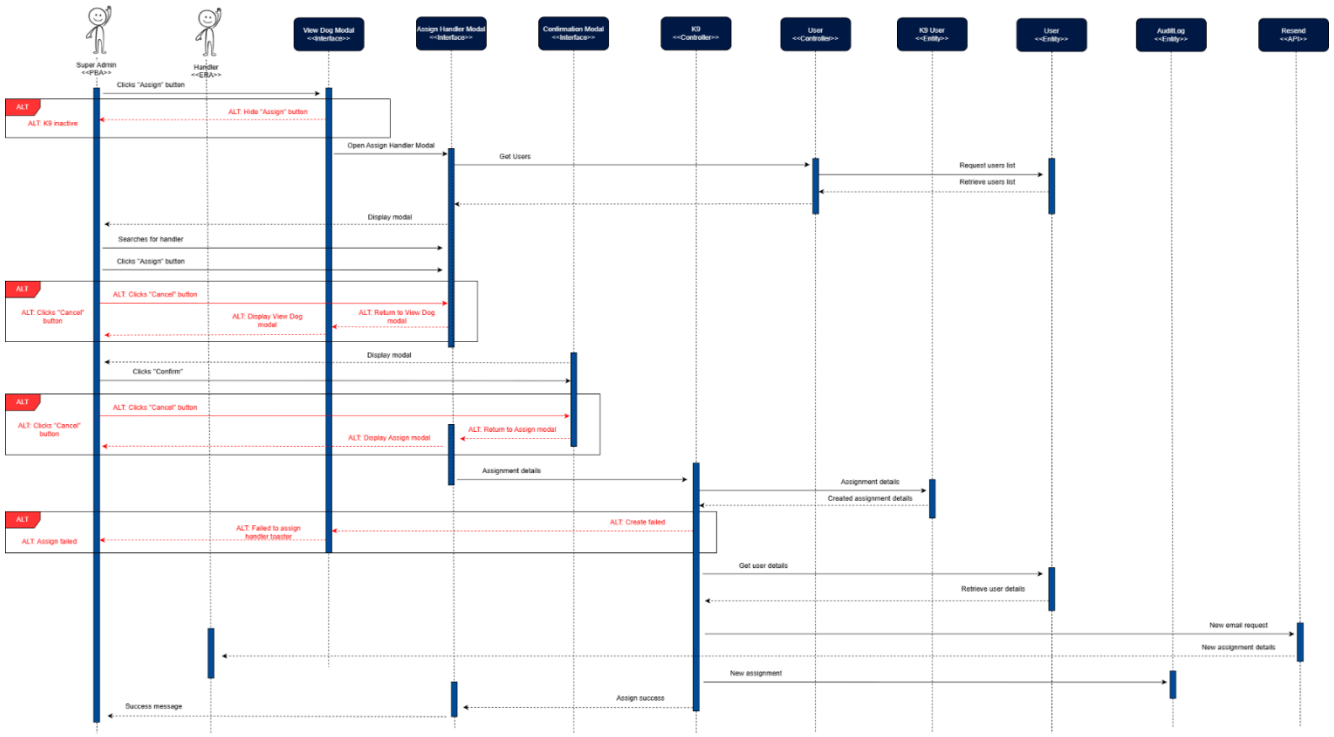


Figure 255: Sequence Diagram 8.6 Assign User to Dog

2.7.7.7 View User Dog-Assignment: Use Case 8.7

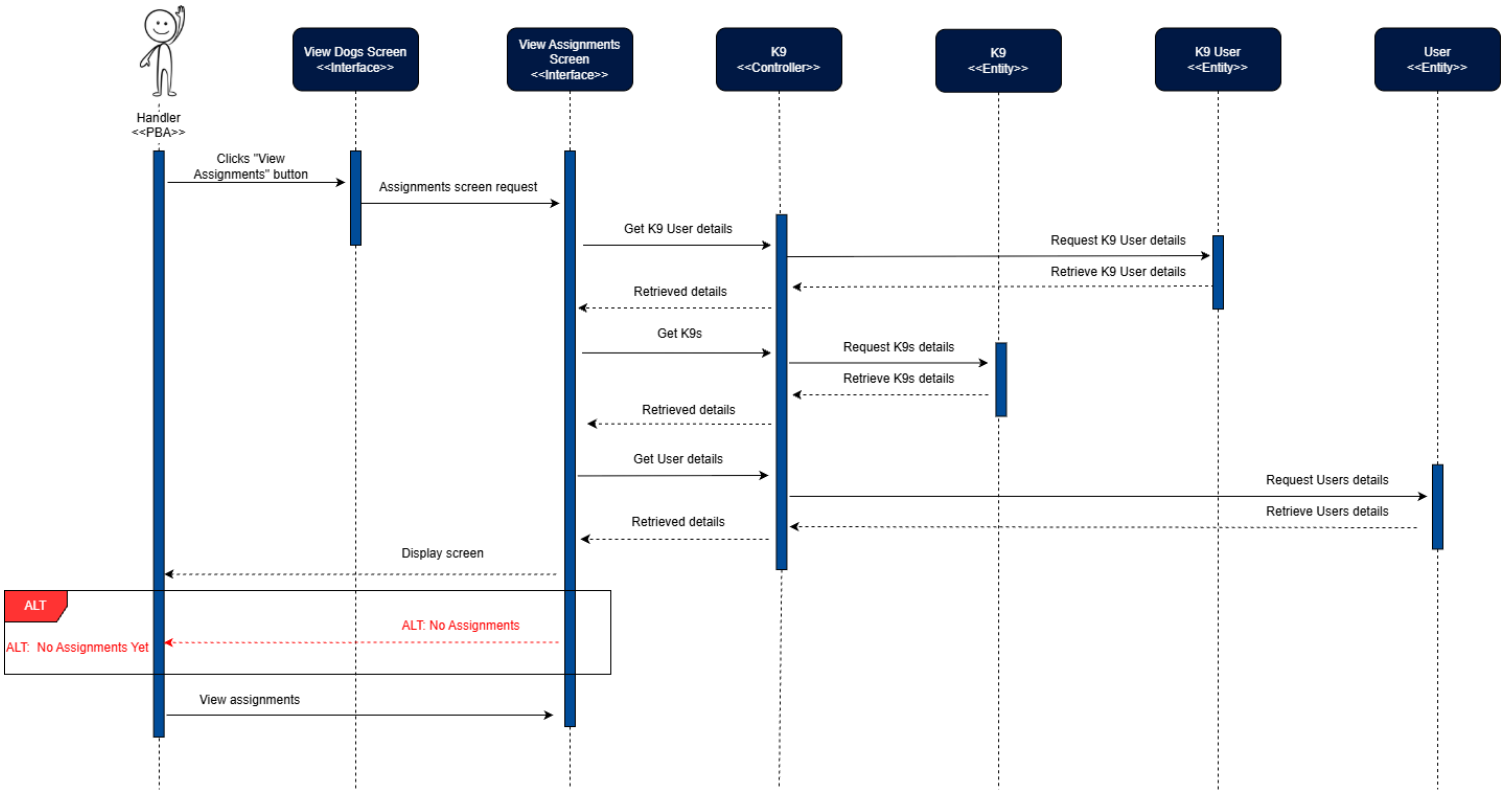


Figure 256: Sequence Diagram 8.7 View User-Dog Assignment

2.7.7.8 Update User-Dog Assignment: Use Case 8.8

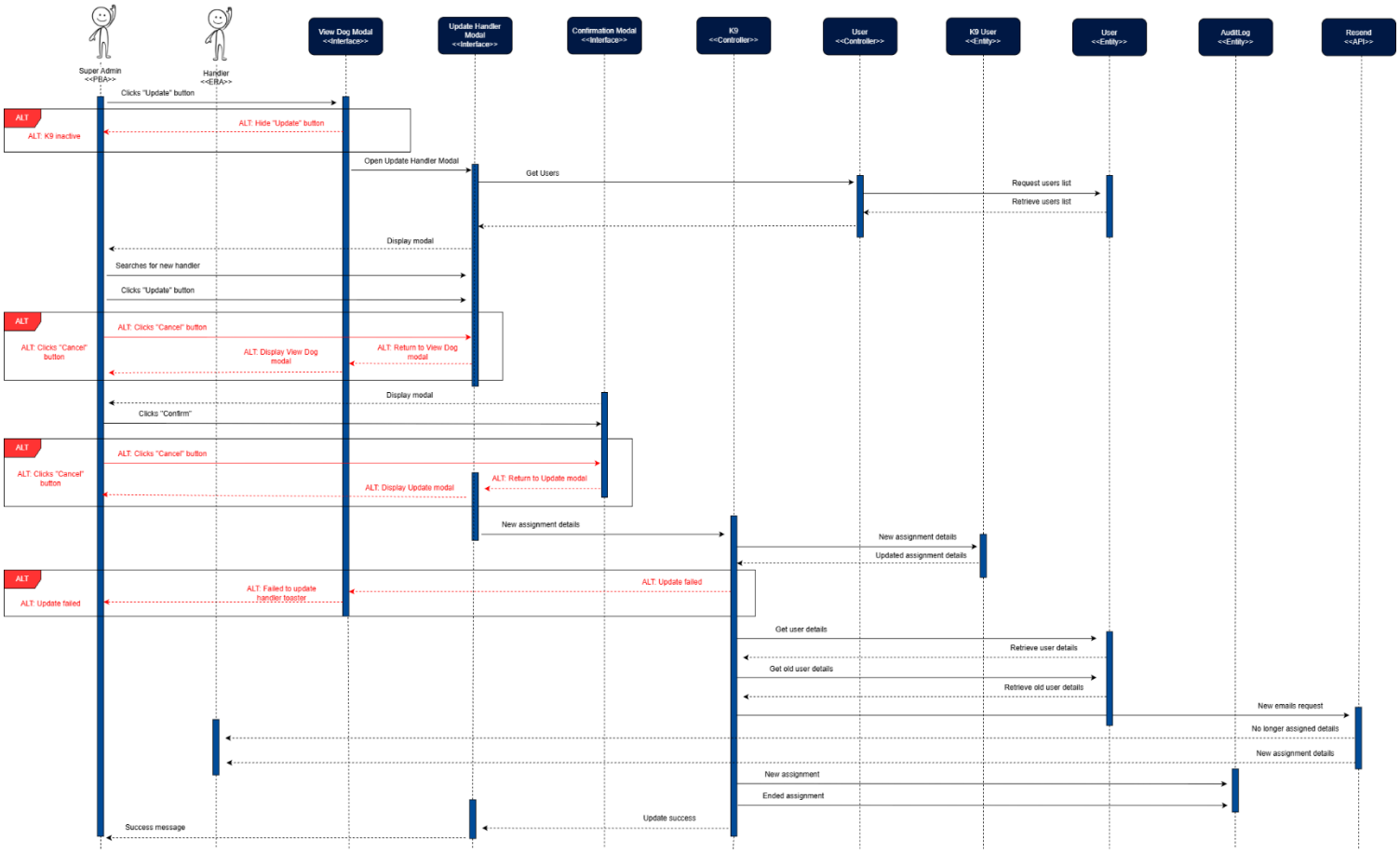


Figure 257: Sequence Diagram 8.8 Update User-Dog Assignment

2.7.7.9 Reactivate User Dog-Assignment: Use Case 8.9

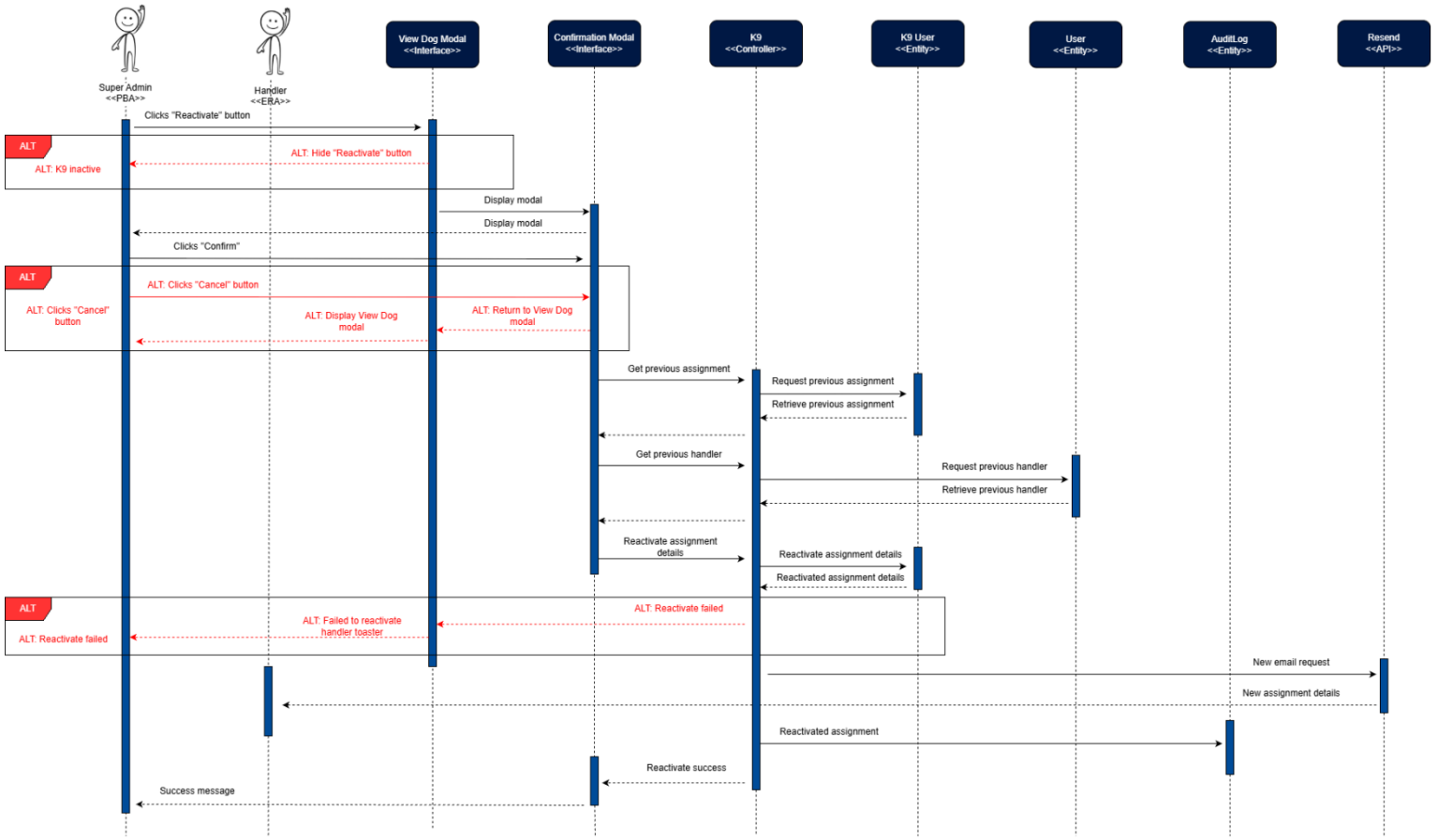


Figure 258: Sequence Diagram 8.9 Reactivate User-Dog Assignment

2.7.7.10 Deactivate User Dog-Assignment: Use Case 8.10

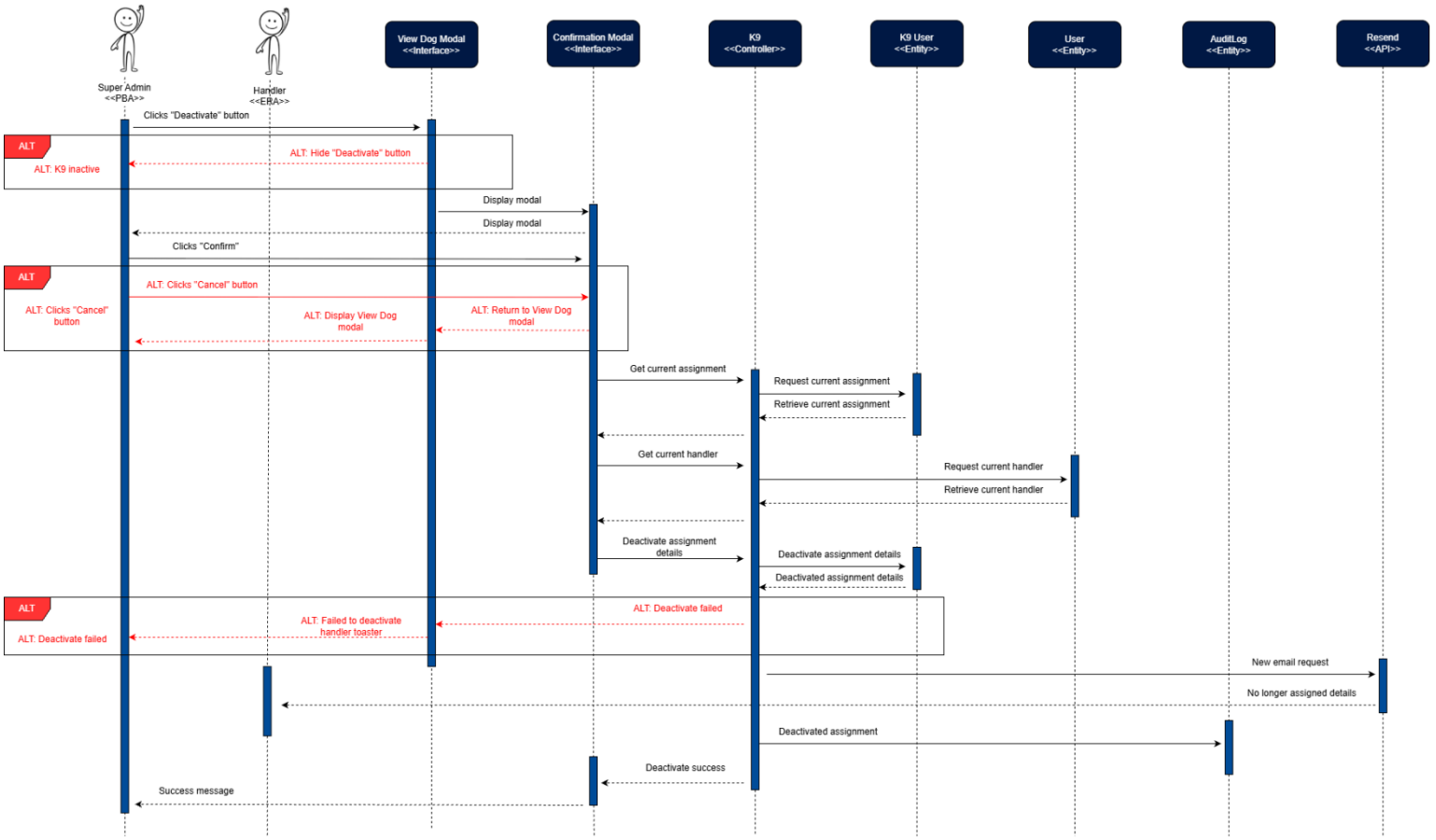


Figure 259: Sequence Diagram 8.10 Deactivate User-Dog Assignment

2.7.7.11 Add K9 Role: Use Case 8.11

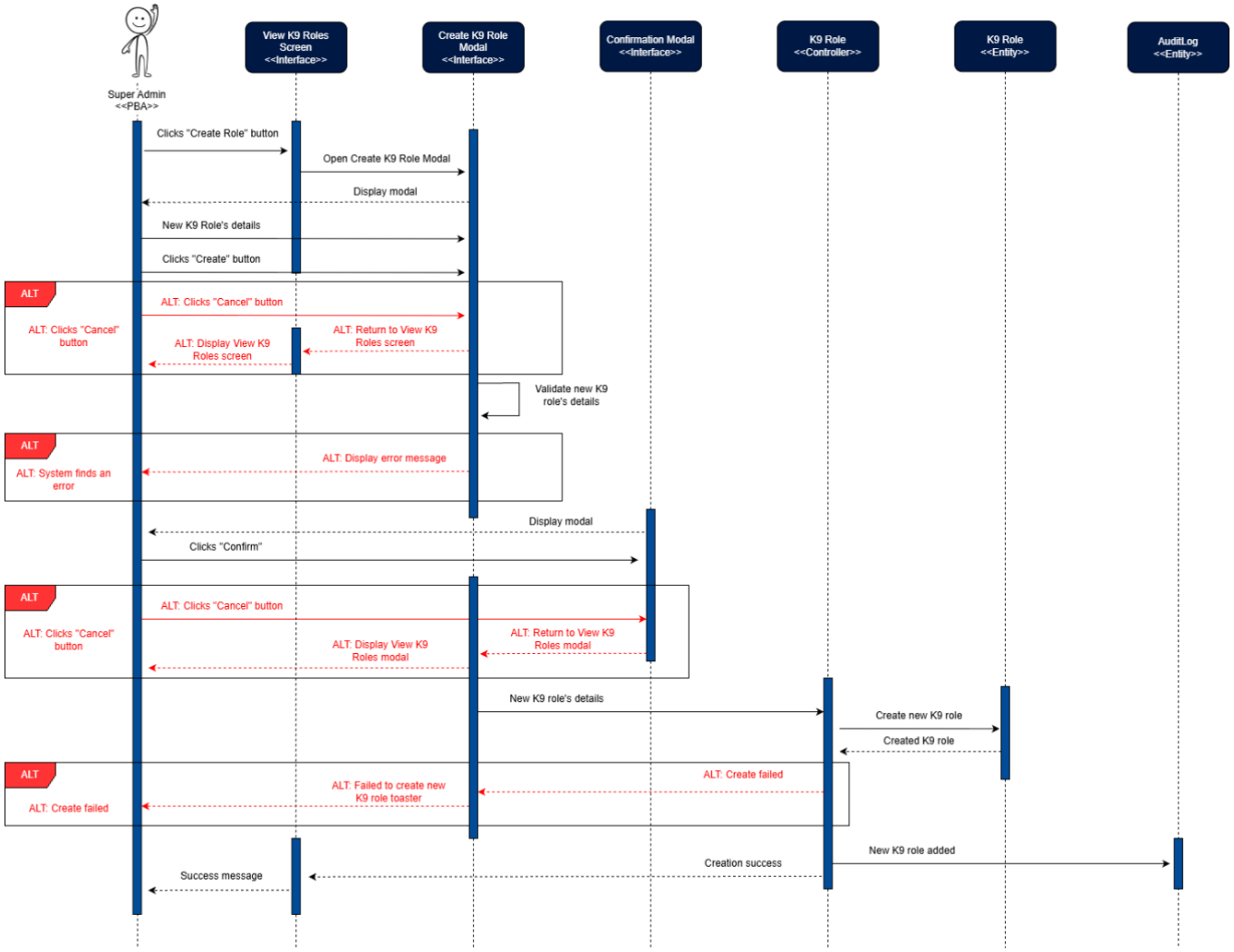


Figure 260: Sequence Diagram 8.11 Add K9 Role

2.7.7.12 Update K9 Role: Use Case 8.12

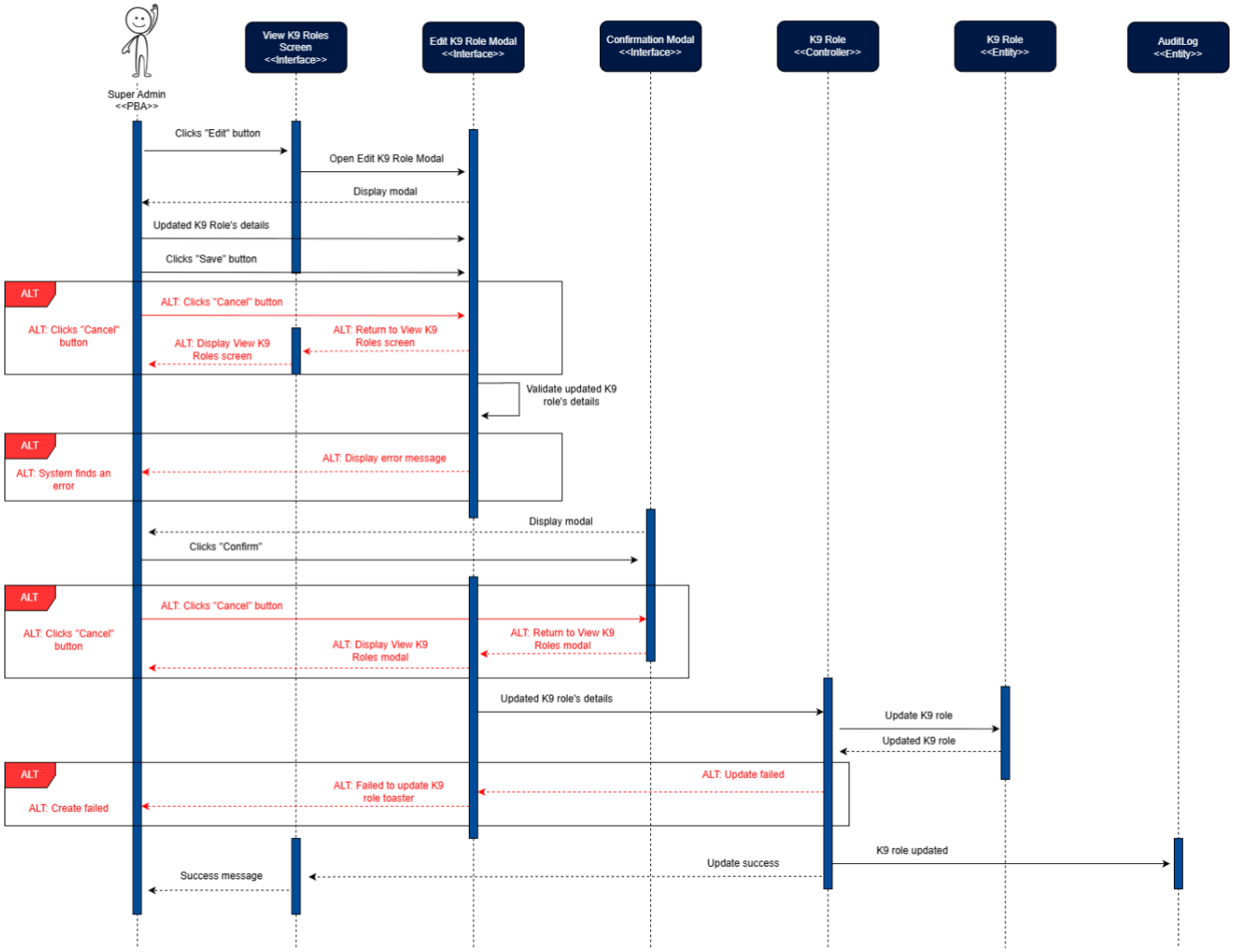


Figure 261: Sequence Diagram 8.12 Update K9 Role

2.7.7.13 View K9 Roles: Use Case 8.13

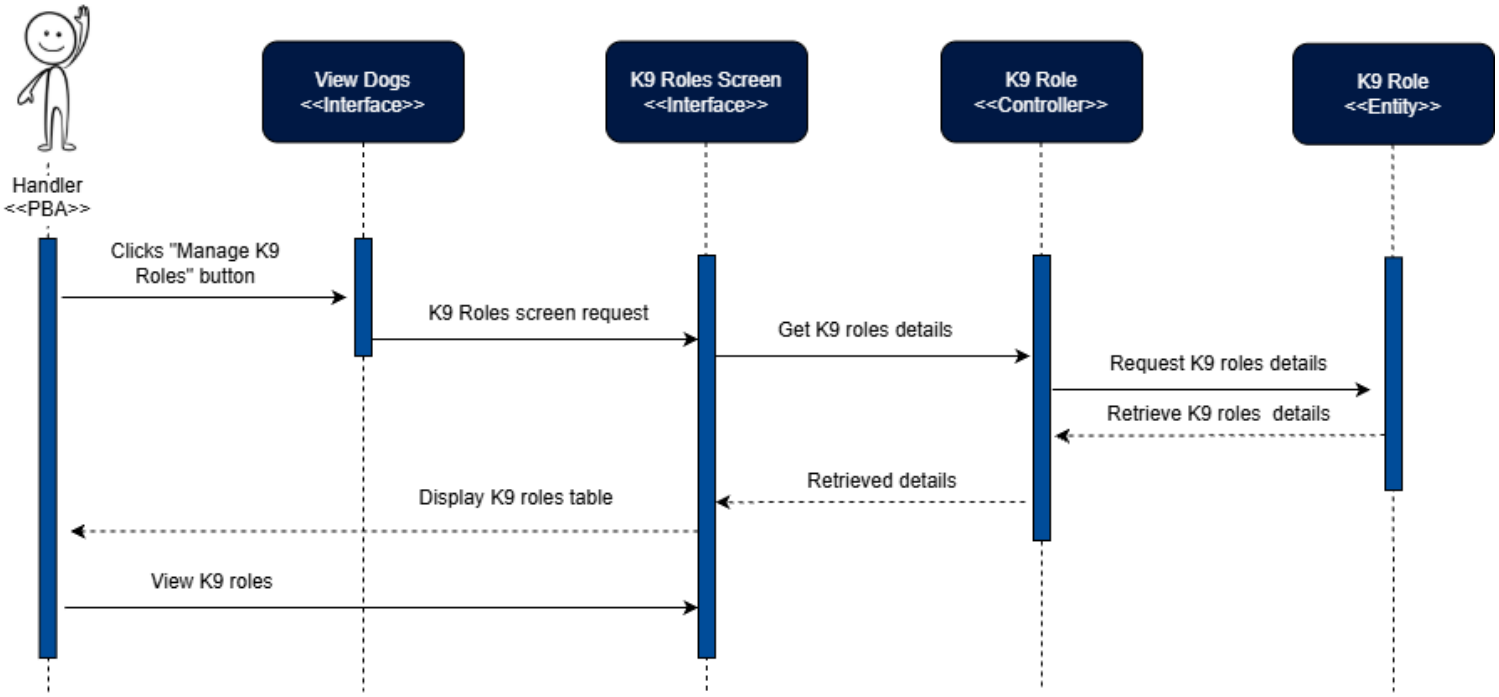


Figure 262: Sequence Diagram 8.13 View K9 Roles

2.7.7.14 Reactivate K9 Role: Use Case 8.14

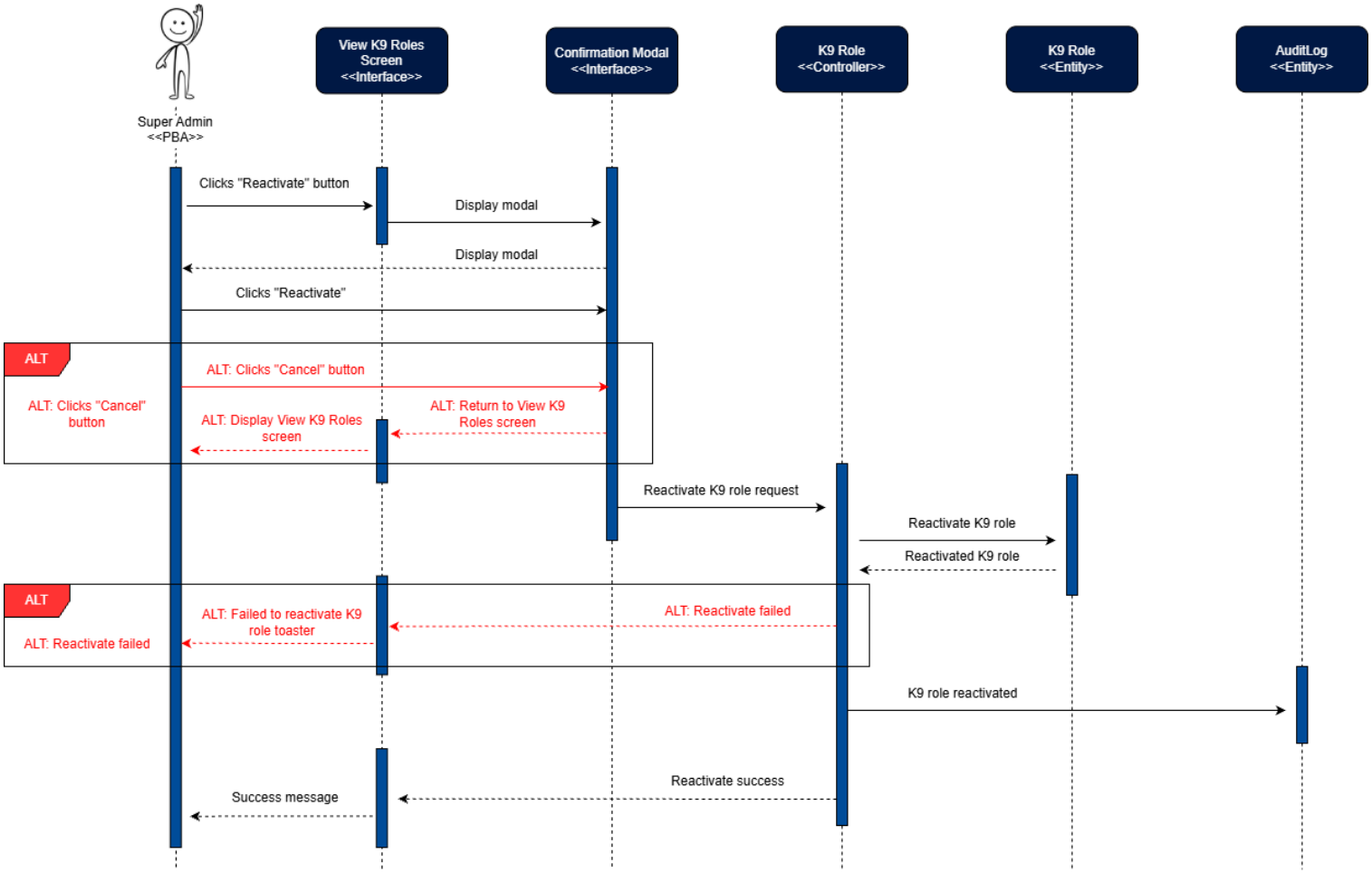


Figure 263: Sequence Diagram 8.14 Reactivate K9 Role

2.7.7.15 Deactivate K9 Role: Use Case 8.15

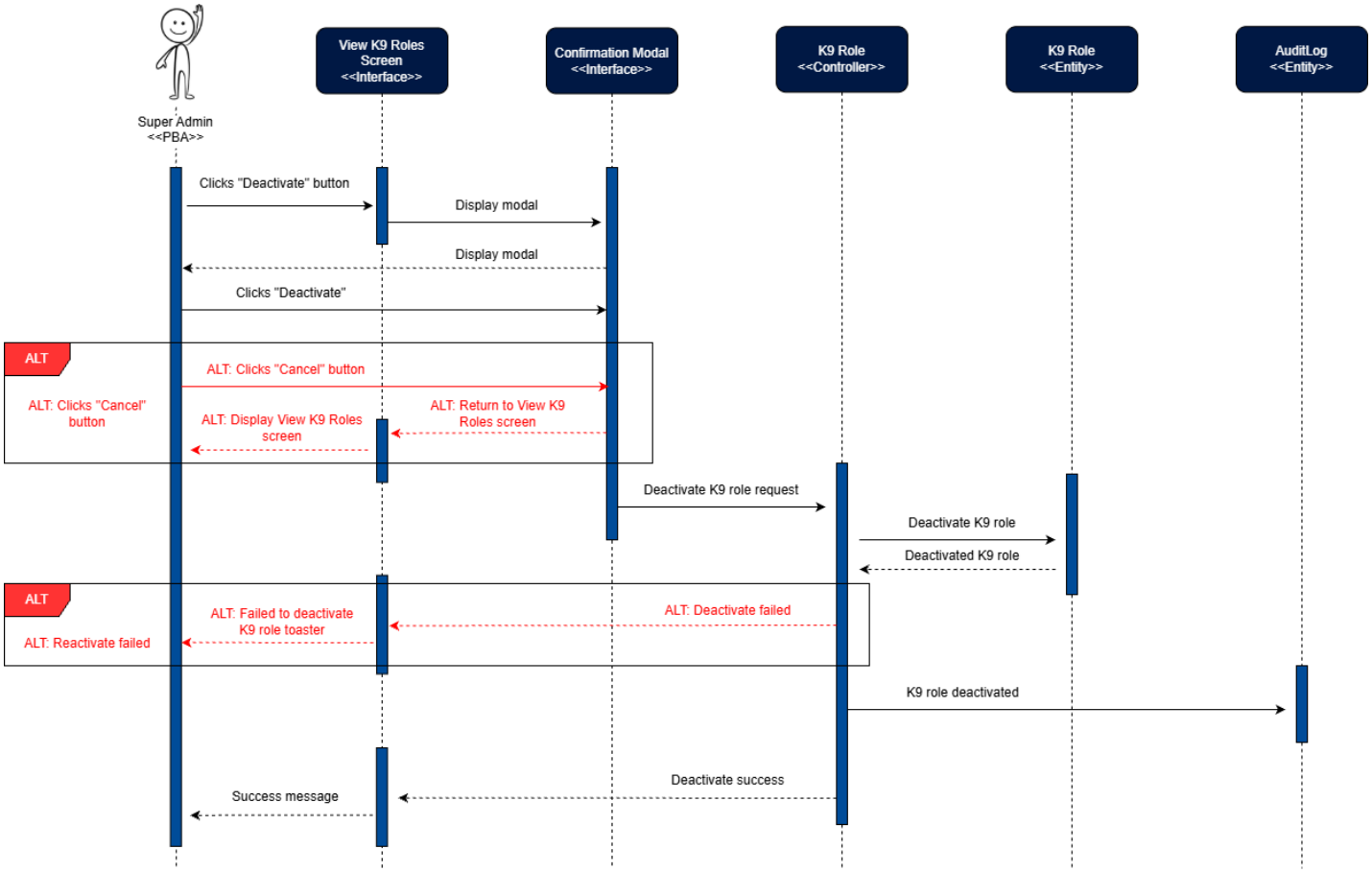


Figure 264: Sequence Diagram 8.15 Deactivate K9 Role

2.7.7.16 View K9 Ancestry: Use Case 8.16

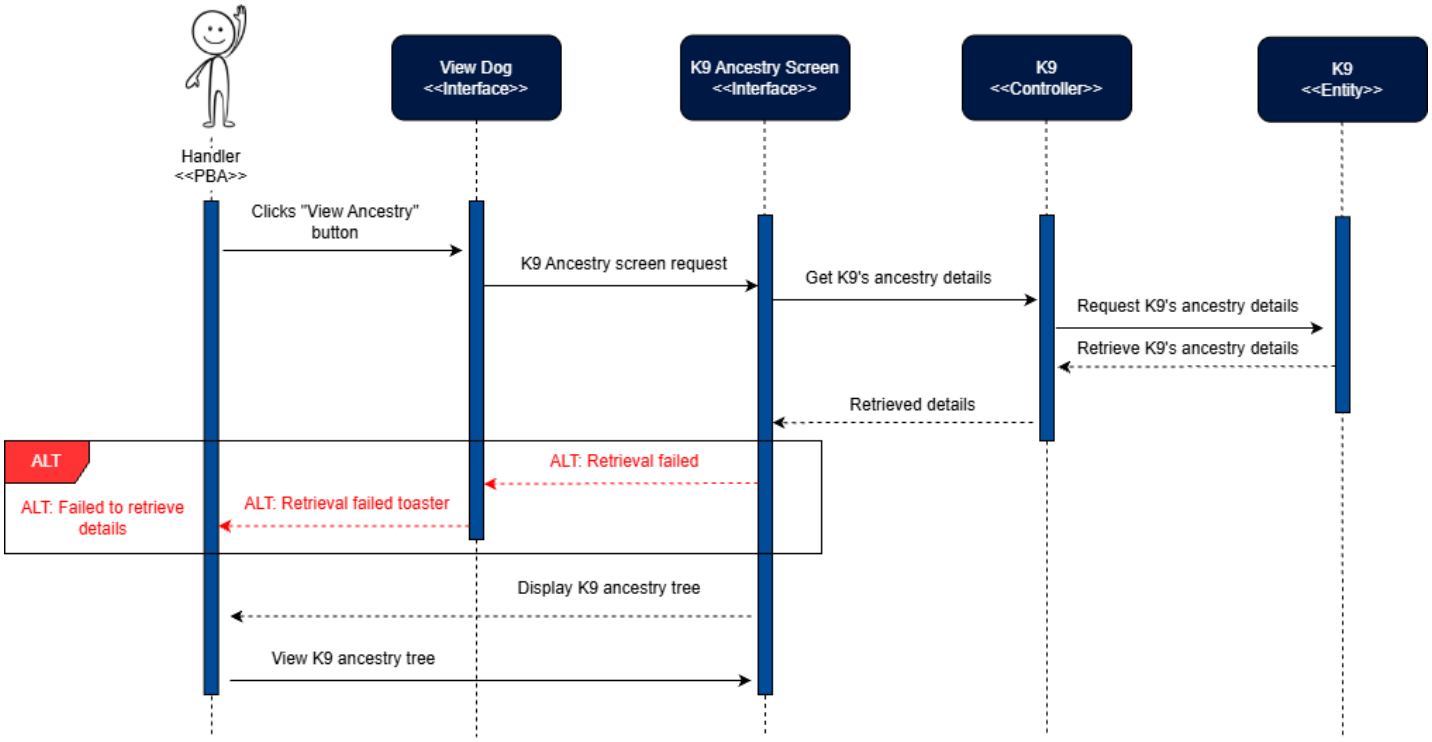


Figure 265: Sequence Diagram 8.16 View K9 Ancestry

2.7.8 *Rhino Subsystem Sequence Diagrams*

2.7.8.1 Add Rhino: Use Case 9.*1*

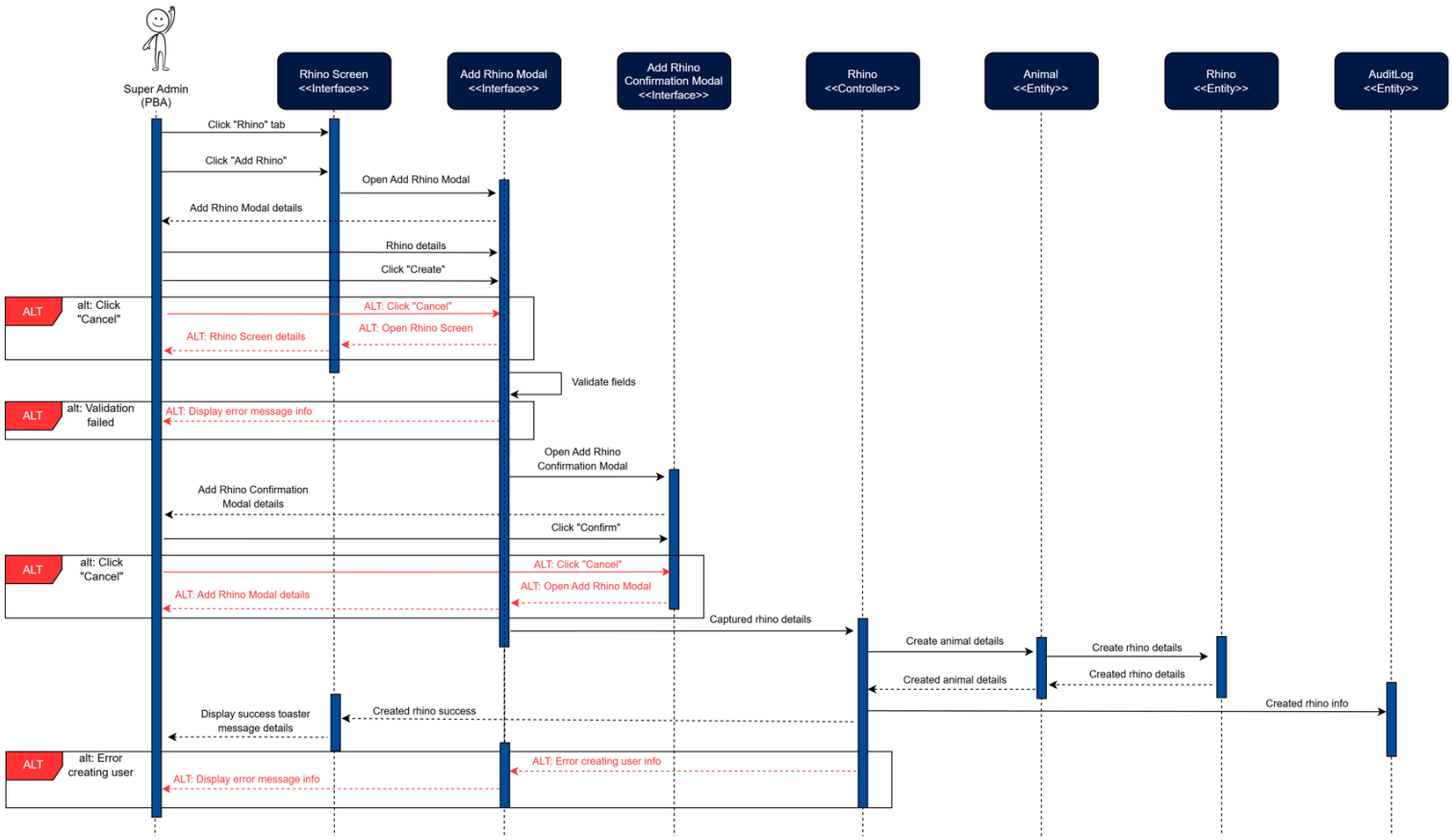


Figure 266: Sequence Diagram 9.1 Add Rhino

2.7.8.2 View Rhino: Use Case 9.2

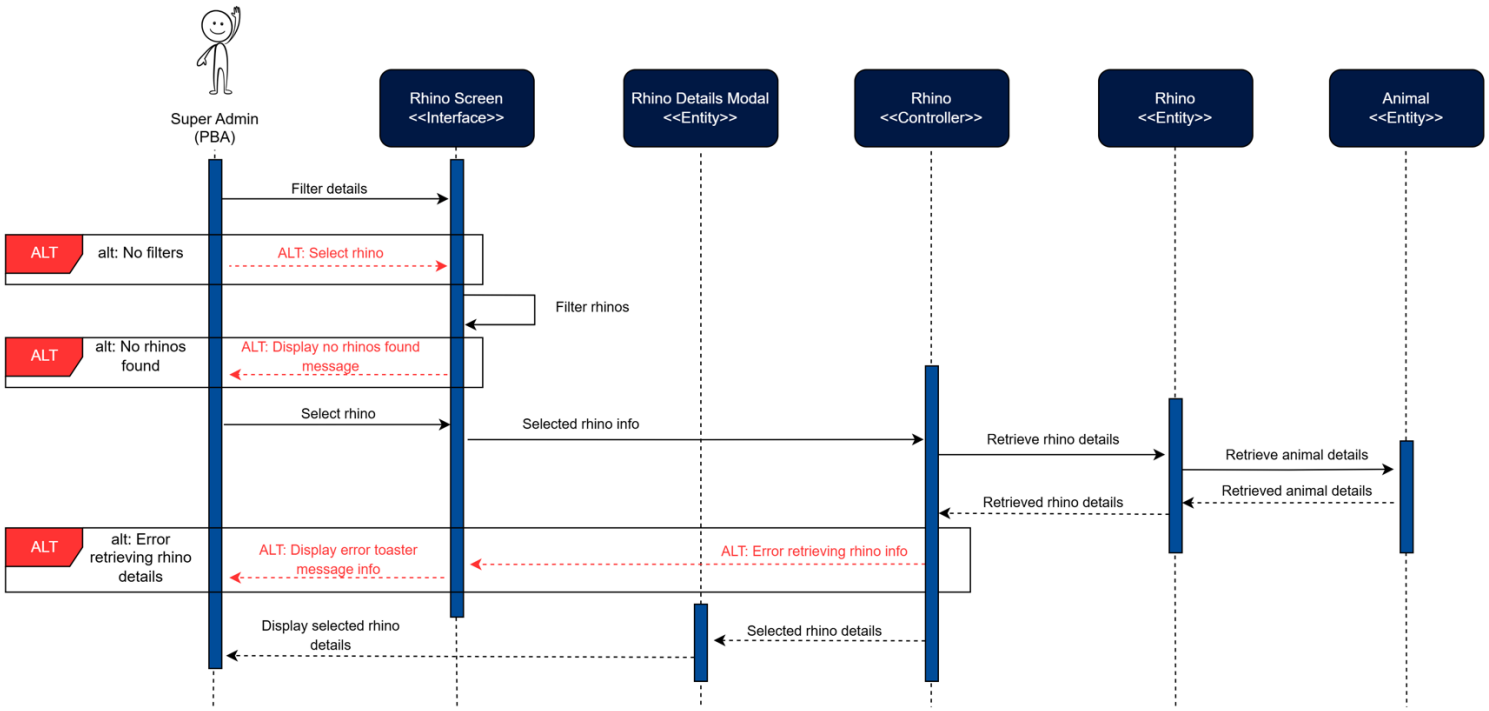


Figure 267: Sequence Diagram 9.2 View Rhino

2.7.8.3 View Rhinos: Use Case 9.3

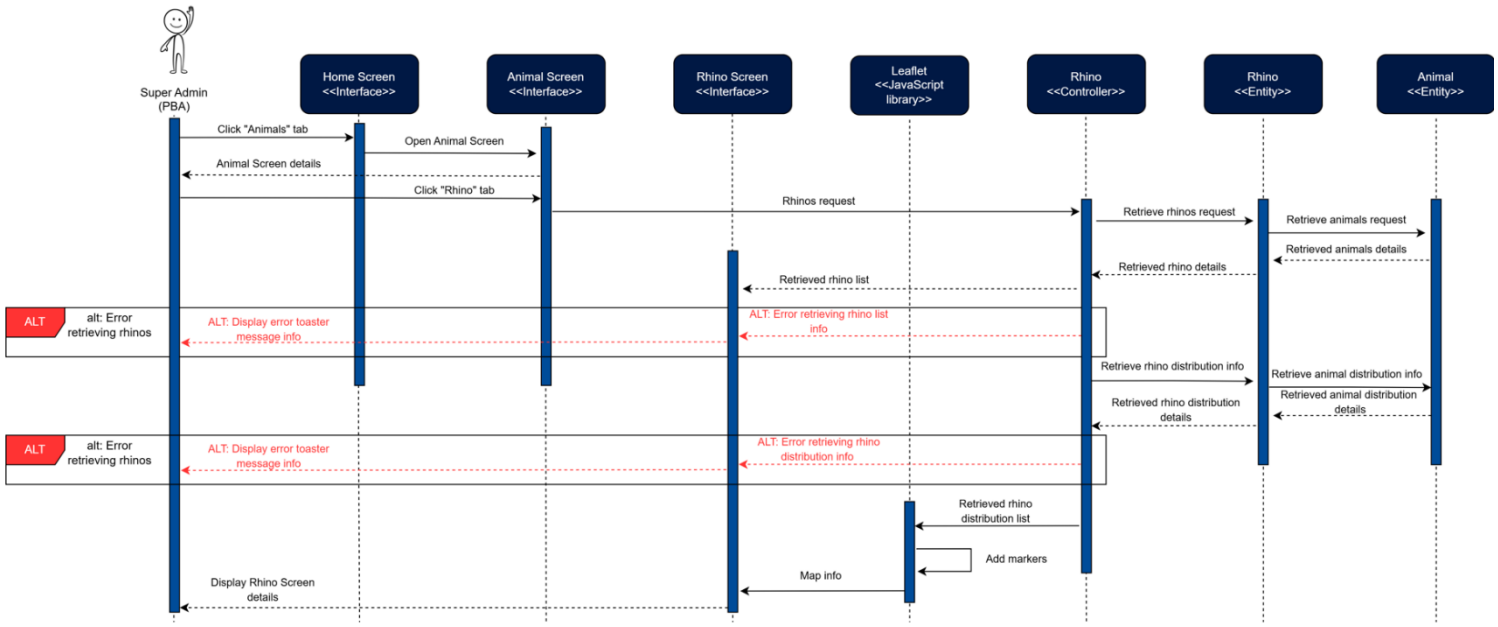


Figure 268: Sequence Diagram 9.3 View Rhinos

2.7.8.4 Update Rhino: Use Case 9.4

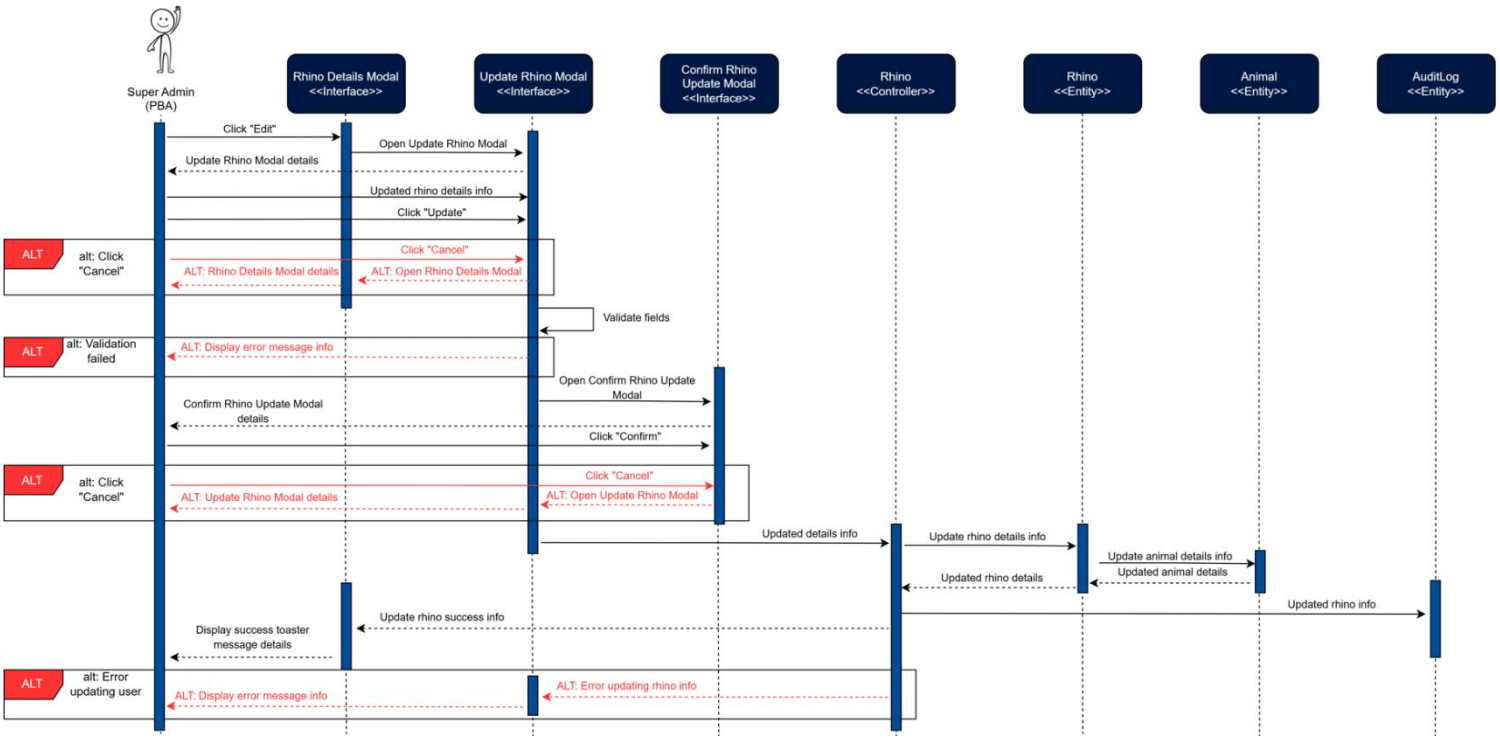


Figure 269: Sequence Diagram 9.4 Update Rhino

2.7.8.5 Deactivate Rhino: Use Case 9.5

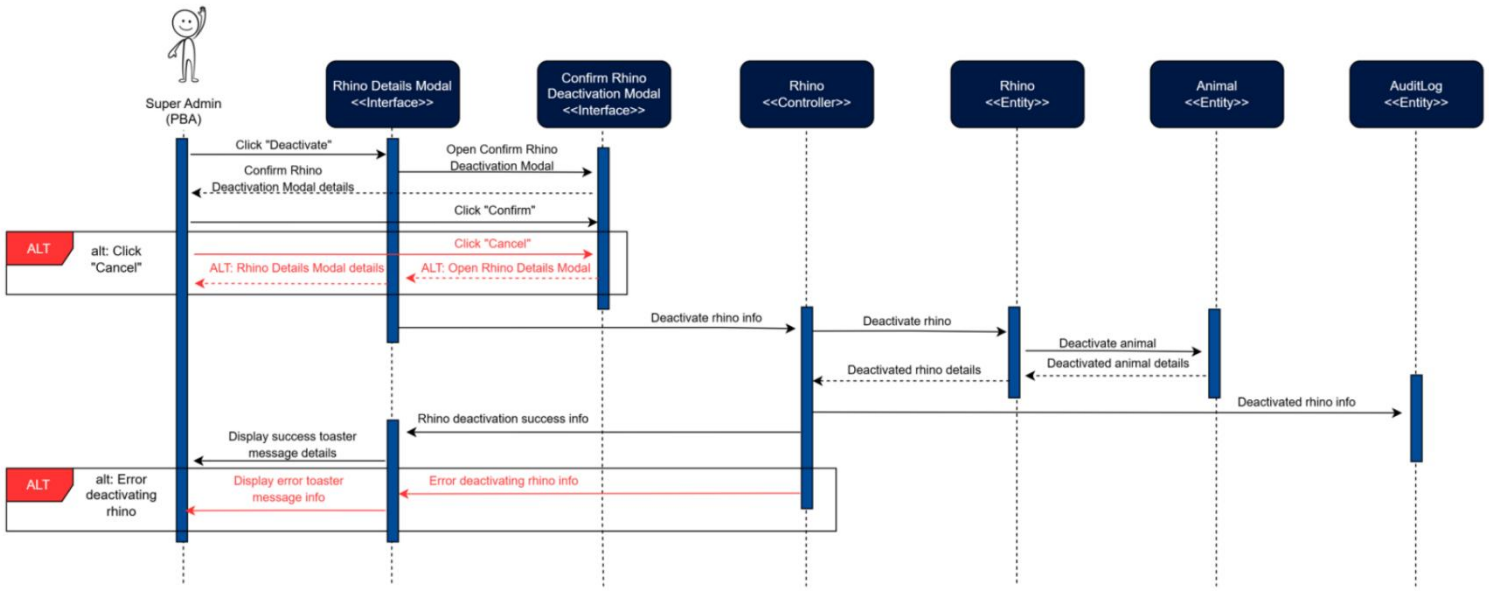


Figure 270: Sequence Diagram 9.5 Deactivate Rhino

2.7.9 Elephant Subsystem Sequence Diagrams

2.7.9.1 Add Elephant: Use Case 10.*1*

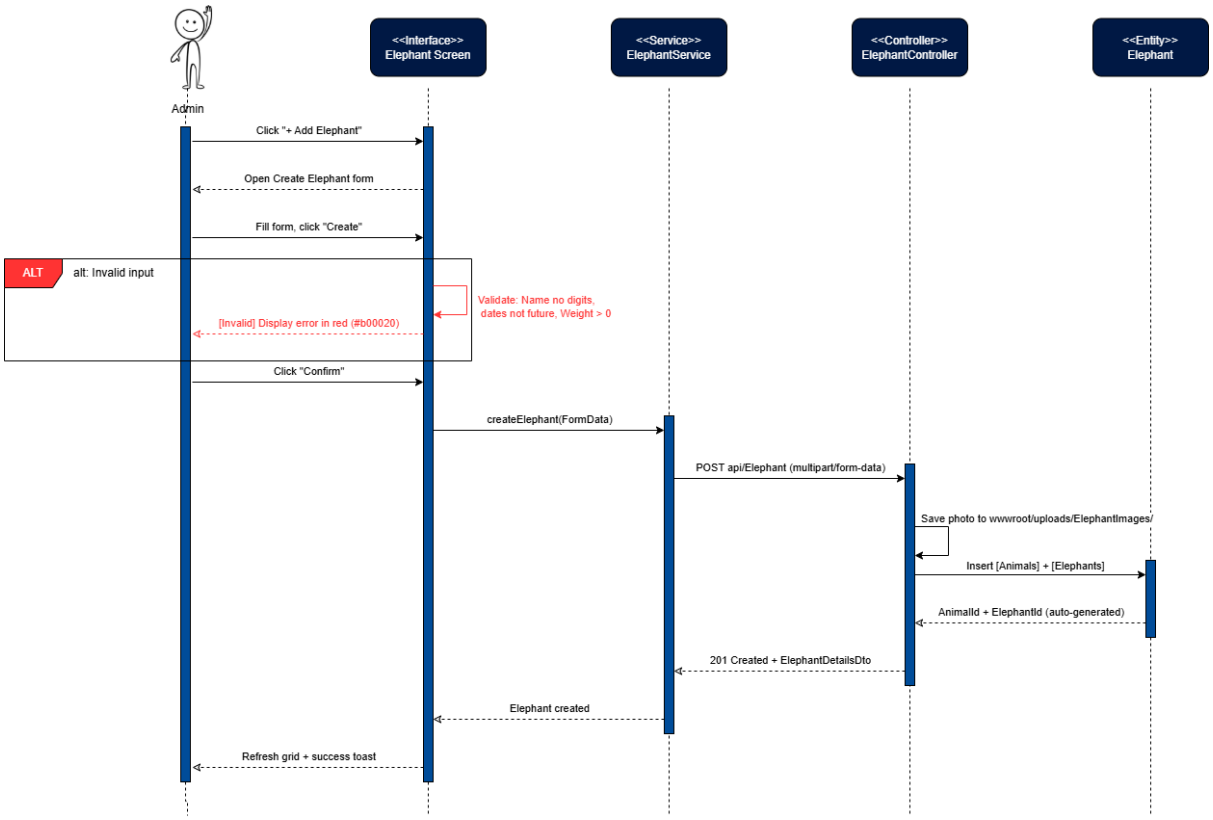


Figure 271: Sequence Diagram 10.1 Add Elephant

2.7.9.2 View Elephant: Use Case 10.2

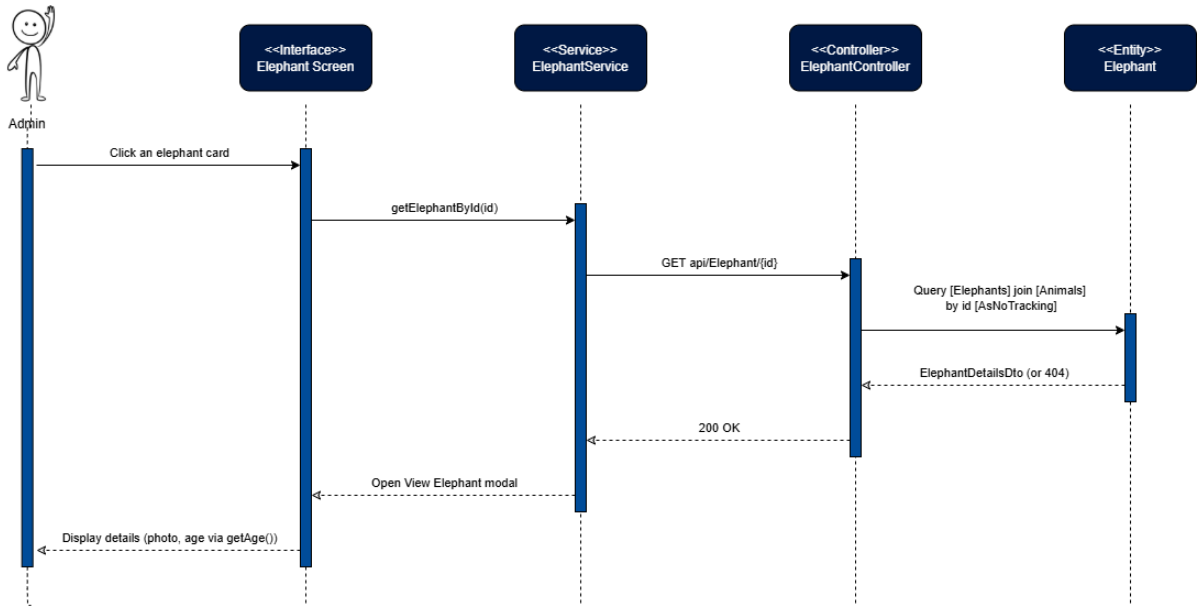


Figure 272: Sequence Diagram 10.2 View Elephant

2.7.9.3 View Elephant: Use Case 10.3

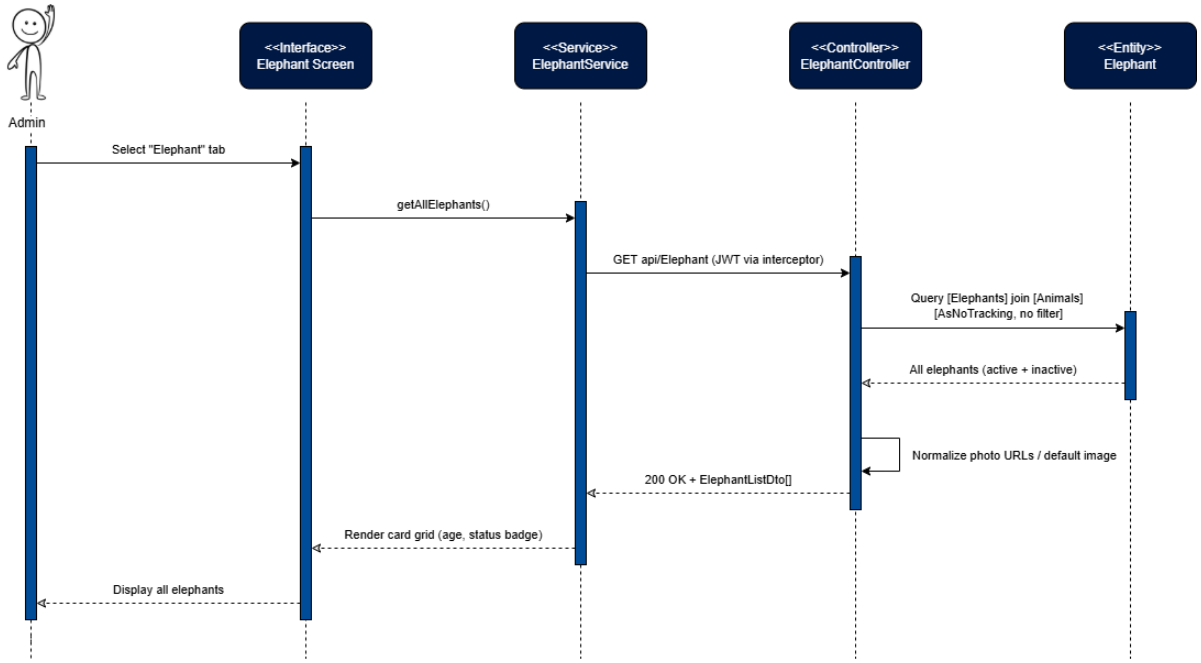


Figure 273: Sequence Diagram 10.3 View Elephant

2.7.9.4 View Elephants: Use Case 10.4

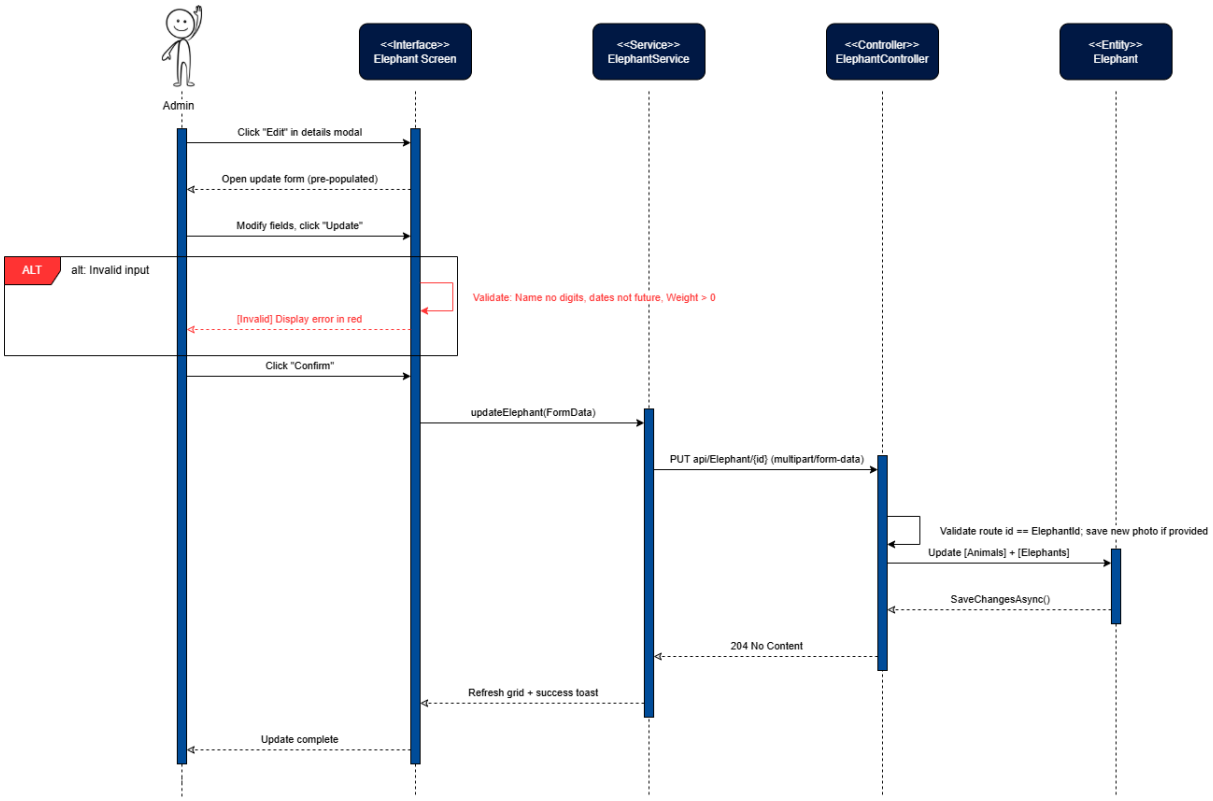


Figure 274: Sequence Diagram 10.4 View Elephants

2.7.9.5 Deactivate Elephant: Use Case 10.5

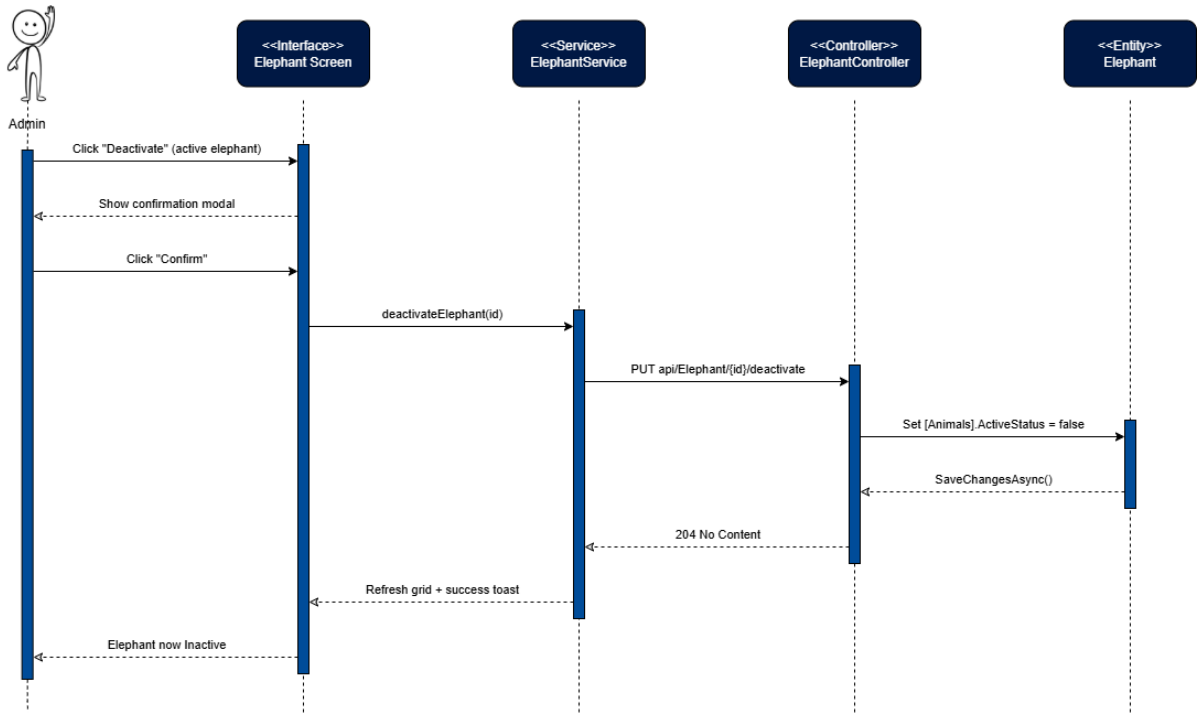


Figure 275: Sequence Diagram 10.5 Deactivate Elephant

2.7.10 *Report Subsystem Sequence Diagrams*

2.7.10.1 Generate Custom Report: Use Case 12.1

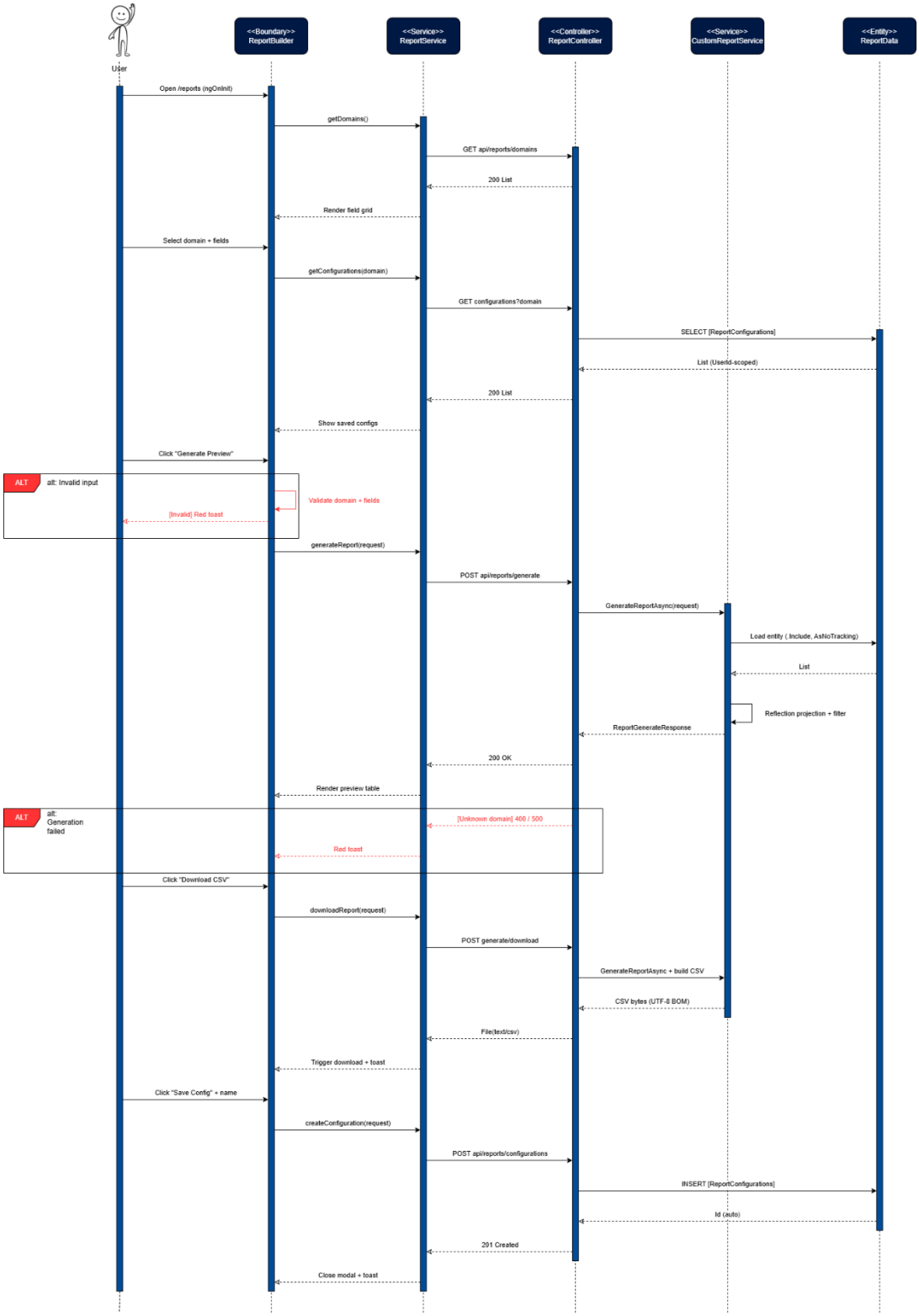


Figure 276: Sequence Diagram 12.1 Generate Custom Report

2.7.10.2 View Custom Report: Use Case 12.2

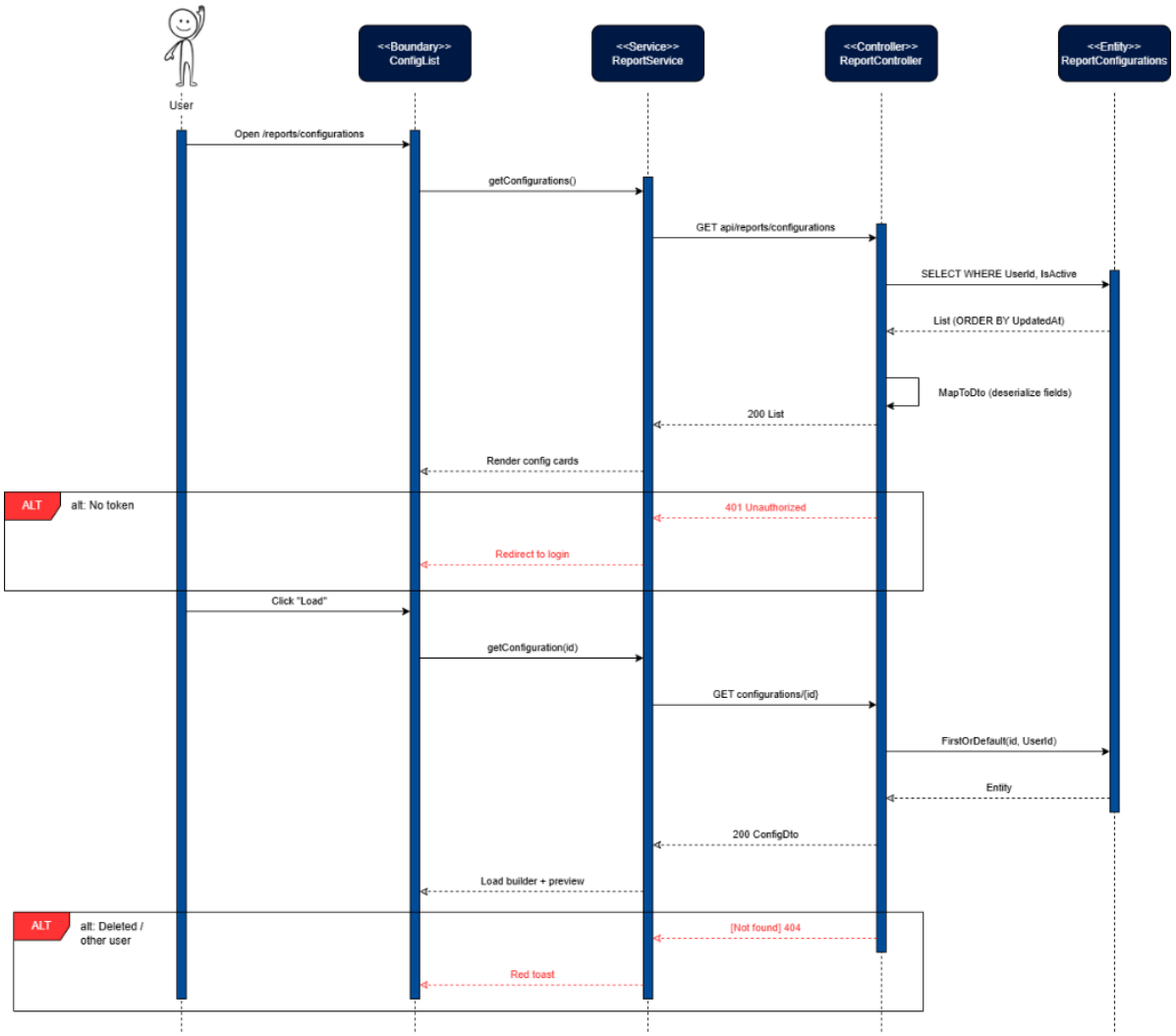


Figure 277: Sequence Diagram 12.2 View Custom Report

2.7.10.3 Update Custom Report: Use Case 12.3

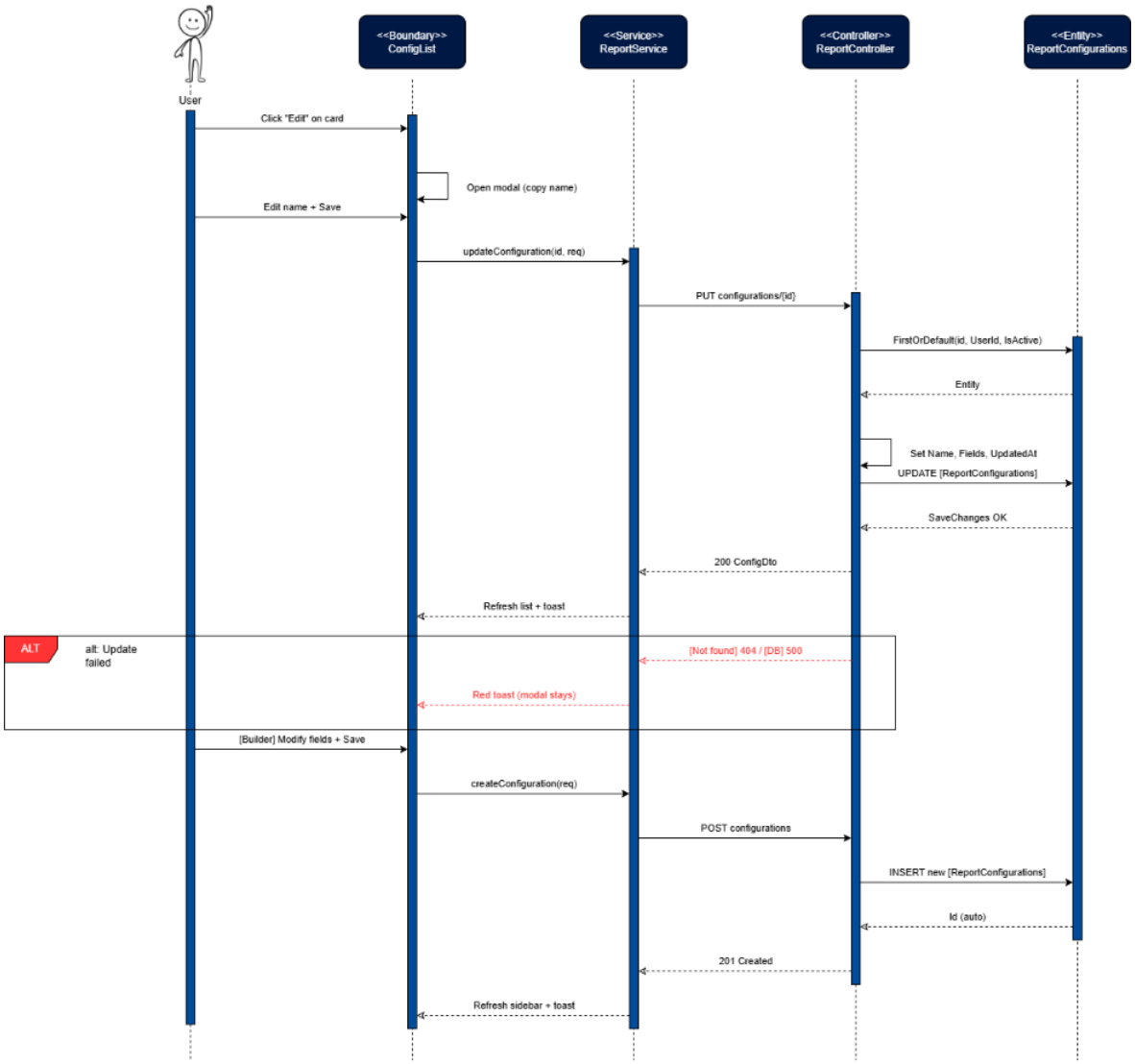


Figure 278: Sequence Diagram 12.3 Update Custom Report

2.7.10.4 Remove Custom Report: Use Case 12.4

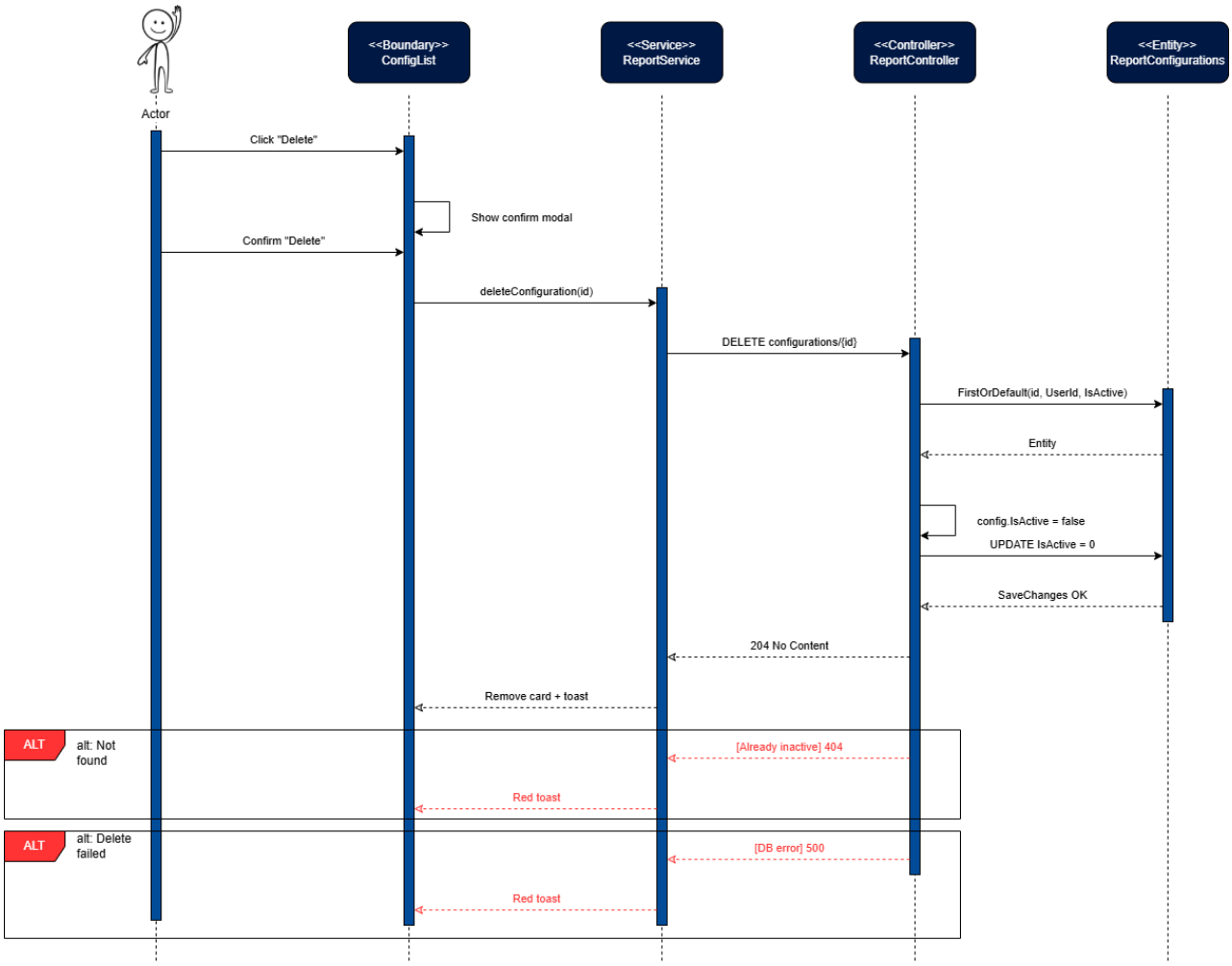


Figure 279: Sequence Diagram 12.4 Remove Custom Report

2.7.11 *Budget Subsystem Sequence Diagrams*

2.7.11.1 View Project Budget: Use Case 14.1

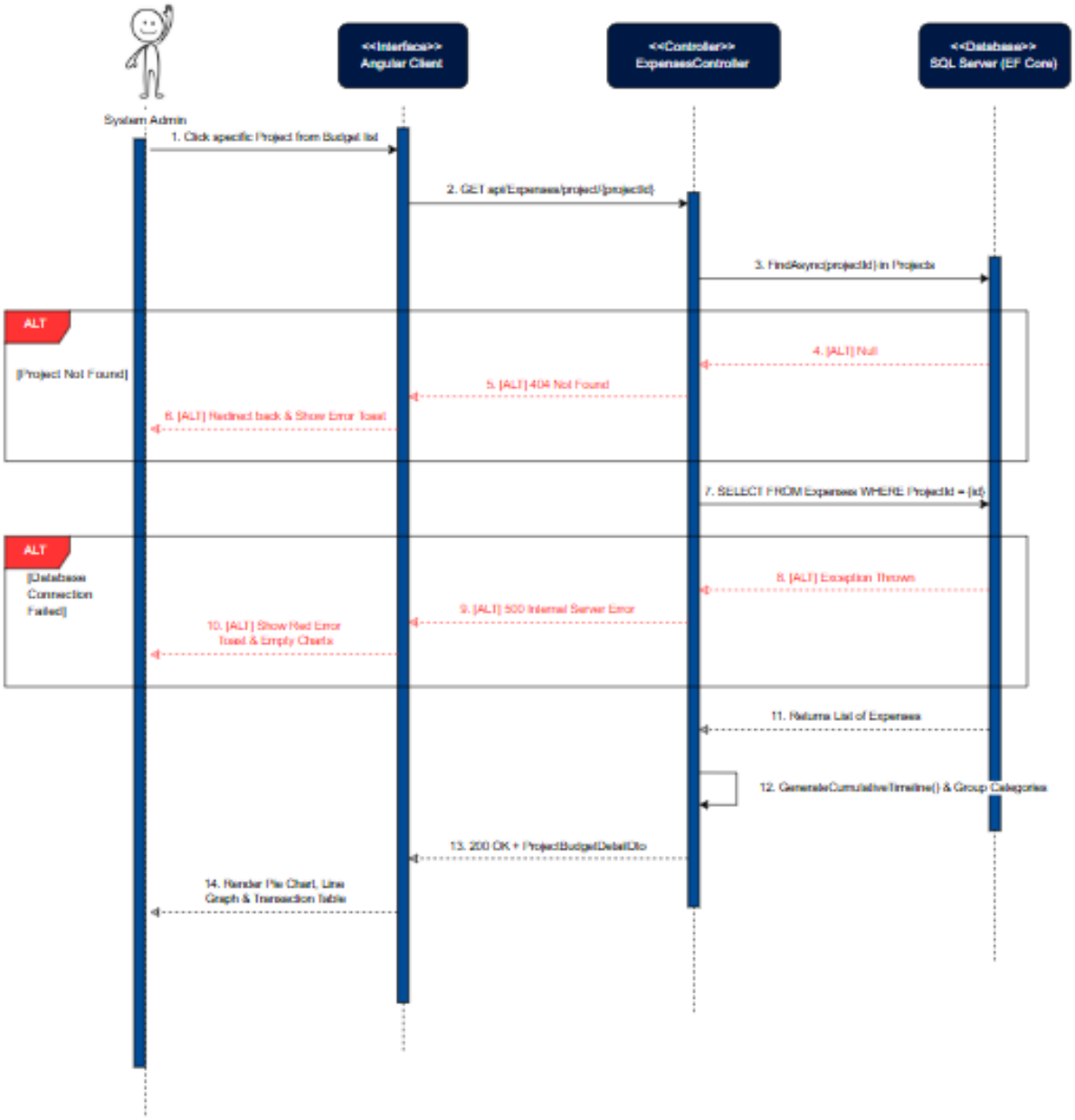


Figure 280: Sequence Diagram 14.1 View Project Budget

2.7.11.2 View Project Budgets: Use Case 14.2

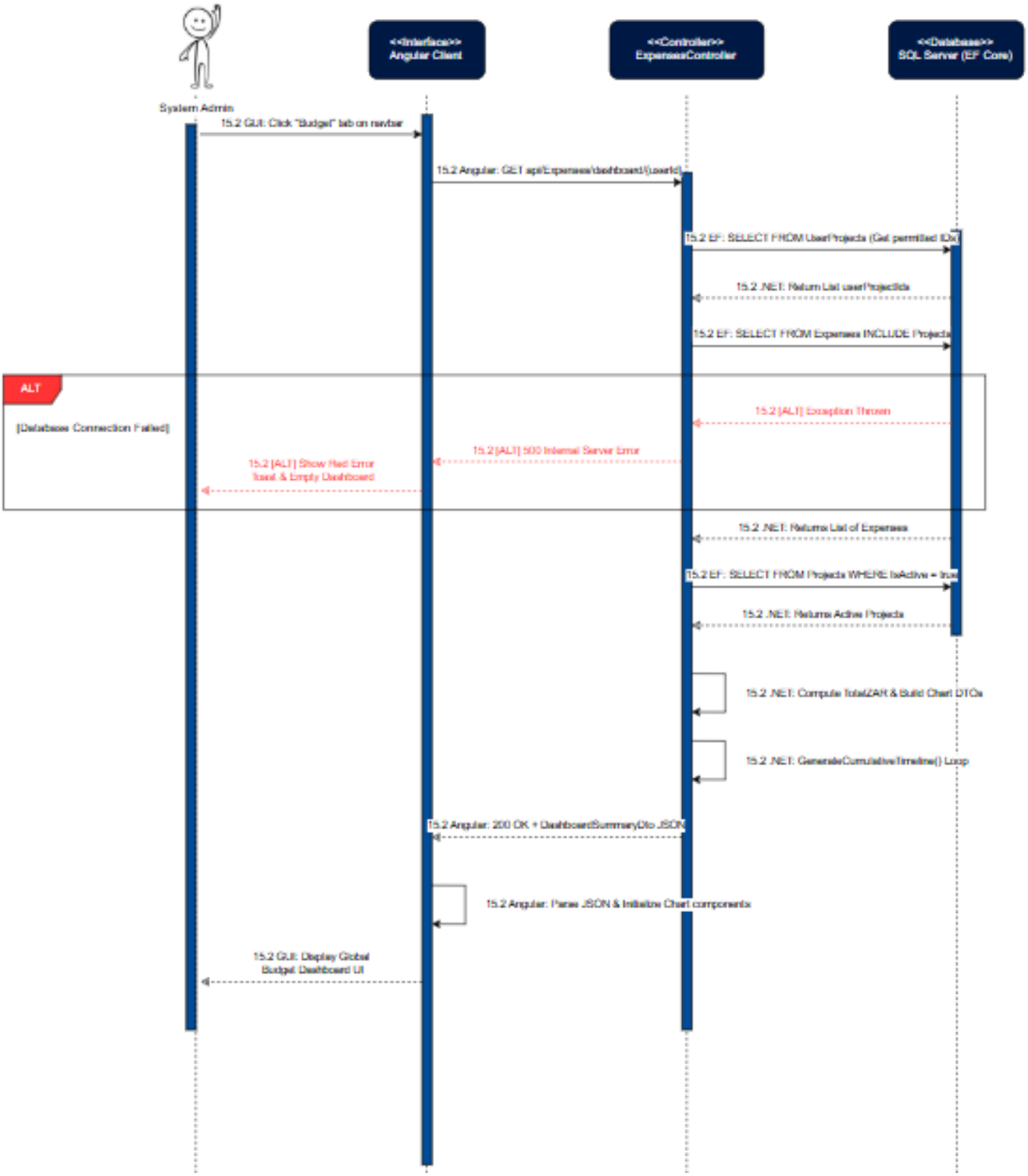


Figure 281: Sequence Diagram 14.2 View Project Budgets

2.7.11.3 Log an Expense: Use Case 14.3

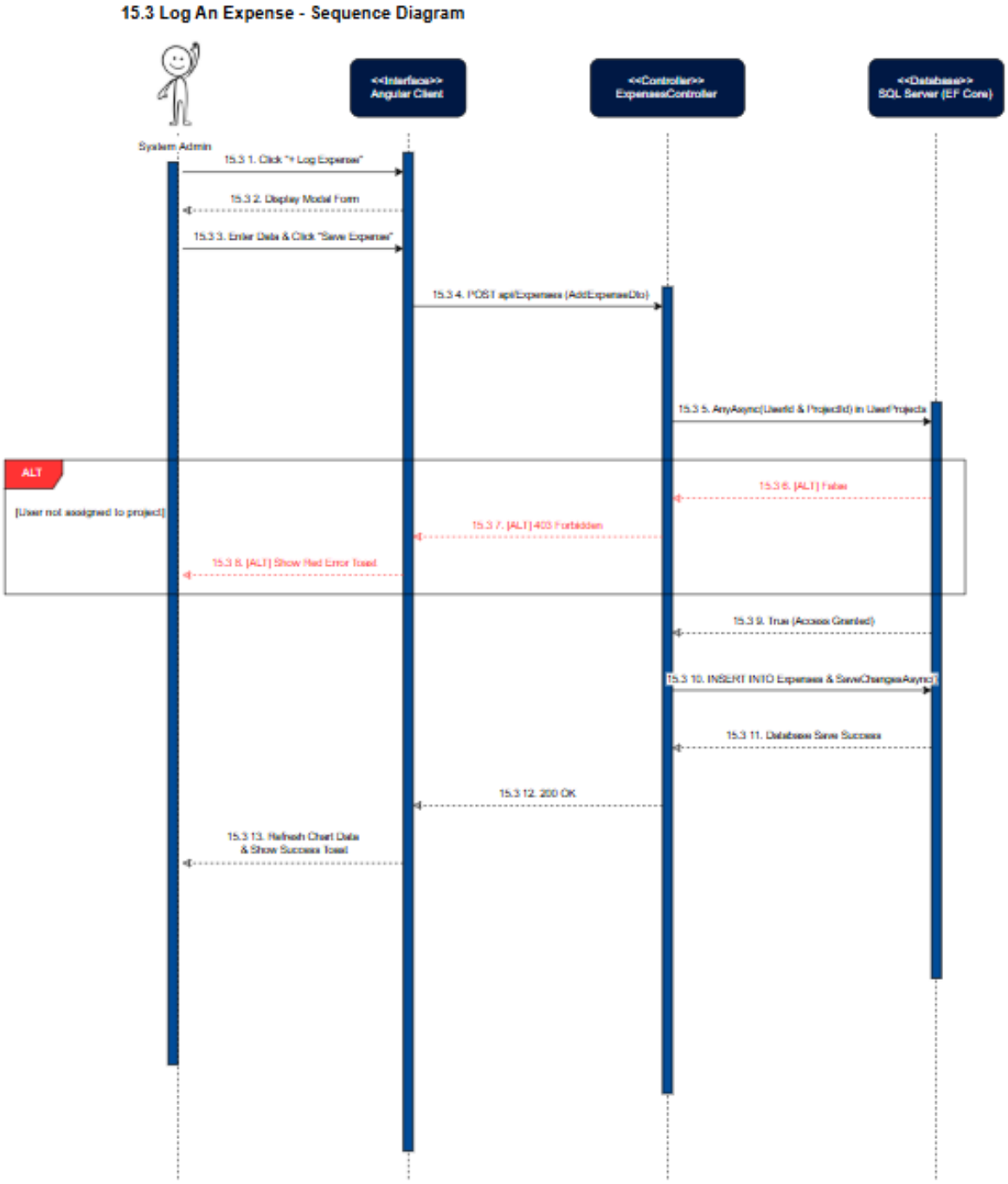


Figure 282: Sequence Diagram 14.3 Log an Expense

2.7.11.4 Update an Expense: 14.4

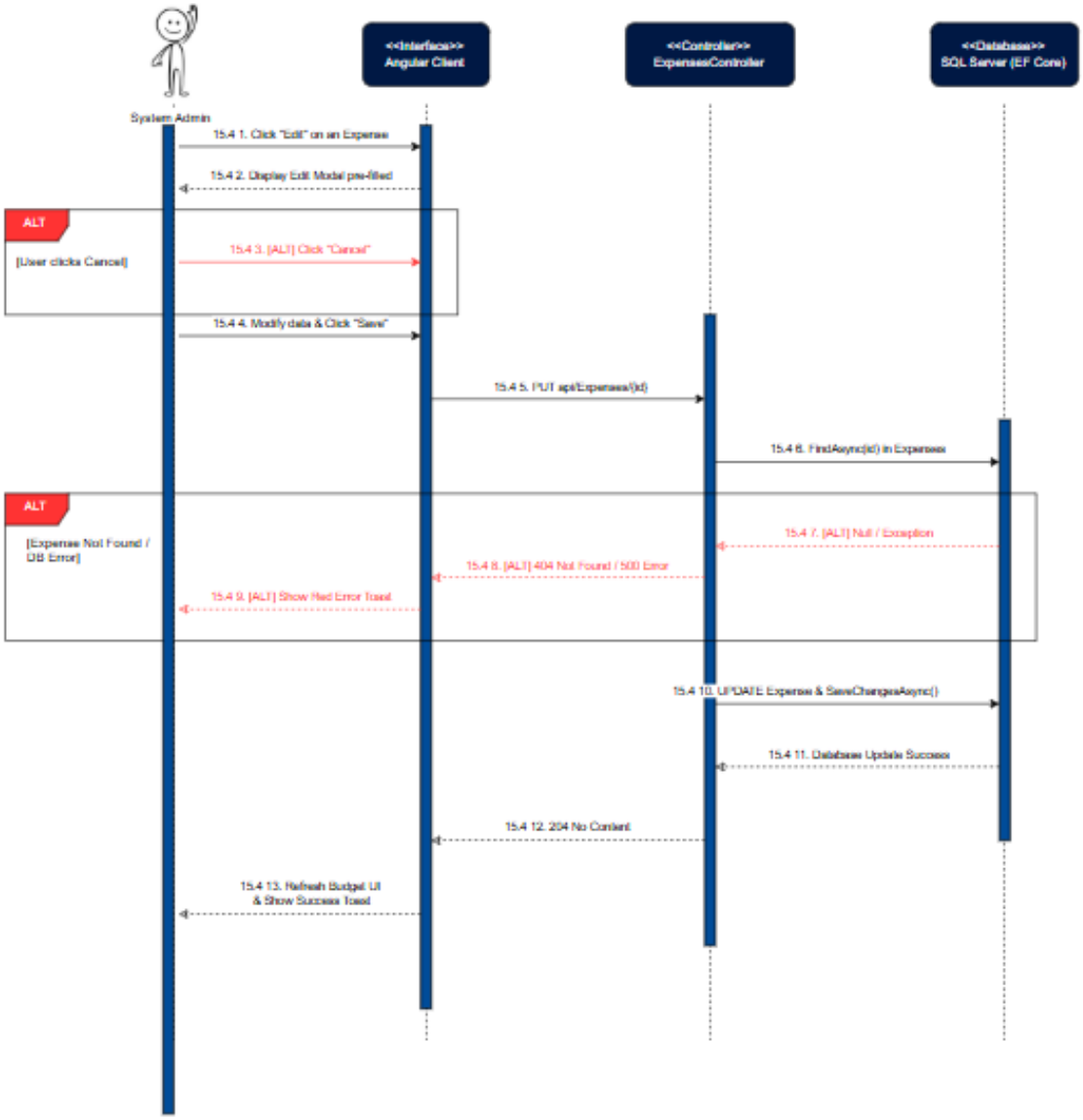


Figure 283: Sequence Diagram 14.4 Update an Expense

2.7.11.5 View Expense: Use Case 14.5

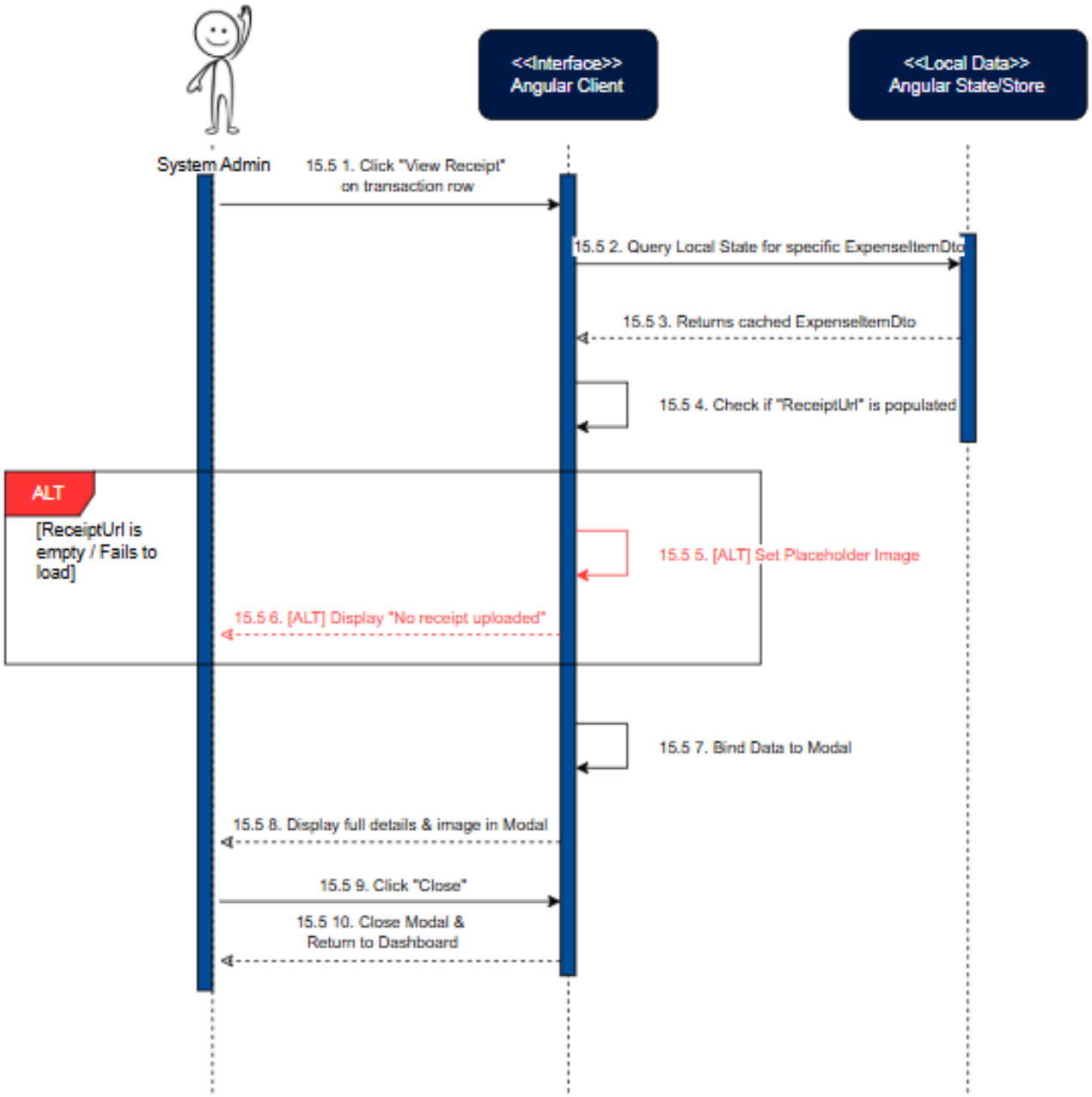


Figure 284: Sequence Diagram 14.5 View Expense

2.7.11.6 Remove Expense: Use Case 14.6

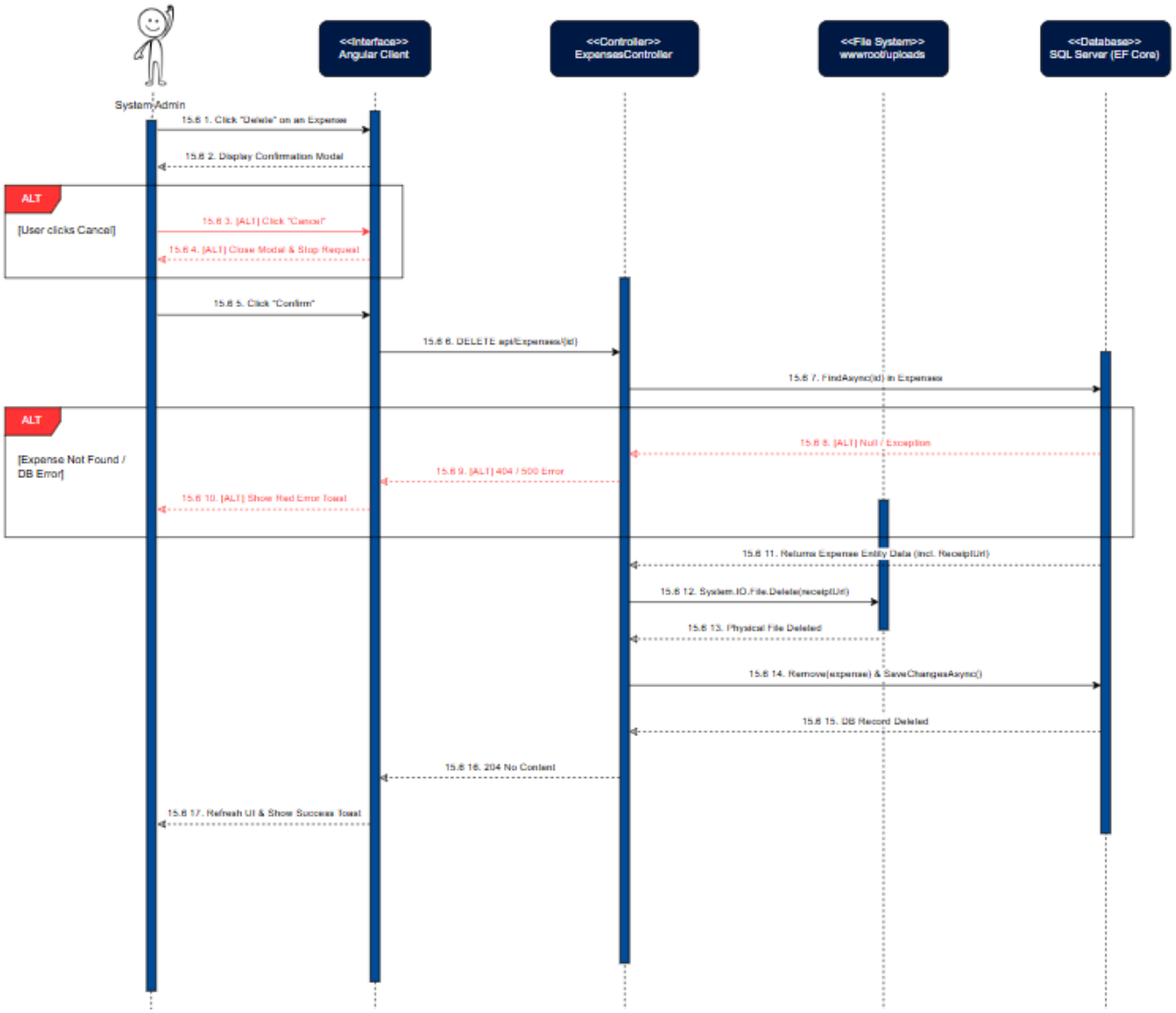


Figure 285: Sequence Diagram 14.6 Remove Expense

2.8 Updated Requirements List

2.8.1 Introduction

This section contains a list, as well as descriptions of all functional requirements.

Tables with changes that have been made to the list are also included.

2.8.2 Updated Functional Requirements Lists

2.8.2.1 User Subsystem

2.8.2.1.1 User Login

2.8.2.1.2 User Logout

2.8.2.1.3 Reset Password

2.8.2.1.4 Create User

2.8.2.1.5 Update User

2.8.2.1.6 View User

2.8.2.1.7 Reactivate User

2.8.2.1.8 Deactivate User

2.8.2.1.9 View FAQ

2.8.2.2 Admin Subsystem

2.8.2.2.1 Create User Profile

2.8.2.2.2 Update User Profile

2.8.2.2.3 View User Profile

2.8.2.2.4 Deactivate User Profile

2.8.2.2.5 Add FAQ

2.8.2.2.2.6 Update FAQ

2.8.2.2.7 Remove FAQ

2.8.2.2.8 Add FAQ Type

2.8.2.2.9 Update FAQ Type

2.8.2.2.10 Remove FAQ Type

2.8.2.2.11 Create Role

2.8.2.2.12 View Role

2.8.2.2.13 Update Role

2.8.2.2.14 Reactivate Role

2.8.2.2.15 Deactivate Role

2.8.2.3 Inventory Subsystem

2.8.2.3.1 Receive Inventory Item

2.8.2.3.2 View Inventory

2.8.2.3.3 Write-Off Inventory

2.8.2.3.4 Allocate Inventory

2.8.2.3.5 Create Inventory Item Type

2.8.2.3.7 View Inventory Item Type

2.8.2.3.7 View Inventory Item Types

2.8.2.3.8 Update Inventory Item Type

2.8.2.3.9 Deactivate Inventory Item Type

2.8.2.4 Group Subsystem

2.8.2.4.1 Create Group

2.8.2.4.2 View Groups

2.8.2.4.3 Update Group

2.8.2.4.4 Deactivate Group

2.8.2.4.5 Create Skill

2.8.2.4.6 View Skills

2.8.2.4.7 View Skill

2.8.2.4.8 Update Skill

2.8.2.4.9 Deactivate Skill

2.8.2.5 Warehouse Subsystem

2.8.2.5.1 Create Warehouse

2.8.2.5.2 View Warehouse

2.8.2.5.3 Update Warehouse

2.8.2.5.4 Reactivate Warehouse

2.8.2.5.5 Deactivate Warehouse

2.8.2.5.6 Inventory Stock Take

2.8.2.6 Vendor Subsystem

2.8.2.6.1 Create Vendor

2.8.2.6.2 View Vendor

2.8.2.6.3 View Vendors

2.8.2.6.4 Edit Vendor

2.8.2.6.5 Reactivate Vendor

2.8.2.6.6 Deactivate Vendor

2.8.2.6.7 Create Vendor Type

2.8.2.6.8 Update Vendor Type

2.8.2.6.9 View Vendor Types

2.8.2.6.10 Reactivate Vendor Type

2.8.2.6.11 Deactivate Vendor Type

2.8.2.7 Project Management Subsystem

2.8.2.7.1 Create Project

2.8.2.7.2 View Project

2.8.2.7.3 Update Project

2.8.2.7.4 Deactivate Project

2.8.2.7.5 Create Project Type

2.8.2.7.6 View Project Type

2.8.2.7.7 Update Project Type

2.8.2.7.8 Deactivate Project Type

2.8.2.7.9 Add Indemnity Form

2.8.2.7.10 Update Indemnity Form

2.8.2.7.11 View Indemnity Form

2.8.2.7.12 Remove Indemnity Form

2.8.2.7.13 Capture Indemnity Form

2.8.2.7.14 Capture Project Feedback

2.8.2.7.15 View Project Feedback

2.8.2.7.16 Create Impact Feedback Form

2.8.2.7.17 View Impact Feedback Form

2.8.2.7.18 Update Impact Feedback Form

2.8.2.7.19 Deactivate Impact Feedback Form

2.8.2.7.20 Capture Impact Form

2.8.2.7.21 View Impact Form

2.8.2.7.22 Add Photo

2.8.2.7.23 View Photo

2.8.2.7.24 Delete Photo

2.8.2.8 K9 Subsystem

2.8.2.8.1 Add Dog

2.8.2.8.2 View Dog

2.8.2.8.3 View Dogs

2.8.2.8.4 Update Dog

2.8.2.8.5 Deactivate Dog

2.8.2.8.6 Assign User to Dog

2.8.2.8.7 View User-Dog Assignment

2.8.2.8.8 Update User-Dog Assignment

2.8.2.8.9 Reactivate User-Dog Assignment

2.8.2.8.10 Deactivate User-Dog Assignment

2.8.2.8.11 Add K9 Role

2.8.2.8.12 Update K9 Role

2.8.2.8.13 View K9 Roles

2.8.2.8.14 Reactivate K9 Role

2.8.2.8.15 Deactivate K9 Role

2.8.2.8.16 View K9 Ancestry

2.8.2.9 Rhino Subsystem

2.8.2.9.1 Add Rhino

2.8.2.9.2 View Rhino

2.8.2.9.3 View Rhinos

2.8.2.9.4 Update Rhino

2.8.2.9.5 Deactivate Rhino

2.8.2.10 Elephant Subsystem

2.8.2.10.1 Add Elephant

2.8.2.10.2 View Elephant

2.8.2.10.3 View Elephants

2.8.2.10.4 Update Elephant

2.8.2.10.5 Deactivate Elephant

2.8.2.11 Message Subsystem

2.8.2.11.1 Create Message

2.8.2.11.2 View Messages

2.8.2.12 Report Subsystem

2.8.2.12.1 Generate Custom Report

2.8.2.12.2 View Custom Report

2.8.2.12.3 Update Custom Report

2.8.2.12.4 Remove Custom Report

2.8.2.12.5 View Bike Impact Report

2.8.2.12.6 View Budget Report

2.8.2.12.7 View Rhino Report

2.8.2.12.8 View Elephant Report

2.8.2.12.9 View K9 Report

2.8.2.12.10 View Machine Learning Dashboard

2.8.2.13. System Subsystem

2.8.2.13.1 Backup System

2.8.2.13.2 Restore System

2.8.2.13.3 View Audit Log

2.8.2.14 Budget Subsystem

2.8.2.14.1 View Project Budget

2.8.2.14.2 View Project Budgets

2.8.2.14.3 Log An Expense

2.8.2.14.4 Update An Expense

2.8.2.14.5 View Expense

2.8.2.14.6 Remove Expense

2.8.3 Updated Functional Requirements Lists Descriptions

2.8.3.1 User Subsystem

Functional Requirement	Explanation
Requirement number:	1.1
Requirement name (use case name):	User Login
Requirement short description:	A User logs into the system using their email address and password.
Requirement detailed description and constraints:	A user logs in to the system using their email address and password. The system compares the email address with the associated hashed password in the database; upon successful authentication the user is directed to the appropriate dashboard based on their role and associated permission level. **Constraints:** • The user should already have created an account on the system. If not, they must sign up. • Time constraint: the use case must be completed before the Project Day in October 2026. • User skill constraint: the use case must be easy to understand and use, as some users will have limited technology abilities. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Secure Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 84: 1.1 User Login – Description

Functional Requirement	Explanation
Requirement number:	1.2
Requirement name (use case name):	User Logout
Requirement short description:	A user can log out of the system.
Requirement detailed description and constraints:	A user can log out of the system, thus terminating their session, ensuring that no one with physical access to their device can access the system. Once they have completed this action, they are directed to the Login page. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • User skill constraint: the use case must be easy to understand and use, as some users will have limited technology abilities. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Secure Access.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 85: 1.2 User Logout Description

Functional Requirement	Explanation
Requirement number:	1.3
Requirement name (use case name):	Reset Password
Requirement short description:	A user could reset their password if they either forgot it or suspect that their credentials might have leaked on another service.
Requirement detailed description and constraints:	A user can reset their password if they either forgot it or suspect that their credentials have leaked on another service. As the system does not save passwords in plain text, but rather a hashed format, the user will be prompted to enter their email address, to which a reset link will be sent. Once they have clicked the link, they have access to a field where they can enter a new password. The password hash is then saved in the database, replacing the previous version. Once these actions have been completed, the user will be redirected to the Login page. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • User skill constraint: the use case must be easy to understand and use, as some users will have limited technology abilities. • The user requires an Internet connection to access this feature. • The password cannot be the same as the previous password, if this edge case occurs, the system will notify that the passwords are the same. • The user should already have created an account on the system. If not, they must sign up.
Business rules applicable to this requirement	• Secure Access. • Ability to recover Account.
Revision date and Revision number:	2026-04-09 Version 3.0

Functional Requirement	Explanation
Criticality/Priority:	High

Table 86: 1.3 Reset Password – Description

Functional Requirement	Explanation
Requirement number:	1.4
Requirement name (use case name):	Create User
Requirement short description:	A person must be able to create a user profile by signing up to the system.
Requirement detailed description and constraints:	A person must be able to create a user profile by signing up to the system. This entails providing login credentials (email address/password), basic personal information such as gender and age, location and selecting skills they possess. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • User skill constraint: the use case must be easy to understand and use, as some users will have limited technology abilities. • The user requires an Internet connection to access this feature. • The new user must have a valid email address.
Business rules applicable to this requirement	• Secure Access.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 87: 1.4 Create User – Description

Functional Requirement	Explanation
Requirement number:	1.5
Requirement name (use case name):	Update User
Requirement short description:	A user must be able to edit their personal user profile.
Requirement detailed description and constraints:	A user must be able to edit their personal user profile. This could entail changing details like location or the skills they possess. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • User skill constraint: the use case must be easy to understand and use, as some users will have limited technology abilities. • The user requires an Internet connection to access this feature. • The user cannot change their age, as this is automatically updated by the system.
Business rules applicable to this requirement	• Secure Access.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 88: 1.5 Update User – Description

Functional Requirement	Explanation
Requirement number:	1.6
Requirement name (use case name):	View User
Requirement short description:	A user must be able to view their personal user profile.
Requirement detailed description and constraints:	A user must be able to edit their personal user profile. Administrators should have the ability to view the personal profiles of all users This could entail changing details like location or the skills they possess. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • User skill constraint: the use case must be easy to understand and use, as some users will have limited technology abilities. • The user requires an Internet connection to access this feature. • Only Administrators can view the full profile details of other users.
Business rules applicable to this requirement	• Secure Access.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 89: 1.6 View User – Description

Functional Requirement	Explanation
Requirement number:	1.7
Requirement name (use case name):	Reactivate User
Requirement short description:	An administrator must be able to reactivate an account.
Requirement detailed description and constraints:	An administrator should be able to reactivate any deactivated user accounts. Once an account has been reactivated, a user can login to the system using their credentials and will appear on the list of active users again. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • User skill constraint: the use case must be easy to understand and use, as some users will have limited technology abilities. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Secure Access. • Data Retention.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 90: 1.7 Reactivate User - Description

Functional Requirement	Explanation
Requirement number:	1.8
Requirement name (use case name):	Deactivate User
Requirement short description:	A user must be able to deactivate their personal user profile.
Requirement detailed description and constraints:	A user must be able to deactivate their personal user profile. Once they have selected the option to deactivate their profile, they will be asked to confirm. If they confirm, the profile will be deactivated, and they will be directed to the login page. Data regarding their involvement in projects will be kept for accurate records on historical data, but they will no longer appear on the users list. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • User skill constraint: the use case must be easy to understand and use, as some users will have limited technology abilities. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Secure Access. • Data Retention.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 91: 1.8 Deactivate User - Description

Functional Requirement	Explanation
Requirement number:	1.9
Requirement name (use case name):	View FAQ
Requirement short description:	A user must be able to click a help button that takes them to a context aware FAQ page.
Requirement detailed description and constraints:	A user must be able to click a help button that takes them to a context aware FAQ page. If the button is pressed on the home page, it will direct the user to the full FAQ document, but if it is pressed while using a subsystem, it will direct the user to the appropriate section as set up by the admin using the FAQ Type use cases. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • User skill constraint: the use case must be easy to understand and use, as some users will have limited technology abilities. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Access to a help function.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 92: 1.9 View FAQ - Description

2.8.3.2 Admin Subsystem

Functional Requirement	Explanation
Requirement number:	2.1
Requirement name (use case name):	Create User Profile
Requirement short description:	A Super Admin must be able to create a new user profile in the system.
Requirement detailed description and constraints:	Super Admin can input necessary details to create a new user. The system validates entries before asking for confirmation. Once confirmed, the user profile is saved. **Constraints:** • Internet connection required. • Mandatory fields must be completed before submission.
Business rules applicable to this requirement	• Only the Super Admin can create user profiles.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 93: 2.1 Create User Profile - Description

Functional Requirement	Explanation
Requirement number:	2.2
Requirement name (use case name):	Update User Profile
Requirement short description:	A Super Admin must be able to update existing user profiles.
Requirement detailed description and constraints:	The Super Admin can modify user information. System validates changes before confirmation. Once confirmed, updates are applied. **Constraints:** • Internet connection required. • Cannot change another user's password without proper authorization.
Business rules applicable to this requirement	• Only the Super Admin can create user profiles.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 94: 2.2 Update User Profile - Description

Functional Requirement	Explanation
Requirement number:	2.3
Requirement name (use case name):	View User Profile
Requirement short description:	The Super Admin must be able to view user profiles.
Requirement detailed description and constraints:	The Super Admin can search and view details of existing users. No changes are allowed in this view. **Constraints:** • Internet connection required.
Business rules applicable to this requirement	• Only the Super Admin can view any profiles.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 95: 2.3 View User Profile - Description

Functional Requirement	Explanation
Requirement number:	2.4
Requirement name (use case name):	Deactivate User Profile
Requirement short description:	The Super Admin can deactivate user profiles.
Requirement detailed description and constraints:	Super Admin can disable a profile temporarily. The system confirms the action before applying it. **Constraints:** • Internet connection required. • Action must be logged in the system audit trail.
Business rules applicable to this requirement	• Only the Super Admin can deactivate profiles.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 96: 2.4 Deactivate User Profile - Description

Functional Requirement	Explanation
Requirement number:	2.5
Requirement name (use case name):	Add FAQ
Requirement short description:	The Super Admin can add new FAQs to the system.
Requirement detailed description and constraints:	The Super Admin input question and answer details. System validates before confirmation. Once confirmed, FAQ is added. **Constraints:** • Internet connection required.
Business rules applicable to this requirement	• Only the Super Admin can add FAQs.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 97: 2.5 Add FAQ – Description

Functional Requirement	Explanation
Requirement number:	2.6
Requirement name (use case name):	Update FAQ
Requirement short description:	The Super Admin can update existing FAQs.
Requirement detailed description and constraints:	The Super Admin can edit FAQ content, which is validated before confirmation. Once confirmed, updates are applied. **Constraints:** • Internet connection required.
Business rules applicable to this requirement	• Only the Super Admin can update FAQs.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 98: 2.6 Update FAQ – Description

Functional Requirement	Explanation
Requirement number:	2.7
Requirement name (use case name):	Remove FAQ
Requirement short description:	The Super Admin can remove FAQs from the system.
Requirement detailed description and constraints:	The Super Admin can select FAQ to delete. System asks for confirmation before removal. **Constraints:** • Internet connection required.
Business rules applicable to this requirement	• Only the Super Admin can remove FAQs from the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 99: 2.7 Remove FAQ – Description

Functional Requirement	Explanation
Requirement number:	2.8
Requirement name (use case name):	Add FAQ Type
Requirement short description:	A Super Admin must be able to capture and commit a new FAQ classification category label to the global lookup directory.
Requirement detailed description and constraints:	The system must provide an interactive Angular Reactive Form template inside an overlay modal layout. The workspace operator must be allowed to input a structural configuration name string, declare a comprehensive functional description scope, and assign an active boolean data visibility property state. **Constraints:** • Time Constraint: The technical pipeline implementation and API route deployment must be completed and audited before the Project Day in October 2026. • User Skill Constraint: The user interface components must utilize responsive, clean, and intuitive validation feedback indicators to accommodate administrative users with minimal systemic training. • Network Constraint: The application client requires a persistent HTTPS web data connection stream to dispatch payloads to host environments. • Entity Constraint: Alphanumeric parameters mapping to the TypeName input boundary must strictly not exceed a data allocation size ceiling threshold of 100 characters.
Business rules applicable to this requirement	• Secure Access. • Classification Domain Uniqueness
Revision date and Revision number:	2026-06-24 Version 1.0
Criticality/Priority:	High

Functional Requirement	Explanation
Requirement number:	2.9
Requirement name (use case name):	Update FAQ Type
Requirement short description:	A Super Admin must be able to alter the alphanumeric properties or status configurations of an existing FAQ classification type entity.
Requirement detailed description and constraints:	The system must launch a contextual edit layout container window and perform an automated patchValue routine to inject current relational data properties into the active form controls. The interface must trap mutations made to the type name or scope variables and package them into an asynchronous HTTP PUT data stream. **Constraints:** • **Time Constraint:** The technical pipeline implementation and API route deployment must be completed and audited before the Project Day in October 2026. • **Operational Constraint:** Form modifications must process asynchronously through specialized data transfer objects (DTOs) to insulate database table layouts from the network stream layer. • **Network Constraint:** The client framework requires a persistent internet stream to hit the targeted server router path (PUT api/FaqType/update/{id}).
Business rules applicable to this requirement	• Secure Access. • Data Overwrite Consistency
Revision date and Revision number:	2026-06-24 Version 1.0
Criticality/Priority:	High

Functional Requirement	Explanation
Requirement number:	2.10
Requirement name (use case name):	Remove FAQ Type
Requirement short description:	A Super Admin must be able to remove an FAQ classification type category row from the system database when it is no longer required.
Requirement detailed description and constraints:	The frontend application must intercept row-level deletion clicks and display an explicit modal verification confirmation overlay block. Upon user affirmation, the framework must compile an asynchronous HTTP DELETE transaction addressing the primary key parameter sequence route (DELETE api/FaqType/remove/{id}). The backend must validate database dependency maps prior to dropping rows from disk storage files. **Constraints:** • **Time Constraint:** The technical pipeline implementation and API route deployment must be completed and audited before the Project Day in October 2026. • **User Error Protection Constraint:** Destructive command routines must never process silent; an explicit user-facing verification warning prompt layout component must always disrupt execution loops. • **Network Constraint:** The system requires stable network connectivity to safely process transaction signatures without corrupting indexing tables.
Business rules applicable to this requirement	• Secure Access.
Revision date and Revision number:	2026-06-24 Version 1.0
Criticality/Priority:	High

Functional Requirement	Explanation
Requirement number:	2.11
Requirement name (use case name):	Create Role
Requirement short description:	The Super Admin can create new roles in the system.
Requirement detailed description and constraints:	The Super Admin enter role name and permissions. System validates input before confirmation. Once confirmed, role is created. **Constraints:** • Internet connection required.
Business rules applicable to this requirement	• The Super Admin can create roles.
Revision date and Revision number:	2026-06-24 Version 3.0
Criticality/Priority:	High

Table 100: 2.8 Create Role – Description

Functional Requirement	Explanation
Requirement number:	2.12
Requirement name (use case name):	View Role
Requirement short description:	The Super Admin can view details of the existing roles.
Requirement detailed description and constraints:	The Super Admin can search and see role details. No changes allowed in view mode. **Constraints:** • Internet connection required.
Business rules applicable to this requirement	• Only the Super Admin can view roles.
Revision date and Revision number:	2026-06-24 Version 3.0
Criticality/Priority:	High

Table 101: 2.9 View Role – Description

Functional Requirement	Explanation
Requirement number:	2.13
Requirement name (use case name):	Update Role
Requirement short description:	The Super Admin can update existing roles.
Requirement detailed description and constraints:	The Super Admin can modify role names or permissions. System validates changes before confirmation. **Constraints:** • Internet connection required.
Business rules applicable to this requirement	• The Super Admin can update roles.
Revision date and Revision number:	2026-06-24 Version 3.0
Criticality/Priority:	High

Table 102: 2.10 Update Role – Description

Functional Requirement	Explanation
Requirement number:	2.14
Requirement name (use case name):	Reactivate Role
Requirement short description:	The Super Admin can reactivate deactivated roles.
Requirement detailed description and constraints:	The Super Admin selects a deactivated role. System confirms before reactivation. **Constraints:** • Internet connection required.
Business rules applicable to this requirement	• Only the Super Admin can reactivate a role.
Revision date and Revision number:	2026-06-24 Version 3.0
Criticality/Priority:	High

Table 103:2.11 Reactivate Role – Description

Functional Requirement	Explanation
Requirement number:	2.15
Requirement name (use case name):	Deactivate Role
Requirement short description:	The Super Admin can deactivate roles.
Requirement detailed description and constraints:	The Super Admin selects a role. System asks for confirmation before applying deactivation. **Constraints:** • Internet connection required. • Action must be logged.
Business rules applicable to this requirement	• Only the Super Admin can deactivate roles.
Revision date and Revision number:	2026-06-24 Version 3.0
Criticality/Priority:	High

Table 104: 2.12 Deactivate Role – Description

Functional Requirement	Explanation
Requirement number:	2.16
Requirement name (use case name):	Update Role Permission
Requirement short description:	The Super Admin modifies existing role permissions. System validates changes before confirmation.
Requirement detailed description and constraints:	The Super Admin modifies existing role permissions. System validates changes before confirmation. **Constraints:** • Internet connection required.
Business rules applicable to this requirement	• The Super Admin can update role permissions.
Revision date and Revision number:	2026-06-24 Version 3.0
Criticality/Priority:	High

Table 105: 2.13 Update Role Permission – Description

Functional Requirement	Explanation
Requirement number:	2.17
Requirement name (use case name):	View Role Permission
Requirement short description:	The Super Admin can view permissions assigned to roles.
Requirement detailed description and constraints:	The Super Admin can search and view role permissions. No edits allowed. **Constraints:** • Internet connection required.
Business rules applicable to this requirement	• The Super Admin can view role permissions.
Revision date and Revision number:	2026-06-24 Version 3.0
Criticality/Priority:	High

Table 106: 2.14 View Role Permission – Description

2.8.3.3 Inventory Subsystem

Functional Requirement	Explanation
Requirement number:	3.1
Requirement name (use case name):	Create Procurement
Requirement short description:	A user must be able to add an item(s) to the system's inventory.
Requirement detailed description and constraints:	A user must be able to add an item(s) to the system's inventory once the item is received from a vendor. The system will allow the admin to enter details regarding the inventory received. The system will validate the details entered before asking for confirmation. Once confirmed the system will add the inventory to the system. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with the required permissions can receive inventory.
Revision date and Revision number:	2026-07-19 Version 3.0
Criticality/Priority:	Medium

Table 107: 3.1 Receive Inventory Item – Description

Functional Requirement	Explanation
Requirement number:	3.2
Requirement name (use case name):	View Procurement
Requirement short description:	A user must be able to view a procurements detail on the system.
Requirement detailed description and constraints:	This use case describes the event in which the user views a procurement on the system. The system will retrieve and display the procurements detail on the system. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with the required permissions can view procurements.
Revision date and Revision number:	2026-07-19 Version 3.0
Criticality/Priority:	Medium

Table 108: 3.2 View Procurement – Description

Functional Requirement	Explanation
Requirement number:	3.3
Requirement name (use case name):	View Inventory
Requirement short description:	A user must be able to view all inventory on the system.
Requirement detailed description and constraints:	This use case describes the event in which the user views the inventory on the system. The system will retrieve and display all the inventory on the system as well as allowing the user to filter. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with the required permissions can view inventory.
Revision date and Revision number:	2026-07-19 Version 3.0
Criticality/Priority:	Medium

Table 109: 3.2 View Inventory – Description

Functional Requirement	Explanation
Requirement number:	3.4
Requirement name (use case name):	Write-Off Inventory.
Requirement short description:	A user with appropriate permissions must be able to write off an inventory item(s).
Requirement detailed description and constraints:	A user with appropriate permissions must be able to edit the details of an inventory item(s). The system will display all the inventory on the system and allow the user to write-off inventory. The system will ask for confirmation before the inventory items are written off. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with the required permissions can write-off inventory.
Revision date and Revision number:	2026-07-19 Version 3.0
Criticality/Priority:	Medium

Table 110: 3.3 Write-Off Inventory - Description

Functional Requirement	Explanation
Requirement number:	3.5
Requirement name (use case name):	Allocate Inventory
Requirement short description:	A user with the appropriate role must be able to allocate inventory to a project.
Requirement detailed description and constraints:	A user with the appropriate role must be able to allocate inventory to a project. The system will display all the inventory and allow the user to allocate inventory to an active project. The system will validate inputted data and ask for confirmation before allocating inventory. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with the required permissions can allocate inventory.
Revision date and Revision number:	2026-07-19 Version 3.0
Criticality/Priority:	Medium

Table 111: 3.4 Allocate Inventory – Description

Functional Requirement	Explanation
Requirement number:	3.6
Requirement name (use case name):	Create Inventory Item Type
Requirement short description:	A user with the appropriate permissions must be able to create a new inventory item type.
Requirement detailed description and constraints:	A user with the appropriate permissions must be able to create a new inventory item type that will then be selectable when adding inventory items. Information including name, description and security level. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can create an inventory item type.
Revision date and Revision number:	2026-07-19 Version 3.0
Criticality/Priority:	Low

Table 112: 3.5 Create Inventory Item Type – Description

Functional Requirement	Explanation
Requirement number:	3.7
Requirement name (use case name):	View Inventory Item Types
Requirement short description:	A user must be able to view all inventory item types.
Requirement detailed description and constraints:	A user must be able to view all inventory item types. The system will retrieve and display all the inventory item types on the system before displaying them to the user. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can view all inventory item types.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	Low

Table 113:3.7 View Inventory Item Types – Description

Functional Requirement	Explanation
Requirement number:	3.8
Requirement name (use case name):	Update Inventory Item Type
Requirement short description:	A user with the appropriate permissions must be able to update an inventory item type.
Requirement detailed description and constraints:	A user with the appropriate permissions must be update inventory item type that will then be selectable when adding inventory items. Information including name, description and security level can be changed. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can update an inventory item type.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	Low

Table 114: 3.8 Update Inventory Item Type – Description

Functional Requirement	Explanation
Requirement number:	3.9
Requirement name (use case name):	Deactivate Inventory Item Type
Requirement short description:	A user with the appropriate permissions must be able to deactivate an inventory item type.
Requirement detailed description and constraints:	A user with the appropriate permissions must be able to deactivate an inventory item type if the type is no longer needed by the organization due to a change in policy or project scope. Data regarding previous use of the inventory item type will be kept for accurate record keeping, but the type will no longer appear when selecting an inventory item type. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can deactivate an inventory item type.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	Low

Table 115: 3.9 Deactivate Inventory Item Type - Description

Functional Requirement	Explanation
Requirement number:	3.10
Requirement name (use case name):	Reactivate Inventory Item Type
Requirement short description:	A user with the appropriate permissions must be able to reactivate an inventory item type.
Requirement detailed description and constraints:	A user with the appropriate permissions must be able to reactivate an inventory item type. The type will appear again when selecting an inventory item type. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can reactivate an inventory item type.
Revision date and Revision number:	2026-07-19 Version 3.0
Criticality/Priority:	Low

Table 116: 3.10 Reactivate Inventory Item Type - Description

2.8.3.4 Group Subsystem

Functional Requirement	Explanation
Requirement number:	4.1
Requirement name (use case name):	Create Group
Requirement short description:	The system must allow a user with appropriate permissions to create a group for a project.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to create a group that is associated with a specific project. A project can have multiple groups within it and group members can see information related to them as well as communicate with each other. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Each group must be linked with an existing project on the system. • Role based access. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 117: 4.1 Create Group – Description

Functional Requirement	Explanation
Requirement number:	4.2
Requirement name (use case name):	View Groups
Requirement short description:	The system must allow users to view the details of the group they are in for a project.
Requirement detailed description and constraints:	The system must allow users on the system to view a group's details. When a user selects to view a group, the system will display information such as the group name, the project it belongs to and the other members. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • User skill constraint: the use case must be easy to understand and use, as some users will have limited technology abilities. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Users can only view groups they are assigned too. • Users must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 118:4.2 View Group – Description

Functional Requirement	Explanation
Requirement number:	4.3
Requirement name (use case name):	Update Group
Requirement short description:	The system must allow a user with appropriate permissions to update the details of a group.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to update a group's details within a project. The editable details will include the group name and ability to add or remove members. Once the user clicks save, the system will validate the updated data and save the changes. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 119: 4.3 Update Group – Description

Functional Requirement	Explanation
Requirement number:	4.4
Requirement name (use case name):	Deactivate Group
Requirement short description:	The system must allow a user with appropriate permissions to deactivate a group.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to deactivate a group within a project when it is no longer used or required. The system will confirm the deactivation and upon confirmation, the group will be deactivated. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 120: 4.4 Deactivate Group - Description

Functional Requirement	Explanation
Requirement number:	4.5
Requirement name (use case name):	Create Skill
Requirement short description:	The system must allow a user with appropriate permissions to create a new skill.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to create new skills that can be assigned to volunteers and staff members. These skills can be used to help assign volunteers or staff members to appropriate teams. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	Medium

Table 121: 4.5 Create Skill - Description

Functional Requirement	Explanation
Requirement number:	4.6
Requirement name (use case name):	View Skills
Requirement short description:	The system must allow users to view skills stored on the system.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to view the skills that have been created on the system. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • User skill constraint: the use case must be easy to understand and use, as some users will have limited technology abilities. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Users must be logged into the system. • Skills must be displayed accurately.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	Medium

Table 122: 4.6 View Skills - Description

Functional Requirement	Explanation
Requirement number:	4.7
Requirement name (use case name):	View Skill
Requirement short description:	The system must allow users to view a skill stored on the system.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to view a skill that is on the system. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • User skill constraint: the use case must be easy to understand and use, as some users will have limited technology abilities. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Users must be logged into the system. • Skills must be displayed accurately.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	Medium

Table 123: View Skill - Description

Functional Requirement	Explanation
Requirement number:	4.8
Requirement name (use case name):	Update Skill
Requirement short description:	The system must allow a user with appropriate permissions to update skills stored on the system.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to update the skills on the system such as correcting the name or spelling. Once the user clicks save, the system will validate the updated data and save the changes. This will ensure that skill information remains accurate. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	Medium

Table 124: 4.7 Update Skill - Description

Functional Requirement	Explanation
Requirement number:	4.9
Requirement name (use case name):	Deactivate Skill
Requirement short description:	The system must allow a user with appropriate permissions to deactivate a skill stored on the system.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to update the skills on the system such as correcting the name or spelling. Once the user clicks save, the system will validate the updated data and save the changes. This will ensure that skill information remains accurate. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	Medium

Table 125: 4.8 Deactivate Skill - Description

2.8.3.5 Warehouse Subsystem

Functional Requirement	Explanation
Requirement number:	5.1
Requirement name (use case name):	Create Warehouse
Requirement short description:	The Super Admin can create new warehouses.
Requirement detailed description and constraints:	The Super Admin can input warehouse name, location, and capacity. System validates details before confirmation. **Constraints:** • Internet connection required. • Location field cannot be empty. • Capacity must be a positive number.
Business rules applicable to this requirement	• Only users with warehouse admin permissions can create warehouses.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 126 :5.1 Create Warehouse - Description

Functional Requirement	Explanation
Requirement number:	5.2
Requirement name (use case name):	View Warehouse
Requirement short description:	The Super Admin can view warehouse details.
Requirement detailed description and constraints:	The Super Admin can search and view warehouses. No edits allowed in view mode. **Constraints:** • Internet connection required. • Access limited to users with warehouse admin privileges.
Business rules applicable to this requirement	• Only warehouse admin users can view warehouses.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 127: 5.2 View Warehouse - Description

Functional Requirement	Explanation
Requirement number:	5.3
Requirement name (use case name):	Update Warehouse
Requirement short description:	The Super Admin can update warehouse details.
Requirement detailed description and constraints:	The Super Admin modifies warehouse information. System validates changes before confirmation. **Constraints:** • Internet connection required. • Capacity updates must be positive numbers. • Changes must be logged.
Business rules applicable to this requirement	• Only warehouse admin users can update warehouses.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 128: 5.3 Update Warehouse - Description

Functional Requirement	Explanation
Requirement number:	5.4
Requirement name (use case name):	Reactivate Warehouse
Requirement short description:	The Super Admin can reactivate deactivated warehouses.
Requirement detailed description and constraints:	The Super Admin selects a deactivated warehouse. System asks for confirmation before reactivation. **Constraints:** • Internet connection required. • Action must be logged.
Business rules applicable to this requirement	• Only the warehouse admin user can reactivate warehouses.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 129 5.4: Reactivate Warehouse - Description

Functional Requirement	Explanation
Requirement number:	5.5
Requirement name (use case name):	Deactivate Warehouse
Requirement short description:	The Super Admin can deactivate warehouses.
Requirement detailed description and constraints:	The Super Admin selects a warehouse. System asks for confirmation before applying deactivation. **Constraints:** • Internet connection required. • Action must be logged.
Business rules applicable to this requirement	• Only warehouse admin users can deactivate warehouses.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 130: 5.5 Deactivate Warehouse - Description

Functional Requirement	Explanation
Requirement number:	5.6
Requirement name (use case name):	Inventory Stock Take.
Requirement short description:	A user with appropriate permissions must be able to perform a stock take.
Requirement detailed description and constraints:	A user with appropriate permissions must be able to perform a stock take. The system will allow the user to select a warehouse and view its inventory. The user will perform a stock take and enter the stock count. The system will determine the difference and capture the difference to adjust the inventory. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with the required permissions can receive perform a stock take.
Revision date and Revision number:	2026-03-29 Version 2.0
Criticality/Priority:	High

Table 131: Inventory Stock Take - Description

2.8.3.6 Vendor Subsystem

Functional Requirement	Explanation
Requirement number:	6.1
Requirement name (use case name):	Create Vendor
Requirement short description:	A user must be able to add a vendor to the system.
Requirement detailed description and constraints:	A user must be able to add a vendor to the system by adding information including vendor name, type, email, phone, contact person, credit facility (if applicable), address, rating and photo. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can add a vendor.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 132: 6.1 Create Vendor - Description

Functional Requirement	Explanation
Requirement number:	6.2
Requirement name (use case name):	View Vendor
Requirement short description:	A user must be able to view any given vendor.
Requirement detailed description and constraints:	A user must be able to view any given vendor. The system will then retrieve and display all relevant information regarding the vendor. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can view a vendor.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 133 6.2: View Vendor - Description

Functional Requirement	Explanation
Requirement number:	6.3
Requirement name (use case name):	View Vendors
Requirement short description:	A user must be able to view all vendors.
Requirement detailed description and constraints:	A user must be able to view all vendors on the system. The system will retrieve and display all vendors and their information for the user. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can view all vendor.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 134: 6.3 View Vendors - Description

Functional Requirement	Explanation
Requirement number:	6.4
Requirement name (use case name):	Edit Vendor
Requirement short description:	A user must be able to edit vendor information
Requirement detailed description and constraints:	A user must be able to edit a vendor on the system by changing information including vendor name, type, email, phone, contact person, credit facility (if applicable), address, rating and photo. **Constraints:** • The user requires an Internet connection to access this feature. • The vendor must already exist on the system.
Business rules applicable to this requirement	• Only a user with appropriate permissions can edit a vendor.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 135: 6.4 Edit Vendor - Description

Functional Requirement	Explanation
Requirement number:	6.5
Requirement name (use case name):	Reactivate Vendor
Requirement short description:	A user must be able to reactivate a vendor.
Requirement detailed description and constraints:	A user must be able to reactivate a vendor on the system. The system will display the vendor and allow the user to reactivate the vendor. The system will ask the user to confirm the reactivation before the vendor is reactivated **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can reactivate a vendor.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 136: 6.5 Reactivate Vendor - Description

Functional Requirement	Explanation
Requirement number:	6.6
Requirement name (use case name):	Deactivate Vendor
Requirement short description:	A user must be able to deactivate any given vendor.
Requirement detailed description and constraints:	A user must be able to deactivate any given vendor in the case that the vendor is no longer in use, or the vendor no longer exists. The system will display the vendor and allow the user to deactivate the vendor. The system will ask the user to confirm the deactivation before the vendor is deactivated. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can deactivate a vendor.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 137: 6.6 Deactivate Vendor - Description

Functional Requirement	Explanation
Requirement number:	6.7
Requirement name (use case name):	Create Vendor Type
Requirement short description:	A user with the appropriate permissions must be able to create a type that vendors can be categorised under.
Requirement detailed description and constraints:	A user with the appropriate permissions must be able to create a type that vendors can be categorised under. These types will be used when updating and creating vendors. It also ensures that vendors can be easily filtered. The details will include type name, description and status. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Vendor management capabilities. • Use of Preferred vendors. • Role based access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 138: 6.7 Create Vendor Type - Description

Functional Requirement	Explanation
Requirement number:	6.8
Requirement name (use case name):	Update Vendor Type
Requirement short description:	A user with the appropriate permissions must be able to update a vendor type.
Requirement detailed description and constraints:	A user with the appropriate permissions must be able to update a vendor type. This includes changes to the type name and description. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature. • Vendor type must already exist on the system.
Business rules applicable to this requirement	• Vendor management capabilities. • Use of Preferred vendors. • Role based access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 139: 6.8 Update Vendor Type - Description

Functional Requirement	Explanation
Requirement number:	6.9
Requirement name (use case name):	View Vendor Types
Requirement short description:	A user with the appropriate permissions must be able to view the vendor types.
Requirement detailed description and constraints:	A user must be able to view the vendor types and then view all the available information regarding the vendor types. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Vendor management capabilities. • Use of Preferred vendors. • Role based access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 140: 6.8 View Vendor Types - Description

Functional Requirement	Explanation
Requirement number:	6.10
Requirement name (use case name):	Reactivate Vendor Type
Requirement short description:	A user with the appropriate permissions must be able to view the vendor types.
Requirement detailed description and constraints:	A user with the appropriate permissions must be able to reactivate a vendor type when it is required again by the organization due to a change in policy or project scope. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature. • Vendor type must already exist on the system.
Business rules applicable to this requirement	• Vendor management capabilities. • Use of Preferred vendors. • Role based access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 141: 6.10 Reactivate Vendor Type - Description

Functional Requirement	Explanation
Requirement number:	6.11
Requirement name (use case name):	Deactivate Vendor Type
Requirement short description:	A user with the appropriate permissions must be able to deactivate a vendor type.
Requirement detailed description and constraints:	A user with the appropriate permissions must be able to deactivate a vendor type if the type is no longer needed by the organization due to a change in policy or project scope. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature. • Vendor type must already exist on the system.
Business rules applicable to this requirement	• Vendor management capabilities. • Use of Preferred vendors. • Role based access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 142: 6.11 Deactivate Vendor Type - Description

2.8.3.7 Project Management Subsystem

Functional Requirement	Explanation
Requirement number:	7.1
Requirement name (use case name):	Create Project
Requirement short description:	A Super-Admin creates a new project representing an NGO goal by inputting project details and team assignments.
Requirement detailed description and constraints:	A Super-Admin will create a new project that will represent a new NGO goal. The system will open a form where the Super-Admin will input the project: name, dates, staff, teams, milestones and deadlines to help plan and organise the project. The system will capture all the information and alert all the relevant staff and create the schedule. **Constraints:** • The staff have all been previously added to the system. • The staff roles have all been previously added to the system. • The project type has been added to the system previously. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 143: 7.1 Create Project - Description

Functional Requirement	Explanation
Requirement number:	7.2
Requirement name (use case name):	View Project
Requirement short description:	A user clicks on a specific project to see all its details.
Requirement detailed description and constraints:	A user clicks on a specific project to see all its details, like the name, staff, dates, project type, milestones, and deadline. They see this in the project's nav bar option. The system retrieves the details and displays them in sections relative to the user's role. **Constraints:** • The user must already have a pre-existing account on the system. The user is logged in. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 144: 7.2 View Project - Description

Functional Requirement	Explanation
Requirement number:	7.3
Requirement name (use case name):	Update Project
Requirement short description:	A Super-Admin will edit an existing project already saved in the system.
Requirement detailed description and constraints:	A Super-Admin will edit an existing project already saved in the system. The system will open the project's form where the Super-Admin will be able to edit the project: name, dates, staff, teams, milestones and deadlines. The system will capture all the updated information and alert all the relevant staff. **Constraints:** • The project has already been created. • The staff have all been previously added to the system. • The staff roles have all been previously added to the system. • The project type has been added to the system previously. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 145: 7.3 Update Project - Description

Functional Requirement	Explanation
Requirement number:	7.4
Requirement name (use case name):	Deactivate Project
Requirement short description:	A Super Admin can deactivate projects after they are completed or no longer support goals.
Requirement detailed description and constraints:	A Super Admin can deactivate projects after they are completed or if the project no longer supports the NPO's organisational goals. The system will load the form and ask the user for confirmation before deactivating the project and notifying the admin of the status change. **Constraints:** • The project is already created and saved on the system. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 146: 7.4 Deactivate Project - Description

Functional Requirement	Explanation
Requirement number:	7.5
Requirement name (use case name):	Create Project Type
Requirement short description:	The Super Admin creates a new category for projects.
Requirement detailed description and constraints:	The Super Admin would like to create a new category for projects that will represent a type of project. The system will display a form where the admin can type the name of the new project type, which the system then verifies and saves. **Constraints:** • The user must already have a pre-existing account on the system. • The user has Super Admin privileges. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 147: 7.5 Create Project Type - Description

Functional Requirement	Explanation
Requirement number:	7.6
Requirement name (use case name):	View Project Type
Requirement short description:	The Super Admin can see, edit, or deactivate project types.
Requirement detailed description and constraints:	The Super Admin can click "View Project types" to be able to see, edit, or deactivate project types to better categorise the projects being completed. The system retrieves recorded project types and displays them in a table for the Super Admin to view. **Constraints:** • The user must already have a pre-existing account on the system. • The user must have Super Admin privileges. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 148: 7.6 View Project Type - Description

Functional Requirement	Explanation
Requirement number:	7.7
Requirement name (use case name):	Update Project Type
Requirement short description:	A Super Admin wants to change the name of a project type.
Requirement detailed description and constraints:	A Super Admin wants to change the name of a project type so users can better understand what those types of projects represent. The system retrieves the recorded project type and displays the name as an editable field for the admin to update. **Constraints:** • The project type has already been added and captured onto the system. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 149: 7.7 Update Project Type - Description

Functional Requirement	Explanation
Requirement number:	7.8
Requirement name (use case name):	Deactivate Project Type
Requirement short description:	The Super Admin deactivates a project type no longer being used.
Requirement detailed description and constraints:	The Super Admin deactivates a project type, and it will remove it from the displayed table of project types. This would be because a project type is no longer being used. The system loads the project type name for confirmation before deactivating it. **Constraints:** • The project type has already been captured on the system. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 150: 7.8 Deactivate Project Type - Description

Functional Requirement	Explanation
Requirement number:	7.9
Requirement name (use case name):	Add Indemnity Form
Requirement short description:	The Super Admin can add a new Indemnity form to projects.
Requirement detailed description and constraints:	The Super Admin can add a new Indemnity form that can be added to projects to make form completions and capturing easier. The system shows fields to place a name for the form and a field to upload a pdf file or paste text. **Constraints:** • The admin is signed in, and the form hasn't been added yet. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 151: 7.9 Add Indemnity Form - Description

Functional Requirement	Explanation
Requirement number:	7.10
Requirement name (use case name):	Update Indemnity Form
Requirement short description:	The Super Admin can update an existing Indemnity form.
Requirement detailed description and constraints:	The Super Admin can update an existing Indemnity form that can be added to projects to make form completions and capturing easier. The system displays editable fields where the admin can change the name, text, or pdf file of the form. **Constraints:** • The admin is signed in, and the form hasn't been added yet. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 152: 7.10 Update Indemnity Form - Description

Functional Requirement	Explanation
Requirement number:	7.11
Requirement name (use case name):	View Indemnity Form
Requirement short description:	The Super Admin can see, edit, add or deactivate Indemnity Forms.
Requirement detailed description and constraints:	The Super Admin can click "View Indemnity Form" to be able to see, edit, add or deactivate Indemnity Forms to better help the projects. The system retrieves the recorded forms and displays them in a table for the admin to manage. **Constraints:** • The user must already have a pre-existing account on the system. • The user must have Super Admin privileges. • Must be completed before project day. • Must be user-friendly and easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 153: 7.11 View Indemnity Form - Description

Functional Requirement	Explanation
Requirement number:	7.12
Requirement name (use case name):	Remove Indemnity Form
Requirement short description:	The Super Admin can remove an existing indemnity form no longer being used.
Requirement detailed description and constraints:	The Super Admin can remove an existing indemnity form that is no longer being used to reduce clutter. The system loads the form content for deactivation confirmation and notifies the user once the form is removed from the table. **Constraints:** • The admin is signed in, and the form hasn't been added yet. • Must be completed before project day. • Must be user-friendly and easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 154: 7.12 Remove Indemnity Form - Description

Functional Requirement	Explanation
Requirement number:	7.13
Requirement name (use case name):	Capture Indemnity Form
Requirement short description:	A Super Admin can attach an indemnity form to a project.
Requirement detailed description and constraints:	A Super Admin can attach an indemnity form to a project which will make it so that anyone who has been added to that project must complete the compulsory indemnity form to view the project details. The system captures the selection and requires signatures from staff. **Constraints:** • The Super Admin must already have a pre-existing account on the system. • Must be completed before project day. • Must be user-friendly and easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 155: 7.13 Capture Indemnity Form - Description

Functional Requirement	Explanation
Requirement number:	7.14
Requirement name (use case name):	Capture Project Feedback
Requirement short description:	A Super Admin can attach a project feedback form to a project.
Requirement detailed description and constraints:	A Super Admin can attach a project feedback form to a project which will make it so that anyone who has been added to that project will be sent a form 1 week after the project commences asking them for a rating and comments. The system captures the selection for all involved staff. **Constraints:** • The project assigned to the feedback form must have commenced. • Must be completed before project day. • Must be user-friendly and easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 156: 7.14 Capture Project Feedback - Description

Functional Requirement	Explanation
Requirement number:	7.15
Requirement name (use case name):	View Project Feedback
Requirement short description:	A Super Admin can view project feedback form ratings and comments.
Requirement detailed description and constraints:	A Super Admin can view all the past projects and see the feedback forms ratings and comments for each project. The system retrieves the records for feedback and displays it neatly for the admin to read. **Constraints:** • The projects have been saved on the system and not been deactivated. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 157: 7.15 View Project Feedback - Description

Functional Requirement	Explanation
Requirement number:	7.16
Requirement name (use case name):	Create Impact Feedback Form
Requirement short description:	The Super Admin can add a new impact form for community members.
Requirement detailed description and constraints:	The Super Admin can add a new impact form that can be added to projects to make form completions and capturing easier for community members. The system shows fields to name the form and buttons to add and define question types. **Constraints:** • The admin creating the form has permissions to create impact forms already. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 158: 7.16 Create Impact Feedback Form - Description

Functional Requirement	Explanation
Requirement number:	7.17
Requirement name (use case name):	View Impact Forms
Requirement short description:	A Super Admin can view project impact form questions and answers.
Requirement detailed description and constraints:	A Super Admin can view all the past projects and see the impact forms questions and answers and see what project they are assigned to. The system retrieves records for impact data and displays it neatly for the admin. **Constraints:** • The projects have been saved on the system and not been deactivated. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 159: 7.17 View Impact Forms - Description

Functional Requirement	Explanation
Requirement number:	7.18
Requirement name (use case name):	Update Impact Form
Requirement short description:	The Super Admin can update an existing impact form.
Requirement detailed description and constraints:	The Super Admin can update an existing impact form that can be added to projects to make form completions and capturing easier. The system shows fields for the form name and the questions currently on the form as editable content. **Constraints:** • The admin is signed in, and the form hasn't been added yet. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 160: 7.18 Update Impact Form - Description

Functional Requirement	Explanation
Requirement number:	7.19
Requirement name (use case name):	Deactivate Impact Form
Requirement short description:	The admin can deactivate an impact form from the list of all forms.
Requirement detailed description and constraints:	The admin can deactivate an impact form from the list of all the forms being displayed; this can remove unused forms and clear clutter from the system. The system loads the form name for deactivation confirmation before removing it from view. **Constraints:** • The impact form has already been created and is displayed on the list. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 161: 7.19 Deactivate Impact Form - Description

Functional Requirement	Explanation
Requirement number:	7.20
Requirement name (use case name):	Capture Impact Form
Requirement short description:	The Super Admin can include an impact form and send it to a community partner.
Requirement detailed description and constraints:	When creating or editing a project the Super Admin can check the "Include impact form" button and choose from the list or create a new one. The system will capture an email address or phone number to send the form to the community partner. **Constraints:** • The impact form has already been captured in the system. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 162: 7.20 Capture Impact Form - Description

Functional Requirement	Explanation
Requirement number:	7.21
Requirement name (use case name):	View Impact Feedback
Requirement short description:	A Super Admin can view all of the project's impact results.
Requirement detailed description and constraints:	A Super Admin can view all of the project's impact results and see the impact form questions and answers, the assigned project, and which community partner answered. The system retrieves and displays these records neatly. **Constraints:** • The impact forms have been saved on the system and not been deactivated. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 163: 7.21 View Impact Feedback - Description

Functional Requirement	Explanation
Requirement number:	7.22
Requirement name (use case name):	Add Photo
Requirement short description:	A user can upload multiple photos to a project's album.
Requirement detailed description and constraints:	A user can click on a project to see all photos from that project, then click the add photos button to upload multiple photos to that album. The system loads a page where images can be dragged, dropped, or uploaded from the device. **Constraints:** • The project's photo group has already been added, and the user has access to that project. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 164: 7.22 Add Photo - Description

Functional Requirement	Explanation
Requirement number:	7.23
Requirement name (use case name):	View Photo
Requirement short description:	A user can see all photos from a specific project.
Requirement detailed description and constraints:	A user can click on a project to see all photos from that project; they can then click the project to see all photos in that album. All projects are displayed in a table with their names and a representative project photo. **Constraints:** • All the project's photo group has already been added, and the user has access to those projects. • Must be completed before project day. • Must be user-friendly and easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 165: 7.23 View Photo - Description

Functional Requirement	Explanation
Requirement number:	7.24
Requirement name (use case name):	Delete Photo
Requirement short description:	A user can remove photos from a project's image folder.
Requirement detailed description and constraints:	A user can click on a project to see all photos and then click the delete photos button; the system then displays an "X" button on photos they uploaded to remove them. The system captures the selection and updates the album. **Constraints:** • The project's photo group has already been added, and the user has access to that project. • The user is deleting photo's they uploaded or are a Super Admin. • Must be completed before project day. • Must be easy to use.
Business rules applicable to this requirement	• Secure Access • Role Based Access
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 166: 7.24 Delete Photo - Description

2.8.3.8 K9 Subsystem

Functional Requirement	Explanation
Requirement number:	8.1
Requirement name (use case name):	Add Dog
Requirement short description:	The system must allow a user with appropriate permissions to add a dog to the system.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to capture and store the details of new dogs used in K9 operations. The details will include the dog's name, breed, age, gender, role, status, handler and parents. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • Each dog must be assigned a handler when applicable. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 167: 8.1 Add Dog - Description

Functional Requirement	Explanation
Requirement number:	8.2
Requirement name (use case name):	View Dog
Requirement short description:	The system must allow a user with appropriate permissions to view a dog on the system.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to view the details of each dog used in K9 operations. This will help users to monitor and manage the K9 unit. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • Dog information must be stored and displayed accurately. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 168: 8.2 View Dog - Description

Functional Requirement	Explanation
Requirement number:	8.3
Requirement name (use case name):	View Dogs
Requirement short description:	The system must allow a user with appropriate permissions to view the dogs on the system.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to view the dogs used in K9 operations. This will help users to monitor and manage the K9 unit. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 169: 8.3 View All Dogs- Description

Functional Requirement	Explanation
Requirement number:	8.4
Requirement name (use case name):	Update Dog
Requirement short description:	The system must allow a user with appropriate permissions to update a dog on the system.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to update the details of an existing dog used in the K9 operations. The editable details will include the dog's name, breed, age, gender, role, and parents. Once the user clicks save, the system will validate the updated data and save the changes. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 170: 8.4 Update Dog - Description

Functional Requirement	Explanation
Requirement number:	8.5
Requirement name (use case name):	Deactivate Dog
Requirement short description:	The system must allow a user with appropriate permissions to deactivate a dog on the system.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to deactivate a dog on the system. This could occur when the dog becomes retired and no longer participating in anti-poaching activities. The system will confirm to deactivate the dog and upon confirmation, the dog will be deactivated. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 171: 8.5 Deactivate Dog - Description

Functional Requirement	Explanation
Requirement number:	8.6
Requirement name (use case name):	Assign User to Dog
Requirement short description:	The system must allow a user with appropriate permissions to assign a user to a dog.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to assign a registered handler to a dog on the system. This will help track who is responsible for the dog and working with the dog during patrols. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • The handler must be registered on the system. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 172: 8.6 Assign User to Dog - Description

Functional Requirement	Explanation
Requirement number:	8.7
Requirement name (use case name):	View User-Dog Assignment
Requirement short description:	The system must allow a user with appropriate permissions to view the assignment of users to dogs.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to view which handlers are assigned to which dogs on the system. This will track responsibilities, ensure accountability, and manage K9 operations effectively. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • All assignments must be stored and displayed accurately. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 173: 8.7 View User-Dog Assignment – Description

Functional Requirement	Explanation
Requirement number:	8.8
Requirement name (use case name):	Update User-Dog Assignment
Requirement short description:	The system must allow a user with appropriate permissions to update the assignment of users to dogs.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to update a dog's handler from one handler to another if needed. Once the user clicks save, the system will validate the updated data and save the changes. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • The new handler must be registered on the system. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 174: 8.8 Update User-Dog Assignment - Description

Functional Requirement	Explanation
Requirement number:	8.9
Requirement name (use case name):	Reactivate User-Dog Assignment
Requirement short description:	The system must allow a user with appropriate permissions to reactivate the assignment of users to dogs.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to reactivate a previous assignment between a handler and a dog. The system will confirm the reactivation and upon confirmation, the user-dog assignment will be reactivated. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 175: 8.9 Reactivate User-Dog Assignment - Description

Functional Requirement	Explanation
Requirement number:	8.10
Requirement name (use case name):	Deactivate User-Dog Assignment
Requirement short description:	The system must allow a user with appropriate permissions to deactivate the assignment of users to dogs.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to deactivate an existing assignment between a handler and a dog. The system will confirm the deactivation and upon confirmation, the user-dog assignment will be deactivated. This will remove the handler from being responsible for the dog. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 176: 8.10 Deactivate User-Dog Assignment - Description

Functional Requirement	Explanation
Requirement number:	8.11
Requirement name (use case name):	Add K9 Role
Requirement short description:	The system must allow a user with appropriate permissions to add a K9 role.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to add a new K9 role. These roles will define the responsibilities of each dog including tracking or attacking. These roles help to assign dogs to activities that match their abilities. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 177: 8.11 Add K9 Role - Description

Functional Requirement	Explanation
Requirement number:	8.12
Requirement name (use case name):	Update K9 Role
Requirement short description:	The system must allow a user with appropriate permissions to update a K9 role.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to update an existing K9's role on the system. The editable details will include the name or spelling of the role. Once the user clicks save, the system will validate the updated data and save the changes. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 178: 8.12 Update K9 Role - Description

Functional Requirement	Explanation
Requirement number:	8.13
Requirement name (use case name):	View K9 Roles
Requirement short description:	The system must allow a user with appropriate permissions to view K9 roles.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to view a list of K9 roles on the system. These roles describe the responsibilities of each dog such as a tracking dog or attacking dog. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • K9 roles must be displayed accurately. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 179: 8.13 View K9 Role - Description

Functional Requirement	Explanation
Requirement number:	8.14
Requirement name (use case name):	Reactivate K9 Role
Requirement short description:	The system must allow a user with appropriate permissions to reactivate a K9 role.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to reactivate a K9 role on the system when it is required again. The system will confirm the reactivation and upon confirmation, the K9 role will be reactivated. Constraints: • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 180: 8.14 Reactivate K9 Role - Description

Functional Requirement	Explanation
Requirement number:	8.15
Requirement name (use case name):	Deactivate K9 Role
Requirement short description:	The system must allow a user with appropriate permissions to remove a K9 role.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to deactivate an existing K9 role from the system when it is no longer required. The system will confirm the deactivation and upon confirmation, the K9 role will be deactivated. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 181: 8.15 Deactivate K9 Role – Description

Functional Requirement	Explanation
Requirement number:	8.16
Requirement name (use case name):	View K9 Ancestry
Requirement short description:	The system must allow a user with appropriate permissions to view the ancestry of a K9.
Requirement detailed description and constraints:	The system must allow a user with appropriate permissions to view a dog's family tree showing the parentage and lineage of the dog. Constraints: • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Role based access. • User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 182: 8.16 View K9 Ancestry - Description

2.8.3.9 Rhino Subsystem

Functional Requirement	Explanation
Requirement number:	9.1
Requirement name (use case name):	Add Rhino
Requirement short description:	The system allows users with appropriate permissions to add a rhino.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to store information about rhinos. The user will enter the required details including name, species, weight, date of birth, gender, horn status, status, last seen, identification notes, approximate location (kept vague for security reasons) and photo. **Constraints:** • Fields must pass input validation before being captured. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can add a rhino. • User must be logged in.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	Medium

Table 183: 9.1 Add Rhino - Description

Functional Requirement	Explanation
Requirement number:	9.2
Requirement name (use case name):	View Rhino
Requirement short description:	The system allows users with appropriate permissions to view a rhino.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to access and view information about a rhino stored on the system. **Constraints:** • The rhino must already exist on the system. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can view a rhino's details. • User must be logged in.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	Medium

Table 184:9.2 View Rhino - Description

Functional Requirement	Explanation
Requirement number:	9.3
Requirement name (use case name):	View Rhinos
Requirement short description:	The system allows users with appropriate permissions to view all rhinos.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to access and view all rhinos stored on the system. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can view all rhinos. • User must be logged in.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	Medium

Table 185: 9.3 View Rhinos - Description

Functional Requirement	Explanation
Requirement number:	9.4
Requirement name (use case name):	Update Rhino
Requirement short description:	The system allows users with appropriate permissions to update rhino details.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to update rhino details that are stored on the system. The system will display the rhino's current details and allow the user to change them. This includes name, species, weight, date of birth, gender, horn status, status, last seen, identification notes, approximate location (kept vague for security reasons) and photo. **Constraints:** • The rhino must already exist on the system. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can update a rhino. • User must be logged in.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	Medium

Table 186: 9.4 Update Rhino - Description

Functional Requirement	Explanation
Requirement number:	9.5
Requirement name (use case name):	Deactivate Rhino
Requirement short description:	The system allows users with appropriate permissions to remove a rhino; historic data will be kept.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to remove a rhino from the system. The user will get a confirmation prompt before the rhino is removed from the system. Historic rhino data will be kept. **Constraints:** • The rhino must already exist on the system. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can deactivate a rhino. • User must be logged in.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	Medium

Table 187: 9.5 9.5 Deactivate Rhino - Description

2.8.3.10 Elephant Subsystem

Functional Requirement	Explanation
Requirement number:	10.1
Requirement name (use case name):	Add Elephant
Requirement short description:	The system allows users with appropriate permissions to add an elephant.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to store information about elephants. The user will enter the required details including name, species, weight, date of birth, gender, tusk status, status, last seen, identification notes, approximate location (kept vague for security reasons) and photo. **Constraints:** • Fields must pass input validation before being captured. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can add an elephant. • User must be logged in.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 188: 10.1 Add Elephant - Description

Functional Requirement	Explanation
Requirement number:	10.2
Requirement name (use case name):	View Elephant
Requirement short description:	The system allows users with appropriate permissions to view an elephant's details.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to access and view information about an elephant stored on the system. **Constraints:** • The elephant must already exist on the system. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can view an elephant's details. • User must be logged in.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 189: 10.2 View Elephant - Description

Functional Requirement	Explanation
Requirement number:	10.3
Requirement name (use case name):	View Elephants
Requirement short description:	The system allows users with appropriate permissions to view all elephants.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to access and view all elephants stored on the system. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can view all elephants. • User must be logged in.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 190: 10.3 View Elephants - Description

Functional Requirement	Explanation
Requirement number:	10.4
Requirement name (use case name):	Update Elephant
Requirement short description:	The system allows users with appropriate permissions to update an elephant's details.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to update elephant details that are stored on the system. The system will display the elephant's current details and allow the user to change them. This includes the name, species, weight, date of birth, gender, tusk status, status, last seen, identification notes, approximate location (kept vague for security reasons) and photo. **Constraints:** • The elephant must already exist on the system. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can update an elephant. • User must be logged in.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 191: 10.4 Update Elephant - Description

Functional Requirement	Explanation
Requirement number:	10.5
Requirement name (use case name):	Deactivate Elephant
Requirement short description:	The system allows users with appropriate permissions to deactivate an elephant; historic data will be kept.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to remove an elephant or elephant from the system. The user will get a confirmation prompt before the rhino or elephant is removed from the system. Historic data will be kept. **Constraints:** • Only users with appropriate permissions can remove an elephant. • The elephant must already exist on the system. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Only a user with appropriate permissions can remove an elephant. • User must be logged in.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 192: 10.5 Deactivate Elephant - Description

2.8.3.11 Messages Subsystem

Functional Requirement	Explanation
Requirement number:	11.1
Requirement name (use case name):	Create Message
Requirement short description:	The system must allow a user to send a message on a group.
Requirement detailed description and constraints:	The system must allow a user to send a message on a group to which they are assigned. The user will type out the message they wish to send and then send the message to a group. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 193: 11.1 Create Message – Description

Functional Requirement	Explanation
Requirement number:	11.2
Requirement name (use case name):	View Messages
Requirement short description:	The system must allow a user to view the messages on a group.
Requirement detailed description and constraints:	The system must allow a user to view a message on a group to which they are assigned. The user selects a group and will be able to see the messages on the group. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• User must be logged into the system.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 194: 11.2 View Message - Description

2.8.3.12 Report Subsystem

Functional Requirement	Explanation
Requirement number:	12.1
Requirement name (use case name):	Generate Custom Report
Requirement short description:	The system allows a user with appropriate permissions to generate a custom report.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to generate a custom report. The user can select fields, and the system will generate a report using those fields. **Constraints:** • Only users with appropriate permissions can generate custom reports. • Users must be logged in.
Business rules applicable to this requirement	• Role-based access.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 195: 12.1 Generate Custom Report - Description

Functional Requirement	Explanation
Requirement number:	12.2
Requirement name (use case name):	View Custom Report
Requirement short description:	The system allows a user with appropriate permissions to view a custom report.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to view a custom report. The user can select and view a custom report that has been created. **Constraints:** • Only users with appropriate permissions can view custom reports. • Report must already exist. • Users must be logged in.
Business rules applicable to this requirement	• Role-based access.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 196: 12.2 View Custom Report - Description

Functional Requirement	Explanation
Requirement number:	12.3
Requirement name (use case name):	Update Custom Report
Requirement short description:	The system allows a user with appropriate permissions to update a custom report.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to update a custom report. The system will display the report's current fields and allow the user to change them. **Constraints:** • Only users with appropriate permissions can update custom reports. • Users must be logged in.
Business rules applicable to this requirement	• Role-based access.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 197: 12.3 Update Custom Report - Description

Functional Requirement	Explanation
Requirement number:	12.4
Requirement name (use case name):	Remove Custom Report
Requirement short description:	The system allows a user with appropriate permissions to remove a custom report.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to remove a custom report and its configuration from the database. **Constraints:** • Only users with appropriate permissions can remove custom reports. • Users must be logged in.
Business rules applicable to this requirement	• Role-based access.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 198: 12.4 Remove Custom Report - Description

Functional Requirement	Explanation
Requirement number:	12.5
Requirement name (use case name):	View Bike Impact Report
Requirement short description:	The system allows a user with appropriate permissions to view a bike impact report.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to view a bike impact report. The user Selects a date range, and the pre-structured report is generated for them to view. **Constraints:** • Only users with appropriate permissions can view bike impact reports. • Users must be logged in.
Business rules applicable to this requirement	• Role-based access.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 199: 12.5 View Bike Impact Report - Description

Functional Requirement	Explanation
Requirement number:	12.6
Requirement name (use case name):	View Budget Report
Requirement short description:	The system allows a user with appropriate permissions to view a budget report.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to view a budget report. The user Selects a date range, and the pre-structured report is generated for them to view. **Constraints:** • Only users with appropriate permissions can view budget reports. • Users must be logged in.
Business rules applicable to this requirement	• Role-based access.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 200: 12.6 View Budget Report – Description

Functional Requirement	Explanation
Requirement number:	12.7
Requirement name (use case name):	View Rhino Report
Requirement short description:	The system allows a user with appropriate permissions to view a rhino report.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to view a rhino report. The user Selects a date range, and the pre-structured report is generated for them to view. **Constraints:** • Only users with appropriate permissions can view rhino reports. • Users must be logged in.
Business rules applicable to this requirement	• Role-based access.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 201: 12.7 View Rhino Report - Description

Functional Requirement	Explanation
Requirement number:	12.8
Requirement name (use case name):	View Elephant Report
Requirement short description:	The system allows a user with appropriate permissions to view an elephant report.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to view an elephant report. The user Selects a date range, and the pre-structured report is generated for them to view. **Constraints:** • Only users with appropriate permissions can view elephant reports. • Users must be logged in.
Business rules applicable to this requirement	• Role-based access.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 202: 12.8 View Elephant Report - Description

Functional Requirement	Explanation
Requirement number:	12.9
Requirement name (use case name):	View K9 Report
Requirement short description:	The system allows a user with appropriate permissions to view a K9 report.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions to view a K9 report. The user Selects a date range, and the pre-structured report is generated for them to view. **Constraints:** • Only users with appropriate permissions can view K9 reports. • Users must be logged in.
Business rules applicable to this requirement	• Role-based access.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 203: 12.9 View K9 Report - Description

Functional Requirement	Explanation
Requirement number:	12.10
Requirement name (use case name):	View Machine Learning Dashboard
Requirement short description:	A user with the appropriate role should be able to view all current predictions.
Requirement detailed description and constraints:	A user with the appropriate role should be able to view all current predictions. Predictions will appear dynamically as users complete tasks and will also be viewable in the Machine Learning Dashboard. Once a user opens the dashboard, the model will give all predictions based on current data. This will include team predictions for ongoing projects, budget predictions for ongoing projects and risk predictions for all ongoing projects **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • User skill constraint: the use case must be easy to understand and use, as some users will have limited technology abilities. • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Intelligent Business Insights.
Revision date and Revision number:	2026-04-28 Version 4.0
Criticality/Priority:	Medium

Table 204: 12.10 View Machine Learning Dashboard - Description

2.8.3.13 System Subsystem

Functional Requirement	Explanation
Requirement number:	13.1
Requirement name (use case name):	Backup System
Requirement short description:	The system must allow a user with appropriate permissions to back up the system.
Requirement detailed description and constraints:	The system will allow users with appropriate permissions make a backup of the system data to ensure that information can be recovered in case of a system failure or data loss. The system backup can be done manually or automatically at scheduled intervals. **Constraints:** • Only users with appropriate permissions can perform system backups.
Business rules applicable to this requirement	• Role-based access. • Ability to backup database.
Revision date and Revision number:	2026-04-28 Version 4.0
Criticality/Priority:	High

Table 205: 13.1 Backup System - Description

Functional Requirement	Explanation
Requirement number:	13.2
Requirement name (use case name):	Restore System
Requirement short description:	The user should be able to restore data on the system.
Requirement detailed description and constraints:	The user should be able to restore data on the system to a previously backed up version. All users will be logged out by the system before the operation takes place. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • User skill constraint: the use case must be easy to understand and use, as some users will have limited technology abilities. • The user requires an Internet connection to access this feature. • Only a Super Administrator can access this feature.
Business rules applicable to this requirement	• Role-based access. • Ability to backup database.
Revision date and Revision number:	2026-04-28 Version 4.0
Criticality/Priority:	Medium

Table 206: 13.2 Restore System - Description

Functional Requirement	Explanation
Requirement number:	13.3
Requirement name (use case name):	View Audit Log
Requirement short description:	An administrator should be able to view the audit log
Requirement detailed description and constraints:	An administrator should be able to view the audit log to see who made which important changes when. This allows them to easily identify the source of misuse. **Constraints:** • Time constraint: the use case must be completed before the Project Day in October 2026. • User skill constraint: the use case must be easy to understand and use, as some users will have limited technology abilities. • The user requires an Internet connection to access this feature. • Only an Administrator can access this feature.
Business rules applicable to this requirement	• Role-based access.
Revision date and Revision number:	2026-04-28 Version 4.0
Criticality/Priority:	Medium

Table 207: 13.3 View Audit Log - Description

2.8.3.14 Budget Subsystem

Functional Requirement	Explanation
Requirement number:	14.1
Requirement name (use case name):	View Project Budget
Requirement short description:	A user with the appropriate permissions should be able to view a project budget.
Requirement detailed description and constraints:	A user with the appropriate permissions should be able to view a project budget. The system will allow the user to select and view a budget's details including expenses. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Financial data such as this is privileged, and thus only persons with appropriate roles may view it.
Revision date and Revision number:	2026-04-28 Version 4.0
Criticality/Priority:	High

Table 208: 14.1 View Project Budget - Description

Functional Requirement	Explanation
Requirement number:	14.2
Requirement name (use case name):	View Project Budgets
Requirement short description:	A user with the appropriate permissions should be able to view all project budgets.
Requirement detailed description and constraints:	A user with the appropriate permissions should be able to view all project budgets. The system will retrieve all project budgets on the system and display them to the user. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Financial data such as this is privileged, and thus only persons with appropriate roles may view it.
Revision date and Revision number:	2026-04-28 Version 4.0
Criticality/Priority:	High

Table 209: 14.2 View Project Budgets - Description

Functional Requirement	Explanation
Requirement number:	14.3
Requirement name (use case name):	Log An Expense
Requirement short description:	A user with appropriate permissions should be able to log an expense.
Requirement detailed description and constraints:	A user with appropriate permissions should be able to log an expense. The system will display the project's budget and allow the user to add an expense by entering the expenses details and the system will validate the inputted data before asking for confirmation **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Financial data such as this is privileged, and thus only persons with appropriate roles may view it.
Revision date and Revision number:	2026-04-28 Version 4.0
Criticality/Priority:	High

Table 210: 14.3 Log An Expense - Description

Functional Requirement	Explanation
Requirement number:	14.4
Requirement name (use case name):	Update An Expense
Requirement short description:	A user with appropriate permissions should be able to update an expense.
Requirement detailed description and constraints:	A user with appropriate permissions should be able to update an expense in the edge case that information is entered incorrectly. The system will display the expense's details and allow the user to update its details. The user will enter the updated details, and the system will validate the updated details before asking the user to confirm the updated details **Constraints:** • The user requires an Internet connection to access this feature. • The expense should already exist on the system.
Business rules applicable to this requirement	• Financial data such as this is privileged, and thus only persons with appropriate roles may view it.
Revision date and Revision number:	2026-04-28 Version 4.0
Criticality/Priority:	High

Table 211: 14.4 Update An Expense - Description

Functional Requirement	Explanation
Requirement number:	14.5
Requirement name (use case name):	View An Expense
Requirement short description:	A user with appropriate permissions should be able to view an expense.
Requirement detailed description and constraints:	A user with appropriate permissions should be able to view an expense. The system will display the project budget and allow the user to select an expense to view. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Financial data such as this is privileged, and thus only persons with appropriate roles may view it.
Revision date and Revision number:	2026-04-28 Version 4.0
Criticality/Priority:	High

Table 212: 14.5 View An Expense - Description

Functional Requirement	Explanation
Requirement number:	14.6
Requirement name (use case name):	Remove An Expense
Requirement short description:	A user with the appropriate permissions should be able to remove an expense.
Requirement detailed description and constraints:	A user with the appropriate permissions should be able to remove an expense. The system will display the projects budget and expenses and allow the user to remove an expense. The system will ask for confirmation before the expense is removed. **Constraints:** • The user requires an Internet connection to access this feature.
Business rules applicable to this requirement	• Financial data such as this is privileged, and thus only persons with appropriate roles may view it.
Revision date and Revision number:	2026-04-28 Version 4.0
Criticality/Priority:	High

Table 213: 16.6 Remove an Expense - Description

2.8.4 Functional Requirements List Detailed Changes Description

2.8.4.1 User Subsystem

User Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
1.1 User Login	1.1 User Login	-
1.2 User Logout	1.2 User Logout	-
1.3 Reset Password	1.3 Reset Password	-
1.4 Create User	1.4 Create User	-
1.5 Update User	1.5 Update User	-
1.6 View User	1.6 View User	-
1.7 Reactivate User	1.7 Reactivate User	-
1.8 Deactivate User	1.8 Deactivate User	-
1.9 View FAQ	1.9 View FAQ	-

Table 214: User Subsystem Modifications

2.8.4.2 Admin Subsystem

Admin Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
2.1 Create User Profile	2.1 Create User Profile	-
2.2 Update User Profile	2.2 Update User Profile	-
2.3 View User Profile	2.3 View User Profile	-
2.4 Deactivate User Profile	2.4 Deactivate User Profile	-
2.5 Add FAQ	2.5 Add FAQ	-
2.6 Update FAQ	2.6 Update FAQ	-
2.7 Remove FAQ	2.7 Remove FAQ	-
-	2.8 Add FAQ Type	Organizes data by creating scalable topic categories instead of a single unmanageable list.
-	2.9 Update FAQ Type	Keeps lookups accurate by instantly cascading updated terminology down to all related FAQs.
-	2.10 Remove FAQ Type	Keeps the system clean by removing obsolete categories while preventing accidental data corruption.
2.8 Create Role	2.11 Create Role	Numbering change
2.9 View Role	2.12 View Role	Numbering change
2.10 Update Role	2.13 Update Role	Numbering change
2.11 Reactivate Role	2.14 Reactivate Role	Numbering change
2.12 Deactivate Role	2.15 Deactivate Role	Numbering change
2.13 Update Role Permission	2.16 Update Role Permission	Numbering change
2.14 View Role Permission	2.17 View Role Permission	Numbering change

Table 215: Admin Subsystem Modifications

2.8.4.3 Inventory Subsystem

Inventory Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
3.1 Receive Inventory Item	3.1 Create Procurement	Name change for better clarity on use case function.
-	3.2 View Procurement	Added to allow admins to view a procurement's details and which procurement an inventory item belongs to.
3.2 View Inventory	3.3 View Inventory	-
3.3 Write Off Inventory	3.4 Write Off Inventory	-
3.4 Allocate Inventory	3.5 Allocate Inventory	-
3.5 Create Inventory Item Type	3.6 Create Inventory Item Type	-
3.6 View Inventory Item Type	Removed	Inventory types are viewed in a table and not individually.
3.7 View Inventory Item Types	3.7 View Inventory Item Types	-
3.8 Update Inventory Item Type	3.8 Update Inventory Item Type	-
3.9 Deactivate Inventory Item Type	3.9 Deactivate Inventory Item Type	-
-	3.10 Reactivate Inventory Item Type	Added to allow super admins the ability to reactivate inventory item types.

Table 216 Inventory Subsystem Modifications

2.8.4.4 Group Subsystem

Group Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
4.1 Create Group	4.1 Create Group	-
4.2 View Groups	4.2 View Groups	-
4.3 Update Group	4.3 Update Group	-
4.4 Deactivate Group	4.4 Deactivate Group	-
4.5 Create Skill	4.5 Create Skill	-
4.6 View Skills	4.6 View Skills	-
4.7 View Skills	4.7 View Skill	-
4.8 Update Skill	4.8 Update Skill	-
4.9 Deactivate Skill	4.9 Deactivate Skill	-

Table 217 Group Subsystem Modifications

2.8.4.5 Warehouse Subsystem

Warehouse Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
5.1 Create Warehouse	5.1 Create Warehouse	-
5.2 View Warehouse	5.2 View Warehouse	-
5.3 Update Warehouse	5.3 Update Warehouse	-
5.4 Reactivate Warehouse	5.4 Reactivate Warehouse	-
5.5 Deactivate Warehouse	5.5 Deactivate Warehouse	-
5.6 Inventory Stock Take	5.6 Inventory Stock Take	-

Table 218: Warehouse Subsystem Modifications

2.8.4.6 Vendor Subsystem

Vendor Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
6.1 Create Vendor	6.1 Create Vendor	-
6.2 View Vendor	6.2 View Vendor	-
6.3 View Vendors	6.3 View Vendors	-
6.4 Edit Vendor	6.4 Edit Vendor	-
6.5 Reactivate Vendor	6.5 Reactivate Vendor	-
6.6 Deactivate Vendor	6.6 Deactivate Vendor	-
6.7 Create Vendor Type	6.7 Create Vendor Type	-
6.8 Update Vendor Type	6.8 Update Vendor Type	-
6.9 View Vendor Types	6.9 View Vendor Types	-
6.10 Reactivate Vendor Type	6.10 Reactivate Vendor Type	-
6.11 Deactivate Vendor Type	6.11 Deactivate Vendor Type	-

Table 219 Vendor Subsystem Modifications

2.8.4.7 Project Management Subsystem

Project Management Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
7.1 Create Project	7.1 Create Project	-
7.2 View Project	7.2 View Project	-
7.3 Update Project	7.3 Update Project	-
7.4 Deactivate Project	7.4 Deactivate Project	-
7.5 Create Project Type	7.5 Create Project Type	-
7.6 View Project Type	7.6 View Project Type	-
7.7 Update Project Type	7.7 Update Project Type	-
7.8 Deactivate Project Type	7.8 Deactivate Project Type	-
7.9 Add Indemnity Form	7.9 Add Indemnity Form	-
7.10 Update Indemnity Form	7.10 Update Indemnity Form	-
7.11 View Indemnity Form	7.11 View Indemnity Form	-
7.12 Remove Indemnity Form	7.12 Remove Indemnity Form	-
7.13 Capture Indemnity Form	7.13 Capture Indemnity Form	-

Project Management Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
7.14 Capture Project Feedback	7.14 Capture Project Feedback	-
7.15 View Project Feedback	7.15 View Project Feedback	-
7.16 Create Impact Feedback Form	7.16 Create Impact Form	-
7.17 View Impact Feedback Form	7.17 View Impact Form	-
7.18 Update Impact Feedback Form	7.18 Update Impact Form	-
7.19 Deactivate Impact Feedback Form	7.19 Deactivate Impact Form	-
7.20 Capture Impact Feedback	7.20 Capture Impact Form	-
7.21 View Impact Feedback	7.21 View Impact Form	-
7.22 Add Photo	7.22 Add Photo	-
7.23 View Photo	7.23 View Photo	-
7.24 Delete Photo	7.24 Delete Photo	-

Table 220 Vendor Subsystem Modifications

2.8.4.8 K9 Subsystem

K9 Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
8.1 Add Dog	8.1 Add Dog	-
8.2 View Dog	8.2 View Dog	-
8.3 View Dogs	8.3 View Dogs	-
8.4 Update Dog	8.4 Update Dog	-
8.5 Deactivate Dog	8.5 Deactivate Dog	-
8.6 Assign User to Dog	8.6 Assign User to Dog	-
8.7 View User-Dog Assignment	8.7 View User-Dog Assignment	-
8.8 Update User-Dog Assignment	8.8 Update User-Dog Assignment	-
8.9 Reactivate User-Dog Assignment	8.9 Reactivate User-Dog Assignment	-
8.10 Deactivate User-Dog Assignment	8.10 Deactivate User-Dog Assignment	-
8.11 Add K9 Role	8.11 Add K9 Role	-
8.12 Update K9 Role	8.12 Update K9 Role	-
8.13 View K9 Role	8.13 View K9 Role	-
8.14 Reactivate K9 Role	8.14 Reactivate K9 Role	-
8.15 Deactivate K9 Role	8.15 Deactivate K9 Role	-
8.16 View Ancestry	8.16 View Ancestry	-

Table 221 K9 Subsystem Modifications

2.8.4.9 Rhino Subsystem

Rhino Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
9.1 Add Rhino	9.1 Add Rhino	-
9.2 View Rhino	9.2 View Rhino	-
9.3 View Rhinos	9.3 View Rhinos	-
9.4 Update Rhino	9.4 Update Rhino	-
9.5 Deactivate Rhino	9.5 Deactivate Rhino	-

Table 222 Rhino Subsystem Modifications

2.8.4.10 Elephant Subsystem

Elephant Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
10.1 Add Elephant	10.1 Add Elephant	-
10.2 View Elephant	10.2 View Elephant	-
10.3 View Elephants	10.3 View Elephants	-
10.4 Update Elephant	10.4 Update Elephant	-
10.5 Deactivate Elephant	10.5 Deactivate Elephant	-

Table 223 Elephant Subsystem Modifications

2.8.4.11 Message Subsystem

Message Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
11.1 Create Message	11.1 Create Message	-
11.2 View Messages	11.2 View Messages	-

Table 224 Message Subsystem Modifications

2.8.4.12 Report Subsystem

Report Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
12.1 Generate Custom Report	12.1 Generate Custom Report	-
12.2 View Custom Report	12.2 View Custom Report	-
12.3 Update Custom Report	12.3 Update Custom Report	-
12.4 Remove Custom Report	12.4 Remove Custom Report	-
12.5 View Bike Impact Report	12.5 View Bike Impact Report	-
12.6 View Budget Report	12.6 View Budget Report	-
12.7 View Rhino Report	12.7 View Rhino Report	-
12.8 View Elephant Report	12.8 View Elephant Report	-
12.9 View K9 Report	12.9 View K9 Report	-
12.10 View Machine Learning Dashboard	12.10 View Machine Learning Dashboard	-

Table 225: Report Subsystem Modifications

2.8.4.13 System Subsystem

System Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
13.1 Backup System	13.1 Backup System	-
13.2 Restore System	13.2 Restore System	-
13.3 View Audit Log	13.3 View Audit Log	-

Table 226: System Subsystem Modifications

2.8.4.14 Budget Subsystem

Budget Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
14.1 View Project Budget	14.1 View Project Budget	-
14.2 View Project Budgets	14.2 View Project Budgets	-
14.3 Log An Expense	14.3 Log An Expense	-
14.4 Update Expense	14.4 Update Expense	-
14.5 View Expense	14.5 View Expense	-
14.6 Remove Expense	14.6 Remove Expense	-

Table 227 Budget Subsystem Modifications

2.9 Technical Use Case Diagram

2.9.1 User Subsystem

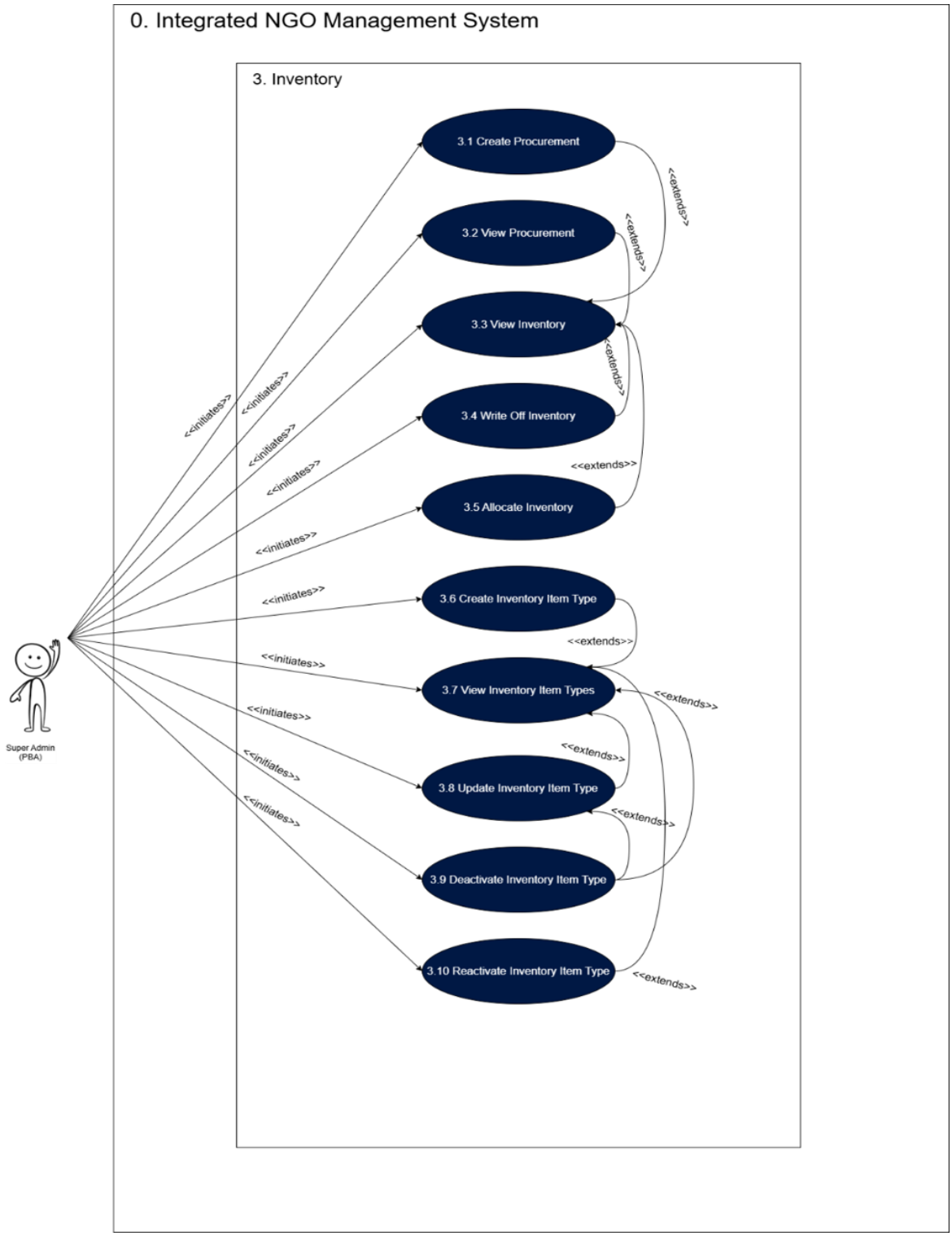


Figure 286: User Technical Use Case Diagram

2.9.2 Admin Subsystem

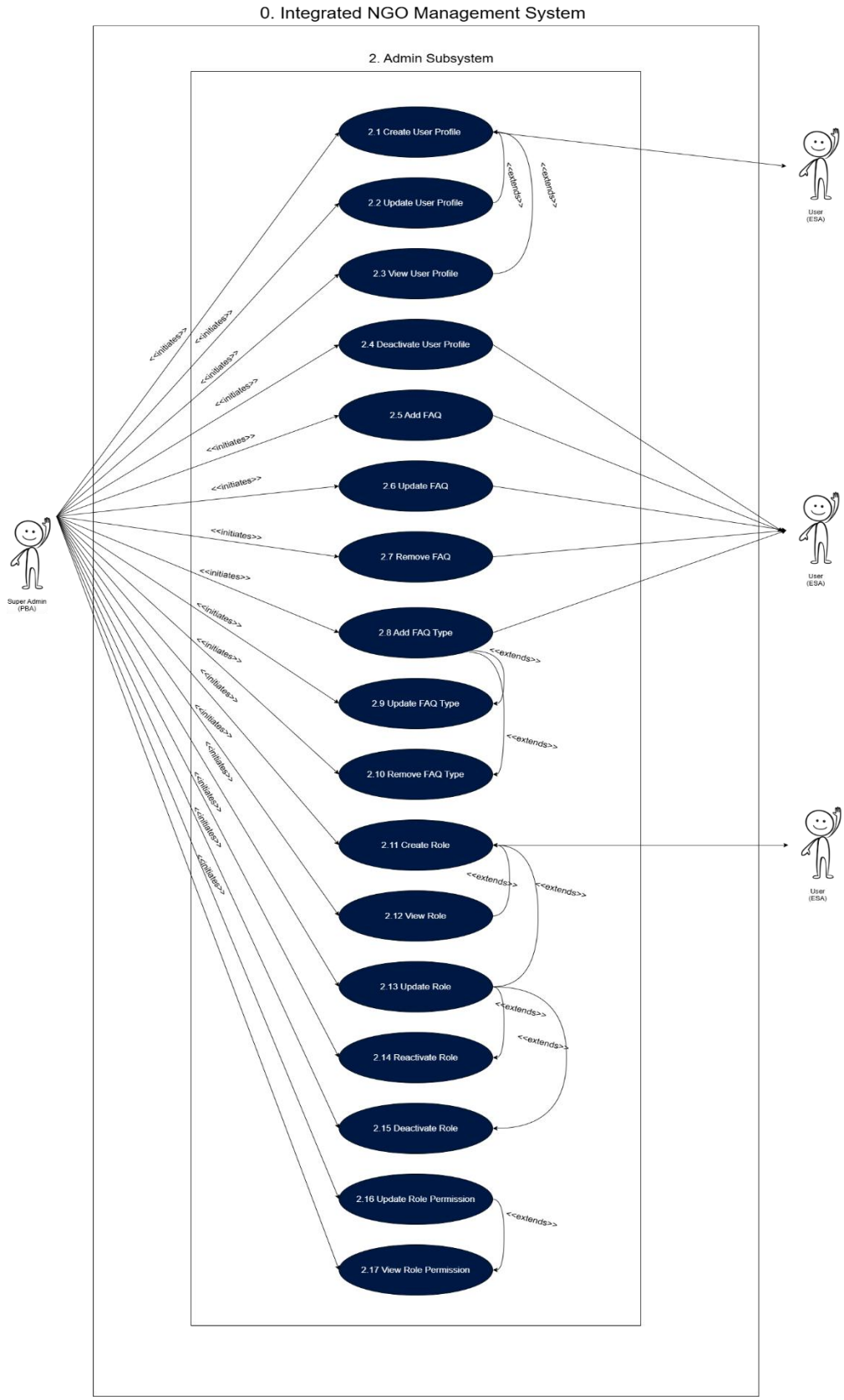


Figure 287: Admin Technical Use Case Diagram

2.9.3 Inventory Subsystem

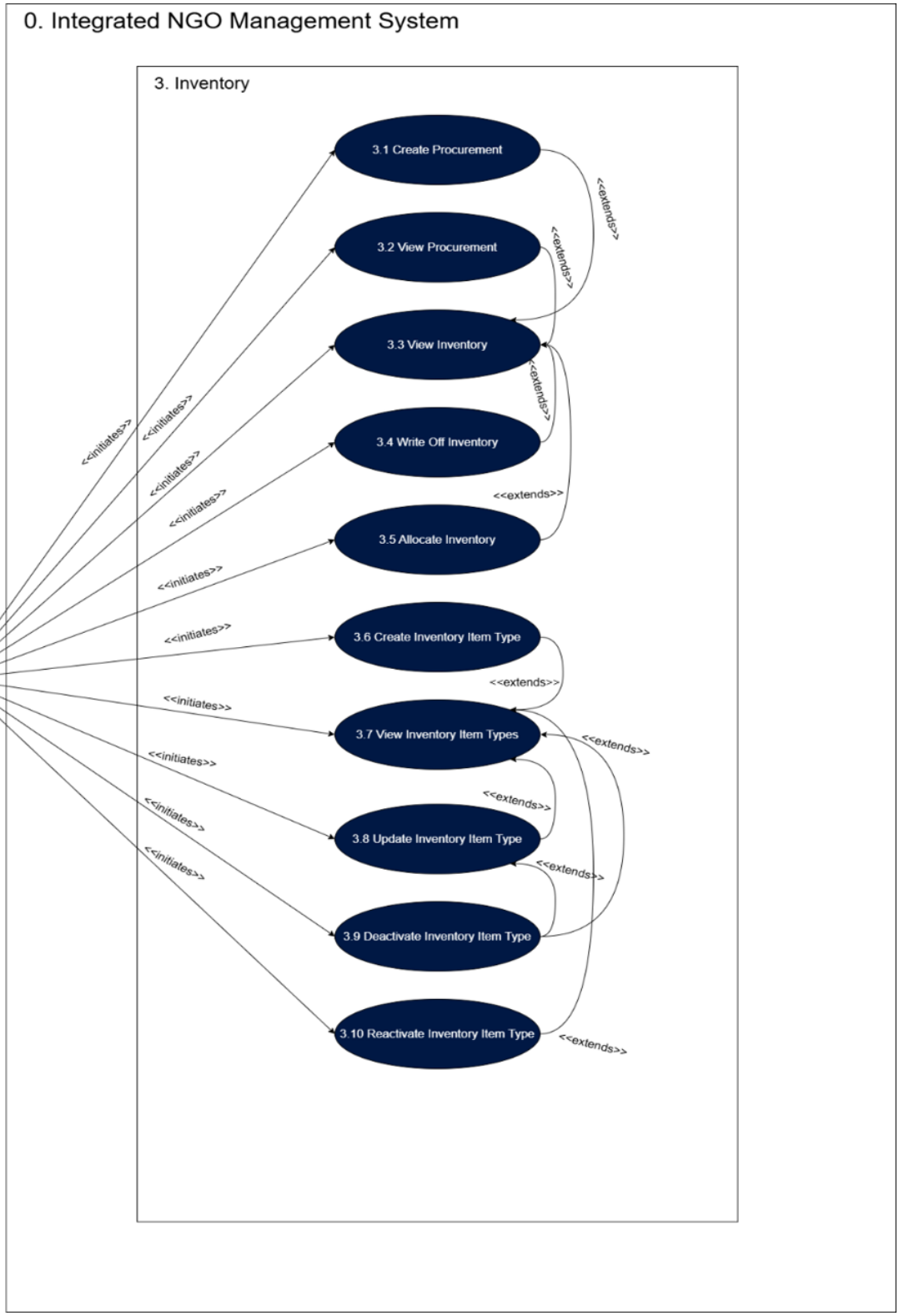


Figure 288: Inventory Technical Use Case Diagram

2.9.4 Warehouse Subsystem

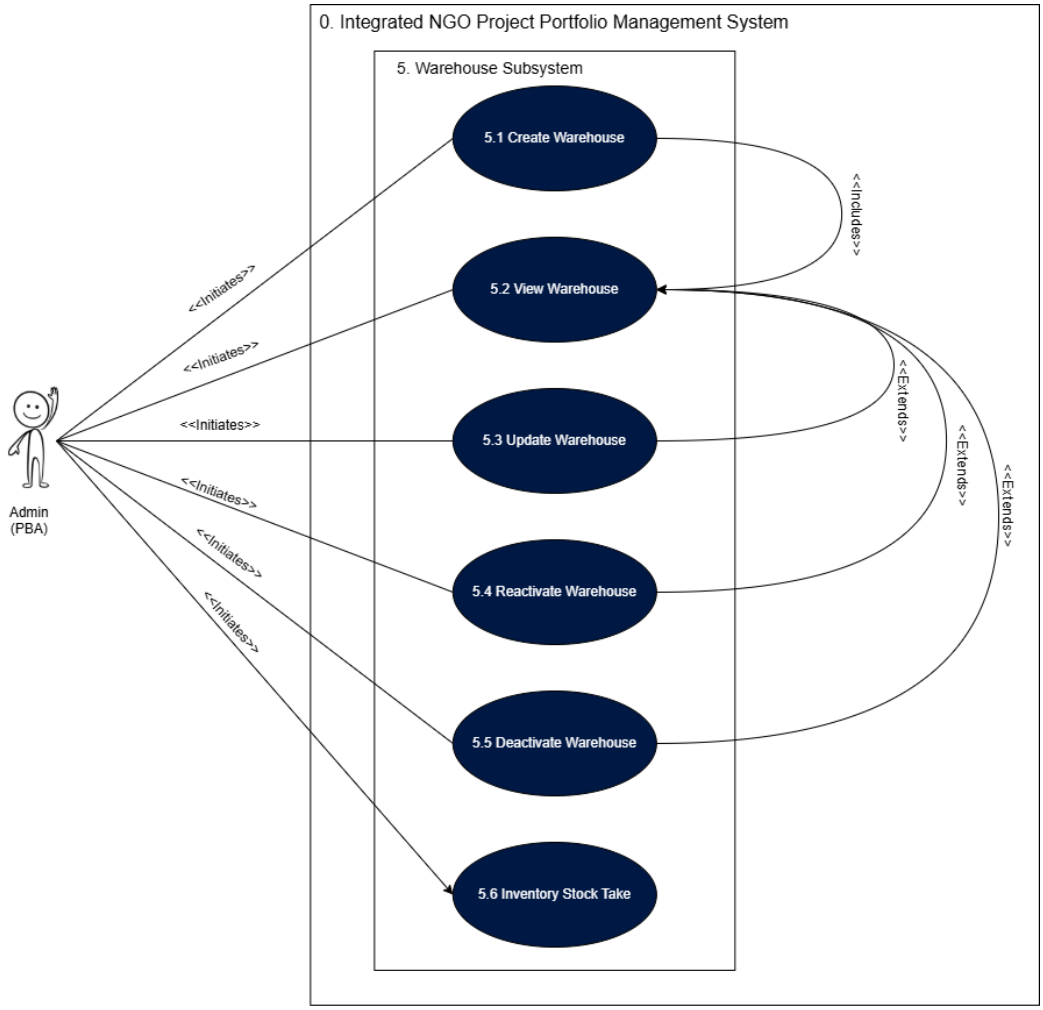


Figure 289: Warehouse Technical Use Case Diagram

2.9.5 Vendor Subsystem

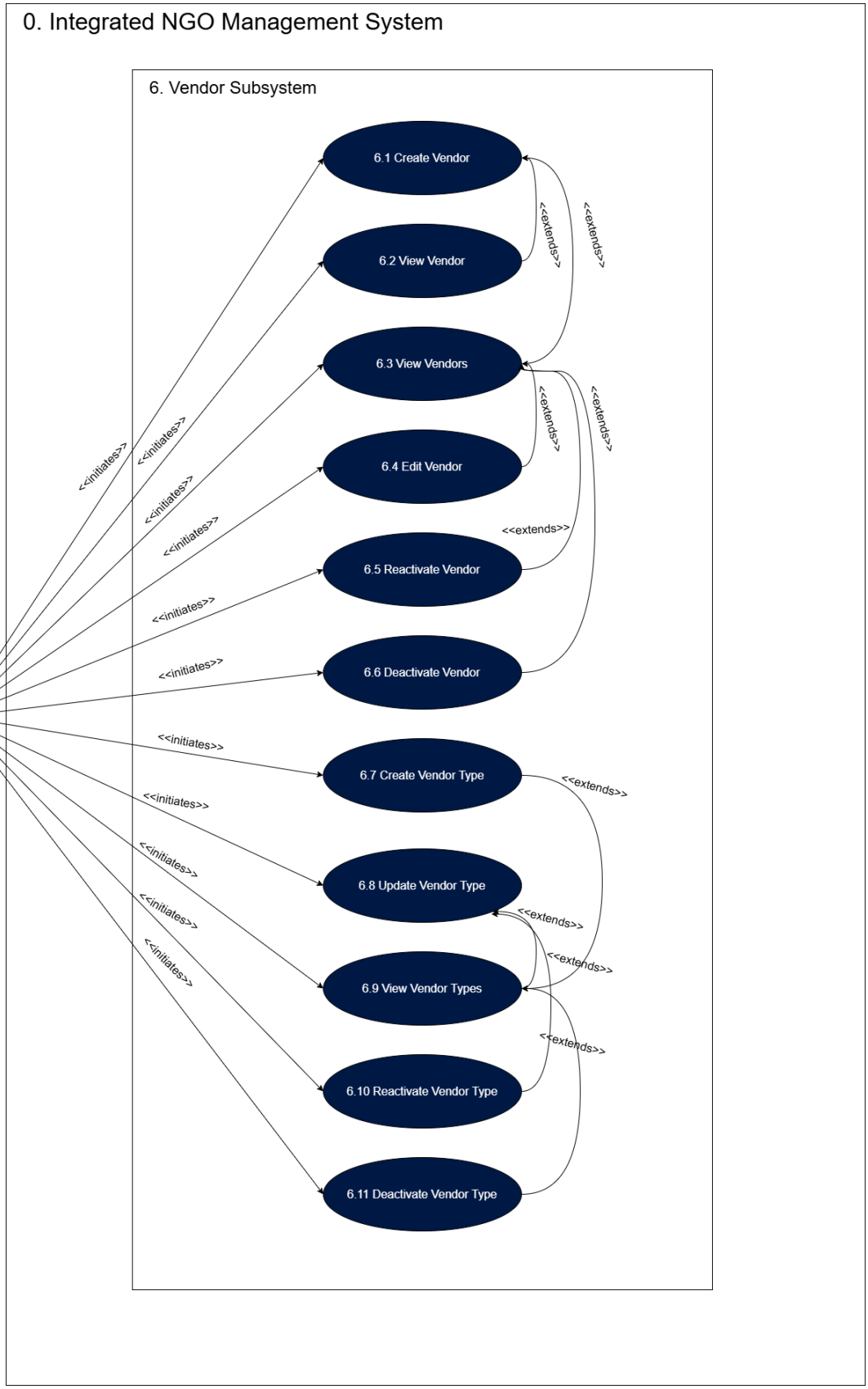


Figure 290: Vendor Technical Use Case Diagram

2.9.6 Project Management

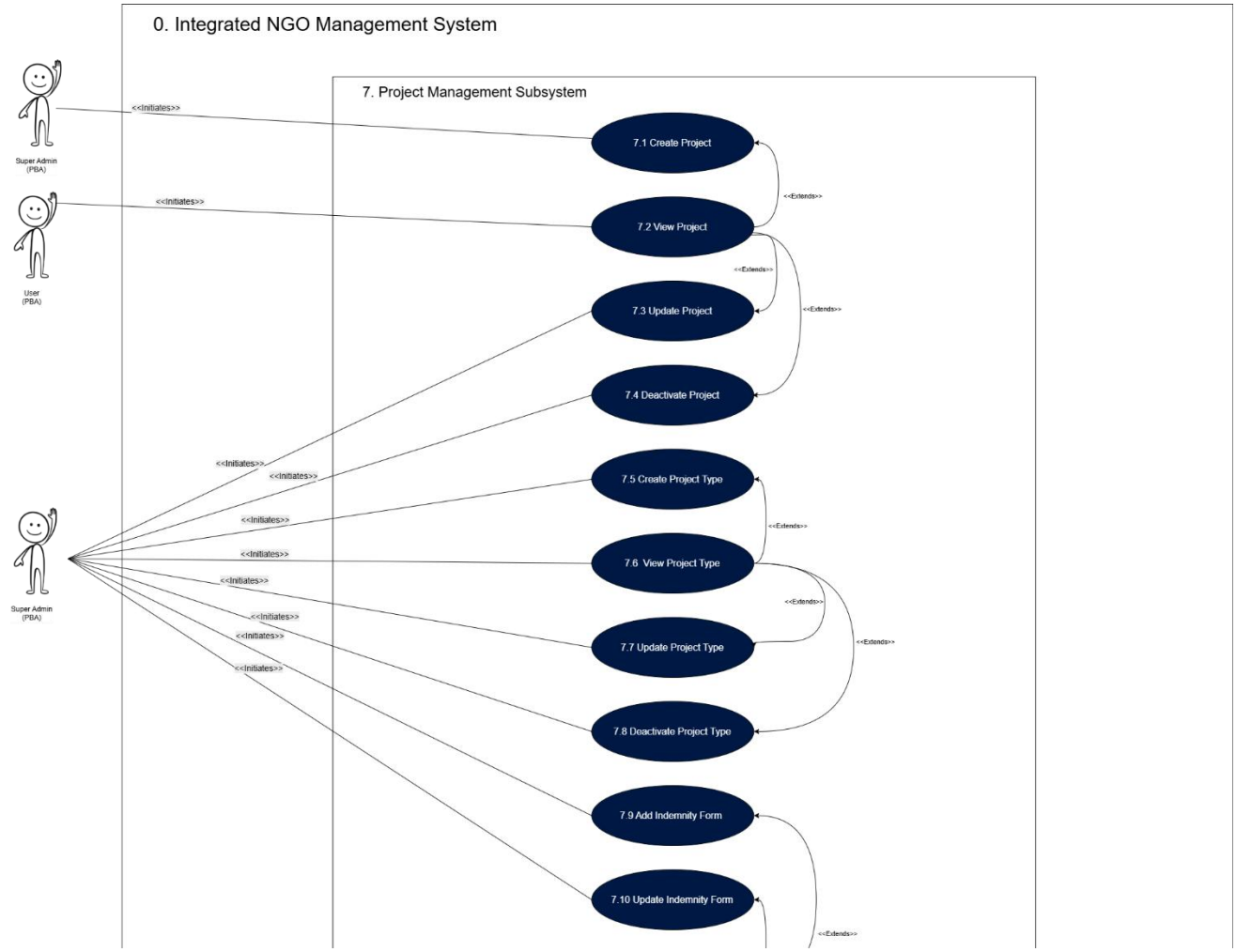


Figure 291: Project Management Technical Use Case Diagram

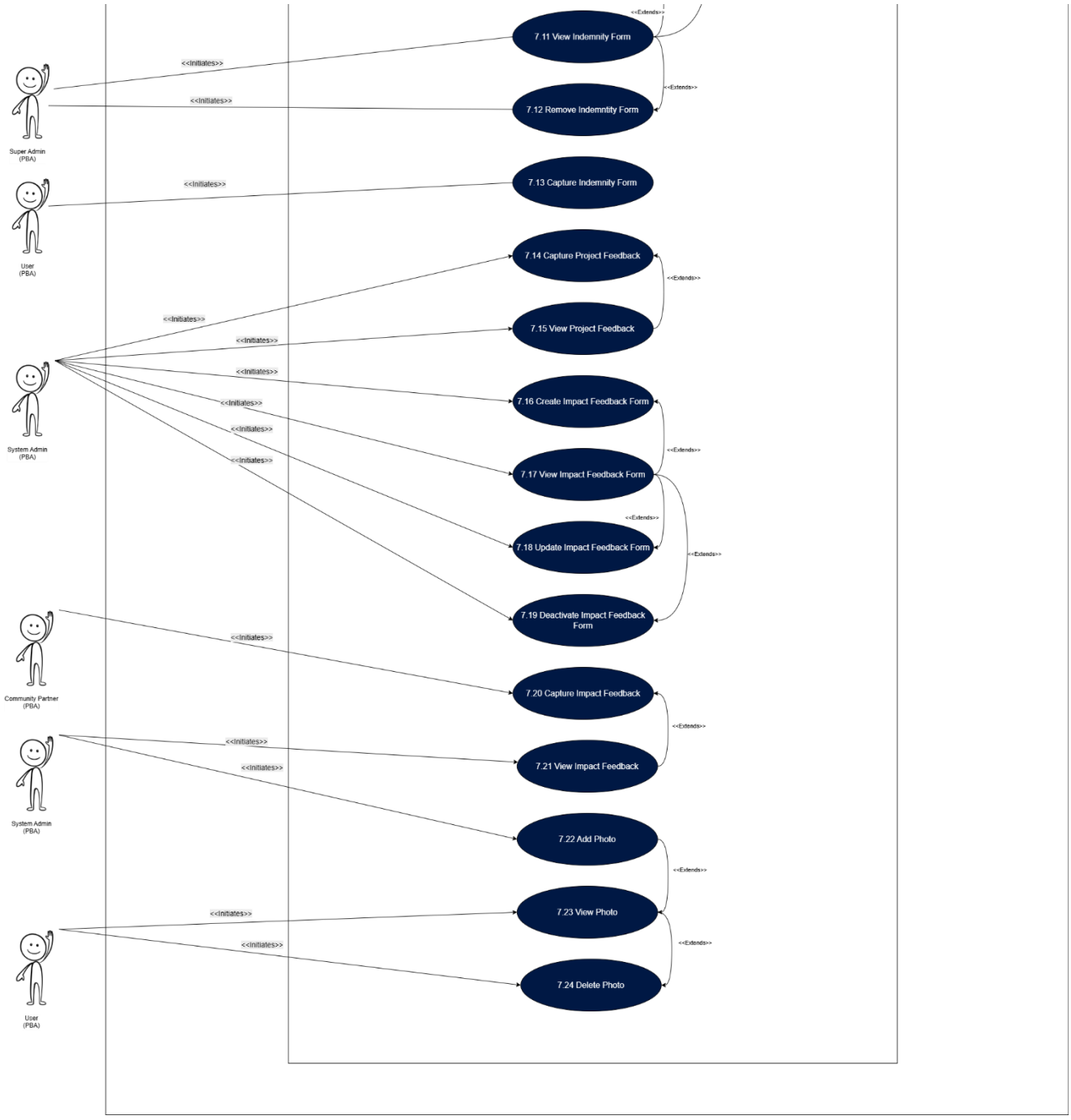


Figure 292: Project Management Technical Use Case Diagram (Continued)

2.9.7 K9 Subsystem

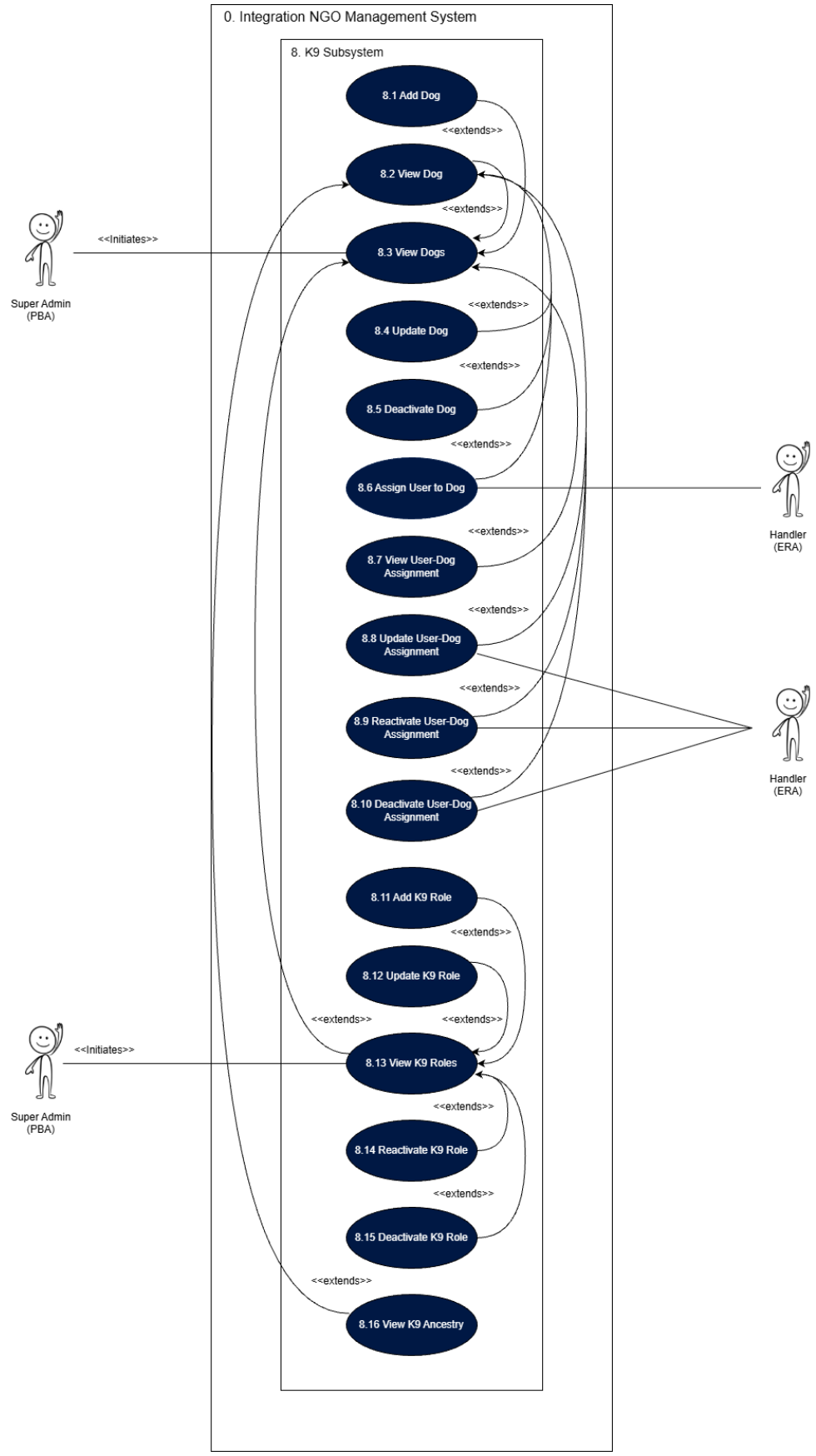


Figure 293: K9 Technical Use Case Diagram

2.9.8 Rhino Subsystem

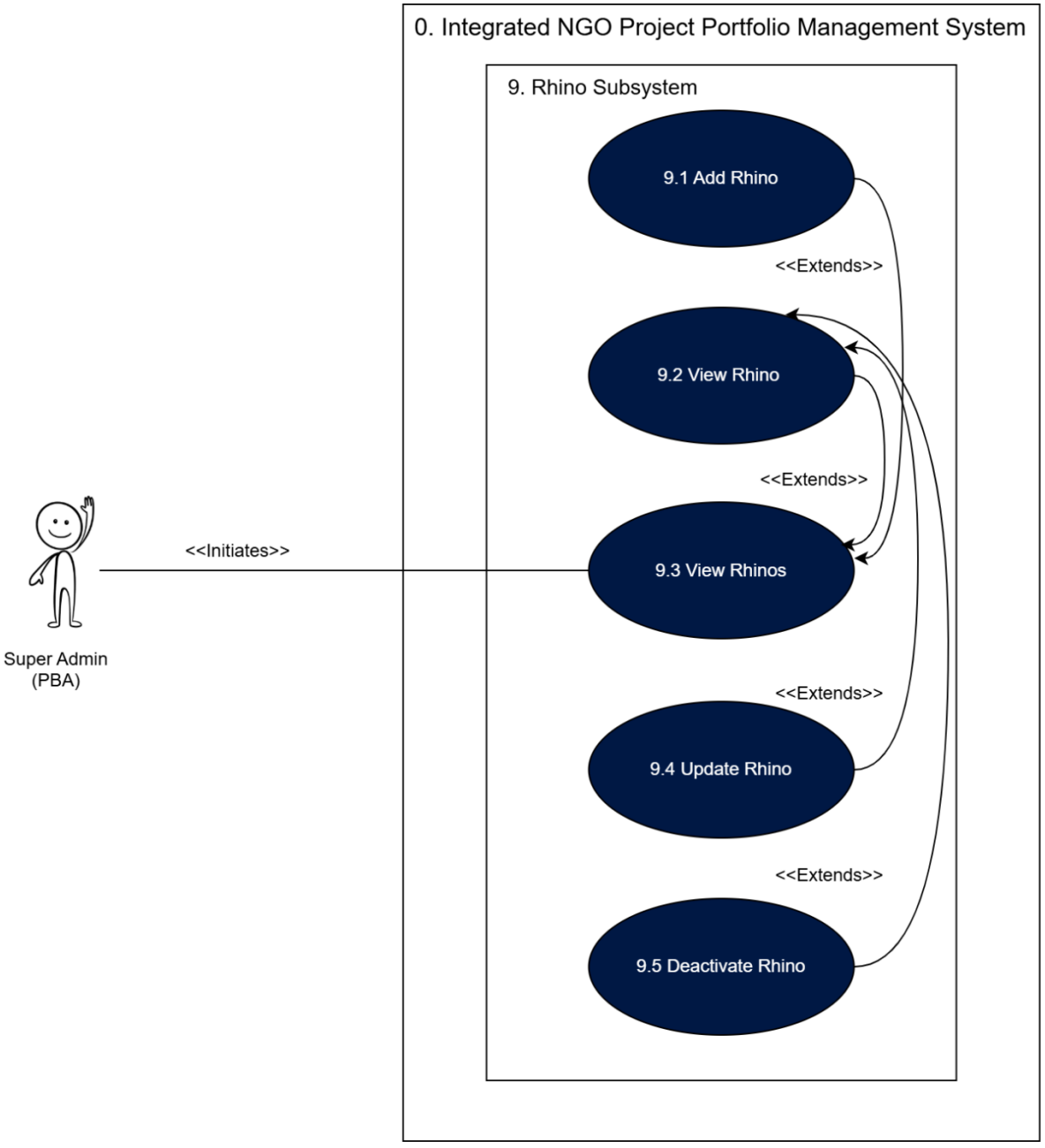


Figure 294: Rhino Technical Use Case Diagram

2.9.9 Elephant Subsystem

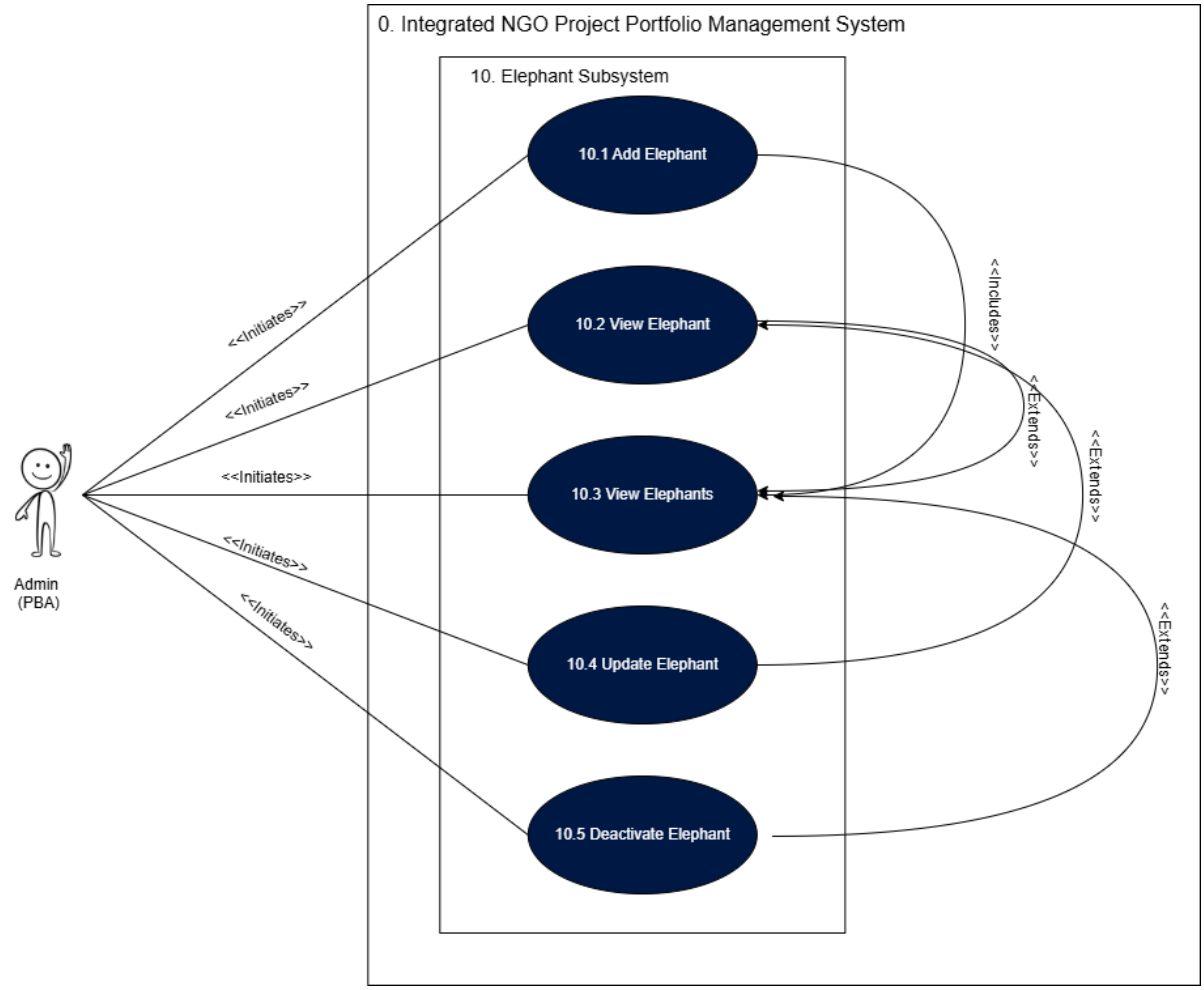


Figure 295: Technical Use Case Diagram

2.9.10 Report Subsystem

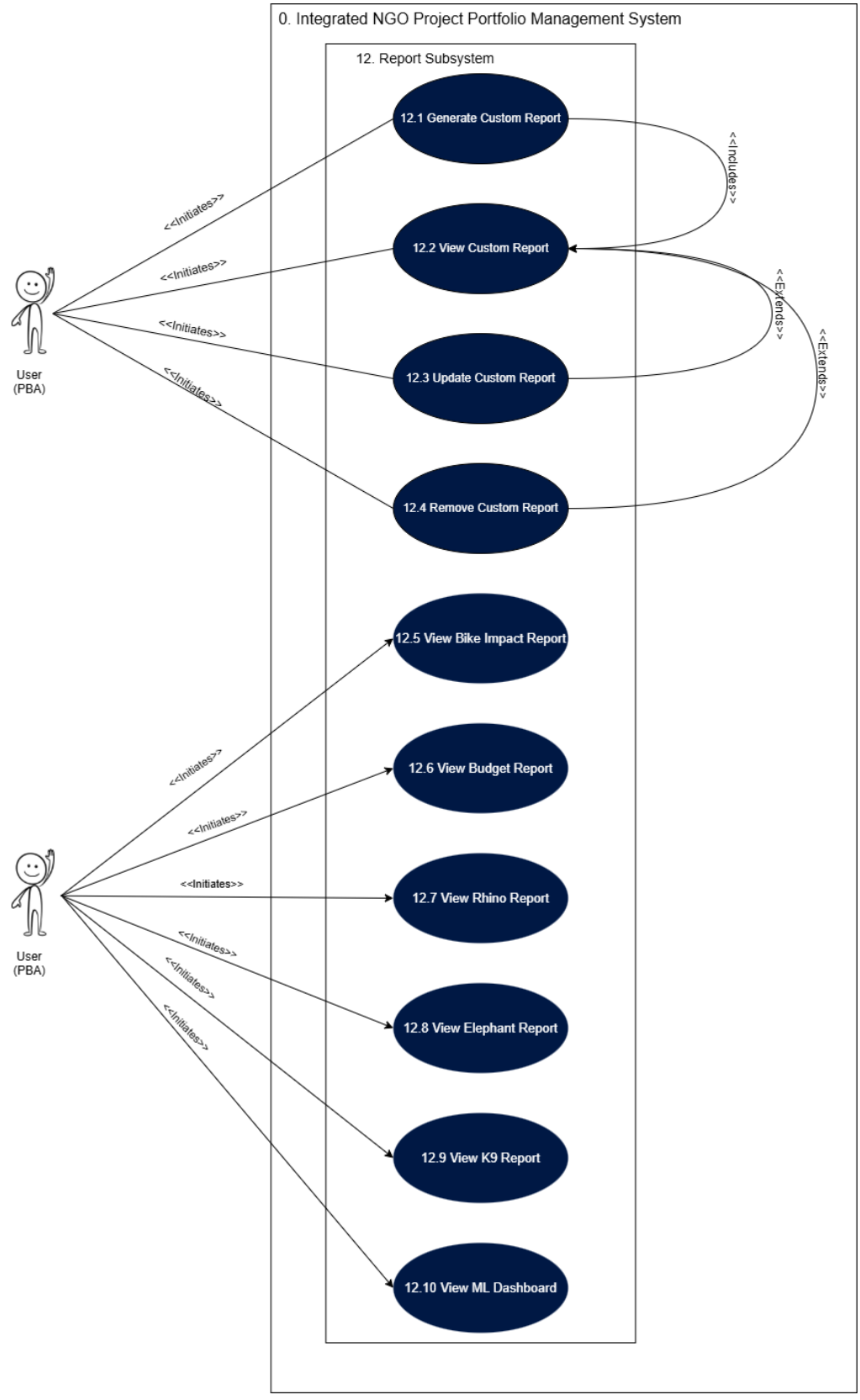


Figure 296: Report Technical Use Case Diagram

2.9.11 Budget Subsystem

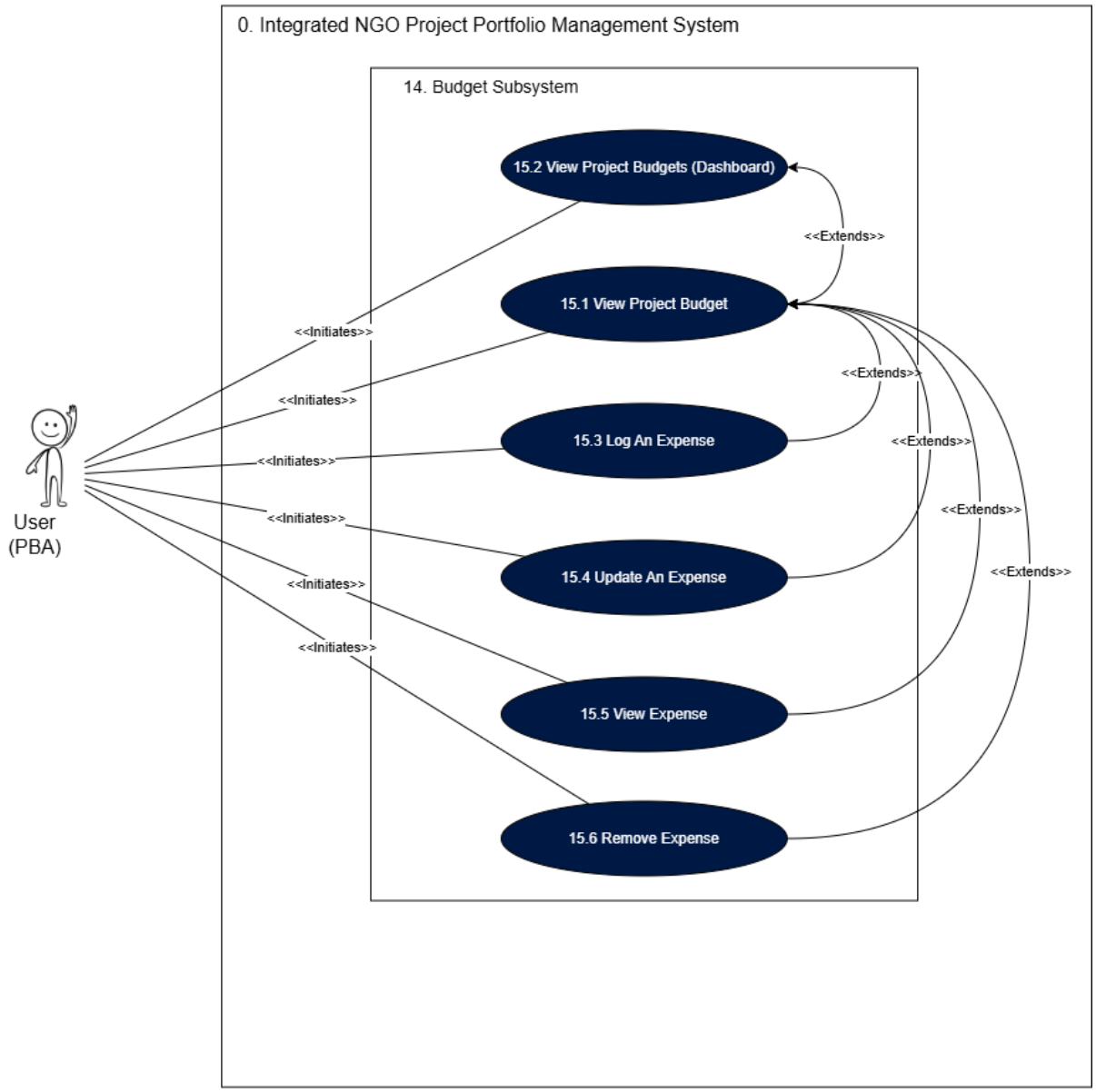
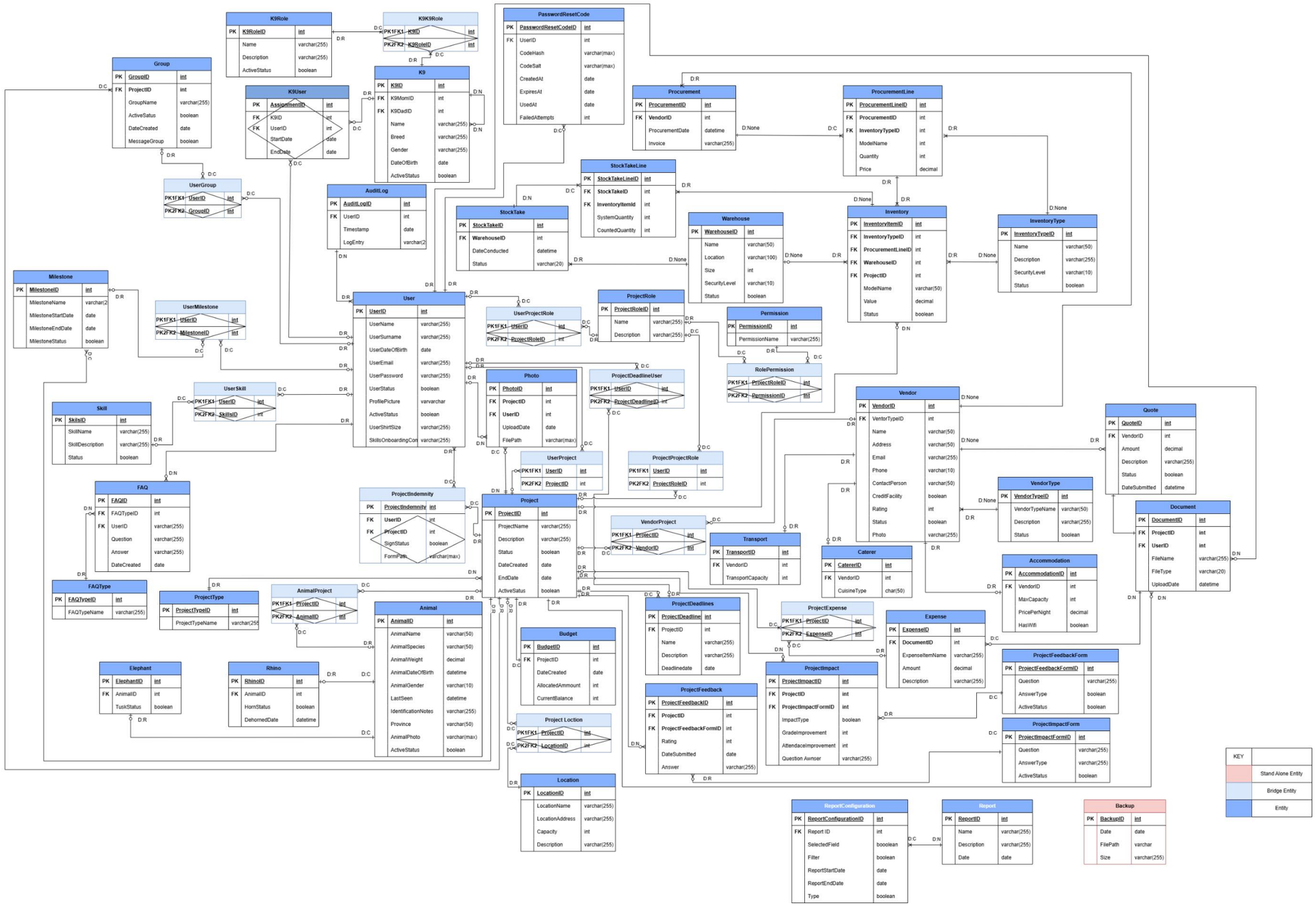


Figure 297: Budget Technical Use Case Diagram

2.10 Technical Entity Relationship Diagram



2.11 State Diagrams

2.11.1 Projects Controller

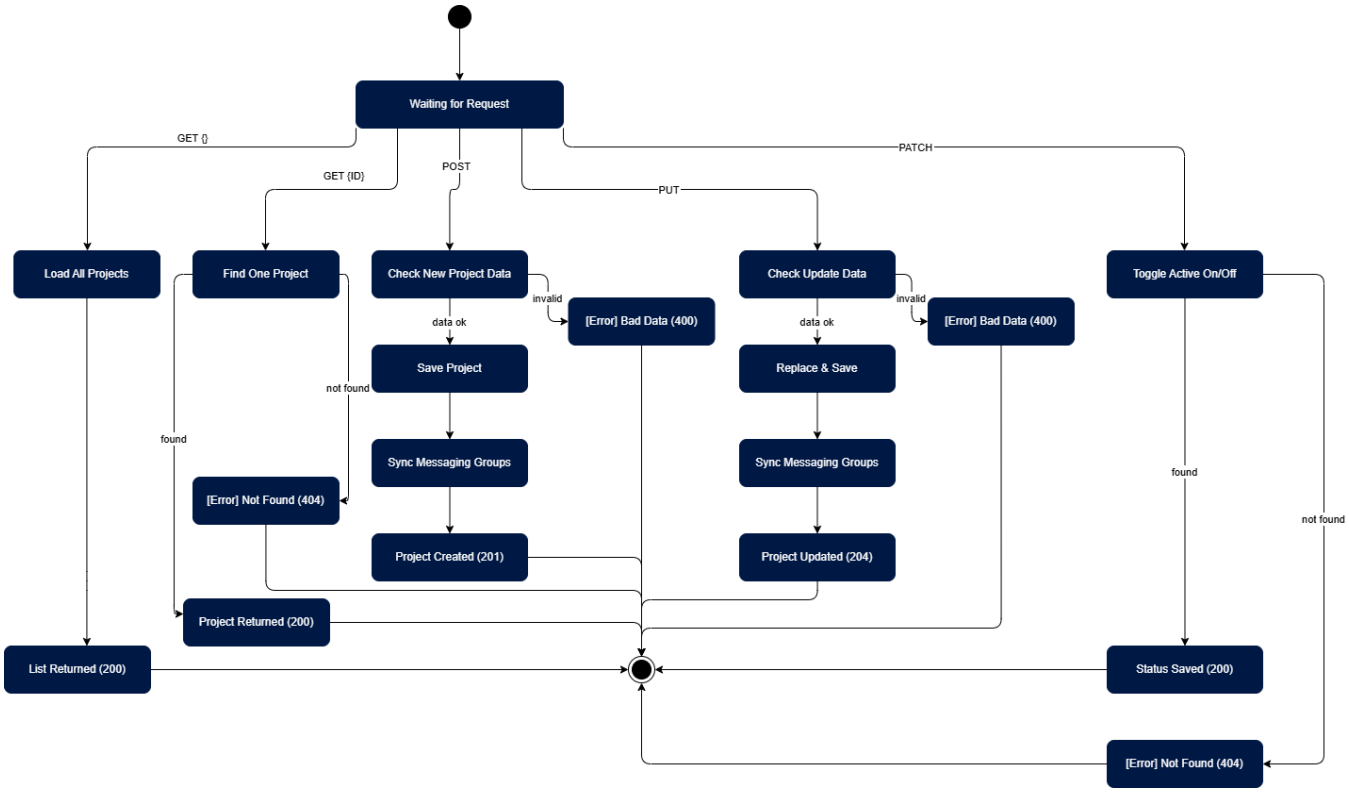


Figure 298: Project Controller State Diagram

2.11.2 Auth Controller

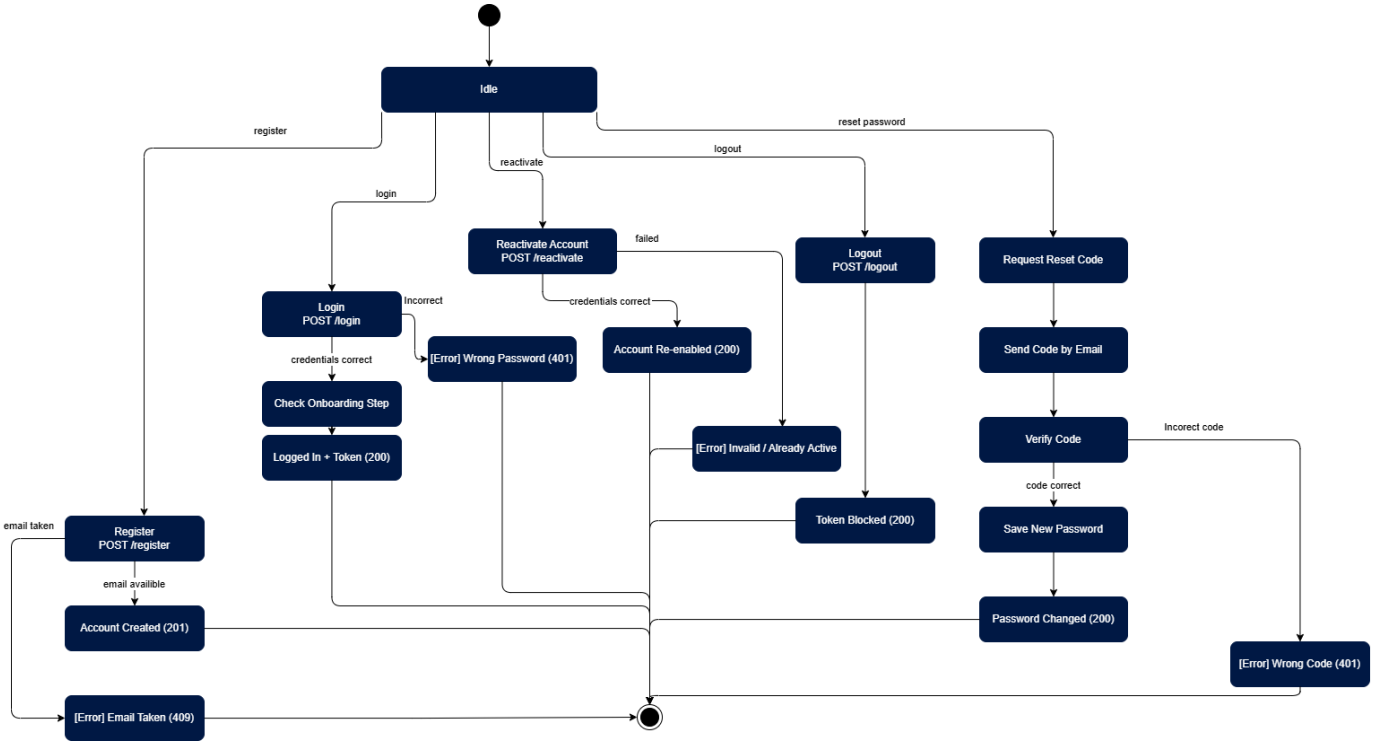


Figure 299: Auth Controller State Diagram

2.11.3 User Controller

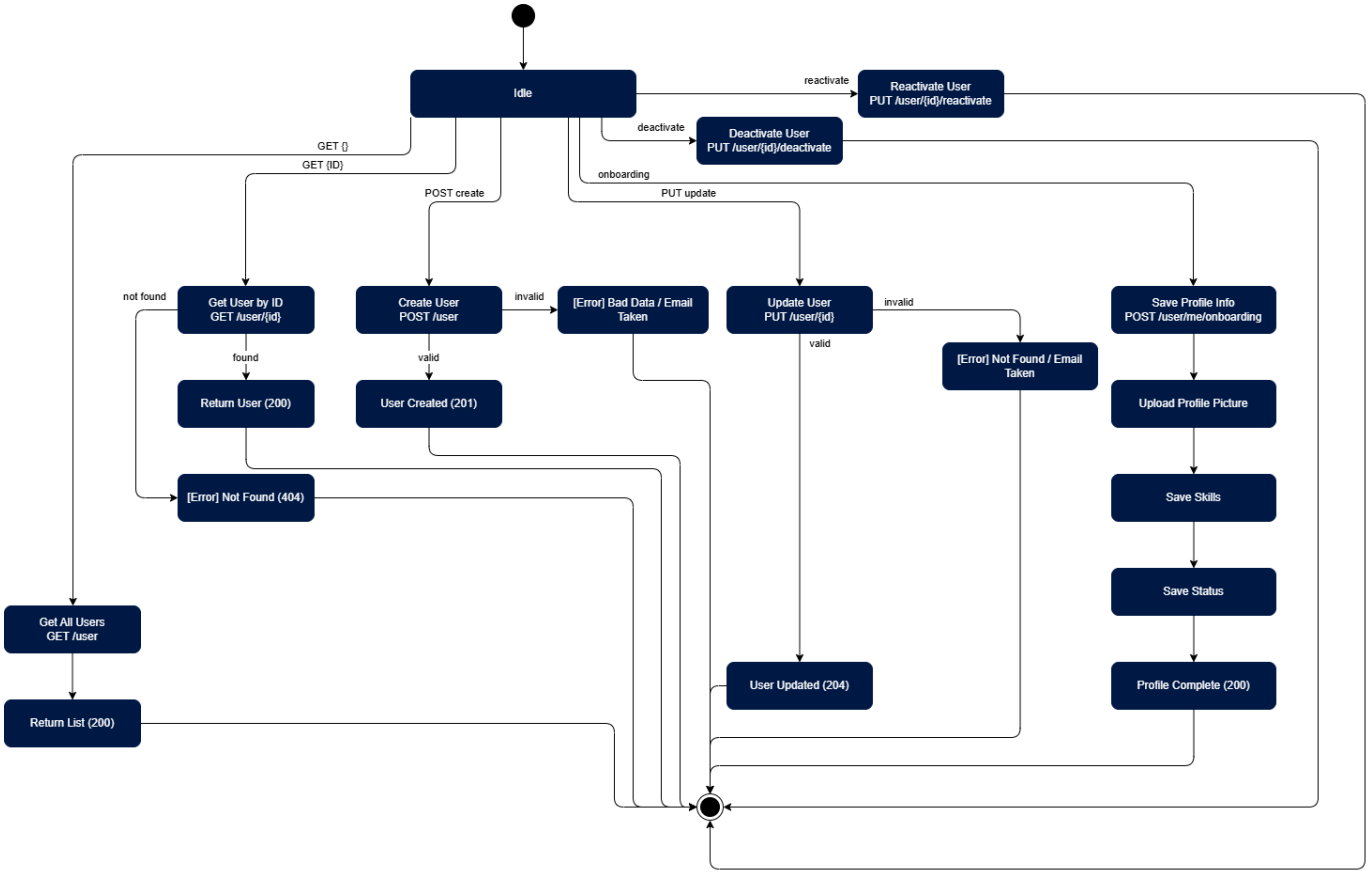


Figure 300: User Controller State Diagram

2.11.4 Project Role Repository

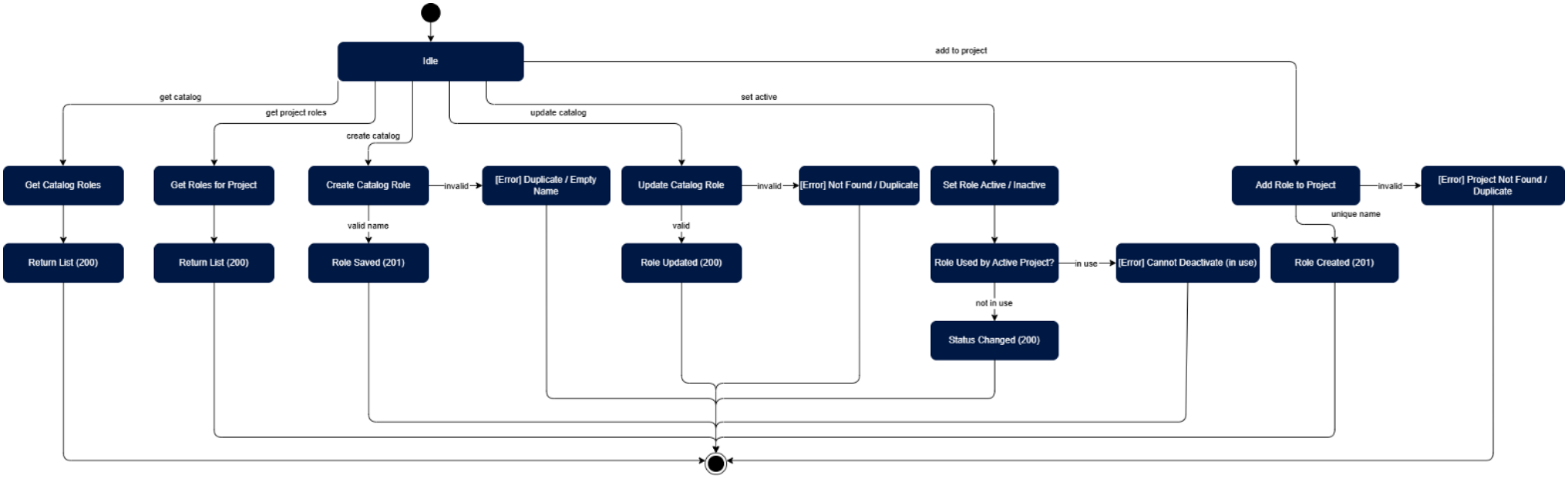


Figure 301: Project Role Repository State Diagram

2.11.5 Group Repository

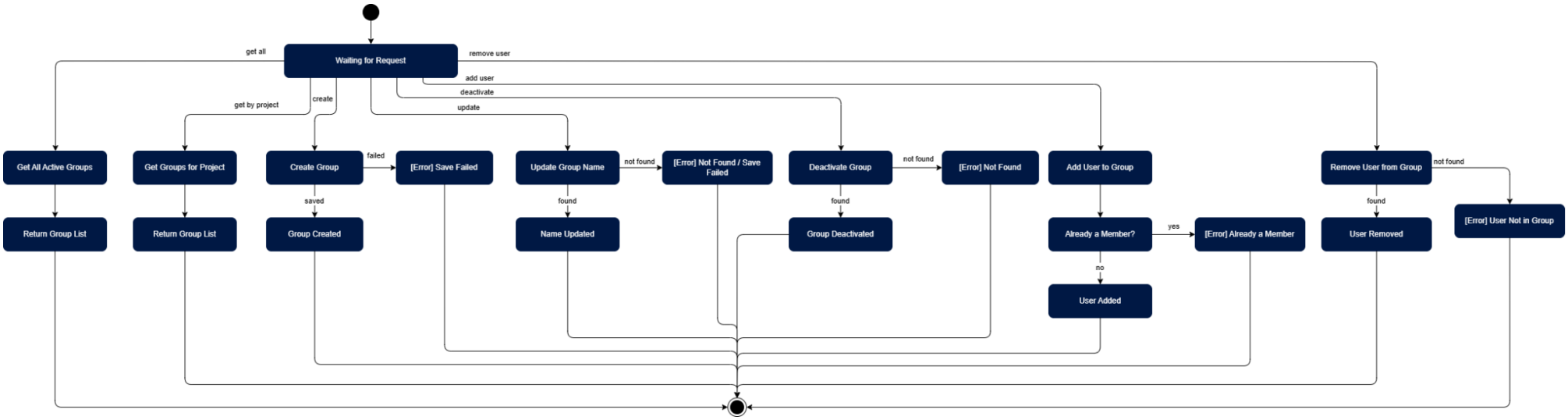


Figure 302: Group Repository State Diagram

3. Document Conclusion

The documentation presented showcased the screen designs for 83 uses assigned to Iteration 5. It also included technical narratives, technical primitive diagrams, technical context diagrams, sequence diagrams, a technical entity relationship diagram, a use case diagram, the updated functional requirement list and state diagrams.

4. Sign-off by Client

Sign-Off Confirmation:

I, the undersigned, have reviewed the contents of this document and approved it as complete and accurate. I acknowledge that the information provided meets the requirements agreed upon with team 35, Canvas.

Client Name: Dané Coetzee

Signature: 

Date: 20/07/2026

5. Sign-off by Team

Team member:

Name: Alexander Steven

Date:25 July 2026

Signature: 

Team member:

Name: Amahle Khoza

Date: 25 July 2026

Signature: 

Team member:

Name: Christopher Bailey

Date: 25 July 2026

Signature: 

Team member:

Name: Nicholas Bailey

Date: 25 July 2026

Signature: 

Team member:

Name: Stephan Bredenhann

Date: 25 July 2026

Signature: 