ITERATION 4

OUR CLIENT:

Group 35

NGO PROJECT PORTFOLIO MANAGEMENT SYSTEM

Our Iteration 4 document focuses on the design and development of 40 use cases for EPI-USE and groupelephant.com. It includes our updated functional requirements with their descriptions, use case diagram, a technical ERD, screen design, technical use case naratives, technical context diagram, technical primitive diagram, a sequence diagram, and lastly state diagram for complex classes.



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

Table of Contents

Table of Tables 4
Table of Figures 9
1. Document Introduction 13
2. Design and Development of the Use Cases 14
2.1 Introduction 14
2.2 Use Case Outline 15
2.3 Screen Design 17
2.3.1 User Subsystem 18
2.3.2 Admin Subsystem 25
2.3.3 Group Subsystem 29
2.3.4 Project Subsystem 34
2.3.5 Message Subsystem 41
2.4 Technical Use Case Narratives 42
2.4.1 User Subsystem Technical Narratives 42
2.4.2 Admin Subsystem Technical Narratives 81
2.4.3 Group Subsystem Technical Narratives 115
2.4.4 Project Subsystem Technical Narratives 156
2.4.5 Message Subsystem Technical Narratives 195
2.5 Technical Context Diagram 203
2.5.1 User Subsystem Context Diagram 203
2.5.2 Admin Subsystem Context Diagram 204
2.5.3 Group Subsystem Context Diagram 205
2.5.4 Project Subsystem Context Diagram 206
2.5.5 Message Subsystem Context Diagram 207
2.6 Technical Primitive Level Diagram 208
2.6.1 User Subsystem Primitive Diagrams 208
2.6.2 Admin Subsystem Primitive Diagrams 218
2.6.3 Group Subsystem Primitive Diagrams 227
2.6.4 Project Subsystem Primitive Diagrams 236
2.6.4 Message Subsystem Primitive Diagrams 249

2.7 Sequence Diagram 251
2.7.1 User Subsystem Sequence Diagrams 251
2.7.2 Admin Subsystem Sequence Diagrams 260
2.7.3 Group Subsystem Sequence Diagrams 269
2.7.4 Project Subsystem Sequence Diagrams 278
2.7.5 Message Subsystem Sequence Diagrams 286
2.8 Updated Requirements List 288
2.8.1 Introduction 288
2.8.2 Updated Functional Requirements Lists 288
2.8.3 Updated Functional Requirements Lists Descriptions 291
2.8.3 Functional Requirement List Detailed Changes Description 421
2.8.5 Conclusion 427
2.9 Technical Use Case Diagram 428
2.9.1 User Subsystem 428
2.9.2 Admin Subsystem 429
2.9.3 Group Subsystem 430
2.9.4 Project Management Subsystem 431
2.9.5 Message Subsystem 432
2.10 Technical Entity Relationship Diagram 433
2.11 State Diagram 434
2.11.1 Projects Controller 434
2.11.2 Auth Controller 435
2.11.3 User Controller 436
2.11.4 Project Role Repository 437
2.11.5 Group Repository 438
3. Document Conclusion 439
4. Sign-off by Client 440
5. Sign-off by Team 441

Table of Tables

Table 1: 1.1 User Login Technical Use Case Narrative 46
Table 2: 1.2 User Logout Technical Use Case Narrative 50
Table 3: 1.3 Reset Password Technical Use Case Narrative 55
Table 4: 1.4 Create User Technical Use Case Narrative 63
Table 5: 1.5 Update User Technical Use Case Narrative 68
Table 6: 1.6 View User Technical Use Case Narrative 72
Table 7: 1.7 Reactivate User Technical Use Case Narrative 76
Table 8: 1.8 Deactivate User Technical Use Case Narrative 80
Table 9: 2.1 Create User Profile Technical Use Case Narrative 84
Table 10: 2.2 Update User Profile Technical Use Case Narrative 88
Table 11: 2.3 View User Profile Technical Use Case Narrative 92
Table 12: 2.4 Deactivate User Technical Use Case Narrative 96
Table 13: 2.8 Create Role Technical Use Case Narrative 100
Table 14: 2.9 View Role Technical Use Case Narrative 103
Table 15: 2.10 Update Role Technical Use Case Narrative 107
Table 16: 2.11 Reactivate Role Technical Use Case Narrative 110
Table 17:2.12 Deactivate Role Technical Use Case Narrative 114
Table 18: 4.1 Create Group Technical Use Case Narrative 120
Table 19: 4.2 View Groups Technical Use Case Narrative 125
Table 20: 4.3 Update Group Technical Use Case Narrative 131
Table 21: 4.4 Deactivate Group Technical Use Case Narrative 135
Table 22: 4.5 Create Skill Technical Use Case Narrative 140
Table 23: 4.6 View Skills Technical Use Case Narrative 143
Table 24: 4.7 View Skill Technical Use Case Narrative 146
Table 25: 4.8 Update Skill Technical Use Case Narrative 151
Table 26: 4.9 Deactivate Skill Technical Use Case Narrative 155
Table 27:7.9 Add Indemnity Form Technical Use Case Narrative 184
Table 28: 7.10 Update Indemnity Form Technical Use Case Narrative 187
Table 29: 7.11 View Indemnity Form Technical Use Case Narrative 189
Table 30: 7.12 Remove Indemnity Form Technical Use Case Narrative 191
Table 31: 7.13 Capture Indemnity Form Technical Use Case Narrative 194
Table 32: 11.1 Create Message Technical Use Case Narrative 197
Table 33: 11.2 View Message Technical Use Case Narrative 202
Table 35: 1.1 User Login – Description 292
Table 36: 1.2 User Logout Description 292
Table 37: 1.3 Reset Password – Description 294
Table 38: 1.4 Create User – Description 295
Table 39: 1.5 Update User – Description 296
Table 40: 1.6 View User – Description 297

Table 41: 1.7 Reactivate User - Description 298
Table 42: 1.8 Deactivate User - Description 299
Table 43: 1.9 View FAQ - Description 300
Table 44: 2.1 Create User Profile - Description 301
Table 45: 2.2 Update User Profile - Description 302
Table 46: 2.3 View User Profile - Description 303
Table 47: 2.4 Deactivate User Profile - Description 304
Table 48: 2.5 Add FAQ – Description 305
Table 49: 2.6 Update FAQ – Description 306
Table 50: 2.7 Remove FAQ – Description 307
Table 51: 2.8 Create Role – Description 308
Table 52: 2.9 View Role – Description 309
Table 53: 2.10 Update Role – Description 310
Table 54:2.11 Reactivate Role – Description 311
Table 55: 2.12 Deactivate Role – Description 312
Table 56: 2.13 Update Role Permission – Description 313
Table 57: 2.14 View Role Permission – Description 314
Table 58: 3.1 Receive Inventory Item – Description 315
Table 59: 3.2 View Inventory – Description 316
Table 60: 3.3 Write-Off Inventory - Description 317
Table 61: 3.4 Allocate Inventory – Description 318
Table 62: 3.5 Create Inventory Item Type – Description 319
Table 63: 3.6 View Inventory Item Type - Description 320
Table 64:3.7 View Inventory Item Types – Description 321
Table 65: 3.8 Update Inventory Item Type – Description 322
Table 66: 3.9 Deactivate Inventory Item Type - Description 323
Table 67: 4.1 Create Group – Description 324
Table 68:4.2 View Group – Description 325
Table 69: 4.3 Update Group – Description 326
Table 70: 4.4 Deactivate Group - Description 327
Table 71: 4.5 Create Skill - Description 328
Table 72: 4.6 View Skills - Description 329
Table 73: View Skill - Description 330
Table 74: 4.7 Update Skill - Description 331
Table 75: 4.8 Deactivate Skill - Description 332
Table 76 :5.1 Create Warehouse - Description 333
Table 77: 5.2 View Warehouse - Description 334
Table 78: 5.3 Update Warehouse - Description 335
Table 79 5.4: Reactivate Warehouse - Description 336
Table 80: 5.5 Deactivate Warehouse - Description 337

Table 81: Inventory Stock Take - Description 338
Table 82: 6.1 Create Vendor - Description 339
Table 83 6.2: View Vendor - Description 340
Table 84: 6.3 View Vendors - Description 341
Table 85: 6.4 Edit Vendor - Description 342
Table 86: 6.5 Reactivate Vendor - Description 343
Table 87: 6.6 Deactivate Vendor - Description 344
Table 88: 6.7 Create Vendor Type - Description 345
Table 89: 6.8 Update Vendor Type - Description 346
Table 90: 6.8 View Vendor Types - Description 347
Table 91: 6.10 Reactivate Vendor Type - Description 348
Table 92: 6.11 Deactivate Vendor Type - Description 349
Table 93: 7.1 Create Project - Description 350
Table 94: 7.2 View Project - Description 351
Table 95: 7.3 Update Project - Description 352
Table 96: 7.4 Deactivate Project - Description 353
Table 97: 7.5 Create Project Type - Description 354
Table 98: 7.6 View Project Type - Description 355
Table 99: 7.7 Update Project Type - Description 356
Table 100: 7.8 Deactivate Project Type - Description 357
Table 101: 7.9 Add Indemnity Form - Description 358
Table 102: 7.10 Update Indemnity Form - Description 359
Table 103: 7.11 View Indemnity Form - Description 360
Table 104: 7.12 Remove Indemnity Form - Description 361
Table 105: 7.13 Capture Indemnity Form - Description 362
Table 106: 7.14 Capture Project Feedback - Description 363
Table 107: 7.15 View Project Feedback - Description 364
Table 108: 7.16 Create Impact Feedback Form - Description 365
Table 109: 7.17 View Impact Forms - Description 366
Table 110: 7.18 Update Impact Form - Description 367
Table 111: 7.19 Deactivate Impact Form - Description 368
Table 112: 7.20 Capture Impact Form - Description 369
Table 113: 7.21 View Impact Feedback - Description 370
Table 114: 7.22 Add Photo - Description 371
Table 115: 7.23 View Photo - Description 372
Table 116: 7.24 Delete Photo - Description 373
Table 117: 8.1 Add Dog - Description 374
Table 118: 8.2 View Dog - Description 375
Table 119: 8.3 View All Dogs- Description 376
Table 120: 8.4 Update Dog - Description 377

Table 121: 8.5 Deactivate Dog - Description 378
Table 122: 8.6 Assign User to Dog - Description 379
Table 123: 8.7 View User-Dog Assignment – Description 380
Table 124: 8.8 Update User-Dog Assignment - Description 381
Table 125: 8.9 Reactivate User-Dog Assignment - Description 382
Table 126: 8.10 Deactivate User-Dog Assignment - Description 383
Table 127: 8.11 Add K9 Role - Description 384
Table 128: 8.12 Update K9 Role - Description 385
Table 129: 8.13 View K9 Role - Description 386
Table 130: 8.14 Reactivate K9 Role - Description 387
Table 131: 8.15 Deactivate K9 Role – Description 388
Table 132: 8.16 View K9 Ancestry - Description 389
Table 133: 9.1 Add Rhino - Description 390
Table 134:9.2 View Rhino - Description 391
Table 135: 9.3 View Rhinos - Description 392
Table 136: 9.4 Update Rhino - Description 393
Table 137: 9.5 9.5 Deactivate Rhino - Description 394
Table 138: 10.1 Add Elephant - Description 395
Table 139: 10.2 View Elephant - Description 396
Table 140: 10.3 View Elephants - Description 397
Table 141: 10.4 Update Elephant - Description 398
Table 142: 10.5 Deactivate Elephant - Description 399
Table 143: 11.1 Create Message – Description 400
Table 144: 11.2 View Message - Description 401
Table 145: 12.1 Generate Custom Report - Description 402
Table 146: 12.2 View Custom Report - Description 403
Table 147: 12.3 Update Custom Report - Description 404
Table 148: 12.4 Remove Custom Report - Description 405
Table 149: 12.5 View Bike Impact Report - Description 406
Table 150: 12.6 View Budget Report – Description 407
Table 151: 12.7 View Rhino Report - Description 408
Table 152: 12.8 View Elephant Report - Description 409
Table 153: 12.9 View K9 Report - Description 410
Table 154: 13.1 View Machine Learning Dashboard - Description 411
Table 155: 14.1 Backup System - Description 412
Table 156: 14.2 Restore System - Description 413
Table 157: 14.3 View Audit Log - Description 414
Table 158: 15.1 View Project Budget - Description 415
Table 159: 15.2 View Project Budgets - Description 416
Table 160: 15.3 Log An Expense - Description 417

Table 161: 15.4 Update An Expense - Description 418
Table 162: 15.5 View An Expense - Description 419
Table 163: 15.6 Remove An Expense – Description 420
Table 164: User Subsystem Modifications 421
Table 165: Admin Subsystem Modifications 421
Table 166 Inventory Subsystem Modifications 422
Table 167 Group Subsystem Modifications 422
Table 168: Warehouse Subsystem Modifications 422
Table 169 Vendor Subsystem Modifications 423
Table 170 Vendor Subsystem Modifications 424
Table 171 K9 Subsystem Modifications 424
Table 172 Rhino Subsystem Modifications 425
Table 173 Elephant Subsystem Modifications 425
Table 174 Message Subsystem Modifications 425
Table 175: Report Subsystem Modifications 426
Table 176: Machine Learning Subsystem Modifications 426
Table 177: System Subsystem Modifications 426
Table 178 Budget Subsystem Modifications 427

Table of Figures

Figure 1: User Login Screen 18
Figure 2: User Logout Screen 18
Figure 3: Reset Password Screen 1 19
Figure 4: Reset Password Screen 2 19
Figure 5: Reset Password Screen 3 20
Figure 6: Sign Up Screen 20
Figure 7: Onboarding Details Screen 21
Figure 8: Onboarding Skills Screen 21
Figure 9: Onboarding Status Screen 22
Figure 10 Update Profile Modal 22
Figure 11: User Profile Screen 23
Figure 12: Reactivate Account Screen 23
Figure 13: Welcome Back Screen 24
Figure 14: Deactivate Confirm Modal 24
Figure 15: Create User Modal 25
Figure 16: Update User Profile Modal 25
Figure 17: View User Profile Screen 26
Figure 18: Deactivate User Profile Screen 26
Figure 19: Create Role Screen 27
Figure 20: View Role Screen 27
Figure 21: Update Role Screen 28
Figure 22: Reactivate Role Screen 28
Figure 23: Deactivate Role Screen 29
Figure 24: Create Group Modal 29
Figure 25: View Groups Modal 30
Figure 26: Update Group Modal 30
Figure 27: Deactivate Group Modal 31
Figure 28: Create Skill Modal 31
Figure 29: View Skills Screen 32
Figure 30: Update Skill Modal 32
Figure 31: Deactivate Skill Screen 33
Figure 32: Create Project Screen 34
Figure 33: View Project Screen 34
Figure 34: Update Project Screen 35
Figure 35: Deactivate Project Screen 35
Figure 36: Create Project Type Screen 36
Figure 37: View Project Type Screen 36
Figure 38: Update Project Type Screen 37
Figure 39: Deactivate Project Type Screen 37

Figure 40: Add Indemnity Form Screen 38
Figure 41: Update Indemnity Form Screen 38
Figure 42: View Indemnity Form Screen 39
Figure 43: Remove Indemnity Form Screen 39
Figure 44: Capture Indemnity Form Screen 40
Figure 45: Message Screen 41
Figure 46: No Group Selected Screen 41
Figure 47: User Subsystem Technical Context Diagram 203
Figure 48: Admin Subsystem Technical Context Diagram 204
Figure 49: Group Subsystem Technical Context Diagram 205
Figure 50: Project Subsystem Technical Context Diagram 206
Figure 51: Message Subsystem Technical Context Diagram 207
Figure 52: 1.1 User Login Technical Primitive Diagram 208
Figure 53: 1.2 User Logout Technical Primitive Diagram 209
Figure 54: 1.3 Reset Password Technical Primitive Diagram 210
Figure 55: 1.4 Create User Technical Primitive Diagram 211
Figure 56: 1.4 Create User Technical Primitive Diagram (Continued) 212
Figure 57: 1.4 Create User Technical Primitive Diagram (Continued) 213
Figure 58: 1.5 Update User Technical Primitive Diagram 214
Figure 59: 1.6 View User Technical Primitive Diagram 215
Figure 60: 1.7 Reactivate User Technical Primitive Diagram 216
Figure 61: 1.8 Deactivate User Technical Primitive Diagram 217
Figure 62: 2.1 Create User Profile Technical Primitive Diagram 218
Figure 63: 2.2 Update User Profile Technical Primitive Diagram 219
Figure 64: 2.3 View User Profile Technical Primitive Diagram 220
Figure 65: 2.4 Deactivate User Technical Primitive Diagram 221
Figure 66: 2.8 Create Role Technical Primitive Diagram 222
Figure 67:2.9 View Role Technical Primitive Diagram 223
Figure 68: 2.10 Update Role Technical Primitive Diagram 224
Figure 69: 2.11 Reactivate Role Technical Primitive Diagram 225
Figure 70:2.12 Deactivate Role Technical Primitive Diagram 226
Figure 71: 4.1 Create Group Technical Primitive Diagram 227
Figure 72: 4.2 View Groups Technical Primitive Diagram 228
Figure 73: 4.3 Update Group Technical Primitive Diagram 229
Figure 74: 4.4 Deactivate Group Technical Primitive Diagram 230
Figure 75: 4.5 Create Skill Technical Primitive Diagram 231
Figure 76: 4.6 View Skills Technical Primitive Diagram 232
Figure 77: 4.7 View Skill Technical Primitive Diagram 233
Figure 78: 4.8 Update Skill Technical Primitive Diagram 234
Figure 79: 4.9 Deactivate Skill Technical Primitive Diagram 235
Figure 80: Add Indemnity Form Technical Primitive Diagram 244

Figure 81: Update Indemnity Form Technical Primitive Diagram 245
Figure 82: View Indemnity Form Technical Primitive Diagram 246
Figure 83: Remove Indemnity Form Technical Primitive Diagram 247
Figure 84: Capture Indemnity Form Technical Primitive Diagram 248
Figure 85: 11.1 Create Message Technical Primitive Diagram 249
Figure 86: 11.2 View Message Technical Primitive Diagram 250
Figure 87: 1.1 User Login Sequence Diagram 251
Figure 88: 1.2 User Logout Sequence Diagram 252
Figure 89: 1.3 Password Reset Sequence Diagram 253
Figure 90: 1.4 Create User Sequence Diagram 254
Figure 91: 1.4 Create User Sequence Diagram (Continued) 255
Figure 92: 1.5 Update User Sequence Diagram..... 256
Figure 93: 1.6 View User Sequence Diagram 257
Figure 94: 1.7 Reactivate User Sequence Diagram..... 258
Figure 95: 1.8 Deactivate User Sequence Diagram 259
Figure 96: 2.1 Create User Profile Sequence Diagram 260
Figure 97: 2.2 Update User Profile Sequence Diagram 261
Figure 98: 2.3 View User Profile Sequence Diagram 262
Figure 99: 2.4 Deactivate User Sequence Diagram 263
Figure 100:2.8 Create Role Sequence Diagram..... 264
Figure 101: 2.9 View Role Sequence Diagram..... 265
Figure 102: 2.10 Update Role Sequence Diagram 266
Figure 103: 2.11 Reactivate Role Sequence Diagram 267
Figure 104: 2.12 Deactivate Role Sequence Diagram 268
Figure 105: 4.1 Create Group Sequence Diagram 269
Figure 106: 4.2 View Groups Sequence Diagram 270
Figure 107: 4.3 Update Group Sequence Diagram 271
Figure 108: 4.4 Deactivate Group Sequence Diagram 272
Figure 109: 4.5 Create Skill Sequence Diagram 273
Figure 110: 4.6 View Skills Sequence Diagram..... 274
Figure 111: 4.6 View Skills Sequence Diagram..... 274
Figure 112: 4.7 View Skill Sequence Diagram 275
Figure 113: 4.8 Update Skill Sequence Diagram 276
Figure 114: 4.9 Deactivate Skill Sequence Diagram 277
Figure 115: 11.1 Create Message Sequence Diagram 286
Figure 116: 11.2 View Message Sequence Diagram 287
Figure 117: User Subsystem Technical Use Case Diagram 428
Figure 118: Admin Subsystem Technical Use Case Diagram 429
Figure 119: Group Subsystem Technical Use Case Diagram..... 430
Figure 120: Project Subsystem Technical Use Case Diagram 431
Figure 121: Message Subsystem Technical Use Case Diagram..... 432

Figure 122: Technical Entity Relationship Diagram 433
Figure 123: Project Controller State Diagram 434
Figure 124: Auth Controller State Diagram 435
Figure 125: User Controller State Diagram 436
Figure 126: Project Role Repository State Diagram 437
Figure 127: Group Repository State Diagram 438

1. Document Introduction

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

Iteration 4 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 4, technical ERD for the data elements involved in Iteration 4. 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, a sequence diagram and the state diagram for the complex classes.

We also included the complexity matrix, the link for our GitHub code and the project and meeting for Iteration 5.

2. Design and Development of the Use Cases

2.1 Introduction

This section of the document outlines the design and development of 40 use cases assigned to Iteration 4. 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, sequence diagrams, and state 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 128 distinct use cases, at least 30% of the use cases need to be completed.

$$128 \times 30\% = 38.4$$

We have decided to complete 40 use cases and based on the outcomes of the dependency matrices; we decided to complete the following use cases:

User Subsystem (Subsystem 1):

1.1 User Login

1.2 User Logout

1.3 Reset Password

1.4 Create User

1.5 Update User

1.6 View User

1.7 Reactivate User

1.8 Deactivate User

Admin Subsystem (Subsystem 2):

2.1 Create User Profile

2.2 Update User Profile

2.3 View User Profile

2.4 Deactivate User Profile

2.8 Create Role

2.9 View Role

2.10 Update Role

2.11 Reactivate Role

2.12 Deactivate Role

Group Subsystem (Subsystem 4):

4.1 Create Group

4.2 View Groups

4.3 Update Group

4.4 Deactivate Group

4.5 Create Skill

4.6 View Skills

4.7 Update Skill

4.8 Deactivate Skill

Project Subsystem (Subsystem 7):

7.1 Create Project

7.2 View Project

7.3 Update Project

7.4 Deactivate Project

7.5 Create Project Type

7.6 View Project Type

7.7 Update Project Type

7.8 Deactivate Project Type

7.9 Add Indemnity Form

7.10 Update Indemnity Form

7.11 View Indemnity Form

7.12 Remove Indemnity Form

7.13 Capture Indemnity Form

Message Subsystem (Subsystem 11):

11.1 Create Message

11.2 View Messages

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 User Login: Use Case 1.1

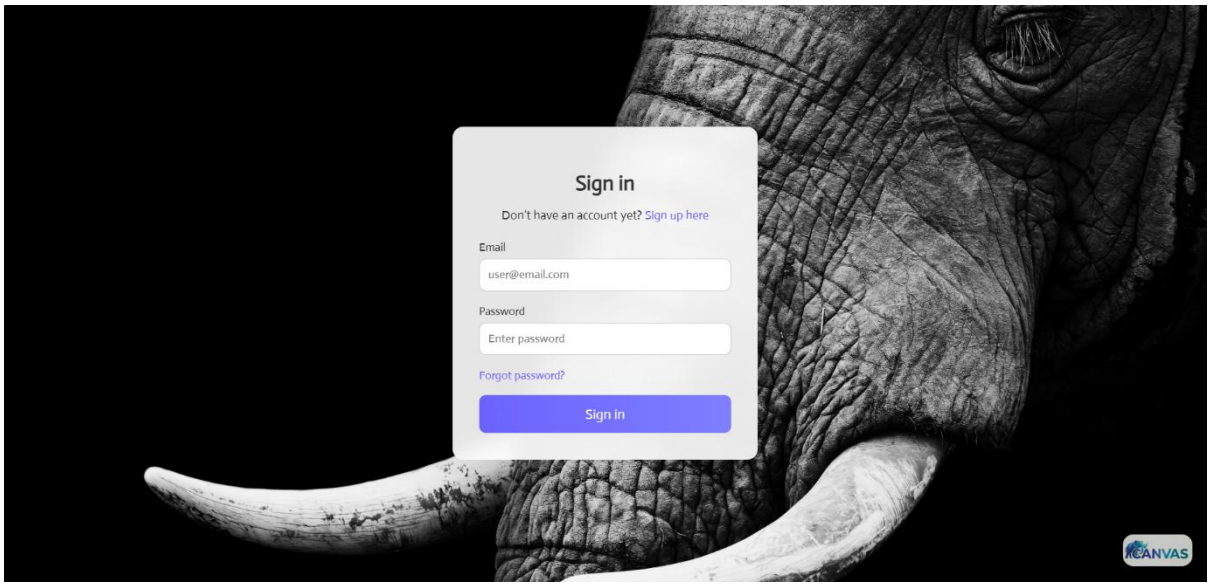


Figure 1: User Login Screen

2.3.1.2 User Logout: Use Case 1.2

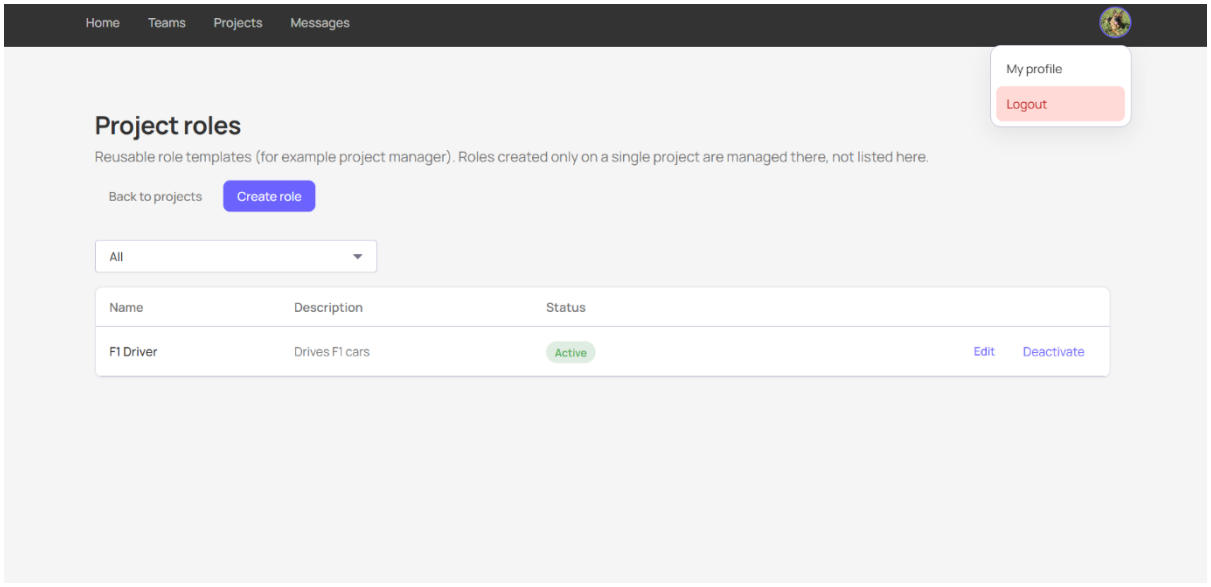


Figure 2: User Logout Screen

2.3.1.3 Reset Password: Use Case 1.3

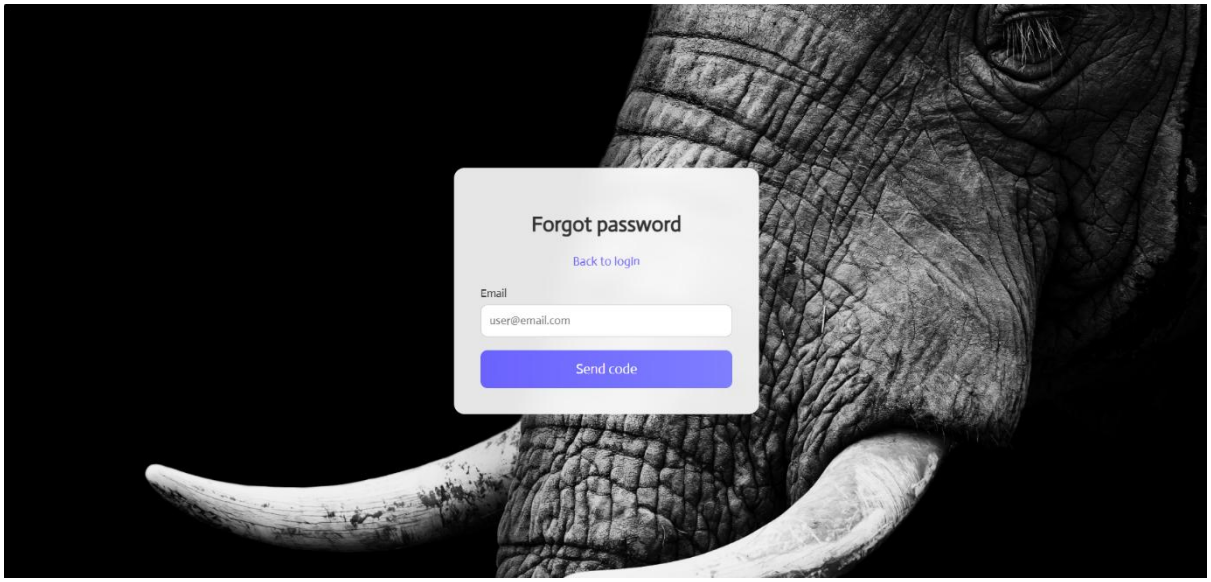


Figure 3: Reset Password Screen 1

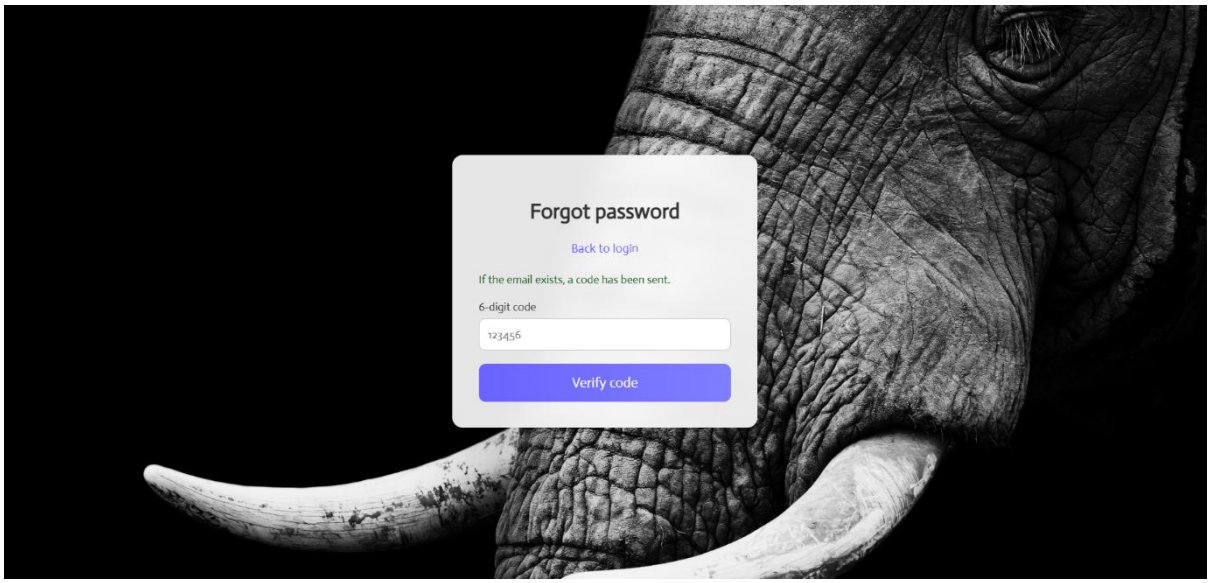


Figure 4: Reset Password Screen 2

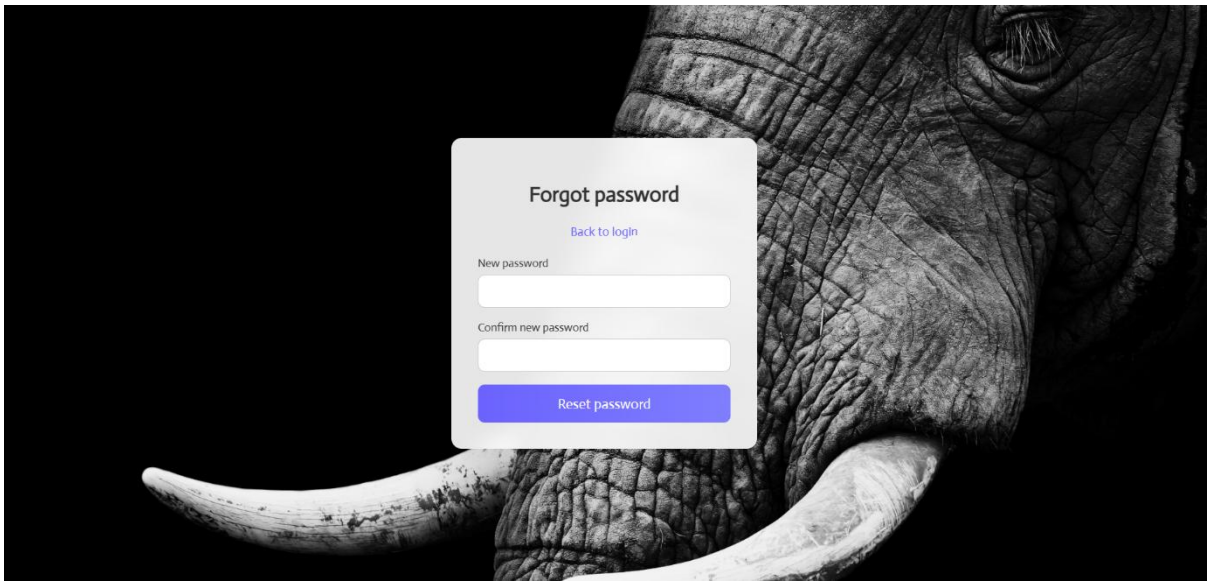


Figure 5: Reset Password Screen 3

2.3.1.4 Create User: Use Case 1.4

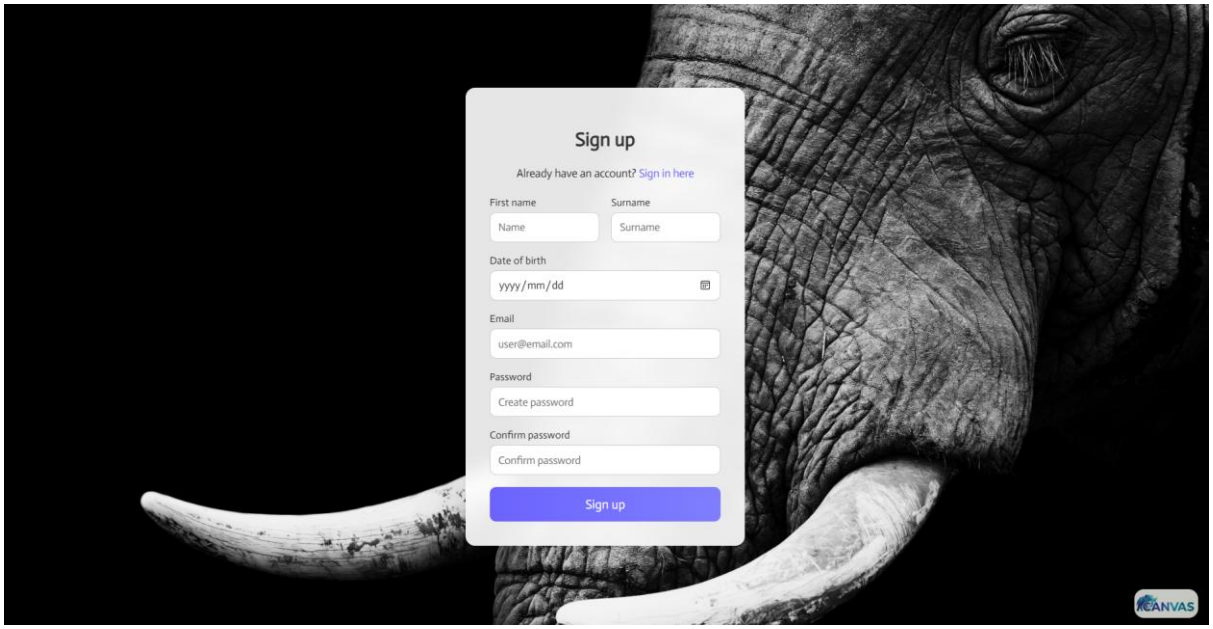


Figure 6: Sign Up Screen

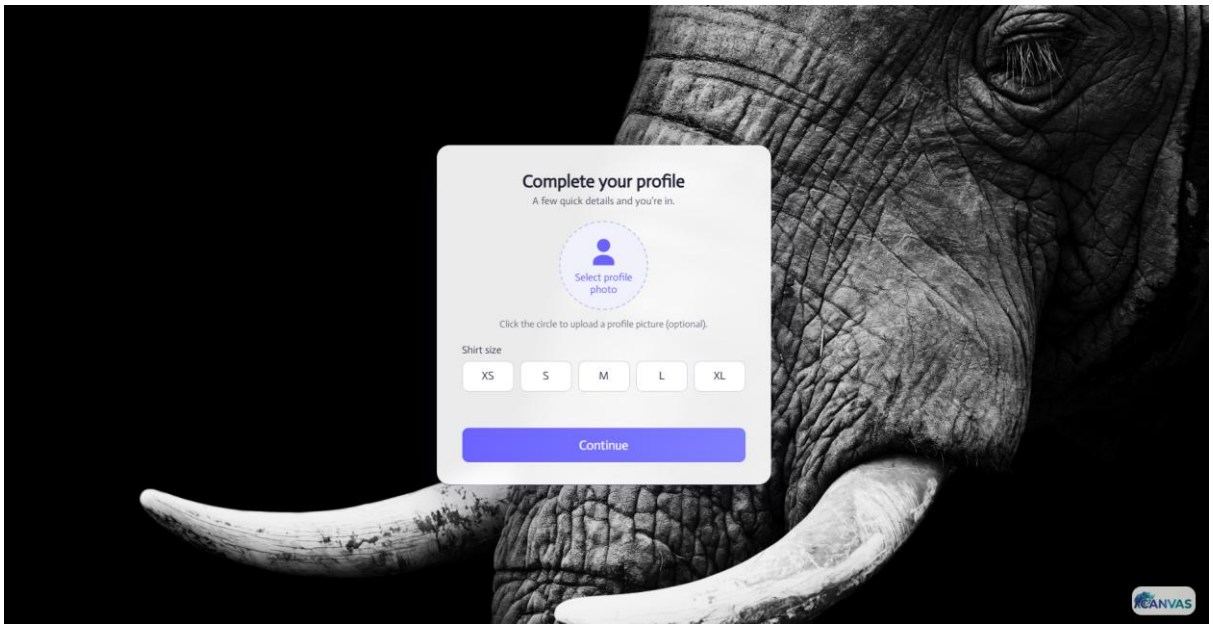


Figure 7: Onboarding Details Screen

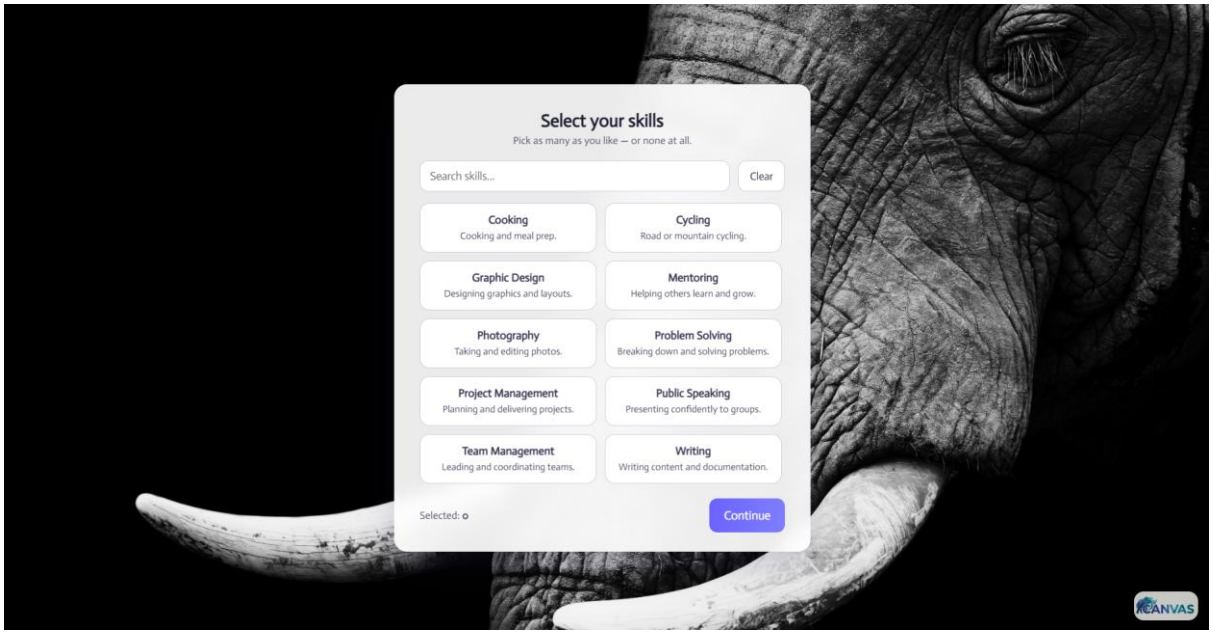


Figure 8: Onboarding Skills Screen

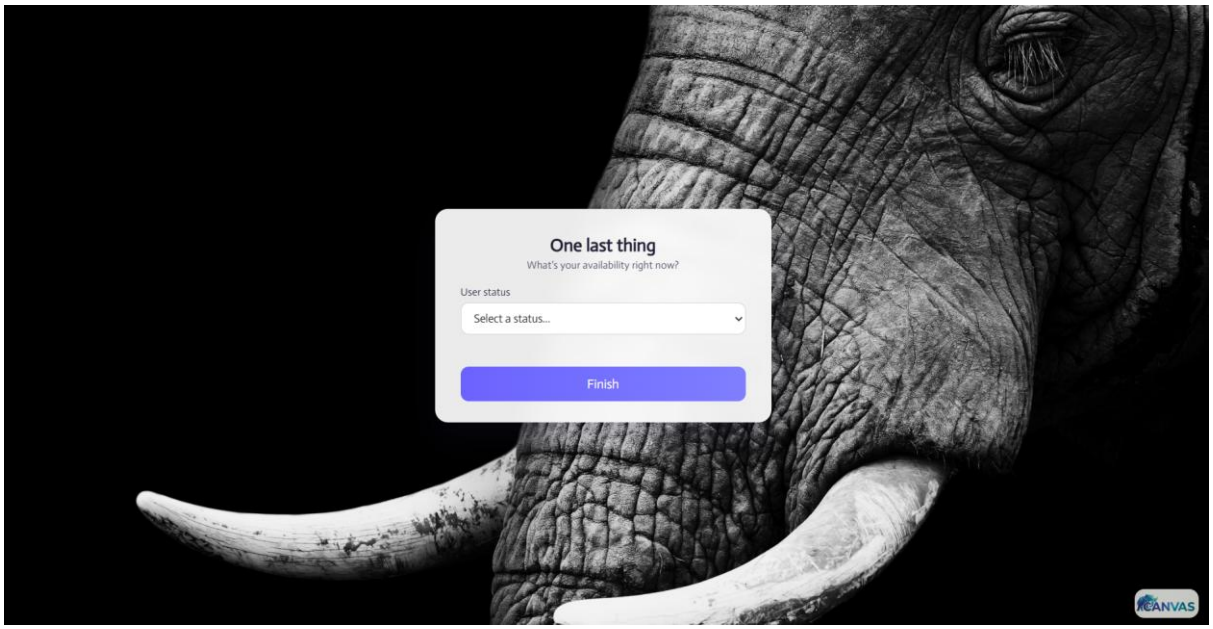


Figure 9: Onboarding Status Screen

2.3.1.5 Update User: Use Case 1.5

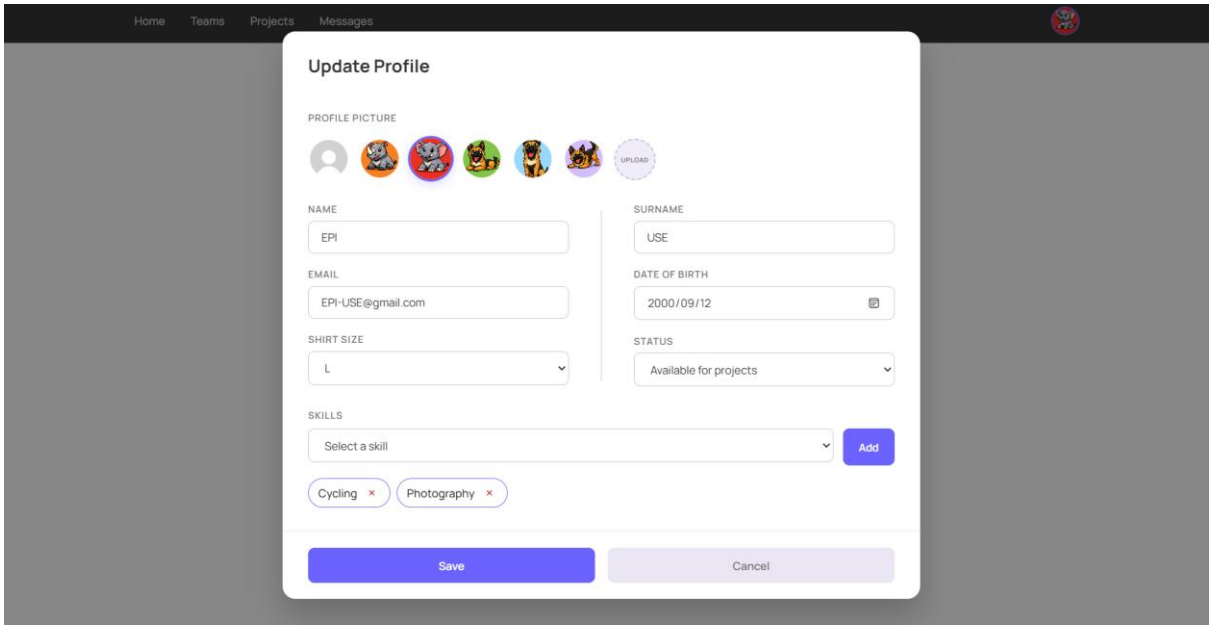


Figure 10 Update Profile Modal

2.3.1.6 View User: Use Case 1.6

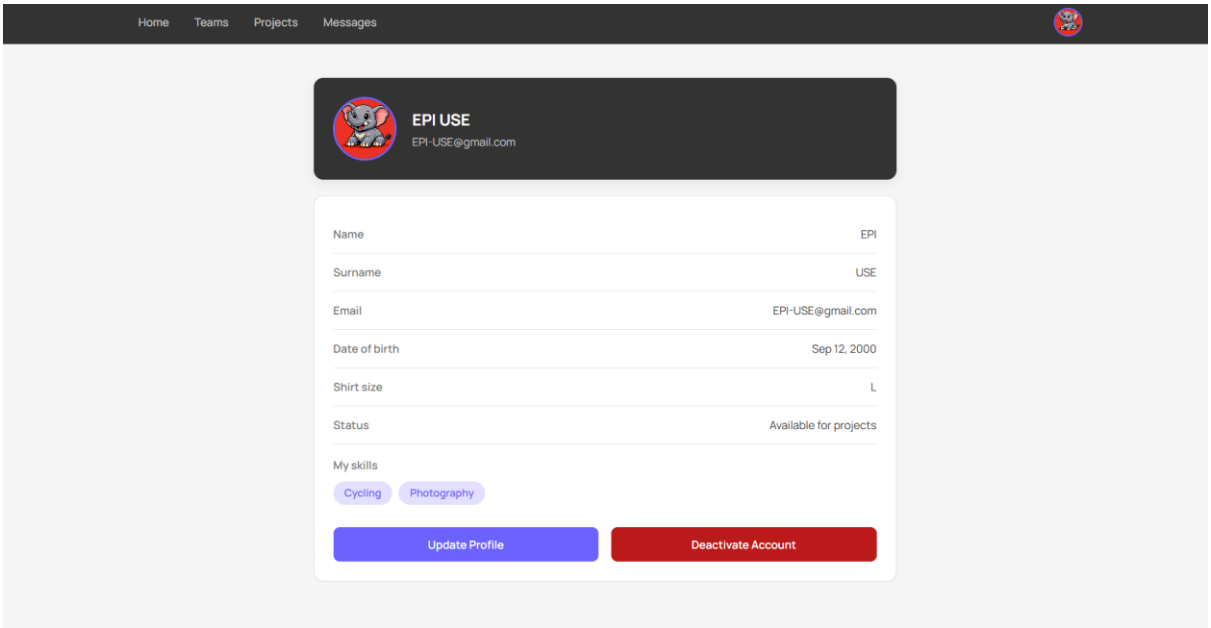


Figure 11: User Profile Screen

2.3.1.7 Reactivate User: Use Case 1.7

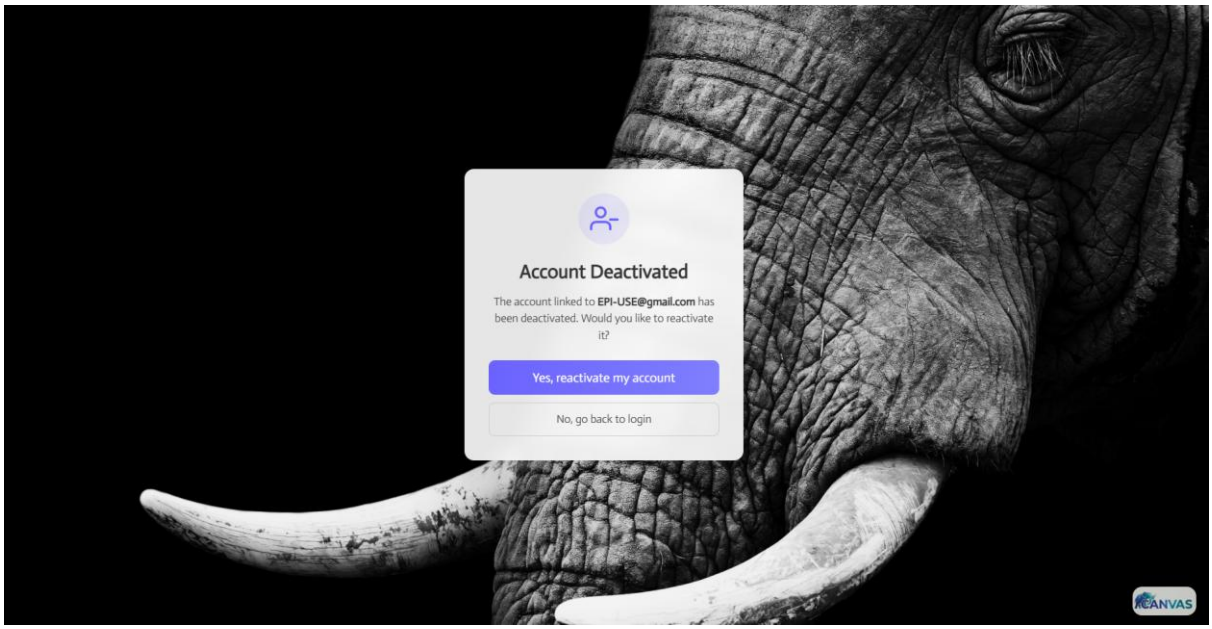


Figure 12: Reactivate Account Screen

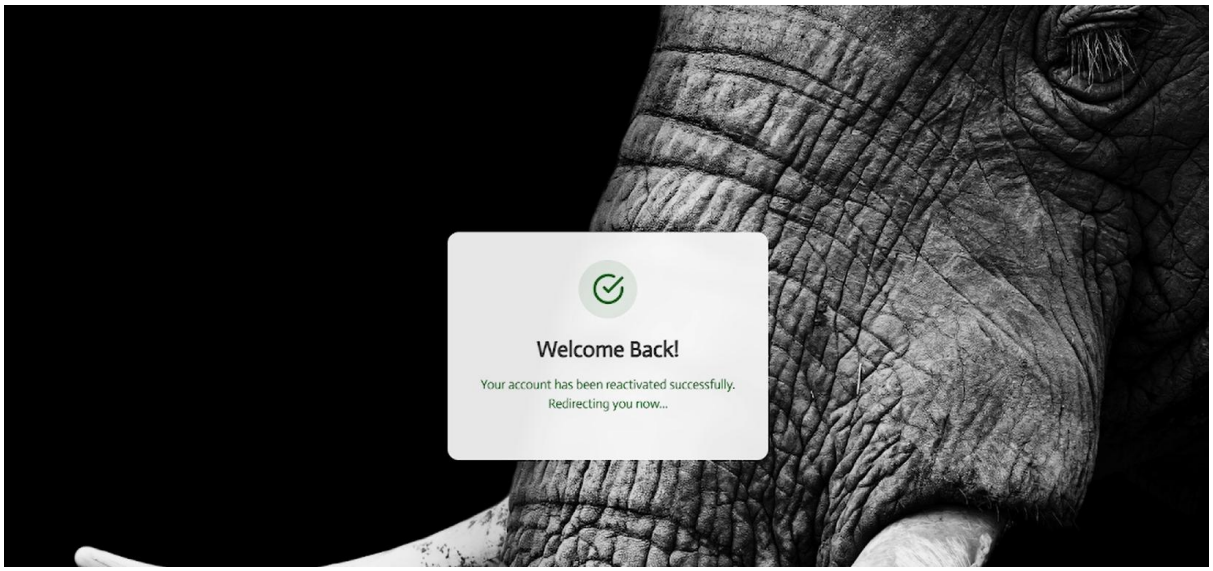


Figure 13: Welcome Back Screen

2.3.1.8 Deactivate User: Use Case 1.8

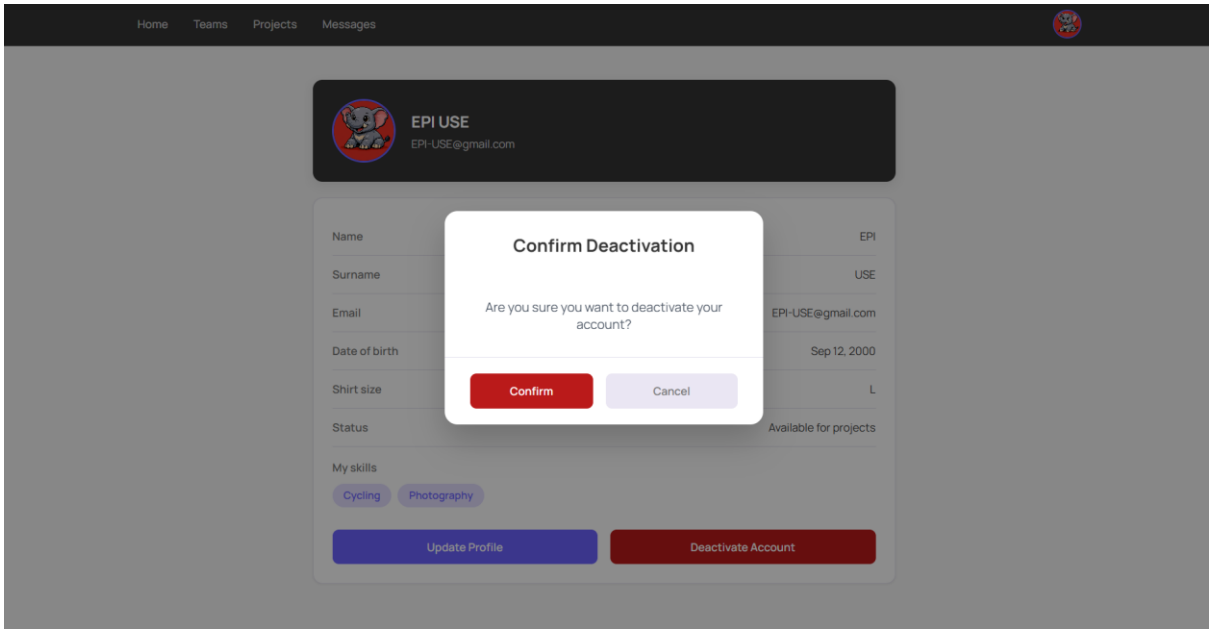


Figure 14: Deactivate Confirm Modal

2.3.2 Admin Subsystem

2.3.2.1 Create User Profile: Use Case 2.1

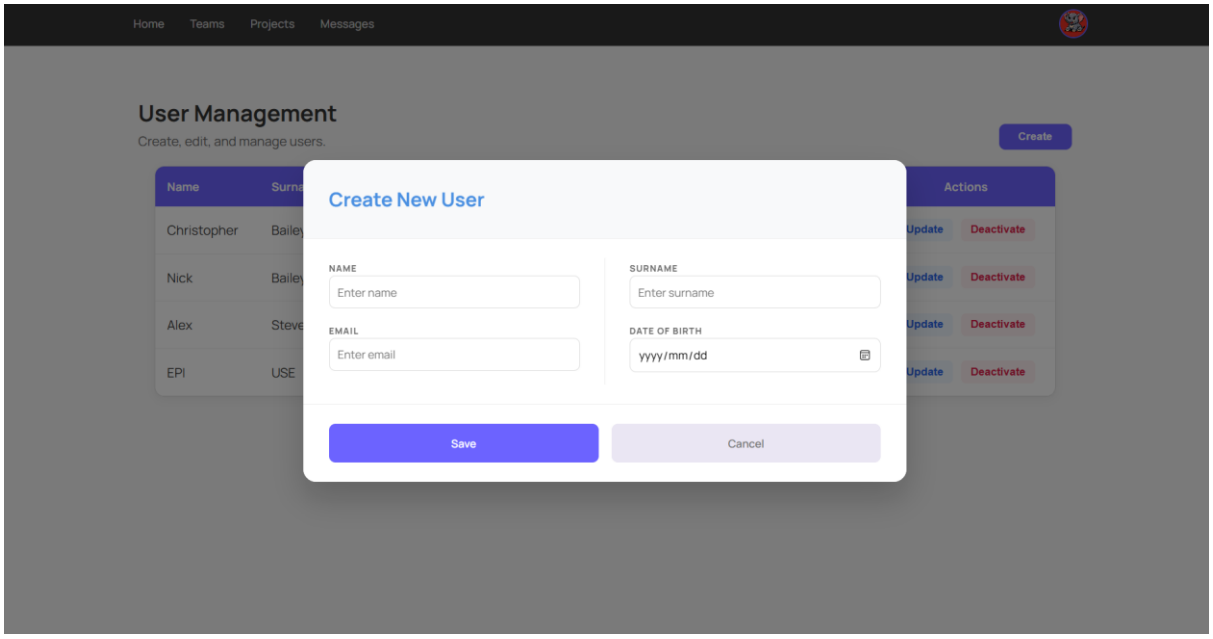


Figure 15: Create User Modal

2.3.2.2 Update User Profile: Use Case 2.2

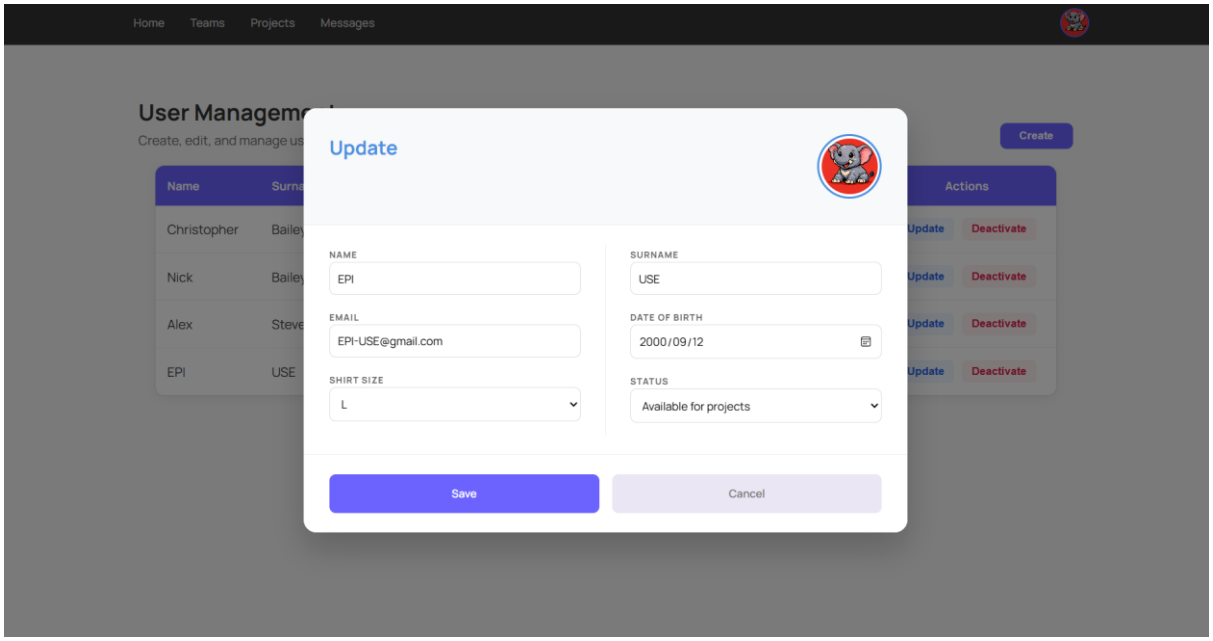


Figure 16: Update User Profile Modal

2.3.2.3 View User Profile: Use Case 2.3

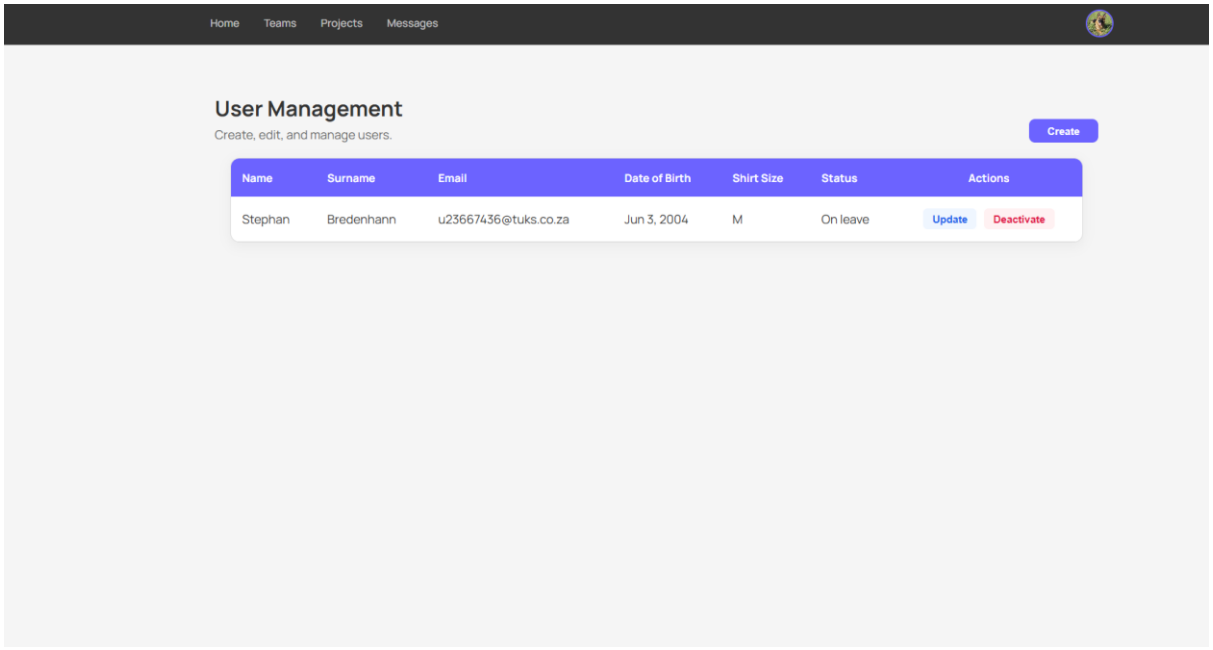


Figure 17: View User Profile Screen

2.3.2.4 Deactivate User Profile: Use Case 2.4

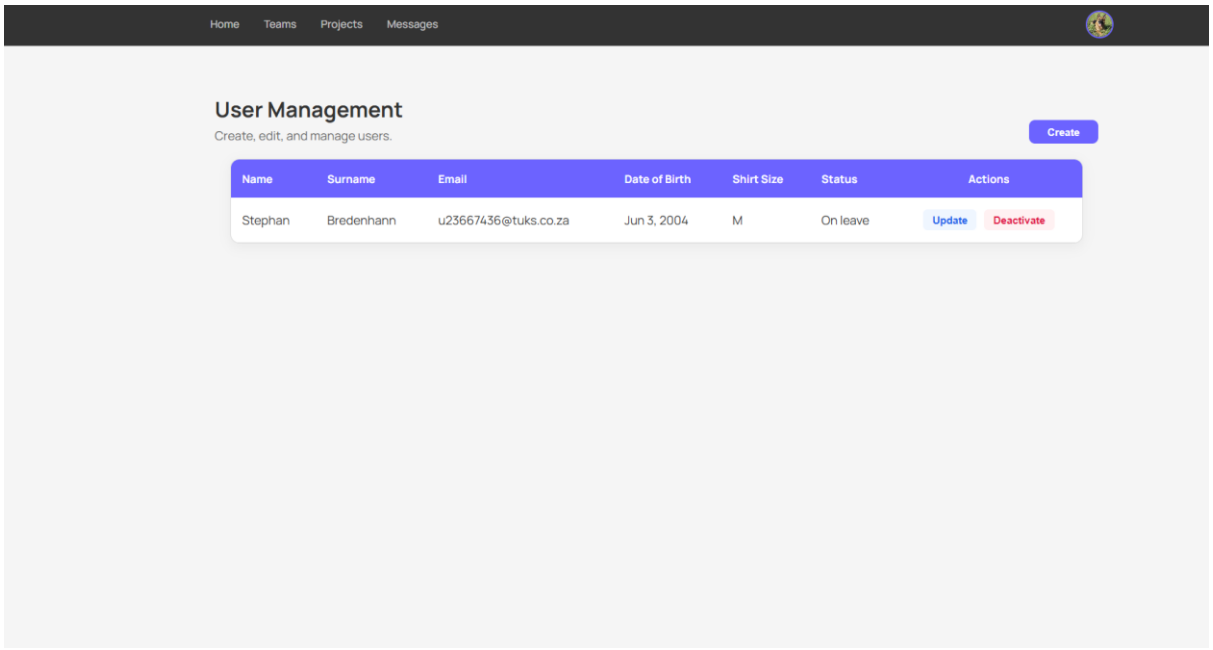


Figure 18: Deactivate User Profile Screen

2.3.2.5 Create Role: Use Case 2.8

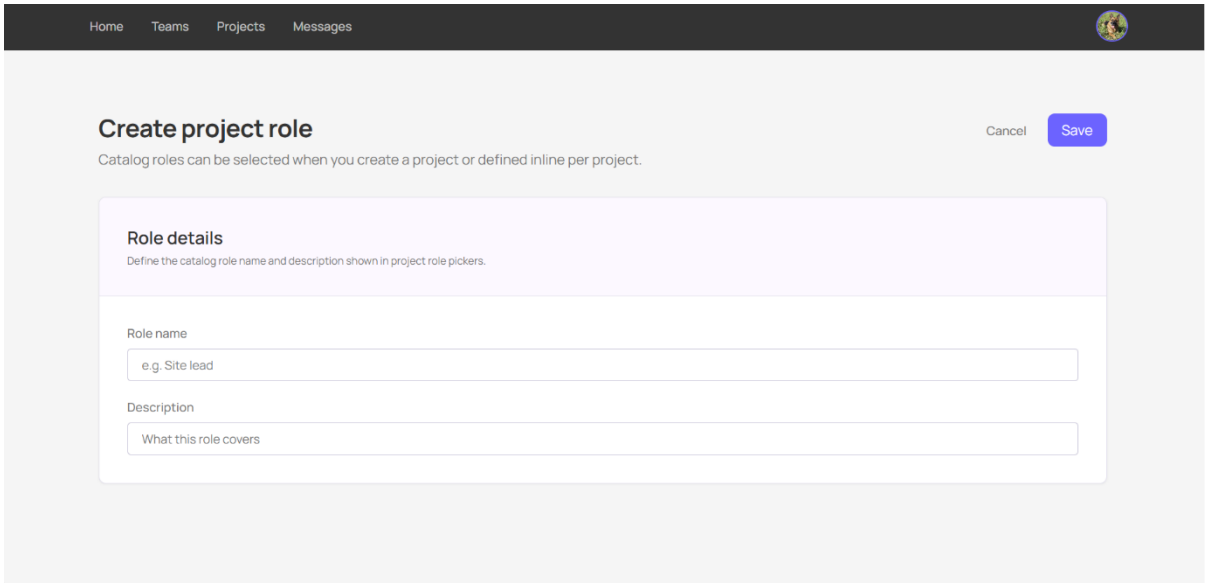


Figure 19: Create Role Screen

2.3.2.6 View Role: Use Case 2.9

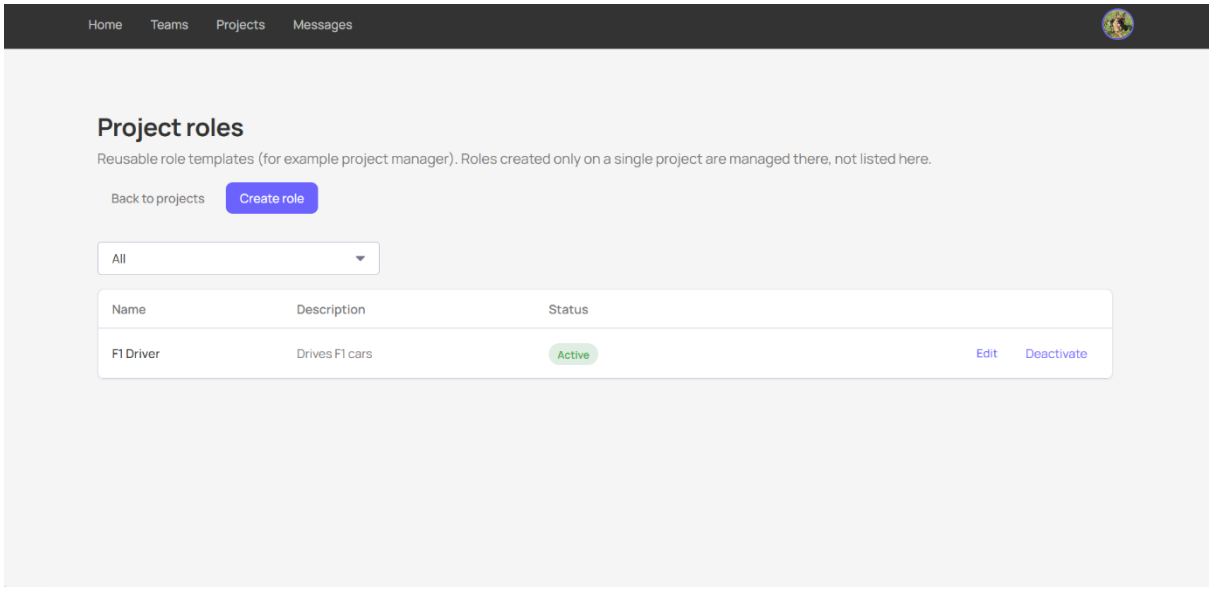


Figure 20: View Role Screen

2.3.2.7 Update Role: Use Case 2.10

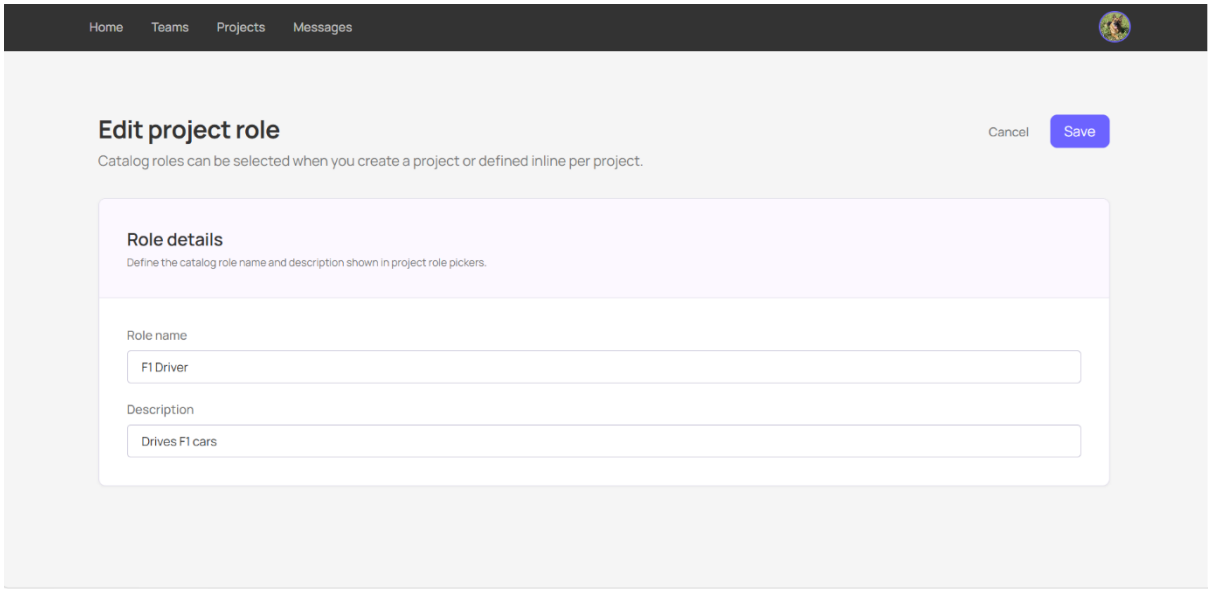


Figure 21: Update Role Screen

2.3.2.8 Reactivate Role: Use Case 2.11

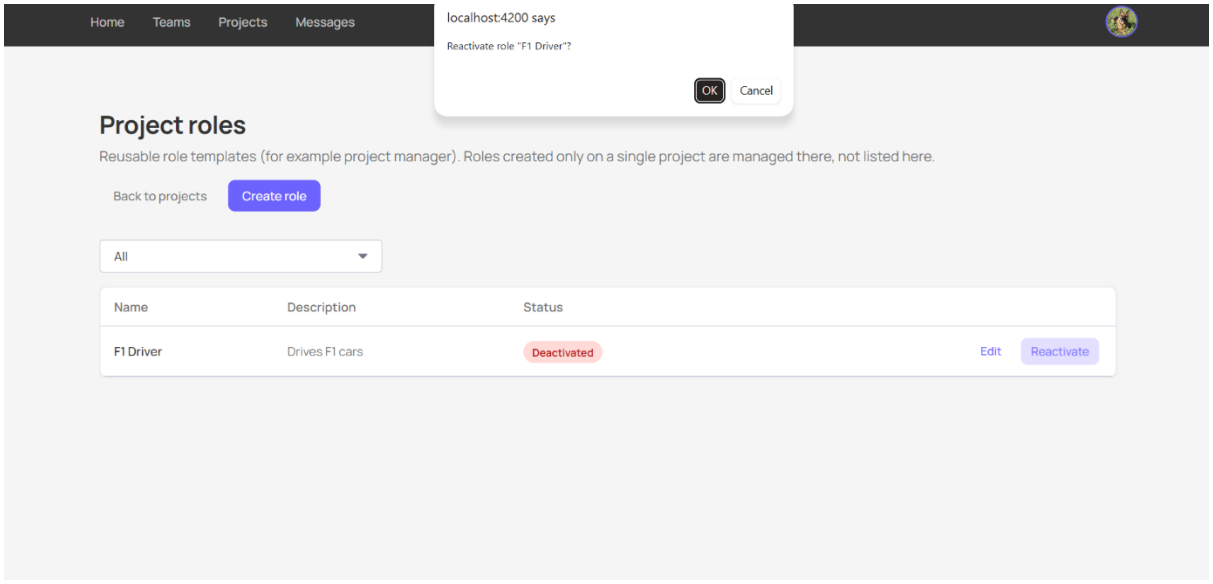


Figure 22: Reactivate Role Screen

2.3.2.8 Deactivate Role: Use Case 2.12

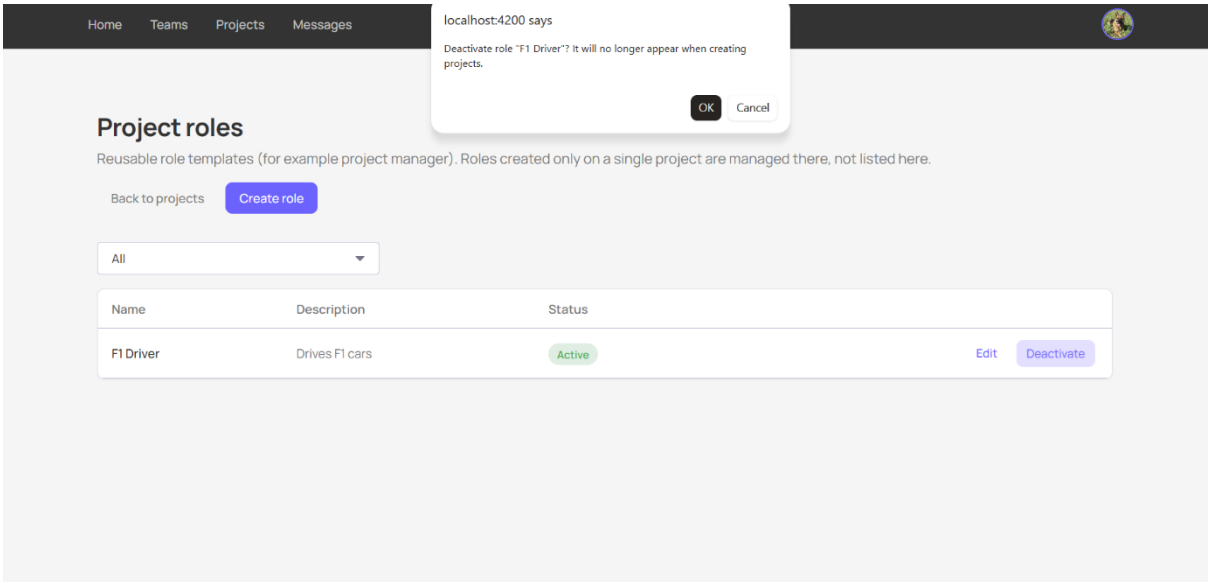


Figure 23: Deactivate Role Screen

2.3.3 Group Subsystem

2.3.3.1 Create Group: Use Case 4.1

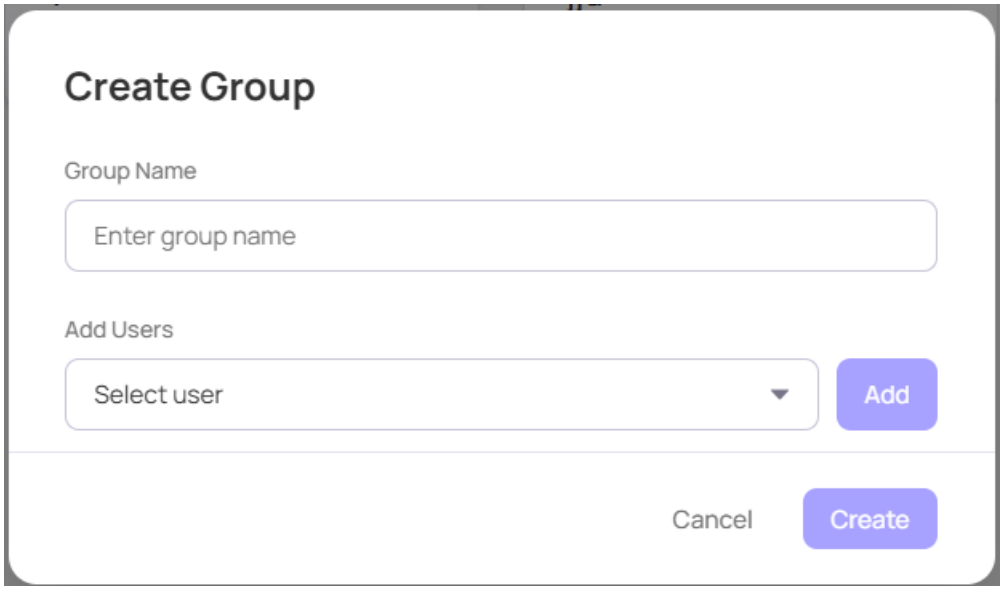


Figure 24: Create Group Modal

2.3.3.2 View Groups: Use Case 4.2

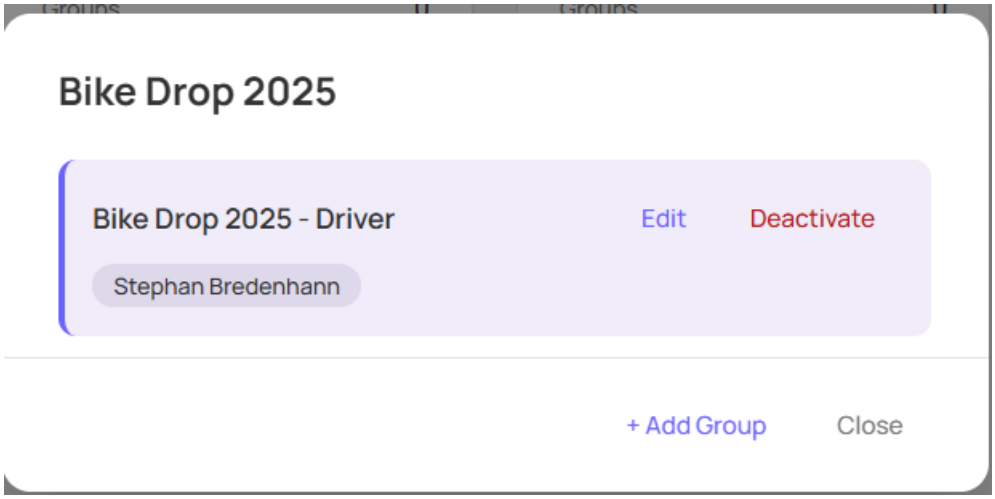


Figure 25: View Groups Modal

2.3.3.3 Update Group: Use Case 4.3

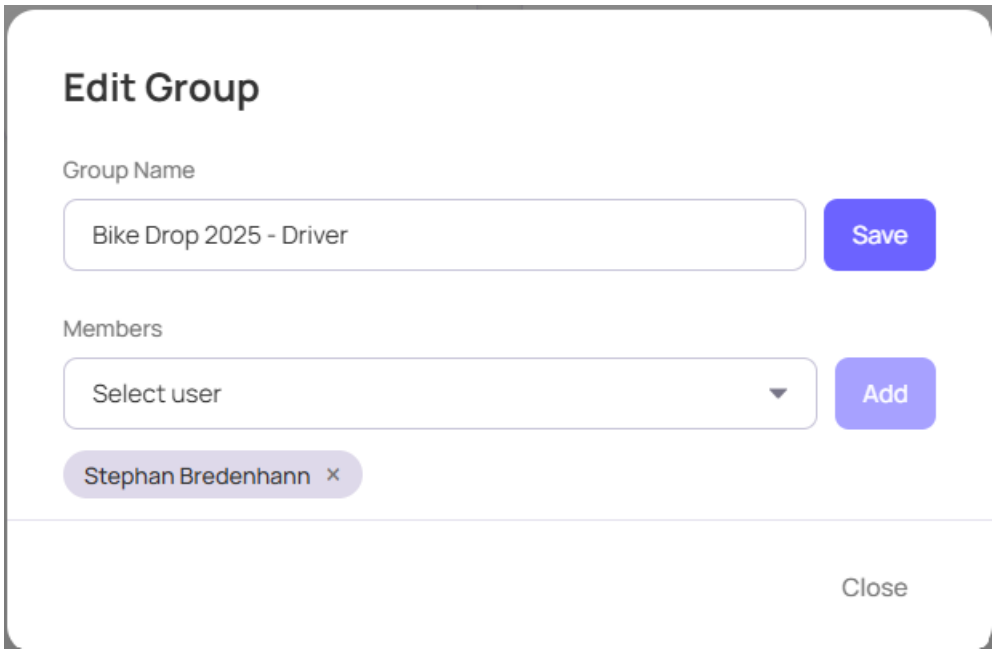


Figure 26: Update Group Modal

2.3.3.4 Deactivate Group: Use Case 4.4

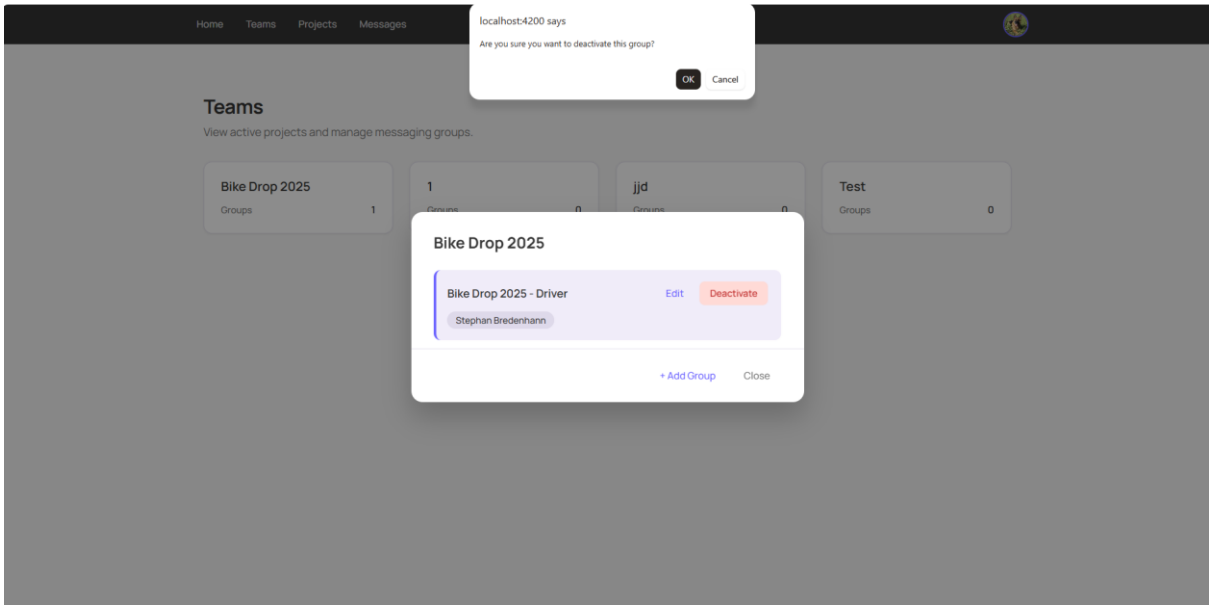


Figure 27: Deactivate Group Modal

2.3.3.5 Create Skill: Use Case 4.5

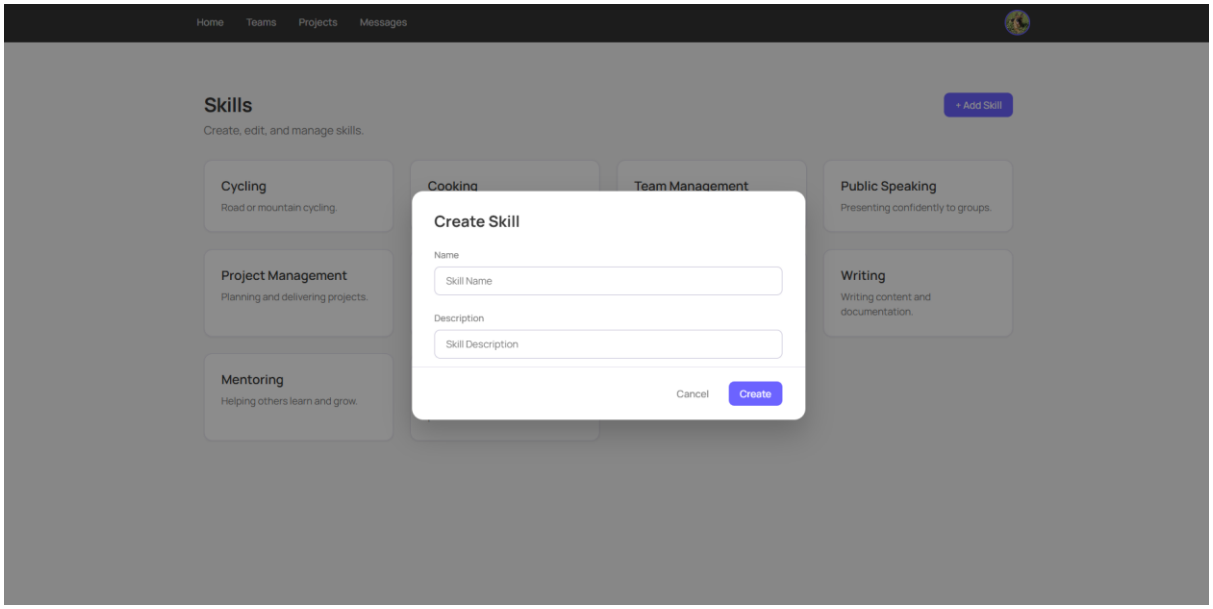


Figure 28: Create Skill Modal

2.3.3.6 View Skills: Use Case 4.6

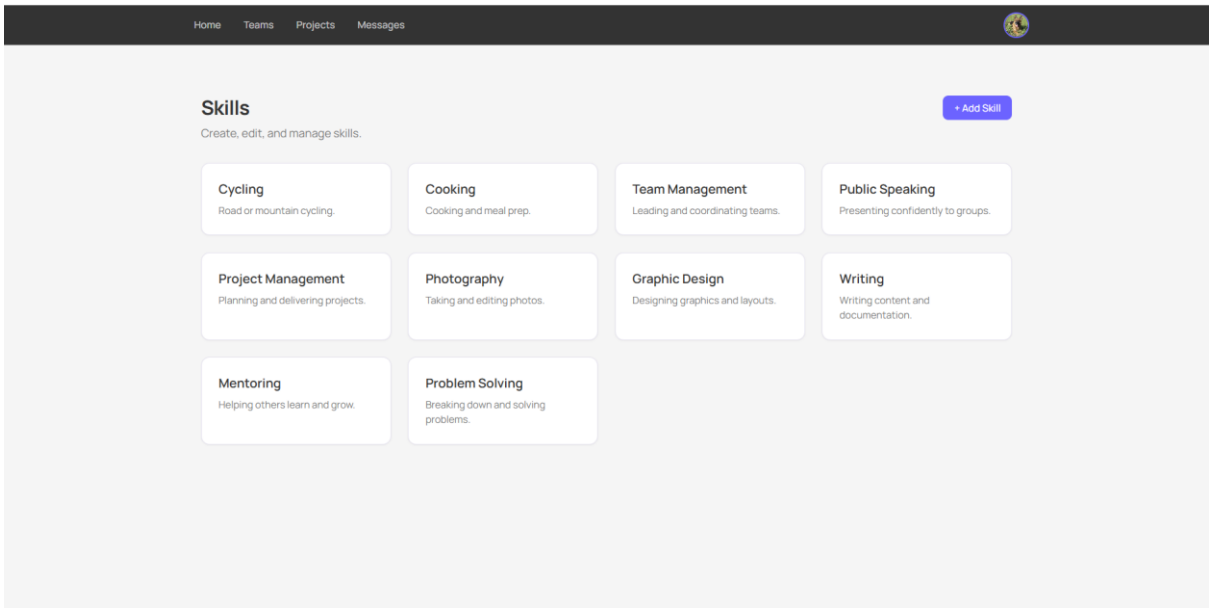


Figure 29: View Skills Screen

2.3.3.7 Update Skill: Use Case 4.7

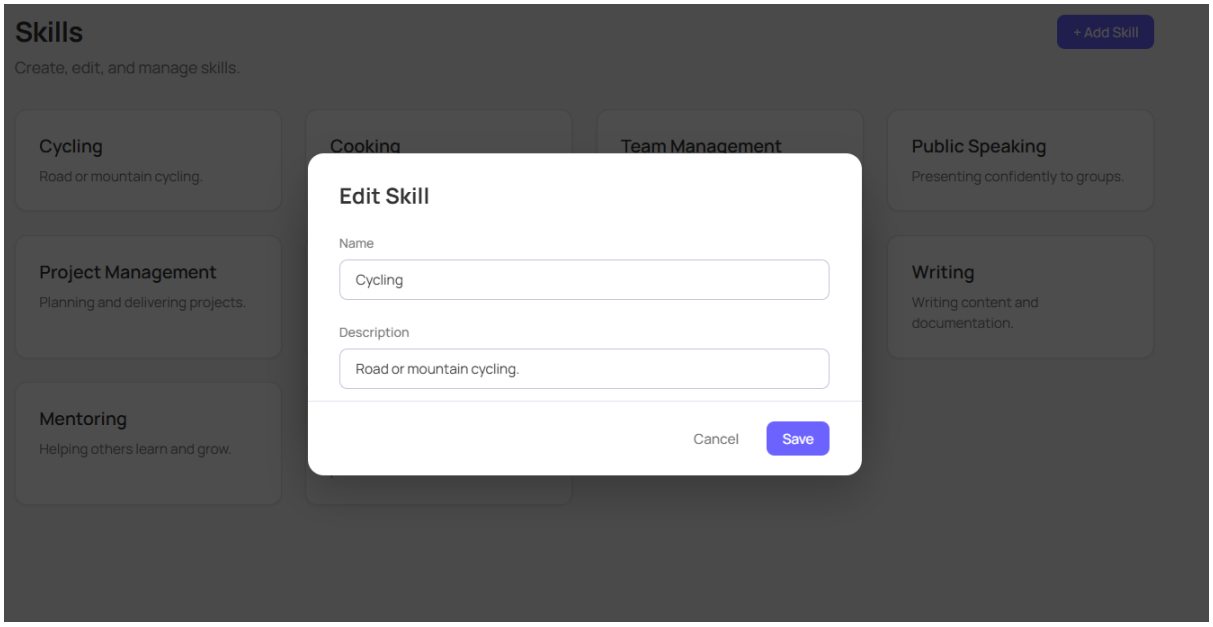


Figure 30: Update Skill Modal

2.3.3.8 Deactivate Skill: Use Case 4.8

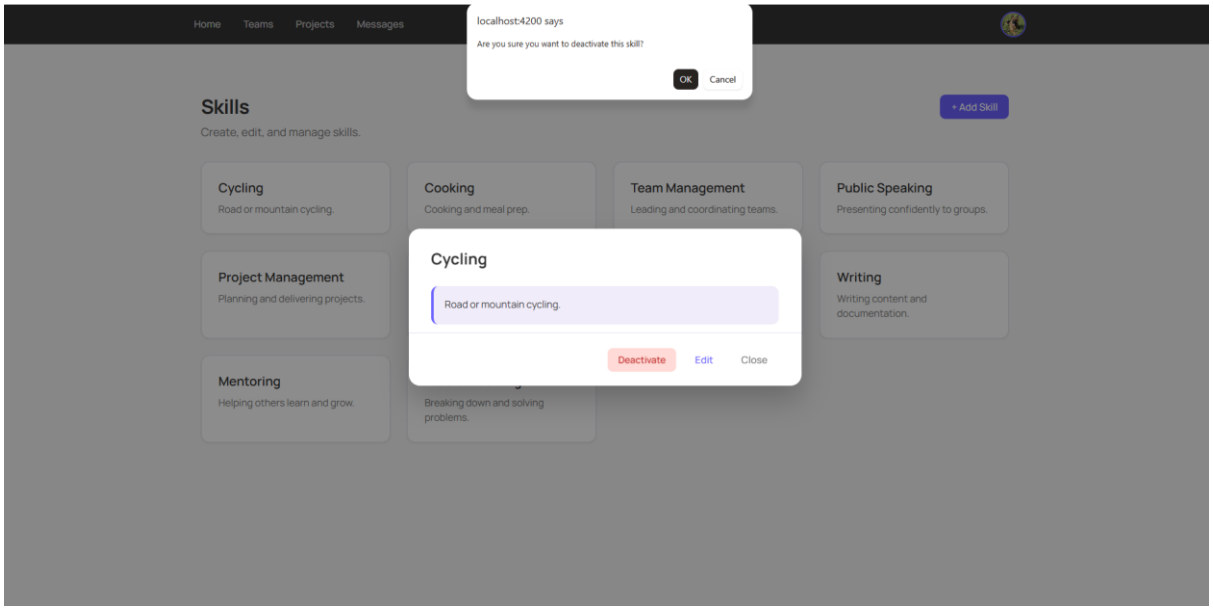


Figure 31: Deactivate Skill Screen

2.3.4 Project Subsystem

2.3.4.1 Create Project: Use Case 7.1

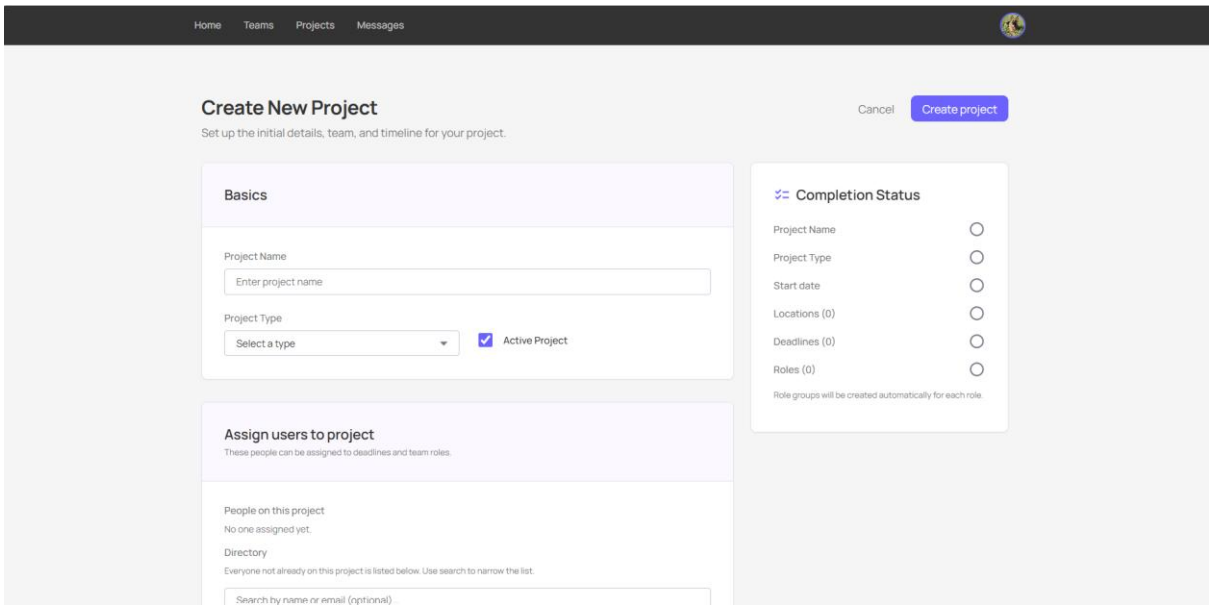


Figure 32: Create Project Screen

2.3.4.2 View Project: Use Case 7.2

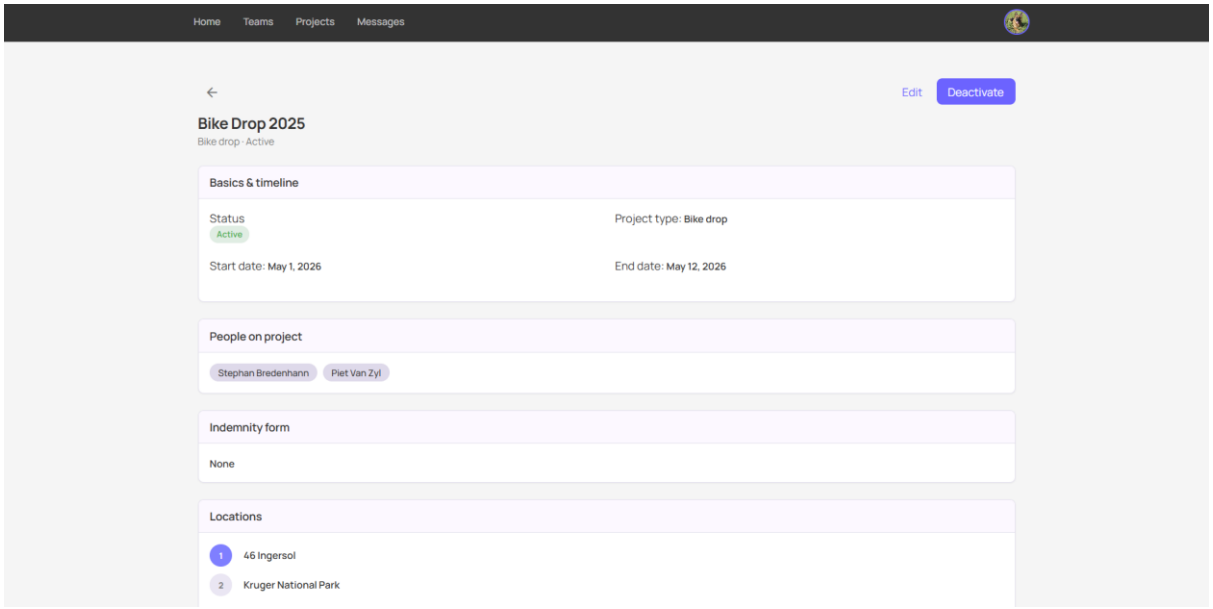


Figure 33: View Project Screen

2.3.4.3 Update Project: Use Case 7.3

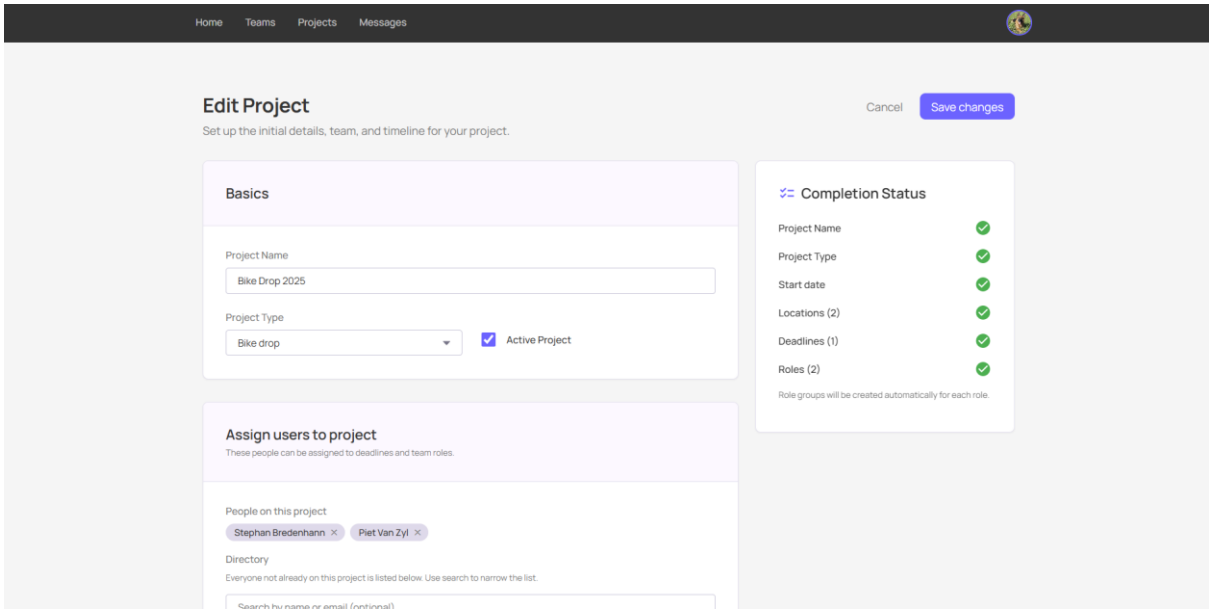


Figure 34: Update Project Screen

2.3.4.4 Deactivate Project: Use Case 7.4

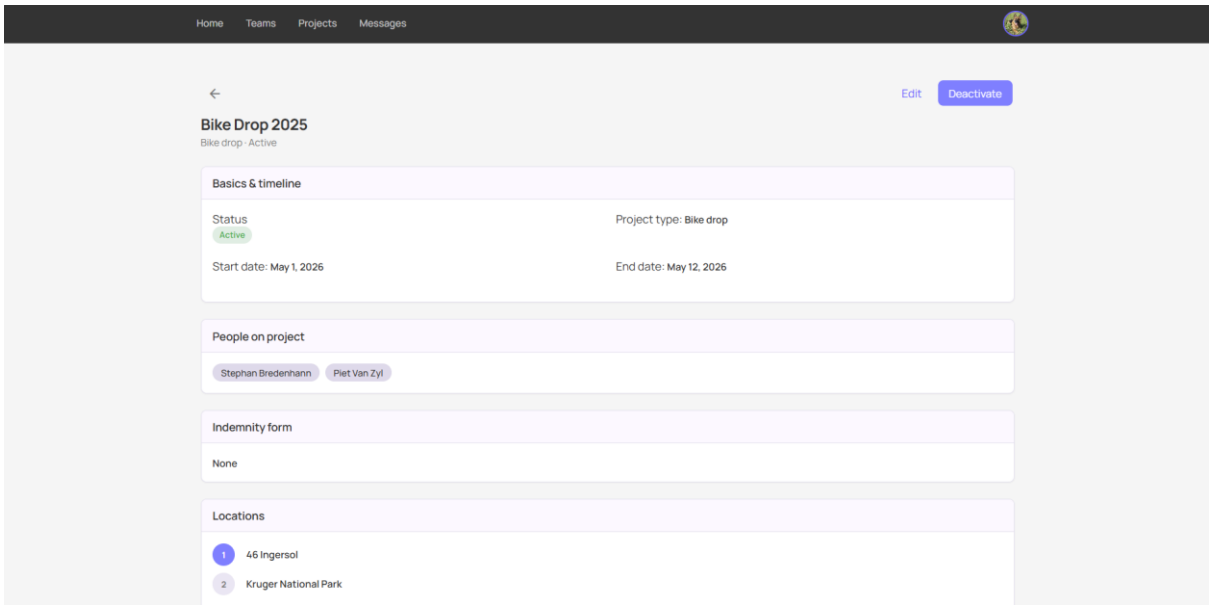


Figure 35: Deactivate Project Screen

2.3.4.5 Create Project Type: Use Case 7.5

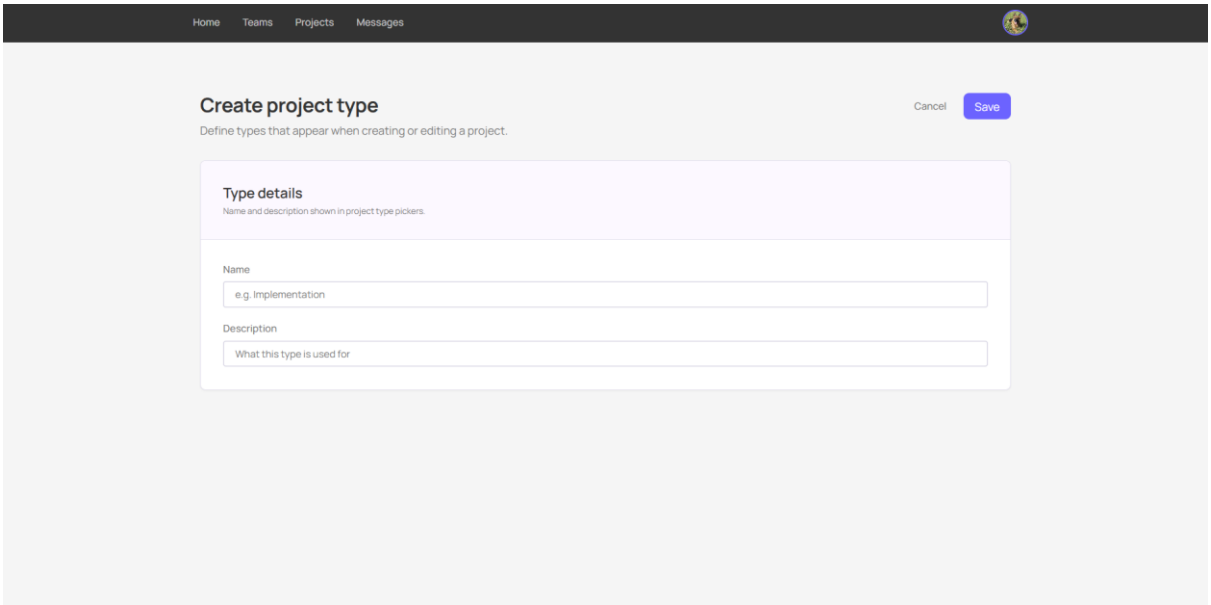


Figure 36: Create Project Type Screen

2.3.4.6 View Project Type: Use Case 7.6

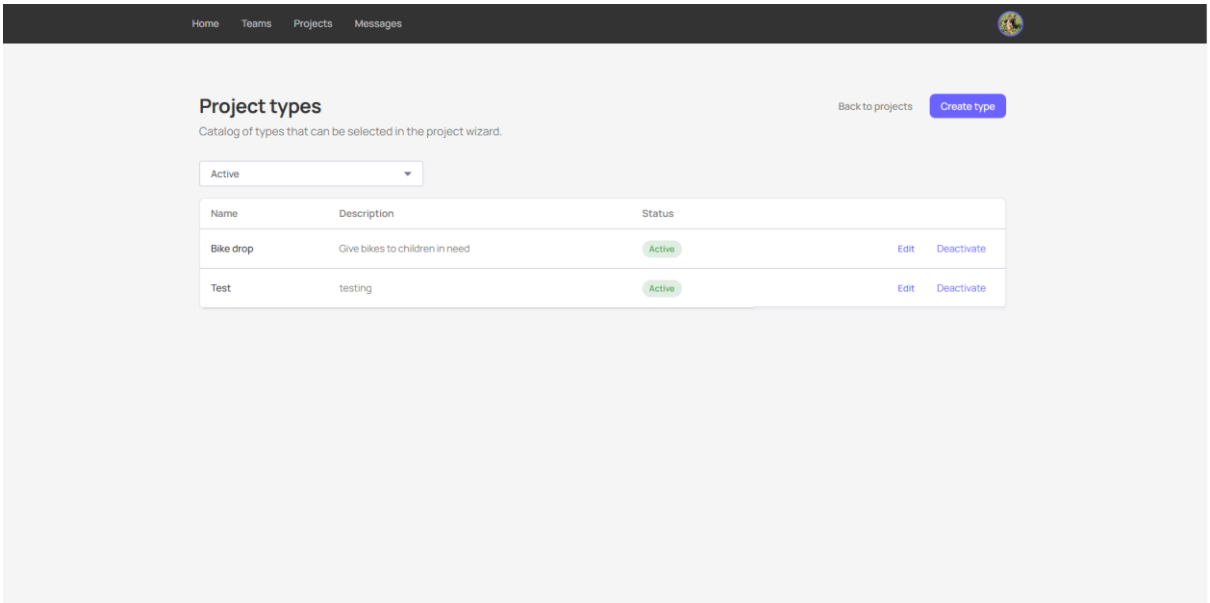


Figure 37: View Project Type Screen

2.3.4.7 Update Project Type: Use Case 7.7

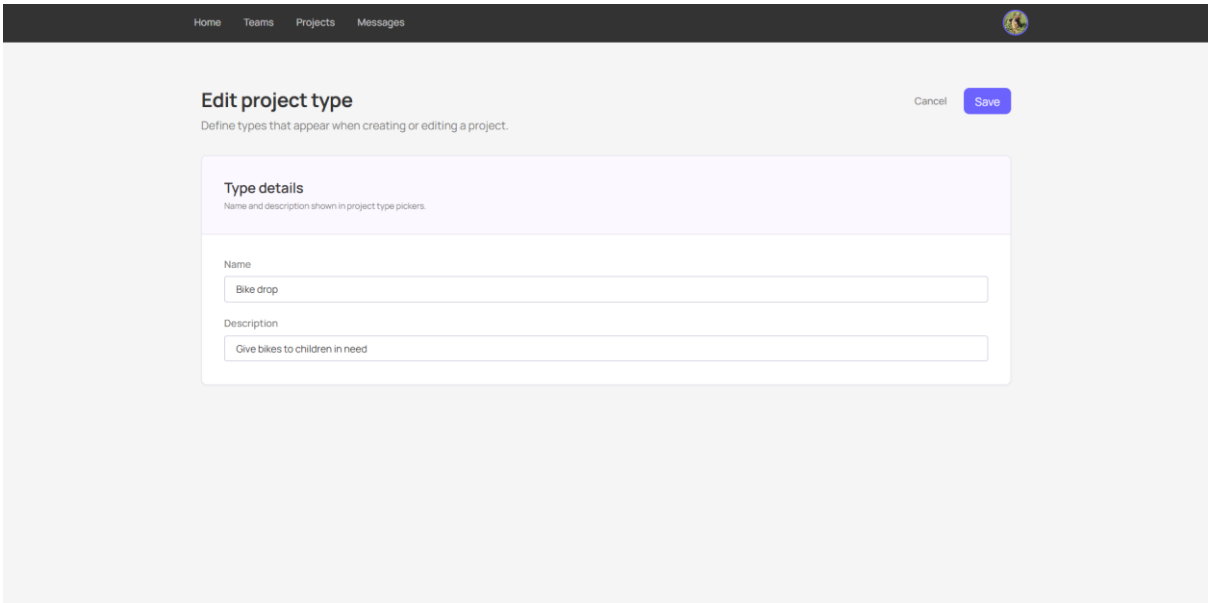


Figure 38: Update Project Type Screen

2.3.4.8 Deactivate Project Type: Use Case 7.8

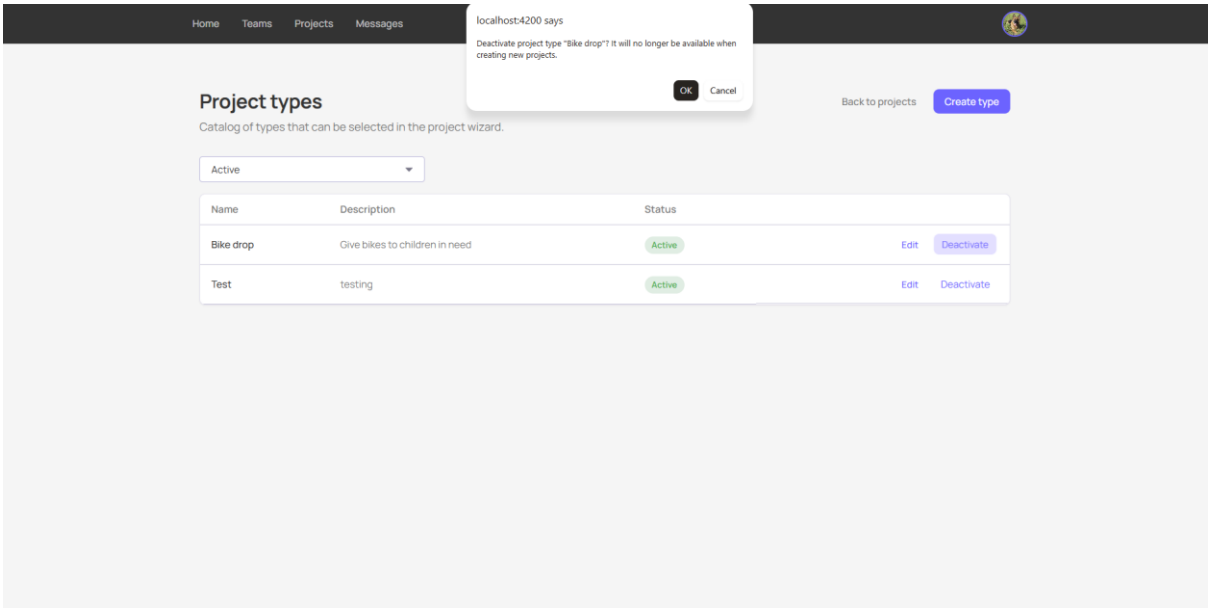


Figure 39: Deactivate Project Type Screen

2.3.4.9 Add Indemnity Form: Use Case 7.9

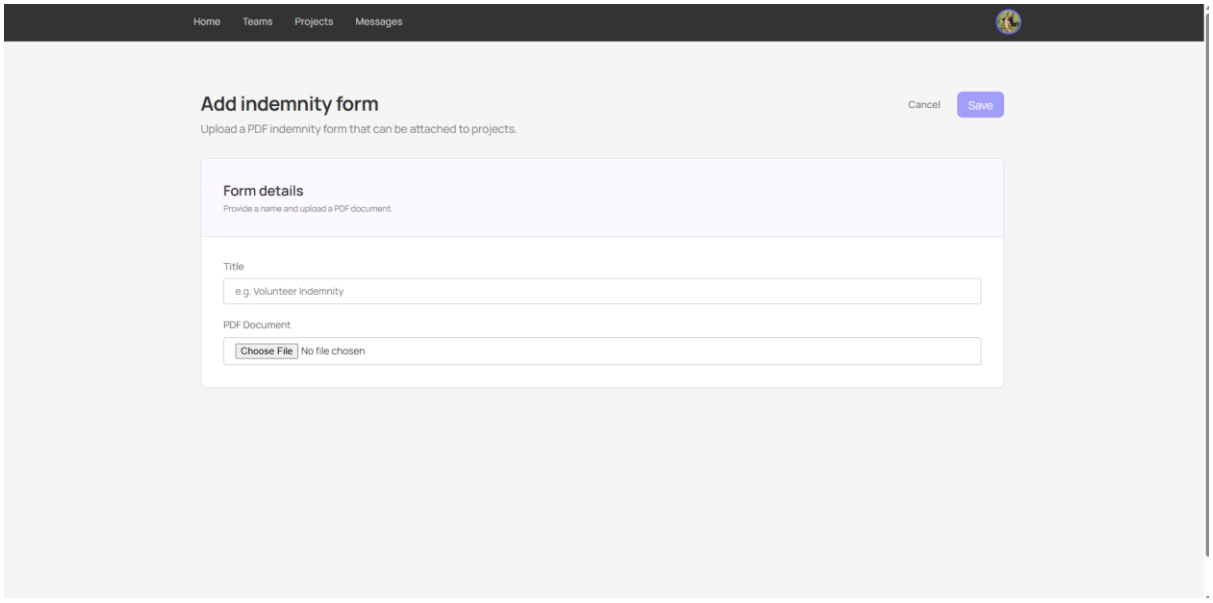


Figure 40: Add Indemnity Form Screen

2.3.4.10 Update Indemnity Form: Use Case 7.10

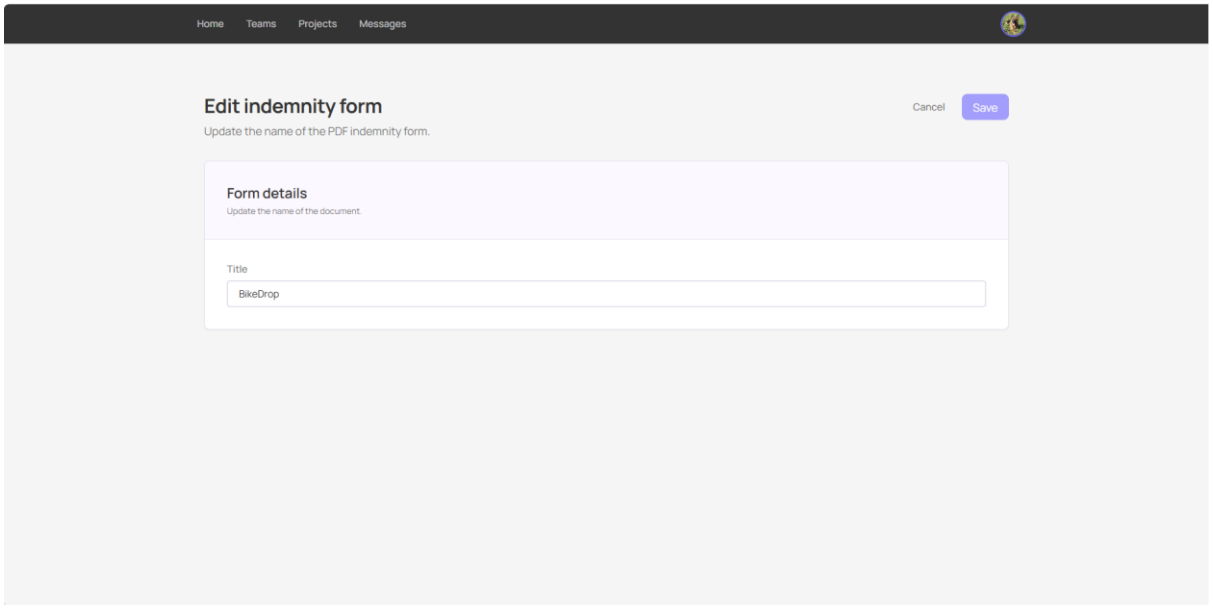


Figure 41: Update Indemnity Form Screen

2.3.4.11 View Indemnity Form: Use Case 7.11

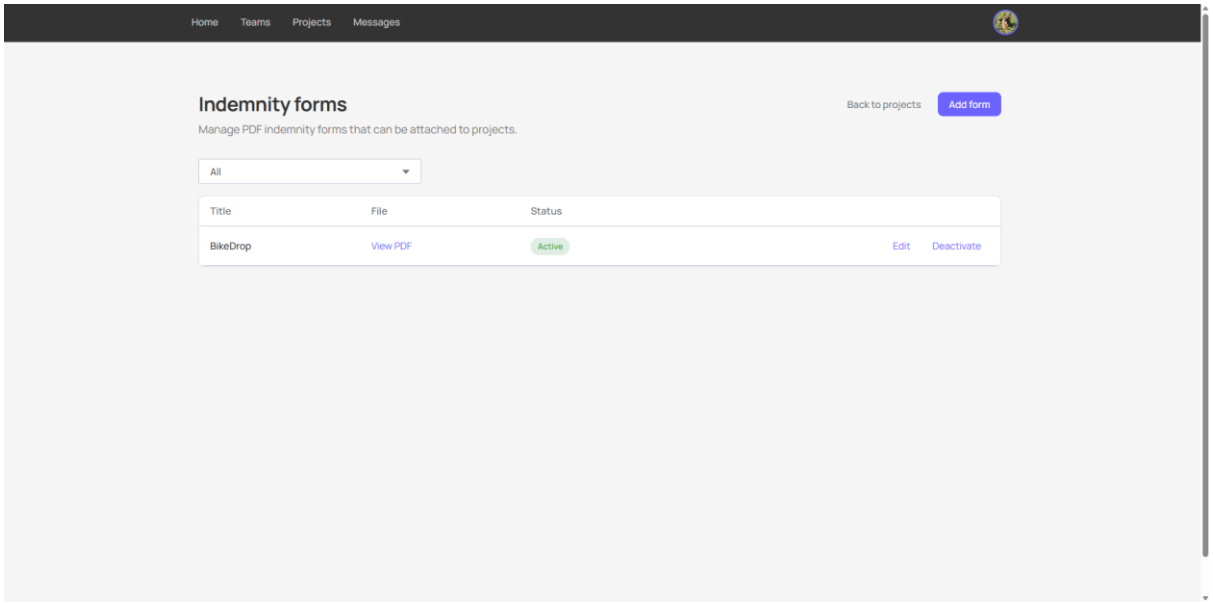


Figure 42: View Indemnity Form Screen

2.3.4.12 Remove Indemnity Form: Use Case 7.12

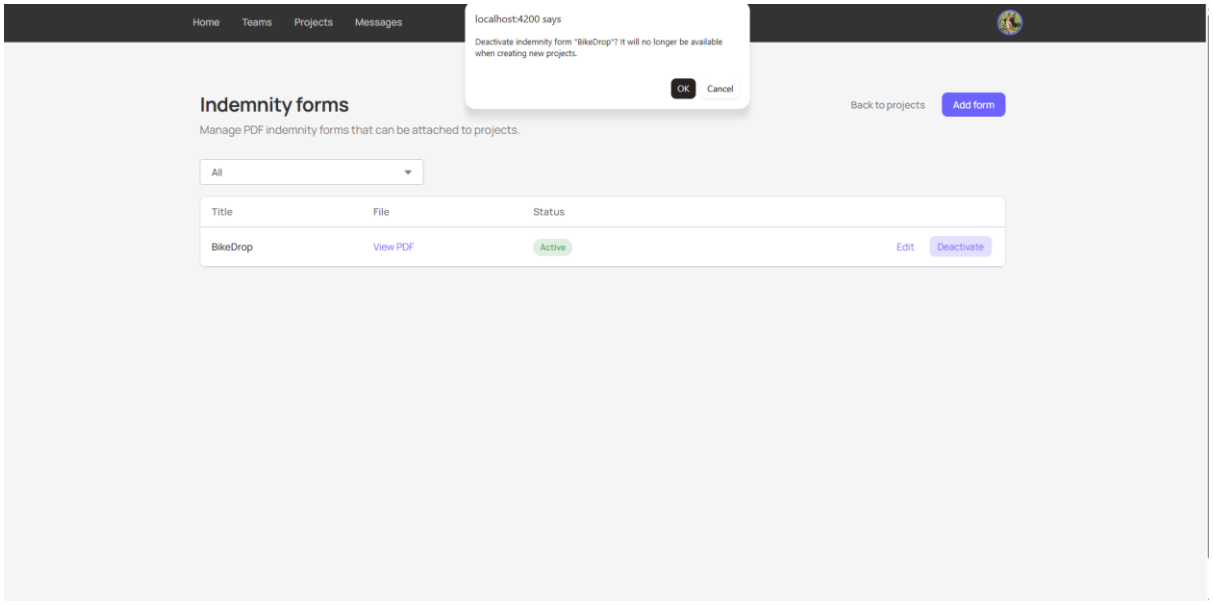


Figure 43: Remove Indemnity Form Screen

2.3.4.14 Capture Indemnity Form: Use Case 7.13

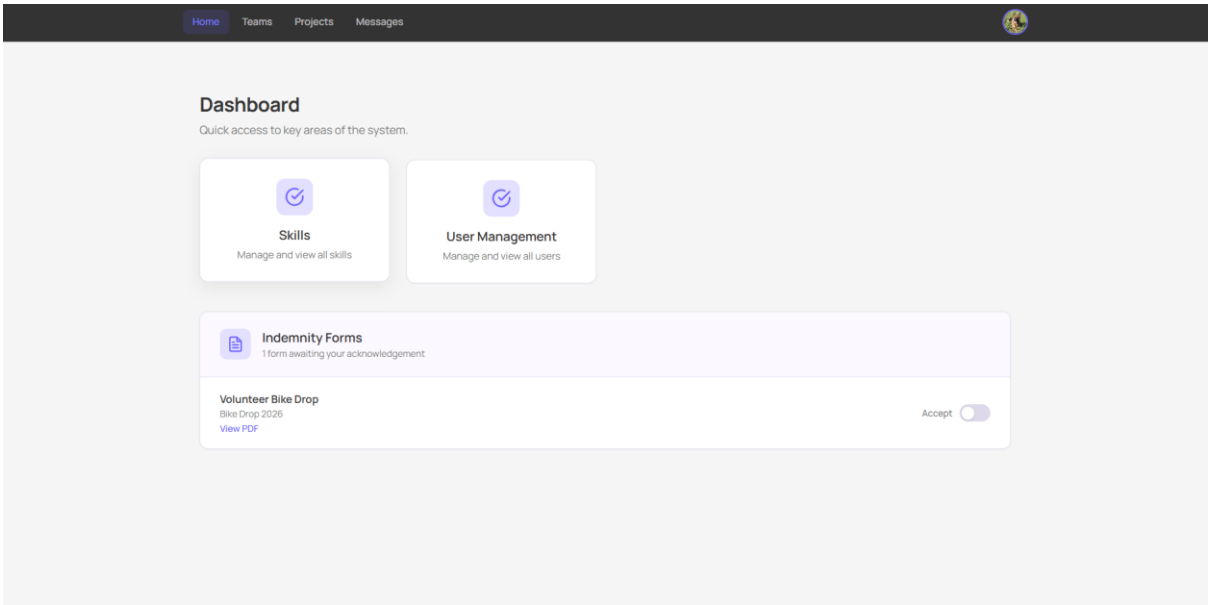


Figure 44: Capture Indemnity Form Screen

2.3.5 Message Subsystem

2.3.5.1 Create Message: Use Case 11.1

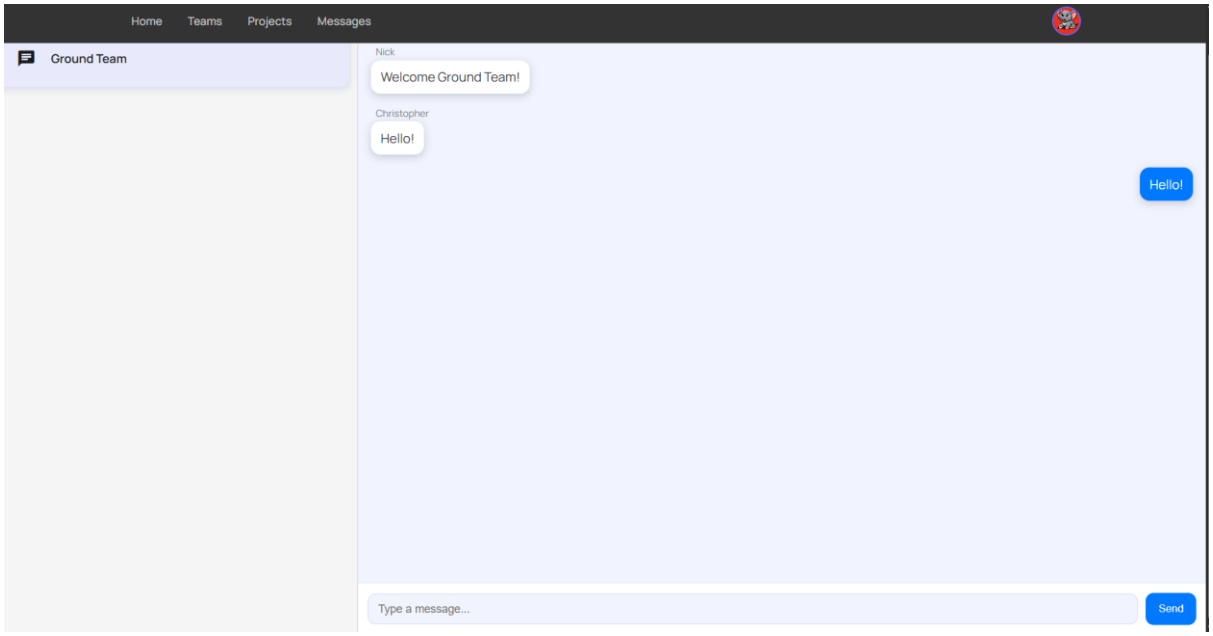


Figure 45: Message Screen

2.3.5.2 View Message: Use Case 11.2

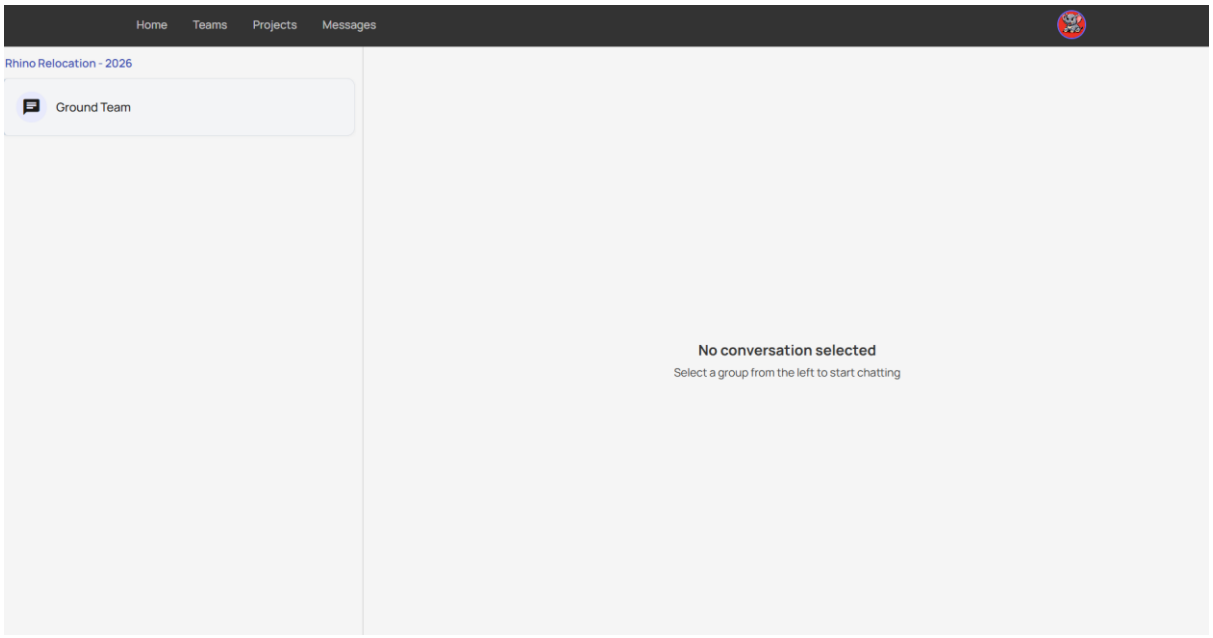


Figure 46: No Group Selected Screen

2.4 Technical Use Case Narratives

2.4.1 User Subsystem Technical Narratives

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>AUTHOR:</td><td colspan="2">Stephan Brednehnn</td><td>Date: 28 April 2025
Version: 1.0</td></tr>
<tr><td>USE CASE NAME:</td><td colspan="2">User Login</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>USE CASE ID:</td><td colspan="2">1.1</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">User</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 an existing user logs onto the system to view and manage their account and projects. The user enters their credentials into the Angular frontend. The system evaluates the reactive form, and upon submission, routes the JSON payload to the corresponding “.NET 8” API endpoint (“POST api/Auth/login”). The backend executes a SQL Read Query via Entity Framework Core on the “User” Entity to validate the entered credentials and account status using the “PasswordHasherService”. The system generates a JWT token embedded with relevant claims and calculates the user's onboarding progression. The use case concludes with the frontend persisting the JWT token in Local Storage, redirecting the user to the appropriate page, and displaying a success toast notification.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The user must be registered on the system and their account must be active.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The user requests to log into their account.</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>
</table>

	Step 1: The user requests to log into their account by navigating to the base URL or "/login" route.		**Step 2:** The Angular Router loads the Login Component. The system renders the user interface containing the following elements on a centralized authentication card: • A Login header reading "Sign In". • An Angular Reactive Form Group containing an Email label and an input field with email formatting validation. • A Password label and an obfuscated input field. • A hyperlink labelled "Forgot password?". • A primary "Sign In" submit button • A label reading "Don't have an Account?". • A hyperlink labelled "Sign Up Here".
	Step 3: The user captures their email address and password into the respective input fields. And selects the "sign In" button. [ALT]		**Step 4**: The system captures the form data and the Angular AuthService sends a POST request to the api/Auth/login endpoint with a LoginRequest JSON payload containing the email and password. The API's AuthController receives the request. The system connects to the ApplicationDbContext and executes an asynchronous query

			(SingleOrDefaultAsync) against the following entity to locate a matching record by UserEmail: **User Entity:** • UserId (PK) • UserName • UserSurname • UserDateOfBirth • UserEmail • PasswordHash • UserStatus • UserAllergies • ProfilePicture • ActiveStatus • UserShirtSize • SkillsOnboardingCompleted
			Step 5: The system validates the retrieved entity against the provided credentials and business rules before proceeding. The system performs the following checks: • Verifies that the user record was found. • Invokes the PasswordHasherService.VerifyPassword() method, hashing the incoming password string

			and validating it against the PasswordHash field stored in the database. • Verifies that ActiveStatus is true (the account is not deactivated). [ALT]
			Step 6: Upon successful validation, the system delegates to the "JwtTokenService" to generate a secure JWT token. The token embeds standard claims alongside the user's ID and Email, granting the user access to authorized endpoints.
			Step 6: The system confirms that the user has completedOnboarding in the User table. [ALT]
			Step 7: The JWT is stored in the user's browser local storage, and they are redirected to the Primary System Dashboard.
Alternate courses:	[ALT] Step 3: The user forgets their login details and selects the "Forgot Password" hyperlink, which triggers the Angular Router to navigate to the password reset component, invoking use case 1.3 Reset Password.		
	[ALT] Step 5**:** The system fails to locate the user's email in the database, or the "PasswordHasherService" returns false for the password check. The API immediately aborts the flow and returns an "HTTP 401 Unauthorized" response with the message "Invalid credentials." The frontend catches the error and renders a red error toast message reading "Invalid credentials, please try again."		
	[ALT] Step 7: The system identifies that the user's "ActiveStatus" is false. The API halts the login process and returns an "HTTP 403 Forbidden" response along with a JSON payload indicating "accountDeactivated = true". The		

	frontend intercepts this specific error and invokes Use Case 1.7 Reactivate User.
Conclusion:	The use case concludes when the system successfully validates the credentials, issues a JWT token and routes the user to the correct application view.
Post-condition:	The user is actively authenticated within the system, holds a valid JWT token in local storage, and is permitted access to secured application features.
Business Rules:	• Secure Access
Implementation constraints and specifications:	• JWT tokens must be securely processed and handled purely in memory and local storage on the client.
Assumptions:	• None
Open issues:	• None

Table 1: 1.1 User Login Technical Use Case Narrative

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>AUTHOR:</td><td colspan="2">Stephan Brednehnn</td><td>Date: 28 April 2025
Version: 1.0</td></tr>
<tr><td>USE CASE NAME:</td><td colspan="2">User Logout</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>USE CASE ID:</td><td colspan="2">1.2</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">User</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 occurs when an authenticated user deliberately chooses to terminate their active session. The frontend initiates the process by calling the secured “POST api/Auth/logout” API endpoint. The backend intercepts the user's Bearer token, extracts the unique JWT identifier (“jti”) and expiration time (“exp”), and registers the token with the “TokenRevocationService” to forcibly invalidate it before its natural expiration. Finally, the frontend purges local state and navigates the user securely back to the login interface.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The user must be currently authenticated with a valid session and actively navigating the system dashboard or internal pages.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The user clicks the "Logout" button located in the application's global navigation bar or user profile dropdown.</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 user opens the profile dropdown menu in the top navigation bar and clicks the "Logout" button.</td><td></td><td>Step 2: The Angular frontend intercepts the click event. The “AuthService” extracts the current JWT token from Local Storage and</td></tr>
</table>

			dispatches a secured "POST" request to the "api/Auth/logout" endpoint, passing the token in the "Authorization: Bearer" header.
			Step 3: The API's "AuthController" receives the request. Because the endpoint is decorated with the "\[Authorize]" attribute, the ASP.NET Core middleware first validates that the token itself is structurally valid and not expired.
			Step 4: The controller manually parses the "Authorization" header using "JwtSecurityTokenHandler.ReadJwtToken()". The system queries the token's embedded claims to extract the "jti" (JWT ID) and the "exp" (Expiration Timestamp) values.
			Step 5: The system invokes the "TokenRevocationService.Revoke()" method, passing in the extracted "jti" and a computed "expiresAt" "DateTimeOffset". The service adds this unique token ID to a secure, server-side blocklist (revocation list) to

			guarantee the token can no longer be used for future authenticated requests, effectively neutralizing it immediately.
			Step 6: The API returns an “HTTP 200 OK” JSON response with a success message reading "Logged out."
			Step 7: The Angular “AuthService” processes the successful HTTP response. The system executes a localized cleanup routine, permanently deleting the JWT token from the browser's Local Storage and clearing any cached user profile information.
			Step 8: The Angular Router executes a programmatic navigation, redirecting the user entirely out of the secure zone and rendering the Login Component interface.
Alternate courses:	None		
Conclusion:	The active session is securely terminated on both the client-side browser and the backend server.		
Post-condition:	The user's access to secured endpoints is completely revoked until a new, successful authentication cycle is completed.		
Business Rules:	• Secure Access		

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

Table 2: 1.2 User Logout Technical Use Case Narrative

System Name: Integrated NGO Management System			
AUTHOR:	Stephan Brednehnn	**Date:** 28 April 2025 **Version:** 1.0	
USE CASE NAME:	Reset Password	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
USE CASE ID:	1.3		
PRIORITY:	High		
SOURCE:	EPI-USE		
Primary Business Actor (PBA):	User		
Primary System Actor (PSA):	• None		
Other Participating Actors:	• None		
Other Interested Stakeholders:	Administrator		
Description:	This use case handles account recovery when a user has forgotten their password or requires a security reset. The user requests a reset code via the frontend, which triggers the backend to generate a cryptographically secure 6-digit PIN. This PIN is hashed, stored in the database ("PasswordResetCode" Entity), and dispatched to the user via the "EmailSenderService". The user then enters the PIN alongside a new password into a validation form. The system verifies the code, enforces retry limits, updates the "PasswordHash" on the "User" Entity, and finalizes the recovery process.		
Pre-condition:	The user must be attempting to log in, and their email address must be registered on the system.		
Trigger:	The user clicks the "Forgot Password?" hyperlink located on the login page.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user clicks the "Forgot Password?" link.		**Step 2:** The Angular Router unmounts the Login Component and mounts the Password Reset Request Component. The UI

			renders a dedicated card displaying a sub-header, a descriptive instruction text, a Reactive Form with an Email input field, a "Back to Login" link, and a "Send Reset Code" primary button.
	Step 3: The user enters their email address and clicks the "Send Reset Code" button.		**Step 4:** The frontend's Reactive Form validates the email format. The system dispatches a “POST” request to “api/Auth/forgot-password/request” containing the “ForgotPasswordRequestDto” JSON payload.
			Step 5: The backend receives the request. The system connects to the database via Entity Framework and performs a query against the “User” table to determine if an account matching the provided email address exists. [ALT]
			Step 6: The system generates a random, secure 6-digit numeric string using the “RandomNumberGenerator” function.
			Step 7: The system calculates a secure hash of the generated code. It creates a 16-byte cryptographic salt, concatenates the code and the salt, and hashes

			the resulting string using the “SHA256” algorithm.
			Step 8: The system initializes a new “PasswordResetCode” Entity, populating it with the user's ID, the generated “CodeSalt”, the “CodeHash”, a “CreatedAt” timestamp, an “ExpiresAt” timestamp strictly set to 10 minutes in the future, and initializes the “FailedAttempts” integer to 0.
			Step 9: The system invokes the “EmailSenderService.SendPasswordResetCodeAsync()” method, commanding the external Email Provider (Resend)to dispatch an automated email containing the raw 6-digit code to the user's email address.
			Step 10: The API returns an “HTTP 200 OK” response with a standard, non-revealing message: "If the email exists, a code has been sent."
	Step 11: The user retrieves the 6-digit code from their email client, enters it into the code input field.		**Step 12:** The system validates the provided code. It concatenates the raw string provided by the user with the “CodeSalt” retrieved from the database, runs

			the “SHA256” hashing algorithm, and performs a comparison against the stored “CodeHash”. The Api returns A ‘HTTP 200 OK’ response. [ALT]
			Step 13: The Angular frontend receives the response and dynamically transitions the UI state. It hides the code input field and displays the new password and confirm password fields.
	Step 14: The user enters their desired new password and re-enters it in the required fields to ensure it is accurately captured.		**Step 15:** The system overwrites the “PasswordHash” property on the “User” Entity with the newly generated hash. Concurrently, the system updates the “PasswordResetCode” Entity, populating the “UsedAt” property with the current UTC timestamp to permanently expire the code.
			Step 16: The API returns an “HTTP 200 OK” response with the message "Password reset successful.". The frontend receives the response and redirects the user to the login component
Alternate courses:	[ALT] Step 5: If the provided email address does not exist in the database, the system intentionally skips code generation and database insertion. To strictly prevent malicious actors from performing email enumeration, the API still		

	returns the exact same “HTTP 200 OK” response ("If the email exists, a code has been sent."), making failures indistinguishable from successes on the frontend.
	[ALT] Step 12: The cryptographic comparison determines that the provided code is incorrect. The system increments the “FailedAttempts” integer property on the “PasswordResetCode” entity and commits the change. If “FailedAttempts” reaches or exceeds 5, the system permanently locks the code and returns an “HTTP 401 Unauthorized”.
Conclusion:	The user's account password is securely overwritten with a new hash, and the user regains access to the system.
Post-condition:	The previous password hash is irrecoverably replaced in the database. The specific 6-digit reset code is marked as consumed.
Business Rules:	• Secure Access
Implementation constraints and specifications:	• None
Assumptions:	• The email provider (Resend) achieves rapid delivery.
Open issues:	• None

Table 3: 1.3 Reset Password Technical Use Case Narrative

<table>
<tr><th colspan="4">INTEGRATED NGO MANAGEMENT SYSTEM</th></tr>
<tr><td>AUTHOR:</td><td colspan="2">Christopher Bailey</td><td>Date: 30 April 2026
Version: 1</td></tr>
<tr><td>USE CASE NAME:</td><td colspan="2">Create User</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>USE CASE ID:</td><td colspan="2">1.4</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">User</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 describes the event in which a user signs up and creates a profile. The user will select to sign up for an account and enter their details into the relevant fields. Once done, the system will allow the user to login. On the first login, the user will complete the onboarding details screens. The use case ends once the new user is registered on the system.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">User does not already have an account.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The user requests to sign up and clicks the “Sign up here” 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 sign up and clicks the “Sign up here” link.</td><td></td><td>Step 2: The system loads and displays the Sign Up form using the Angular frontend with the following controls:
• A Sign up title.</td></tr>
</table>

			• A Already have an account? Sign in here label. • A First name label and input field. • A Surname label and input field. • A Date of birth label and date picker. • Email label and input field. • Password label and input field. • Confirm password label and input field. • Sign up button.
	Step 3: The user enters their details and clicks the Sign up button. [ALT]		**Step 4:** The system validates the details entered, validating the following: • All fields are completed. • The email is not already in use. • The date of birth is not in the future. • Password and Confirm password match. [ALT]
			Step 5: The Angular frontend calls the User service to send a POST request to the C# Web API backend User controller through the CreatUser endpoint. The backend creates a new user in the following entities:

			User Entity: • UserId (PK) • UserName • UserSurname • UserDateOfBirth • UserEmail • PasswordHash • ActiveStatus [ALT]
			Step 6: The system invokes use case 1.1 (User Login) before completing the onboarding details screens.
			Step 7: The system loads and displays the Onboarding Details form using the Angular frontend with the following controls: • A Complete your profile title. • A few quick details and you're in label. • A circle with a user icon and the text, Select Profile Photo. • A Click the circle to upload a profile picture (optional) label. • A Shirt size label. • Buttons for sizes: ○ XS ○ S ○ M ○ L ○ XL • A Continue button.

	Step 8: The user provides the details and clicks Continue.		**Step 9:** The system validates the details entered, validating the following: • All required fields are completed. [ALT]
			Step 10: The Angular frontend calls the User service to send a POST request to the C# Web API backend User controller through the CompleteOnboarding endpoint. The backend adds the following details to the user in the following entities: User Entity: • ProfilePicture • UserShirtSize [ALT]
			Step 11: The system loads and displays the Onboarding Skills form using the Angular frontend with the following controls: • A Select your skills title. • A Pick as many as you like – or none at all label. • A search bar with a Search skills placeholder. • A Clear button. • A table of the different skills

			with a short description for each skill. • A Selected label with a counter for how many were selected. • A Continue button. The skills table is populated with skills retrieved from the Skill table. The Angular frontend calls the Skills service to send a GET request to the C# Web API backend Skills controller through the GetSkills endpoint. Skill Entity: • SkillName • SkillDescription
	Step 12: The user selects their skills can clicks Continue.		**Step 13:** The Angular frontend calls the User service to send a POST request to the C# Web API backend User controller through the SaveMySkills endpoint. The backend adds the following details to the user's account in the following entities: UserSkill Entity: • UserId (PK, FK) • SkillId (PK, FK) [ALT]

			Step 14: The system loads and displays the Onboarding Status form using the Angular frontend with the following controls: • A One last thing title. • A What's your availability right now label. • A User Status label. • A dropdown with a Select a status placeholder. • A Finish button.
	Step 15: The user selects their status and clicks Finish.		**Step 16:** The system validates the entered details. The system validates the following: • An option has been selected from the dropdown. [ALT]
			Step 17: The Angular frontend calls the User service to send a POST request to the C# Web API backend User controller through the SaveMyStatus endpoint. The backend adds the following details to the user's account in the following entities: User Entity: • UserStatus [ALT]

			Step 18: 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 user. The system will store information such as the authentication user performing the action, the timestamp of the operation, and a descriptive log entry indicating that a new user has been created. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
Alternate courses:	[ALT] Step 3: The user selects the "Already have an account? Sign in here" link. Invoke use case 1.1 User Login.		
	[ALT] Step 4: The validation fails. The system displays an error message. Go back to step 3.		
	[ALT] Step 5: The system fails to add the new user's details. The system will display an error message based on the status code.		
	[ALT] Step 9: The validation fails. The system displays an error message. Go back to step 8.		
	[ALT] Step 10: The system fails to add the new users onboarding details. The system will display an error message based on the status code.		
	[ALT] Step 13: The system fails to add the new users onboarding skills. The system will display an error message based on the status code.		

	[ALT] Step 16: The validation fails. The system displays an error message. Go back to step 15.
	[ALT] Step 17: The system fails to add the new users onboarding status. The system will displays an error message based on the status code.
Conclusion:	The user is successfully captured on the system.
Post-condition:	The user's profile is saved on the system.
Business Rules:	• Valid email address is required.
Implementation constraints and specifications:	• Passwords are hashed before storage in the database.
Assumptions:	• The user has access to a valid email address.
Open issues:	• None

Table 4: 1.4 Create User Technical Use Case Narrative

<table>
<tr><th colspan="4">INTEGRATED NGO MANAGEMENT SYSTEM</th></tr>
<tr><td>AUTHOR:</td><td colspan="2">Christopher Bailey</td><td>Date: 30 April 2026
Version: 1</td></tr>
<tr><td>USE CASE NAME:</td><td colspan="2">Update User</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>USE CASE ID:</td><td colspan="2">1.5</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">User</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 describes the event in which a user updates their profile. The user will navigate to the profile icon in the navbar and select profile. The system will display the user’s current details on the system and allow the user to update their details. The user will update the details they wish to and then select save. The system will validate the details entered before updating the user’s details on the system. The use case ends once the user’s details are successfully updated on the system.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">User is logged into the system and on their account.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The user requests to update their 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 user requests to update their details on the system.</td><td></td><td></td></tr>
</table>

	Step 2: The user navigates to and selects the profile icon. They then select the My Profile option.		**Step 3:** The system invokes use case 1.6 (View User).
	Step 4: The user clicks the Update Profile button.		**Step 5:** The system loads and displays the Update Profile modal using the Angular frontend with the following controls populated with the users current details: • An Update Profile title. • A Profile Picture label. • A section displaying selectable avatar images. • A circular upload option with an UPLOAD label for adding a custom profile picture. A form divided into two columns: • Left column: • A Name label with an input field. • An Email label with an input field. • A Shirt Size label with a dropdown selection (XS, S, M, L, XL). • Right column: • A Surname label with an input field.

			• A Date of Birth label with a date picker. • A Status label with a dropdown selection. • A Skills label. • A dropdown to select a skill. • An Add button to add the selected skill. • A list of added skills displayed as removable pills (each with an "×" remove option). • A Save button. • A Cancel button.
	Step 6: The user updates their details and clicks the Save button. [ALT]		**Step 7:** The system validates the details entered, validating the following: • All fields are completed. • Date of birth is not in the future. • Email does not already exist. [ALT]
			Step 8: The Angular frontend calls the User service to send a PUT request to the C# Web API backend User controller through the UpdateUser endpoint. The backend updates the user details in the following entities:

			User Entity: • UserName • UserSurname • UserDateOfBirth • UserEmail • UserShirtSize • UserStatus UserSkill Entity: • UserId (PK, FK) • SkillId (PK, FK) [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 user. 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 user has been updating. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry

Alternate courses:	[ALT] Step 6: The user clicks the cancel button. Use case terminates.
	[ALT] Step 7: The validation fails. The system displays an error label. Go back to step 6.
	[ALT] Step 8: The system fails to update user's details. The system will display an error message based on the status code.
Conclusion:	The user's updated details are successfully captured on the system.
Post-condition:	The user's profile is updated on the system.
Business Rules:	• The user has an account on the system.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 5: 1.5 Update User Technical Use Case Narrative

<table>
<tr><th colspan="4">INTEGRATED NGO MANAGEMENT SYSTEM</th></tr>
<tr><td>AUTHOR:</td><td colspan="2">Christopher Bailey</td><td>Date: 30 April 2026
Version: 1</td></tr>
<tr><td>USE CASE NAME:</td><td colspan="2">View User</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>USE CASE ID:</td><td colspan="2">1.6</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">User</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 user views their profile. The user will navigate to the profile icon in the navbar and select profile. The system will retrieve and display the user’s current details on the system. The use case ends once the user can view their profile.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">User is logged into the system and on their account.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The user requests to view their profile on the system.</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 user requests to view their profile on the system.</td><td></td><td></td></tr>
<tr><td>Step 2: The user navigates to and selects the profile icon. They then select the My Profile option.</td><td></td><td>Step 3: The Angular frontend calls the User service to send a GET request to the C# Web</td></tr>
</table>

			API backend User controller through the GetUserById endpoint. The backend retrieves the user's details from the following entities: User Entity: • UserId (PK) • UserName • UserSurname • UserDateOfBirth • UserEmail • UserStatus • ProfilePicture • UserShirtSize UserSkill Entity: • UserId (PK, FK) • SkillId (PK, FK) Skill Entity: • SkillId (PK) • SkillName [ALT]
			Step 4: The system loads and displays the User Profile screen using the Angular frontend with the following controls populated with the user details: A header section containing: • The user's profile image. • A display of the user's full name

			(name and surname). • A display of the user's email address. A profile details card containing: • A row showing the Name label and the user's name. • A row showing the Surname label and the user's surname. • A row showing the Email label and the user's email. • A row showing the Date of birth label and the user's date of birth. • A row showing the Shirt size label and the user's selected shirt size. • A row showing the Status label and the user's current status. A My skills section displaying the user's skills as individual skill pills. A fallback message indicating "No skills added yet" if no skills exist.

			A set of profile actions: • A Update Profile button • A button to toggle account status, displaying either Deactivate Account or Reactivate Account depending on the current state.
Alternate courses:	[ALT] Step 3: The system fails to retrieve the user's details. The system will display an error message based on the status code.		
Conclusion:	The user can view their profile.		
Post-condition:	The user's profile is displayed.		
Business Rules:	• The user has an account on the system.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 6: 1.6 View User Technical Use Case Narrative

INTEGRATED NGO MANAGEMENT SYSTEM			
AUTHOR:	Christopher Bailey	**Date: 30 April 2026** **Version:** 1	
USE CASE NAME:	Reactivate User	**USE CASE TYPE**	
USE CASE ID:	1.7	Business Requirements: □	
PRIORITY:	High	System Analysis: □	
SOURCE:	EPI-USE	System Design: ✓	
Primary Business Actor (PBA):	User		
Primary System Actor (PSA):	• None		
Other Participating Actors:	• None		
Other Interested Stakeholders:	Admin		
Description:	This use case describes the event in which a user reactivates their account. The user will login into their account before being asked if they would like to reactivate their account. Once reactivated the system will display a success popup. The use case ends once the users account is successfully reactivated on the system.		
Pre-condition:	User's account is deactivated.		
Trigger:	The user requests to reactivate their profile on the system.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The user requests to reactivate their profile on the system.		**Step 2:** The system invokes use case 1.1 (User Login).
			Step 3: The system loads and displays the Reactivate Account

			screen using the Angular frontend with the following controls: • An icon inside a circular container. • An Account Deactivated title. • A label indicating that the account linked to the user's email address has been deactivated, asking whether the user would like to reactivate it. • A Yes, reactivate my account button • A No, go back to login button
	Step 4: The user clicks the Yes, reactivate my account button. [ALT]		**Step 5:** The Angular frontend calls the Auth service to send a POST request to the C# Web API backend Auth controller through the ReactivateAccount endpoint. The backend sets the user's active status to true (1) in the following entities: User Entity: • AcitveStatus [ALT]
			Step 6: 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 user. 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 user has been reactivated. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 7: The system loads and displays the Welcome Back screen using the Angular frontend with the following controls: • An icon inside a circular container • A Welcome Back! title. • A label confirming that the account has been successfully reactivated. A label indicating that the user will be redirected automatically.
Alternate courses:	[ALT] Step 4: The user clicks No, go back to login. The user is then redirected back to the login screen using the Angular frontend with the following		

	controls: • A Sign in title. • A label asking if the user does not have an account, with a link to Sign up here. A login form containing: • An Email label with an input field and placeholder text "user@email.com". • A Password label with an input field and placeholder text "Enter Password". • A Forgot password? link. • A Sign in button • A CANVAS logo displayed at the bottom of the page.
	[ALT] Step 5: The system fails to update the user's active status. The system will display an error message based on the status code.
Conclusion:	The user's account is reactivated.
Post-condition:	The user's profile is active on the system.
Business Rules:	• Admins can reactivate users' accounts.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 7: 1.7 Reactivate User Technical Use Case Narrative

<table>
<tr><th colspan="4">INTEGRATED NGO MANAGEMENT SYSTEM</th></tr>
<tr><td>AUTHOR:</td><td colspan="2">Christopher Bailey</td><td>Date: 30 April 2026
Version: 1</td></tr>
<tr><td>USE CASE NAME:</td><td colspan="2">Deactivate User</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>USE CASE ID:</td><td colspan="2">1.8</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">User</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 describes the event in which a user deactivates their account. The user will navigate to the profile icon in the navbar and select profile. The system will display the user's current details on the system and allow the user to deactivate their account. Once deactivate the user is logged out of the system. The use case ends once the users account is successfully deactivated on the system.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">User is logged into the system and on their account.
User's account is active.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The user requests to deactivate their profile 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 user requests to deactivate their profile on the system.</td><td></td><td></td></tr>
</table>

	Step 2: The user navigates to and selects the profile icon. They then select the profile option.		**Step 3:** The system invokes use case 1.6 (View User).
	Step 4: The user clicks the Deactivate Account button.		**Step 5:** The system displays a confirmation modal with the following controls: • Confirm Deactivation title. • A Are you sure you want to deactivate your account label. • A Confirm button • A Cancel button
	Step 6: The user clicks Confirm. [ALT]		**Step 7:** The Angular frontend calls the User service to send a PUT request to the C# Web API backend User controller through the DeactivateUser endpoint. The backend sets the user's active status to false (0) in the following entities: User: • AcitveStatus [ALT]
			Step 8: The system logs the user out and loads and displays the Sign In screen using angular with the following controls: • A Sign in title. • A label asking if the user does not have an

			account, with a link to Sign up here. A login form containing: • An Email label with an input field and placeholder text “user@email.com”. • A Password label with an input field and placeholder text “Enter Password”. • A Forgot password? link. • A Sign in button • A CANVAS logo displayed at the bottom of the page.
			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 deactivation of the user. 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 user has been deactivated.

			AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
Alternate courses:	[ALT] Step 6: The user clicks Cancel. Go back to step 4.		
	[ALT] Step 7: The system fails to update the user's active status. The system will display an error message based on the status code.		
Conclusion:	The user's account is deactivated.		
Post-condition:	The user's profile is deactivated on the system.		
Business Rules:	• Admins can deactivate users' accounts.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 8: 1.8 Deactivate User Technical Use Case Narrative

2.4.2 Admin Subsystem Technical Narratives

INTEGRATED NGO MANAGEMENT SYSTEM			
AUTHOR:	Christopher Bailey	**Date: 1 May 2026** **Version:** 1	
USE CASE NAME:	Create User Profile	**USE CASE TYPE**	
USE CASE ID:	2.1	Business Requirements:	□
PRIORITY:	High	System Analysis:	□
SOURCE:	EPI-USE	System Design:	✓
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 a new user profile. The system will display all the users on the system and allow the super admin to create a new user profile. The super admin will enter the user's details and select create. The use case ends once the user's profile is successfully saved on the system.		
Pre-condition:	Super Admin is logged into the system and on their account.		
Trigger:	The super admin wants to create a user profile.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The super admin wants to create a user profile.		
	Step 2: The user navigates to the home screen and		**Step 3:** The system invokes use case 2.3 (View User Profile)

	selects the user management widget.		
	Step 4: The super admin clicks the Create button		**Step 5:** The system loads and displays the Create User modal using the Angular frontend with the following controls: • A Create User Profile title. • A Name label and input field. • A Surname label and input field. • An Date of Birth label and date picker. • An Email label and input field. • A Password label and input field. • A Confirm Password label and input field. • A Create button. • A Cancel button.
	Step 6: The super admin enters the user's details and clicks the Create button. [ALT]		**Step 7:** The system validates the details entered, validating the following: • All fields are completed. • The email is not already in use. • The date of birth is not in the future. • Password and Confirm password match. [ALT]
			Step 8: The Angular frontend calls the User service to send a POST request to the C# Web API backend User

			controller through the CreatUser endpoint. The backend creates a new user in the following entities: User Entity: • UserId (PK) • UserName • UserSurname • UserDateOfBirth • UserEmail • PasswordHash • ActiveStatus [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 user. 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 user has been created. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp LogEntry

Alternate courses:	[ALT] Step 6: The super admin clicks the Cancel button. The super admin is redirected to the User Management screen.
	[ALT] Step 7: The validation fails. The system displays an error message. Go back to step 6.
	[ALT] Step 7: The system fails to add the new user's details. The system will display an error message based on the status code.
Conclusion:	The user is successfully created on the system.
Post-condition:	The user can login and complete the onboarding screens.
Business Rules:	• Only super admins can create user profiles.
Implementation constraints and specifications:	• Password must be encrypted in the database.
Assumptions:	• Super admin has the correct user details.
Open issues:	• None

Table 9: 2.1 Create User Profile Technical Use Case Narrative

INTEGRATED NGO MANAGEMENT SYSTEM			
AUTHOR:	Christopher Bailey	**Date: 1 May 2026** **Version:** 1	
USE CASE NAME:	Update User Profile	**USE CASE TYPE**	
USE CASE ID:	2.2	Business Requirements: □	
PRIORITY:	High	System Analysis: □	
SOURCE:	EPI-USE	System Design: ✓	
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 a user's profile. The system will display all the users on the system and allow the super admin to select a user profile. The system will display the user's current details and allow the user to update their details. The use case ends once the user's profile is successfully updated on the system.		
Pre-condition:	Super Admin is logged into the system and on their account.		
Trigger:	The super admin wants to update a user profile.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The super admin wants to create a user profile.		
	Step 2: The user navigates to the home screen and selects the user management widget.		**Step 3:** The system invokes use case 2.3 (View User Profile)

	Step 4: The super admin selects a user.		**Step 5:** The system loads and displays the Update User Profile modal using the Angular frontend with the following controls: • A header section containing an Update title. • A circular avatar displaying the user's current profile picture. A form divided into two columns. Left column: • A Name label with an input field. • An Email label with an input field. • A Shirt Size label with a dropdown selection (XS, S, M, L, XL). Right column: • A Surname label with an input field. • A Date of Birth label with a date picker. • A Status label with a dropdown selection. • A Skills label. A list of the user's skills displayed as skill pills. • A Save button to submit the changes.

			A Cancel button to close the modal.
	Step 6: The super admin enters the updated user's details and clicks the Save button. [ALT]		**Step 7:** The system validates the details entered, validating the following: • All fields are completed. • Date of birth is not in the future. • Email does not already exist. [ALT]
			Step 8: The Angular frontend calls the User service to send a PUT request to the C# Web API backend User controller through the UpdateUser endpoint. The backend updates the user details in the following entities: User • UserName • UserSurname • UserDateOfBirth • UserEmail • UserShirtSize • UserStatus [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 user. 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 user has been updated. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
Alternate courses:	[ALT] Step 6: The super admin clicks 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 user's details. The system will display an error message based on the status code.		
Conclusion:	The user is successfully updated on the system.		
Post-condition:	The user's details are updated.		
Business Rules:	• Only super admins can update user profiles.		
Implementation constraints and specifications:	• None		
Assumptions:	• Super admin has the correct updated user details.		
Open issues:	• None		

Table 10: 2.2 Update User Profile Technical Use Case Narrative

<table>
<tr><th colspan="4">INTEGRATED NGO MANAGEMENT SYSTEM</th></tr>
<tr><td>AUTHOR:</td><td colspan="2">Nicholas Bailey</td><td>Date: 30/04/2026
Version: 1</td></tr>
<tr><td>USE CASE NAME:</td><td colspan="2">View User Profile</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>USE CASE ID:</td><td colspan="2">2.3</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 when an admin views the details of all the users within the system. The system displays the skills in a table containing the user’s name, surname, email, date of birth, shirt size, and status. The use case ends when the admin sees all the users and their details.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The admin must be logged into the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The admin wants to view the users 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 admin wants to view the users on the system.</td><td></td><td></td></tr>
<tr><td>Step 2: The admin selects the “User</td><td></td><td>Step 3: The system sends the request through the Angular frontend and user service to the relevant HTTP GET API</td></tr>
</table>

	Management" widget on home dashboard.		endpoint in Visual Studio (UserController – GetAllUsers), which retrieves all users from the User table in the database. User Entity: • UserId (PK) • UserName • UserSurname • UserDateOfBirth • UserEmail • UserStatus • UserShirtSize
			Step 4: The system will then display the User Management page using the Angular frontend, showing all retrieved users in a table based on the data returned from the API. • Page header section: ○ Page title ("User Management") ○ Description text ("Create, edit, and manage users.") ○ Create user button (Top right of table.) • Table display section: ○ Users are displayed in a table • The table includes the following columns: ○ Name ○ Surname ○ Email

			○ Date of Birth ○ Shirt Size (displayed as a styled tag) ○ Status (displays current user status or "N/A" if not available) ○ Actions • Actions column: ○ Update button (opens the update functionality for the selected user) ○ Deactivate/Reactivate button: ▪ Displays "Deactivate" if the user is active ▪ Displays "Reactivate" if the user is inactive ▪ Button styling changes dynamically based on user status [ALT]
	Step 5: The admin can view the users.		
Alternate courses:	ALT-Step 4: The system does not find any users. Terminate use case.		
Conclusion:	All users' details are displayed in a table.		

Post-condition:	The system displays all the current users in a table.
Business Rules:	• Admins and super admins can view the users.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 11: 2.3 View User Profile Technical Use Case Narrative

<table>
<tr><th colspan="4">INTEGRATED NGO MANAGEMENT SYSTEM</th></tr>
<tr><td>AUTHOR:</td><td colspan="2">Nicholas Bailey</td><td>Date: 30/04/2026
Version: 1</td></tr>
<tr><td>USE CASE NAME:</td><td colspan="2">Deactivate User</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>USE CASE ID:</td><td colspan="2">2.4</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 how a Super Admin deactivates a user in the system. The system allows the Super Admin to select deactivate on a user and will then confirm the deactivation. Once confirmed, the system updates the user’s status to inactive in the database. The use case ends when the user’s status is changed to inactive.</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 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 super admin wants to deactivate a user on the system.</td><td></td><td>Step 2: The system invokes UC 2.3 (View User Profile) to display the users in a table. This is done using the Angular frontend and users service, which routes the request via Angular HTTP Client to the corresponding HTTP</td></tr>
</table>

			GET API endpoint in Visual Studio (UserController).
	Step 3: The super admin selects the "Deactivate" button on a specific user.		**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 user deactivation") • Confirmation message label ○ Displays message: "Are you sure you want to deactivate this user?" • Action buttons: ○ Confirm button ○ Cancel button
	Step 5: The super admin selects the "Confirm" button. [ALT]		**Step 6:** The system will then deactivate the selected user by routing the action through the Angular frontend and user service, which sends the request to the appropriate HTTP DELETE endpoint in Visual Studio (UserController – DeactivateUser). The API endpoint will validate the user and update its status in the database using Entity Framework Core in MS SQL Server. This ensures that the user is soft deleted. The following entity is involved in the process: User Entity: • UserId (PK) • UserName • UserSurname • UserDateOfBirth • UserEmail

			• UserPassword • ProfilePicture • ActiveStatus • UserStatus • UserShirtSize [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 user. 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 user has been deactivated. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 8: The system will then display a confirmation message indicating that the user has been successfully deactivated. This is implemented in the Angular frontend using a toaster notification component. The confirmation message will display the text: “User deactivated successfully”.
Alternate courses:	ALT-Step 5: The super admin does not confirm the changes by selecting the “Cancel” button. Go back to step 2.		

	ALT-Step 6: The system fails to deactivate the skill and displays a toaster message: "Failed to deactivate user". Terminate use case.
Conclusion:	The active user has been deactivated, and a confirmation message is shown.
Post-condition:	The system updates the user's status to inactive in the User table.
Business Rules:	• Only super admin can deactivate a user.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 12: 2.4 Deactivate User Technical Use Case Narrative

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>AUTHOR:</td><td colspan="2">Stephan Brednehnn</td><td>Date: 28 April 2025
Version: 1.0</td></tr>
<tr><td>USE CASE NAME:</td><td colspan="2">Create Role</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>USE CASE ID:</td><td colspan="2">2.8</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">Administrator</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 administrative use case defines the creation of a new global catalogue role within the system. The Super Admin interacts with a specialized form interface to define the role's Name and Description. The Angular frontend transmits this payload to the secured “POST api/projectroles” endpoint. The backend's “IProjectRoleCatalogService” applies uniqueness validations before saving the new definition as a “ProjectRole” Entity using Entity Framework, thereby making the role available globally for future project assignments.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The Administrator must be logged in with a valid JWT token.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The Admin clicks the primary "Create Role" action button within the UI</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 Admin navigates to the Roles Management view and clicks the "Create Role" button.</td><td></td><td>Step 2: The Angular frontend triggers an event that opens a dynamic Dialog/Modal component layered over the active view. The</td></tr>
</table>

			system renders an Angular Reactive Form containing: • A "Role Name" standard text input field (configured with a mandatory required validator). • A "Role Description" multi-line text area input. • A secondary "Cancel" button. • A primary "Save Role" submit button.
	Step 3: The Admin populates the input fields, typing the desired Role Name and describing its intended responsibilities.		**Step 4:** The Reactive Form's internal state updates in real-time. Once the mandatory "Role Name" field is populated, the "Save Role" button's “disabled” state changes to false, allowing interaction.
	Step 5: The Admin confirms the details by clicking the "Save Role" button.		**Step 6:** The frontend extracts the form values and maps them into a JSON payload. The Angular service issues a secured “POST” request to the “api/projectroles” API endpoint, transmitting the “CreateProjectRoleDto” data structure in the request body.
			Step 7: The API's “ProjectRolesController” intercepts the request. Before executing business logic, the system executes the “access.EnsureCanMana

			geCatalogAsync(User)" method to strictly verify the incoming JWT claims, ensuring the user possesses the correct authorization level to change catalogue roles.
			Step 8: The controller invokes the "IProjectRoleCatalogService.CreateAsync()" method. The backend system queries the database to enforce a critical business rule: it checks if any existing "ProjectRole" entity already uses the exact same "RoleName". [ALT]
			Step 9: Passing validation, the system instantiates a new ProjectRole entity in server memory. It maps the DTO properties to the entity framework model: **ProjectRole Entity:** • ProjectRoleId (Auto-generated PK) • ProjectId (Explicitly set to null to denote this as a global catalogue definition, rather than a scoped role) • RoleName (Mapped from payload) • RoleDescription (Mapped from payload) • IsActive (Initialized to true)

			Step 10: The system utilizes _dbContext.ProjectRoles.Add() to commit the new database record and returns a 'HTTP 201 Created' response.
			Step 12: The Angular frontend receives the successful response, dismisses the dialog modal, pushes the newly returned role object into the local state array bound to the data table (preventing a full page reload), and triggers an Angular Material Snackbar rendering a green success toast: "Role created successfully."
Alternate courses:	[ALT] Step 8: The backend database query detects an existing record with a matching RoleName. A respone is returned with error message: "A catalog role with this name already exists." The frontend prevents the modal from closing, highlights the Role Name input in red		
Conclusion:	A new role is added that can be used in the creation of any project on the system.		
Post-condition:	A new, normalized ProjectRole entity is permanently committed to the SQL database with ProjectId assigned to null.		
Business Rules:	• Project Role management.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 13: 2.8 Create Role Technical Use Case Narrative

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>AUTHOR:</td><td colspan="2">Stephan Brednehnn</td><td>Date: 28 April 2025
Version: 1.0</td></tr>
<tr><td>USE CASE NAME:</td><td colspan="2">View Role</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>USE CASE ID:</td><td colspan="2">2.9</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">Administrator</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 details the retrieval and visualization of the existing catalogue roles defined within the system. The Angular frontend initializes the Roles Management component, that dispatches a ‘GET’ request to the “api/projectroles” endpoint. The backend’s “IProjectRoleCatalogService” queries the database via Entity Framework Core, gathers all the “ProjectRole” entities, and transmits them back. The UI displays these roles within a data table.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The Administrator must be logged in with a valid JWT token.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The Admin selects the “Roles Management" link from the system navigation bar.</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 Admin navigates into the Roles Management view.</td><td></td><td>Step 2: The Angular Router redirects to the Role Management module and sends a secured “GET” request to</td></tr>
</table>

			the “api/projectroles” API endpoint. **Step 3:** The “ProjectRolesController” receives the request. The system immediately executes “access.EnsureCanViewCatalogAsync(User)” to validate the presence of appropriate permissions embedded within the incoming JWT.
			Step 4: The controller invokes the “IProjectRoleCatalogService.ListAsync()” method. The backend runs an Entity Framework LINQ query targeting the “ProjectRole’ table. The system filters the query specifically for records where the “ProjectId” property is “null” (filtering out localized project-scoped roles to guarantee only global catalogue definitions are fetched).
			Step 5: The system retrieves the matching ProjectRole entities from the database: **ProjectRole Entity Fields Retrieved:** • ProjectRoleId • RoleName • RoleDescription • IsActive
			Step 6: The system renders the UI, binding

			the data source to an Angular Material Data Table. The table constructs rows dynamically, displaying columns for: • Role Name • Role Description • Status (rendered as a dynamically coloured badge: Green for Active, Grey/Red for Inactive) • An "Actions" column populated with icon buttons tailored to the row's state (e.g., Edit button, Deactivate button if active, Reactivate button if inactive).
Alternate courses:	None		
Conclusion:	The administrator can view all global roles that can be assigned to projects.		
Post-condition:	The Angular front end table is fully populated with data.		
Business Rules:	• Project Role Management		
Implementation constraints and specifications:	• None		
Assumptions:	• There are Roles to populate the front end with.		
Open issues:	• None		

Table 14: 2.9 View Role Technical Use Case Narrative

System Name: Integrated NGO Management System			
AUTHOR:	Stephan Brednehnn	**Date:** 28 April 2025 **Version:** 1.0	
USE CASE NAME:	Update Role	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
USE CASE ID:	2.10		
PRIORITY:	High		
SOURCE:	EPI-USE		
Primary Business Actor (PBA):	Administrator		
Primary System Actor (PSA):	• None		
Other Participating Actors:	• None		
Other Interested Stakeholders:	None		
Description:	This use case describes the modification of pre-existing catalogue role definitions. The Admin initiates the flow from the data table, launching a pre-populated Angular Reactive Form. Edits are c routed to the “PUT api/projectroles/{id}” endpoint. The backend validates the payload, ensuring the uniqueness constraints are not violated, before applying changes directly to the target “ProjectRole” Entity and synchronizing the state with the database.		
Pre-condition:	The Administrator must be logged in with a valid JWT token, and a role exists.		
Trigger:	The Admin clicks the "Edit" action icon button located on a specific row within the Roles Management data table.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The Admin locates the desired role in the data table and clicks the corresponding "Edit" icon button.		**Step 2:** The Angular frontend captures the selected role's ID. The system opens an "Edit Role" Modal component.

			Step 3: The UI initializes a Reactive Form, pre-populating the "Role Name" and "Role Description" input fields with the existing values passed from the table row.
	Step 4: The Admin modifies the values within the "Role Name" and/or "Role Description" fields, then clicks the "Update Role" button.		**Step 5:** The Reactive Form validates the input, ensuring mandatory fields remain populated. The frontend extracts the updated values, constructs the “UpdateProjectRoleDto” payload, and executes a secured “PUT” request to “api/projectroles/{id}”, where “{id}” corresponds to the “ProjectRoleId”.
			Step 6: The API's “ProjectRolesController” receives the request and checks access via “_access.EnsureCanManageCatalogAsync(User)”.
			Step 7: The controller hands over to “IProjectRoleCatalogService.UpdateAsync()”. The backend queries the database using “FindAsync” to fetch the specific “ProjectRole” entity corresponding to the provided ID.
			Step 8: The system executes a conflict check query. It scans the database to determine if the newly provided

			"RoleName" matches any other existing role (excluding the role currently being edited). [ALT]
			Step 9: Passing validations, the system overwrites the entity's properties in memory, updating "RoleName" and "RoleDescription" with the payload's strings.
			Step 10: The system executes "_dbContext.SaveChangesAsync()" to store the updated role details.
			Step 11: The API returns an "HTTP 200 OK" response containing the updated role object JSON.
			Step 12: The Angular frontend receives the successful HTTP response. The system closes the dialog modal. It locates the specific object within the local data table array and overwrites it with the returned object, triggering an immediate UI change of that specific row.
Alternate courses:	**[ALT] Step 8:** The uniqueness query discovers a conflict; the new "RoleName" is already claimed by a different entity. The API responds with an "HTTP 409		

	Conflict". The frontend keeps the modal open and displays an inline validation error prompting the user to choose a different name.
Conclusion:	The role's name and/or description is successfully changed.
Post-condition:	he specific "ProjectRole" entity retains its original Primary Key but contains an updated Name and/or description
Business Rules:	• Project role management.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 15: 2.10 Update Role Technical Use Case Narrative

System Name: Integrated NGO Management System			
AUTHOR:	Stephan Brednehnn	**Date:** 28 April 2025 **Version:** 1.0	
USE CASE NAME:	2.11	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
USE CASE ID:	Reactivate Role		
PRIORITY:	High		
SOURCE:	EPI-USE		
Primary Business Actor (PBA):	Administrator		
Primary System Actor (PSA):	• None		
Other Participating Actors:	• None		
Other Interested Stakeholders:	None		
Description:	This use case permits an Admin to restore a previously deactivated catalogue role to an active state, thereby enabling it to be assigned to users within projects again. The interaction is triggered via an action button in the data table. The Angular service issues a “PUT api/projectroles/{id}/reactivate” request. The backend locates the specific “ProjectRole” Entity and modifies its Boolean ‘IsActive’ attribute , concluding with an updated UI rendering the new state.		
Pre-condition:	The Admin must be authenticated. A target role must exist in the database and possess an “IsActive” property set to “false”.		
Trigger:	The Admin locates an inactive role and clicks the "Reactivate" button in the data table row.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The Admin clicks the "Reactivate" action button associated with an inactive role row.		**Step 2:** The Angular frontend intercepts the click event. To prevent accidental changes, the UI invokes a lightweight confirmation dialog ("Are

			you sure you wish to reactivate this role?").
			Step 4: The frontend extracts the "ProjectRoleId" from the row's data context and sends "PUT" request directly to the "api/projectroles/{id}/reactivate" API endpoint.
			Step 5: The "ProjectRolesController" receives the incoming request and validates the administrator's JWT claims utilizing "_access.EnsureCanManageCatalogAsync(User)".
			Step 6: The controller invokes "IProjectRoleCatalogService.ReactivateAsync(id)". The backend issues a query to Entity Framework to locate the target "ProjectRole" entity using the provided ID.
			Step 8: The system invokes "_dbContext.SaveChangesAsync()" to commit the specific boolean state change to the SQL database.
			Step 9: The API constructs an "HTTP 200 OK" response, embedding the updated JSON representation of the role within the body.

			Step 10: The Angular frontend processes the response. The data table's specific row object is updated. The UI changes the row, dynamically transitioning the status badge from a "Inactive" state to a "Active" state, and swaps the action button back to the "Deactivate" option.
Alternate courses:	None		
Conclusion:	The role is updated to show an active state and will once again appear for selection when creating a project.		
Post-condition:	The target 'ProjectRole' entity's 'IsActive' Attribute is changed to true		
Business Rules:	• Project Role Management		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 16: 2.11 Reactivate Role Technical Use Case Narrative

System Name: Integrated NGO Management System			
AUTHOR:	Stephan Brednehnn	**Date:** 28 April 2025 **Version:** 1.0	
USE CASE NAME:	Deactivate Role	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
USE CASE ID:	2.12		
PRIORITY:	High		
SOURCE:	EPI-USE		
Primary Business Actor (PBA):	Administrator		
Primary System Actor (PSA):	• None		
Other Participating Actors:	• None		
Other Interested Stakeholders:	None		
Description:	This use case describes the process of retiring a catalogue role. The system prevents destructive deletion to maintain historical integrity. Instead, the UI triggers a “PUT api/projectroles/{id}/deactivate” request. The backend’s “IProjectRoleCatalogService” performs mandatory relational dependency checks to guarantee the role is not actively assigned to users within ongoing projects. If clear, the “ProjectRole” Entity's status flag is toggled, hiding it from future selection when creating a project..		
Pre-condition:	The Admin must be authenticated. The target role must be designated as active in the database.		
Trigger:	The Admin clicks the "Deactivate" icon button on a specific role within the Role Management data table.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The Admin clicks the "Deactivate" button located on an active role's row.		**Step 2:** The Angular frontend detects the action. To prevent unintended changes, the UI displays a warning toast: "Deactivating this

			role will prevent it from being assigned in the future. Are you sure?"
	Step 3: The Admin reads the prompt and confirms the deactivation action. [ALT]		**Step 4:** The frontend extracts the "ProjectRoleId" and executes a "PUT" request targeting the "api/projectroles/{id}/deactivate" endpoint.
			Step 5: The API receives the request. The controller asserts the administrator's JWT claims via "_access.EnsureCanManageCatalogAsync(User)"
			Step 6: "IProjectRoleCatalogService.DeactivateAsync(id)" is invoked. The backend queries the database using Entity Framework to locate the target "ProjectRole" entity.
			Step 7: The system executes a dependency analysis query. It actively scans tables such as "UserProjectRole" to verify if the role is currently bound to any users in active system projects. [ALT]
			Step 8: Upon confirming the role is safely unassigned and isolated, the system changes the entity in memory, setting the boolean "IsActive" property to "false".

			Step 9: The system calls “dbContext.SaveChanges Async()” to commit the deactivation transaction to the SQL database.
			Step 10: The API returns an “HTTP 200 OK” JSON response containing the modified role object.
			Step 11: The Angular frontend receives the successful payload. The UI updates the corresponding object within the data table. Angular's change detection triggers an update of changeing the status badge from "Active" to "Inactive," and the button swaps to the "Reactivate" option.
Alternate courses:	[ALT] Step 3: The admin does not confirm and the use case is terminated.		
	[ALT] Step 7: The rational dependency query detects that the role is actively assigned, the use case terminates with an error toast: "This role is still in use on an active project. Remove those team assignments or user project roles before deactivating."		
Conclusion:	The role is marked as deactivated, and will no longer be available when creating a project.		
Post-condition:	The “ProjectRole” Entity's “IsActive” attribute changes to false. It will no longer populate dropdown menus.		
Business Rules:	• Role Management		
Implementation constraints and specifications:	• None		

Assumptions:	• None
Open issues:	• None

Table 17:2.12 Deactivate Role Technical Use Case Narrative

2.4.3 Group Subsystem Technical Narratives

INTEGRATED NGO MANAGEMENT SYSTEM			
AUTHOR:	Nicholas Bailey	**Date: 30/04/2026** **Version:** 1	
USE CASE NAME:	Create Group	**USE CASE TYPE**	
USE CASE ID:	4.1	Business Requirements:	□
PRIORITY:	Medium	System Analysis:	□
SOURCE:	EPI-USE	System Design:	✓
Primary Business Actor (PBA):	Super Admin		
Primary System Actor (PSA):	• None		
Other Participating Actors:	• None		
Other Interested Stakeholders:	• Users		
Description:	This use case describes how a Super Admin creates a new group within a selected project. The system allows the user to enter the group's details, assign users, and then the system will validate the input, and confirm the creation before saving the group to the database. The group is then stored, linked to each user, and a confirmation message is displayed. The transaction is also recorded in the audit log.		
Pre-condition:	The super admin must be logged into the system.		
Trigger:	The super admin wants to add a new group 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 group to the system.		**Step 2:** The system invokes UC 4.2 (View Groups) to display a pop-up showing all the groups' details. This is done using Angular frontend and the group service, which routes the request

			via Angular HTTP Client to the corresponding HTTP GET API endpoint in Visual Studio. (Group Controller)
	Step 3: The super admin selects the “Add Group” button.		**Step 4:** The system will then load and display the Create Group Modal with the following components: • Create Group modal title • Group name section: ○ Group name label and input field ○ Placeholder text: “Enter group name” • User selection section: ○ Add user label ○ Dropdown list ▪ Default option: “Select user” ▪ Displays username and user surname • Add button (Disabled when no user is selected) • Selected users section: ○ List of selected users displayed as tags ○ Displays username and surname ○ Remove (×) Button for each user • Action buttons section: • Create button (Disabled if the group name is empty) • Cancel button

			• The User dropdown 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
	Step 5: The super admin enters the group's details, adds users and then selects the "Create" button. [ALT]		**Step 6:** The system validates the entered details using Angular form validation and frontend checks. The system validates the following: • The Group Name input field is not null or empty. • The Group Name input does not contain only whitespace characters. • Duplicate users are not added to the selectedUsers list. [ALT]
			Step 7: The system displays a custom Angular confirmation modal, which is triggered when the Super Admin clicks the "Create" button. • Confirmation modal title ("Confirm group creation") • Confirmation message label ○ Displays message: "Are you sure you want to create this group?" • Action buttons: ○ Confirm button ○ Cancel button

	Step 8: The super admin selects the "Confirm" button. [ALT]		**Step 9:** The system will then capture the new Group's details and route the data to the corresponding API endpoints using the Angular frontend and group service to route the request to the appropriate HTTP POST endpoint in Visual Studio. (GroupController – CreateGroup). The API endpoint will validate the incoming data and create the new Group in the database using Entity Framework Core in MS SQL Server. The following entities are involved in the process: Group Entity: • GroupId (PK) • ProjectId (FK) • GroupName • ActiveStatus • DateCreated UserGroup Entity (Junction Table): • UserId (PK, FK) • GroupId (PK, FK) [ALT]
			Step 10: The system will synchronise the newly created Group with Firestore to support real time messaging between users.
			Step 11: 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 group. 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 group has been created. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 12: The system will then display a confirmation message indicating that the group has been successfully created. This is implemented in the Angular frontend using a toaster notification component. The confirmation message will display the text: "Group 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 displays a toaster message: "Invalid details". Go back to step 4.		
	ALT-Step 8: The super admin does not confirm the changes by selecting the "Cancel" button. Go back to step 4.		
	ALT-Step 9: The system fails to save the new group and displays a toaster message: "Failed to save new group". Terminate use case.		
Conclusion:	The new group's details have been saved onto the system, and a confirmation message is shown.		
Post-condition:	The system saves the new group onto the Group table and the assigned users onto the UserGroup table.		

Business Rules:	• Only super admin can add a new group.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 18: 4.1 Create Group Technical Use Case Narrative

<table>
<tr><th colspan="4">INTEGRATED NGO MANAGEMENT SYSTEM</th></tr>
<tr><td>AUTHOR:</td><td colspan="2">Nicholas Bailey</td><td>Date: 30/04/2026
Version: 1</td></tr>
<tr><td>USE CASE NAME:</td><td colspan="2">View Groups</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>USE CASE ID:</td><td colspan="2">4.2</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">User</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 when a user wants to view the groups for a specific project on the system The system retrieves active projects along with their group count and displays them in a card using the Angular frontend. The user can select a project, and the system retrieves all related groups, including their details and assigned users through the relevant API endpoints and displayed in a pop-up modal. The groups are shown with their names and members.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The user must be logged into the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The user wants to view the groups for a specific project to which they are assigned.</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 wants to view the groups for a specific project to which they are assigned.</td><td></td><td></td></tr>
</table>

	Step 2: The user selects the "Teams" tab to navigate to the teams' page.		**Step 3:** The system sends the request through the Angular frontend and project card service to the relevant HTTP GET API endpoint in Visual Studio (GroupController – GetProjectCards) which retrieves all projects and their associated group counts from the Project and Group tables in the database. **Project Entity:** • ProjectId (PK) • ProjectName **Group Entity:** • GroupId (PK) • ProjectId (FK) • GroupName • ActiveStatus • DateCreated Only active groups with are included in the results.
			Step 4: The system displays a list of projects and the number of groups associated with each project based on the user's role permissions. The retrieved projects and their number of groups are shown in the Angular frontend using a dynamic card layout. The system will load and display the cards with the following components: • Page title: ○ "Active Projects" • Project card for each project: ○ Project name ○ Group count

			○ Clickable card event (Allows the user to select a project) The system ensures that only projects and groups that the user is assigned to are displayed. [ALT]
	Step 5: The user selects a project card.		**Step 6:** The system retrieves all groups associated with the selected project, including their details and the users assigned to each group through the group service in Angular frontend. The service sends a request via Angular HTTP Client to the HTTP GET endpoint in Visual Studio (GroupController – GetByProject). The API then retrieves data from the following tables: Group Entity: • GroupId (PK) • ProjectId (FK) • GroupName • ActiveStatus • DateCreated UserGroup Entity (Junction Table): • UserId (PK, FK) • GroupId (PK, FK) User Entity: • UserId (PK) • UserName • UserSurname
			Step 7: The system displays a pop-up modal on the Angular frontend containing all the retrieved group details for the selected project using an

			Angular modal that is triggered when a project card is selected. The view modal has the following components: • Modal heading: ○ Project name The system displays each group with the following details: • Group name • Group actions: ○ Edit button ○ Deactivate button • Group members section: ○ Displays assigned users ○ Each user is shown as a tag containing: ▪ Username ▪ User Surname The modal shows a view of all active groups within the selected project. [ALT]
	Step 8: The user can view the groups.		
Alternate courses:	ALT-Step 4: The system does not find any active projects. Terminate use case.		
	ALT-Step 7: The system does not find any active groups in the selected project. Terminate use case.		
Conclusion:	The system displays all the groups' details.		
Post-condition:	All the related groups' details are displayed.		

Business Rules:	• Users can only view groups they are assigned to. • Admins and super admins can view all groups on the system.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 19: 4.2 View Groups Technical Use Case Narrative

INTEGRATED NGO MANAGEMENT SYSTEM			
AUTHOR:	Nicholas Bailey	**Date: 30/04/2026** **Version:** 1	
USE CASE NAME:	Update Group	**USE CASE TYPE**	
USE CASE ID:	4.3	Business Requirements:	□
PRIORITY:	Medium	System Analysis:	□
SOURCE:	EPI-USE	System Design:	✓
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 an existing group within a selected project. The system retrieves and displays the current group details, including the group name and assigned users. The Super Admin can update the group name and manage group membership by adding or removing users. Once the changes are made, the system validates the updated information before submitting it to the backend API. The use case ends when the system successfully saves the updated group's details to the database.		
Pre-condition:	The super admin must be logged into the system.		
Trigger:	The super admin wants to update a group'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 group's details on the system.		**Step 2:** The system invokes UC 4.2 (View Groups) to display a pop-up showing all the groups' details. This is done using Angular frontend and the group service, which routes the request

			via Angular HTTP Client to the corresponding HTTP GET API endpoint in Visual Studio. (Group Controller)
	Step 3: The super admin selects the "Edit" button on a specific group.		**Step 4:** The system displays the Edit Group Modal with the group name pre-populated and editable. The modal contains the following components: • Edit Group modal title • Group name section: ○ Input field ○ Pre-populated with the selected group's current name ○ Allows the user to modify the group name • Add User Section: ○ Dropdown list populated via the user service ○ Displays: ▪ Username ▪ User surname ○ Add Button ○ Prevents duplicate selection • Users Section: ○ Displays a list of all users currently assigned to the group ○ Each user is shown as a tag containing: ▪ Username ▪ User surname

			○ Each user tag includes a remove button • The User dropdown 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
	Step 5: The super admin updates the group's details and selects the "Save" button. [ALT]		**Step 6:** The system validates the updated group's details using Angular form validation and frontend checks before allowing the update to proceed. The system validates the following: • The Group Name input field is not null or empty. • The Group Name input does not contain only whitespace characters. • Duplicate users are not added to the group's user list when adding members. [ALT]
			Step 7: The system displays a custom Angular confirmation modal, which is triggered when the Super Admin clicks the "Save" button. • Confirmation modal title ("Confirm group update") • Confirmation message label ○ Displays message: "Are you sure you want to update this group?" • Action buttons:

			○ Confirm button ○ Cancel button
	Step 8: The super admin selects the "Confirm" button. [ALT]		**Step 9:** The system will then capture the updated Group details and route the data to the corresponding API endpoint using the Angular frontend and group service, which sends the request to the appropriate HTTP PUT endpoint in Visual Studio (GroupController – UpdateGroup). The API endpoint will validate the incoming data and update the existing Group record in the database using Entity Framework Core in MS SQL Server. The following entities are involved in the process: Group Entity: • GroupId (PK) • ProjectId (FK) • GroupName • ActiveStatus • DateCreated UserGroup Entity (Junction Table): • UserId (PK, FK) • GroupId (PK, FK) The system also updates the UserGroup table to reflect any changes made, including newly added users and removed users. [ALT]
			Step 10: The system synchronises the updated group data with Firestore to support real-time updates across connected users.

			Step 11: 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 group. 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 group has been updated. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 12: The system will then display a confirmation message indicating that the group has been successfully updated. This is implemented in the Angular frontend using a toaster notification component. The confirmation message will display the text: “Group 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 displays a toaster message: “Invalid details”. Go back to step 4.		
	ALT-Step 8: The super admin does not confirm the changes by selecting the “Cancel” button. Go back to step 4.		
	ALT-Step 9: The system fails to save the new group and displays a toaster message: “Failed to update group”. Terminate use case.		

Conclusion:	The updated group's details have been saved onto the system, and a confirmation message is shown.
Post-condition:	The system saves the changes to the Group and UserGroup tables.
Business Rules:	• Only super admin can update a group's details.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 20: 4.3 Update Group Technical Use Case Narrative

INTEGRATED NGO MANAGEMENT SYSTEM			
AUTHOR:	Nicholas Bailey	**Date: 30/04/2026** **Version:** 1	
USE CASE NAME:	Deactivate Group	**USE CASE TYPE**	
USE CASE ID:	4.4	Business Requirements:	□
PRIORITY:	Medium	System Analysis:	□
SOURCE:	EPI-USE	System Design:	✓
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 deactivates an existing group within the system. The system displays the selected group and allows the Super Admin to select deactivate. The group is then soft deleted by changing the status to inactive, to ensure data integrity and historical tracking. The system will no longer show the group in the active group listings. The use case ends when the group has been successfully deactivated and is no longer visible as an active group within the system.		
Pre-condition:	The super admin must be logged into the system.		
Trigger:	The super admin wants to deactivate a group on the system.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The super admin wants to deactivate a group on the system.		**Step 2:** The system invokes UC 4.2 (View Groups) to display a pop-up showing all the groups' details. This is done using Angular frontend and the group service, which routes the request

			via Angular HTTP Client to the corresponding HTTP GET API endpoint in Visual Studio. (Group Controller)
	Step 3: The super admin selects the "Deactivate" button on a specific group. [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 group deactivation") • Confirmation message label ○ Displays message: "Are you sure you want to deactivate this group?" • Action buttons: ○ Confirm button ○ Cancel button
	Step 5: The super admin selects the "Confirm" button. [ALT]		**Step 6:** The system will then deactivate the selected group by routing the action through the Angular frontend and group service, which sends the request to the appropriate HTTP DELETE endpoint in Visual Studio (GroupController – DeactivateGroup). The API endpoint will validate the group and update its status in the database using Entity Framework Core in MS SQL Server. This ensures that the group is soft deleted. The following entity is involved in the process: Group Entity: • GroupId (PK) • ProjectId (FK) • GroupName • ActiveStatus

			• DateCreated The deactivated group is then no longer displayed in the active group 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 group. 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 group has been deactivated. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 8: The system will then display a confirmation message indicating that the group has been successfully deactivated. This is implemented in the Angular frontend using a toaster notification component. The confirmation message will display the text: “Group deactivated successfully”.
Alternate courses:	ALT-Step 3: The super admin changes their mind and closes the pop-up by selecting the “Cancel” button. Terminate use case.		
	ALT-Step 5: The super admin does not confirm the changes by selecting the “Cancel” button. Go back to step 2.		

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

Table 21: 4.4 Deactivate Group Technical Use Case Narrative

INTEGRATED NGO MANAGEMENT SYSTEM			
AUTHOR:	Nicholas Bailey	**Date: 30/04/2026** **Version:** 1	
USE CASE NAME:	Create Skill	**USE CASE TYPE**	
USE CASE ID:	4.5	Business Requirements:	□
PRIORITY:	Medium	System Analysis:	□
SOURCE:	EPI-USE	System Design:	✓
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 creates a new skill within the system. The system allows the user to enter the skill's name and description, validates the input, and requests confirmation before saving the skill. Once confirmed, the system sends the data to the backend API endpoint, where the skill is created in the database. The new skill is then displayed on the Skills page as a card, and a confirmation message is shown.		
Pre-condition:	The super admin must be logged into the system.		
Trigger:	The super admin wants to add a new skill 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 skill to the system.		**Step 2:** The system invokes UC 4.6 (View Skills) to display all the current skills. This is done using the Angular frontend and skills service, which routes the request via Angular HTTP Client to the corresponding HTTP GET API endpoint in Visual Studio

			(SkillController). The retrieved skills are then displayed as skill cards on the Skills page.
	Step 3: The super admin selects the "Add Skill" button.		**Step 4:** The system will then load and display the Create Skill Angular Modal with the following components: • Create Skill modal title • Skill name section: ○ Skill name label and input field ○ Placeholder text: "Skill Name" • Skill description section: ○ Skill description label and input field ○ Placeholder text: "Skill Description" • Action buttons section: ○ Create button (Disabled if the skill name or description is empty) ○ Cancel button
	Step 5: The super admin enters the skill's details and then selects the "Create" button. [ALT]		**Step 6:** The system validates the entered details using Angular form validation and frontend checks. The system validates the following: • The Skill Name input field is not null or empty. • The Skill Name input does not contain only whitespace characters. • The Skill Description input field is not null or empty.

			• The Skill Description input does not contain only whitespace characters. [ALT]
			Step 7: The system displays a custom Angular confirmation modal, which is triggered when the Super Admin clicks the "Create" button. • Confirmation modal title ("Confirm skill creation") • Confirmation message label ○ Displays message: "Are you sure you want to create this skill?" • Action buttons: ○ Confirm button ○ Cancel button
	Step 8: The super admin selects the "Confirm" button. [ALT]		**Step 9:** The system will then capture the new Skill's details and route the data to the corresponding API endpoints using the Angular frontend and skills service, which sends the request to the appropriate HTTP POST endpoint in Visual Studio (SkillController – Create). The API endpoint will validate the incoming data and create the new Skill in the database using Entity Framework Core in MS SQL Server. The following entity is involved in the process: Skill Entity: • SkillId (PK) • SkillName • SkillDescription

			• IsActive [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 skill. 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 skill has been created. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 11: The system will then display a confirmation message indicating that the skill has been successfully created. This is implemented in the Angular frontend using a toaster notification component. The confirmation message will display the text: “Skill 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 displays a toaster message: “Invalid details”. Go back to step 4.		

	ALT-Step 8: The super admin does not confirm the changes by selecting the "Cancel" button. Go back to step 4.
	ALT-Step 9: The system fails to save the new skill and displays a toaster message: "Failed to save new skill". Terminate use case.
Conclusion:	The new skill's details have been saved onto the system, and a confirmation message is shown.
Post-condition:	The system saves the new skill onto the Skills table.
Business Rules:	• Only super admin can add a new skill.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 22: 4.5 Create Skill Technical Use Case Narrative

INTEGRATED NGO MANAGEMENT SYSTEM			
AUTHOR:	Nicholas Bailey	**Date: 30/04/2026** **Version:** 1	
USE CASE NAME:	View Skills	**USE CASE TYPE**	
USE CASE ID:	4.6	Business Requirements:	□
PRIORITY:	Low	System Analysis:	□
SOURCE:	EPI-USE	System Design:	✓
Primary Business Actor (PBA):	Admin		
Primary System Actor (PSA):	• None		
Other Participating Actors:	• None		
Other Interested Stakeholders:	• None		
Description:	This use case describes when an admin views the details of all the skill within the system. The system displays the skills as cards containing the skill's name and description. The use case ends when the admin sees all the skills displays as cards.		
Pre-condition:	The admin must be logged into the system.		
Trigger:	The admin wants to view the skills on the system.		

Typical Course of Events:	Actor Action	System Response	
		Manual Action	**Automated Action**
	Step 1: The admin wants to view the skills on the system.		
	Step 2: The admin selects the "Skills" widget on the home dashboard.		**Step 3:** The system sends the request through the Angular frontend and skills service to the relevant HTTP GET API endpoint in Visual Studio (SkillController – GetAll), which

			retrieves all active skills from the Skills table in the database. Skill Entity: • SkillId (PK) • SkillName • SkillDescription • IsActive Only active skills are shown.
			Step 4: The system will then display the Skills page using the Angular frontend, showing all retrieved skills as interactive skill cards based on the data returned from the API. • The Skills page includes the following components: • Page title (“Skills”) • Action section: ○ Add Skill button (navigates to Create Skill modal) • Skills display section: ○ Skills are displayed as cards in a grid layout ○ Each skill card displays: ▪ Skill Name ▪ Skill Description • Interaction behaviour: • Each skill card is clickable [ALT]
	Step 5: The admin can view the skills.		

Alternate courses:	ALT-Step 4: The system does not find any active skills. Terminate use case.
Conclusion:	All skills' details are displayed in cards.
Post-condition:	The system displays all the current skills in cards.
Business Rules:	• Admins and super admins can view the skills.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 23: 4.6 View Skills Technical Use Case Narrative

INTEGRATED NGO MANAGEMENT SYSTEM		
AUTHOR:	Nicholas Bailey	**Date: 30/04/2026** **Version:** 1
USE CASE NAME:	View Skill	**USE CASE TYPE**
USE CASE ID:	4.7	Business Requirements: □
PRIORITY:	Low	System Analysis: □
SOURCE:	EPI-USE	System Design: ✓
Primary Business Actor (PBA):	Admin	
Primary System Actor (PSA):	• None	
Other Participating Actors:	• None	
Other Interested Stakeholders:	• None	
Description:	This use case describes when an admin views the details of a selected skill within the system. The admin will select a skill card, and it will open a modal. From this modal, the user can view full skill information and access actions such as editing or deactivating the skill. The use case ends when the admin can view the skill's details.	
Pre-condition:	The admin must be logged into the system.	
Trigger:	The admin wants to view the specific skill's details on the system.	

Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The admin wants to view the specific skill's details on the system.		**Step 2:** The system invokes UC 4.6 (View Skills) to display all the current skills. This is done using the Angular frontend and skills service, which routes the request via Angular HTTP Client to the corresponding HTTP GET API endpoint in Visual Studio

			(SkillController). The retrieved skills are then displayed as skill cards on the Skills page.
	Step 3: The admin selects a skills card.		**Step 4:** The system retrieves the selected skill details through the skills service in the Angular frontend. The service sends a request via Angular HTTP Client to the HTTP GET endpoint in Visual Studio (SkillController – Get), which retrieves the skill based on the selected SkillId. The API then retrieves data from the following table: **Skill Entity:** • SkillId (PK) • SkillName • SkillDescription • IsActive
			Step 5: The system displays a View Skill Modal on the Angular frontend containing the selected skill details. The modal is triggered when a skill card is selected. The view modal has the following components: • Modal heading: ○ Skill name • Skill details section: ○ Displays the skill description • Action buttons section: ○ Edit button ○ Deactivate button ○ Close button

	Step 6: The admin can view the skill.		
Alternate courses:	None		
Conclusion:	The system displays a modal showing the skill's details.		
Post-condition:	The skill's details are displayed.		
Business Rules:	• Admins and super admins can view the skills.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 24: 4.7 View Skill Technical Use Case Narrative

INTEGRATED NGO MANAGEMENT SYSTEM			
AUTHOR:	Nicholas Bailey	**Date: 30/04/2026** **Version:** 1	
USE CASE NAME:	Update Skill	**USE CASE TYPE**	
USE CASE ID:	4.8	Business Requirements:	□
PRIORITY:	Low	System Analysis:	□
SOURCE:	EPI-USE	System Design:	✓
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 an existing skill in the system. The system allows the user to modify the skill's name and description on an edit modal. Once the user confirms the changes, the system validates the input and updates the skill in the database. The updated skill information is then reflected on the skills page.		
Pre-condition:	The super admin must be logged into the system.		
Trigger:	The super admin wants to update a skill'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 skill's details on the system.		**Step 2:** The system invokes UC 4.7 (View Skill) to display the selected skill details in a pop-up modal. This is done using the Angular frontend and skills service, which routes the request via Angular HTTP Client to the corresponding HTTP GET API endpoint

			in Visual Studio (SkillController). The retrieved skill details are then used to populate the edit form within the modal.
	Step 3: The super admin selects the "Edit" button.		**Step 4:** The system displays the Edit Skill Modal with the selected skill's details pre-populated and editable. The modal contains the following components: • Edit Skill modal title • Skill name section: ○ Input field ○ Pre-populated with the selected skill's current name • Skill description section: ○ Input field ○ Pre-populated with the selected skill's current description • Action buttons section: ○ Save button ○ Cancel button
	Step 5: The super admin updates the skill's details and selects the "Save" button. [ALT]		**Step 6:** The system validates the updated details using Angular form validation and frontend checks. The system validates the following: • The Skill Name input field is not null or empty. • The Skill Name input does not contain only whitespace characters. • The Skill Description input field is not null or empty.

			• The Skill Description input does not contain only whitespace characters. [ALT]
			Step 7: The system displays a custom Angular confirmation modal, which is triggered when the Super Admin clicks the "Save" button. • Confirmation modal title ("Confirm skill update") • Confirmation message label ○ Displays message: "Are you sure you want to update this skill?" • Action buttons: ○ Confirm button ○ Cancel button
	Step 8: The super admin selects the "Confirm" button. [ALT]		**Step 9:** The system will then capture the updated Skill details and route the data to the corresponding API endpoint using the Angular frontend and Skills service, which sends the request to the appropriate HTTP PUT endpoint in Visual Studio (SkillController – Update). The API endpoint will validate the incoming data and update the existing Skill record in the database using Entity Framework Core in MS SQL Server. The following entity is involved in the process: Skill Entity: • SkillId (PK) • SkillName • SkillDescription • IsActive

			[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 group. 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 skill has been updated. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 11: The system will then display a confirmation message indicating that the skill has been successfully updated. This is implemented in the Angular frontend using a toaster notification component. The confirmation message will display the text: “Skill 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 displays a toaster message: “Invalid details”. Go back to step 4.		
	ALT-Step 8: The super admin does not confirm the changes by selecting the “Cancel” button. Go back to step 4.		
	ALT-Step 9: The system fails to save the new skill and displays a toaster message: “Failed to update skill”. Terminate use case.		

Conclusion:	The updated skill's details have been saved onto the system, and a confirmation message is shown.
Post-condition:	The system saves the updated skill's details onto the Skills table.
Business Rules:	• Only super admin can update a skill.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 25: 4.8 Update Skill Technical Use Case Narrative

INTEGRATED NGO MANAGEMENT SYSTEM			
AUTHOR:	Nicholas Bailey	**Date: 30/04/2026** **Version:** 1	
USE CASE NAME:	Deactivate Skill	**USE CASE TYPE**	
USE CASE ID:	4.9	Business Requirements:	□
PRIORITY:	Low	System Analysis:	□
SOURCE:	EPI-USE	System Design:	✓
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 deactivates an existing skill in the system. The system allows the Super Admin to select deactivate on a skill and will then confirm the deactivation. Once confirmed, the system updates the skill's status to inactive in the database, which will remove it from the active skills list. The use case ends when the skill is no longer visible within the active skill listings.		
Pre-condition:	The super admin must be logged into the system.		
Trigger:	The super admin wants to deactivate a skill on the system.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The super admin wants to deactivate a skill on the system.		**Step 2:** The system invokes UC 4.7 (View Skill) to display the selected skill details in a pop-up modal. This is done using the Angular frontend and skills service, which routes the request via Angular HTTP Client to the

			corresponding HTTP GET API endpoint in Visual Studio (SkillController).
	Step 3: The super admin selects the "Deactivate" button on a specific group. [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 skill deactivation") • Confirmation message label ○ Displays message: "Are you sure you want to deactivate this skill?" • Action buttons: ○ Confirm button ○ Cancel button
	Step 5: The super admin selects the "Confirm" button. [ALT]		**Step 6:** The system will then deactivate the selected skill by routing the action through the Angular frontend and skills service, which sends the request to the appropriate HTTP DELETE endpoint in Visual Studio (SkillController – Deactivate). The API endpoint will validate the skill and update its status in the database using Entity Framework Core in MS SQL Server. This ensures that the skill is soft deleted. The following entity is involved in the process: Skill Entity: • SkillId (PK) • SkillName • SkillDescription • IsActive

			The deactivated skill is then no longer displayed in the active skills 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 skill. 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 skill has been deactivated. AuditLog Entity: • AuditLogID (PK) • UserID (FK) • Timestamp • LogEntry
			Step 8: The system will then display a confirmation message indicating that the skill has been successfully deactivated. This is implemented in the Angular frontend using a toaster notification component. The confirmation message will display the text: “Skill deactivated successfully”.
Alternate courses:	ALT-Step 3: The super admin changes their mind and closes the pop-up by selecting the “Cancel” button. Terminate use case.		
	ALT-Step 5: The super admin does not confirm the changes by selecting the “Cancel” button. Go back to step 2.		
	ALT-Step 6: The system fails to deactivate the skill and displays a toaster message: “Failed to deactivate skill”. Terminate use case.		

Conclusion:	The active skill has been deactivated, and a confirmation message is shown.
Post-condition:	The system updates the skill's status to inactive in the Skill table.
Business Rules:	• Only super admin can deactivate a skill.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 26: 4.9 Deactivate Skill Technical Use Case Narrative

2.4.4 Project Subsystem Technical Narratives

System Name: Integrated NGO Management System		
AUTHOR:	Alexander Steven	**Date:** 28 April 2025 **Version:** 1.0
USE CASE NAME:	Create Project	**USE CASE TYPE**
USE CASE ID:	7.1	Business Requirements: □
PRIORITY:	High	System Analysis: □
SOURCE:	EPI-USE	System Design: ✓
Primary Business Actor (PBA):	System Admin	
Primary System Actor (PSA):	• None	
Other Participating Actors:	• None	
Other Interested Stakeholders:	Users in the project	
Description:	This use case describes the administrative process of initializing a new project on the system. The user chooses a project name and description, a start date, and optionally an end date. They then select a project type from a pre-defined list populated via the system API ("GET api/ProjectTypes") and select a project photo. The process then moves onto the locations step where the geographic route is sequenced. Following this, the user navigates to the teams section where they select from a list of user roles using the system API ("GET api/ProjectRoles") and subsequently add users to those roles by fetching the user directory via the API ("GET api/User"). The user then defines the project deadlines before concluding with the indemnity forms section. Here, the system retrieves available forms from the system list ("GET api/IndemnityForms") allowing the admin to attach a mandatory legal waiver. The use case ends when the frontend dispatches the full project hierarchy to the backend via the API ("POST api/Projects").	

Pre-condition:	The Admin must be authenticated. Selectable Project Types, Catalog Roles, and Users must exist in the database.		
Trigger:	The Admin selects the "+ Create Project" button on the Project Portfolio dashboard.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The System Admin clicks the "+ Create Project" button on the Portfolio dashboard.		**Step 2:** The system loads and displays the Project Identity step using the Angular frontend (“ProjectWizardComponent”) with the following controls: • A Project Identity title. • A Project Name label with a text input field and placeholder text “e.g. Rhino Relocation 2026”. • A Project Type label with a dropdown selection populated from the API (“GET api/ProjectTypes”). • A Project Profile Picture label with a selectable gallery of 8 project icons. • A Project Start Date label with a date picker. • An Include project end date checkbox which, when selected, displays an End Date date picker. •

			A Project Visibility Status section with radio buttons for Active Project and Inactive/Hidden.
	Step 3: The Admin enters the project details and clicks the Next Section button.		**Step 4**: The system loads and displays the Team Allocation step using the Angular frontend with the following controls: • A Team Allocation title. • An Add Project Role button which opens an inline selector for roles retrieved via the API (“GET api/ProjectRoles”). • A list of role blocks, each containing: o A role icon and name. o A Create Chat checkbox for enabling SignalR messaging groups. o A list of member chips with a remove “×” option. o An Add Member button which opens an inline dropdown of users retrieved via the API (“GET api/User”).
	Step 5: The Admin enters the project details and clicks the Next Section button.		**Step 6:** The system loads and displays the Project Route step using the Angular

			frontend with the following controls: • A Project Route title. • An Add Location button. • A sequenced list of location rows, each containing a stop number, an address text input field, and a Delete button.
	Step 7: The Admin adds the sequences the route for the project and clicks the Next Section button.		**Step 8:** The system loads and displays the Project Deadlines step using the Angular frontend with the following controls: • A Project Deadlines title. • An Add Deadline button. • A grid of deadline cards, each containing a name input field, a datetime-local picker, a description textarea, and a remove “x” button.
	Step 9: The Admin defines the deadlines and clicks the Next Section button.		**Step 10:** The system loads and displays the Legal & Compliance step using the Angular frontend with the following controls: • A Legal & Compliance title.

			• An Indemnity Form label with a dropdown selection of active forms retrieved via the API (“GET api/IndemnityForms”). • A Create Project button.
	Step 11: The Admin clicks the Create Project button. [ALT]		**Step 12**: The system sets the loading state to true and dispatches the project object to the backend API (“POST api/Projects”).
			Step 13: The backend (“ProjectsController”) receives the payload and iterates through the team roles to set the “EntityState” of pre-existing users and roles to “Unchanged” within the database context (“AppDbContext”).
			Step 14: The system adds the new record to the database (“Project” Entity), saves the changes, and returns an HTTP 201 Created response. [ALT]
			Step 15: The system redirects the Admin back to the Portfolio dashboard and

			displays a success notification.
Alternate courses:	[ALT] Step 11: The system detects that the Project Name or Project Type fields are empty. The Create Project button remains disabled, and validation messages are displayed under the respective fields.		
	[ALT] Step 14: The system fails to save the project due to a database connection error. The system displays an error message: “Failed to create project. Check console for details.”		
Conclusion:	A new project is successfully created and stored in the system.		
Post-condition:	The project is visible in the Portfolio and available for management in the Command Centre.		
Business Rules:	• Secure Access		
Implementation constraints and specifications:	• JWT tokens must be securely processed and handled purely in memory and local storage on the client.		
Assumptions:	• None		
Open issues:	• None		

Table 27: 7.1 Create Project Technical Use Case Narrative

System Name: Integrated NGO Management System			
AUTHOR:	Alexander Steven	**Date:** 28 April 2025 **Version:** 1.0	
USE CASE NAME:	View Project	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
USE CASE ID:	7.2		
PRIORITY:	High		
SOURCE:	EPI-USE		
Primary Business Actor (PBA):	System Users		
Primary System Actor (PSA):	• None		
Other Participating Actors:	• None		
Other Interested Stakeholders:	None		
Description:	This use case describes the process where a user selects a specific project from the portfolio list which triggers a request to the backend API ("GET api/Projects/{id}"). The system retrieves all associated data including the project type, assigned locations, chronological deadlines, and the team structure. This information is then displayed in the command center where the user can monitor the operational status and see the specific roles and members involved. The User can filter by active projects at the top of the screen.		
Pre-condition:	The project must exist in the database.		
Trigger:	The User clicks on a project card or selects "View Project" from the Portfolio or from Teams page.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The User clicks the View Project button on a project card.		**Step 2:** The system loads the View project task using the Angular frontend ("ProjectDetailComponent") and extracts the ID

			from the route parameters.
			Step 3: The system invokes the project service (“ProjectService”) to fetch project details via the API (“GET api/Projects/{id}”).
			Step 4: The backend (“ProjectsController”) performs an eager-load query on the database (“Project” Entity) using “.Include” statements to gather related locations, deadlines, and team members. [ALT]
			Step 5: The system returns the populated project object to the frontend.
			Step 6: The system renders the Command Center UI with the following panels: • An Identity Panel displaying the project photo, name, type, and start/end dates. • A Team & Personnel Panel showing each role and the assigned members. • A Route Sequence Panel displaying a

			vertical timeline of addresses. • A Critical Deadlines Panel listing milestones by date and time. • A Compliance Panel showing the status of the attached indemnity form. • A status indicator showing a pulse animation for "Active" or "Completed".
Alternate courses:	[ALT] Step 4: The system cannot find the project with the specified ID. The API returns an HTTP 404 response, and the frontend redirects the user back to the projects list.		
Conclusion:	The user successfully views the detailed project status.		
Post-condition:	The user is presented with options to edit or close.		
Business Rules:	• Secure Access		
Implementation constraints and specifications:	• JWT tokens must be securely processed and handled purely in memory and local storage on the client.		
Assumptions:	• None		
Open issues:	• None		

Table 28: 7.2 View Project Technical Use Case Narrative

System Name: Integrated NGO Management System			
AUTHOR:	Alexander Steven	**Date:** 28 April 2025 **Version:** 1.0	
USE CASE NAME:	Update Project	**USE CASE TYPE**	
USE CASE ID:	7.3	Business Requirements: □	
PRIORITY:	High	System Analysis: □	
SOURCE:	EPI-USE	System Design: ✓	
Primary Business Actor (PBA):	System Admin		
Primary System Actor (PSA):	• None		
Other Participating Actors:	• None		
Other Interested Stakeholders:	Users in the project		
Description:	This use case describes the process of modifying an existing project record. The user chooses to modify an existing project which opens the wizard interface in edit mode. The system populates the form using data retrieved from the backend. The user can update the project identity, change the team allocation by interacting with the project role API ("GET api/ProjectRoles"), or adjust the timeline and locations. Once the changes are finalized, the system updates the record by sending the data to the backend API ("PUT api/Projects/{id}").		
Pre-condition:	: The user must be an authorized System Admin and the project must already exist.		
Trigger:	The User selects the "Edit Project" button within the View project panel.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The User selects the "Edit Project" button within the View project panel.		**Step 2:** The system navigates to the project wizard in edit mode ("ProjectWizardComponent") and sets "isEditMode" to true.

			Step 3: The system fetches the existing project details via the API ("GET api/Projects/{id}") and populates all wizard controls.
	Step 4: The User modifies the necessary fields across the five wizard steps (Identity, Team, Route, Deadlines, or Legal).		
	Step 5: The User clicks the Update Project button on the final step.		**Step 6:** The system dispatches the updated project object to the backend API ("PUT api/Projects/{id}"). [ALT]
			Step 7: The backend ("ProjectsController") marks the "Project" Entity as modified in the database context ("AppDbContext") and executes "SaveChangesAsync()".
			Step 8: Upon receiving an HTTP 204 No Content response, the system redirects the User back to the Project Portfolio**.**
Alternate courses:	[ALT] Step 6: The system fails to validate the updated dates (e.g., end date is before start date). An error message is displayed, and the update is blocked.		
Conclusion:	The project information is successfully updated in the database.		

Post-condition:	The updated details are immediately reflected in the View Project Info page and Projects List page.
Business Rules:	• Secure Access
Implementation constraints and specifications:	• JWT tokens must be securely processed and handled purely in memory and local storage on the client.
Assumptions:	• None
Open issues:	• None

Table 29: 7.3 Update Project Technical Use Case Narrative

System Name: Integrated NGO Management System			
AUTHOR:	Alexander Steven	**Date:** 28 April 2025 **Version:** 1.0	
USE CASE NAME:	Deactivate Project	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
USE CASE ID:	7.4		
PRIORITY:	High		
SOURCE:	EPI-USE		
Primary Business Actor (PBA):	System Admin		
Primary System Actor (PSA):	• None		
Other Participating Actors:	• None		
Other Interested Stakeholders:	Users in the project		
Description:	This use case describes the process of suspending a project's operational status. The user selects the toggle option within the Project Info page which triggers a request to the backend API ("PATCH api/Projects/{id}/toggle-active"). The system modifies the "IsActive" boolean on the "Project" Entity in the database and saves the change. Once the update is confirmed, the frontend updates the status indicator and the pulse animation to reflect that the project is now "Completed".		
Pre-condition:	The user must be registered on the system and their account must be active. The User must be a System Admin The project must exist and currently be active.		
Trigger:			
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The User clicks the Deactivate Project button in the Project Info page.		**Step 2:** The System asks the Admin for confirmation they want to deactivate the project.

	Step 3: The Admin confirms or clicks back		**Step 4:** The system invokes the toggle method in the project detail component ("ProjectDetailComponent") which calls the project service.
			Step 5: The system dispatches a request to the backend API ("PATCH api/Projects/{id}/toggle-active").
			Step 6: The backend ("ProjectsController") retrieves the project from the database ("Project" Entity) and inverts the "IsActive" status.
			Step 7: The system executes "SaveChangesAsync()" on the database context ("AppDbContext") and returns the new status in an HTTP 200 OK response.
			Step 8: The system updates the frontend UI to change the button text to "Activate Project".
			Step 9: The system updates the status pulse indicator from a green "Online" state to a grey "Off-grid" state.
Alternate courses:	[ALT] Step 3: The Admin clicks back and returns to the Project View Page		

	[ALT] Step 6: The system cannot find the project record to update. The API returns an HTTP 404 response and the status remains unchanged.
Conclusion:	The project is successfully moved to an inactive state.
Post-condition:	The project remains in the database but is marked as inactive in the portfolio.
Business Rules:	• Secure Access
Implementation constraints and specifications:	• JWT tokens must be securely processed and handled purely in memory and local storage on the client.
Assumptions:	• None
Open issues:	• None

Table 30: 7.4 Deactivate Project Technical Use Case Narrative

System Name: Integrated NGO Management System			
AUTHOR:	Alexander Steven	**Date:** 28 April 2025 **Version:** 1.0	
USE CASE NAME:	Create Project Type	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓	
USE CASE ID:	7.5		
PRIORITY:	High		
SOURCE:	EPI-USE		
Primary Business Actor (PBA):	System Admin		
Primary System Actor (PSA):	• None		
Other Participating Actors:	• None		
Other Interested Stakeholders:	None		
Description:	This use case describes the creation of a new project category used to classify different NGO initiatives. The user enters a unique name and a detailed description for the project type. Upon saving, the frontend dispatches the data to the system API ("POST api/ProjectTypes"). The backend receives the request, adds the new category to the "ProjectType" Entity, and commits the record to the SQL database. This new type then becomes available for selection when administrators are using the project creation wizard.		
Pre-condition:	The Admin must be registered on the system and their account must be active. The Admin must be authenticated.		
Trigger:	The Admin selects the "Create Project Type" option in the Admin Dashboard		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The Admin clicks the Create Project Type button on the View Project Types page		**Step 2:** The system loads and displays the Create Project Type modal using the Angular frontend with the following controls:

			• A Create Project Type title. • A Name label with a text input field. • A Description label with a textarea field. • A Save button.
	Step 3: The User enters the name (e.g., “Wildlife Relocation”) and the description and clicks save.		**Step 4**: The system dispatches the project type object to the backend API (“POST api/ProjectTypes”).
			Step 5: The backend (“ProjectTypesController ”) adds the new record to the database (“ProjectType” Entity) via the database context (“AppDbContext”). [ALT]
			Step 6: The system executes “SaveChangesAsync()” and returns the created object with an HTTP 200 OK response.
			Step 7: The system closes the modal and refreshes the project type list to include the new entry.
Alternate courses:	[ALT] Step 5: The system detects that a project type with the same name already exists in the database. The API returns an error and the user is prompted to choose a unique name.		

Conclusion:	A new project type is added to the system catalog.
Post-condition:	The project type can now be assigned to new projects.
Business Rules:	• Secure Access
Implementation constraints and specifications:	• JWT tokens must be securely processed and handled purely in memory and local storage on the client.
Assumptions:	• None
Open issues:	• None

Table 31: 7.5 Create Project Type Technical Use Case Narrative

System Name: Integrated NGO Management System			
AUTHOR:	Alexander Steven		**Date:** 28 April 2025 **Version:** 1.0
USE CASE NAME:	View Project Type		**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓
USE CASE ID:	7.6		
PRIORITY:	High		
SOURCE:	EPI-USE		
Primary Business Actor (PBA):	System Admin		
Primary System Actor (PSA):	• None		
Other Participating Actors:	• None		
Other Interested Stakeholders:	None		
Description:	This use case describes how the administrator views the list of project categories currently configured in the system. The user navigates to the project type management page, which triggers a request to the backend API (“GET api/ProjectTypes”). The system retrieves all records from the “ProjectType” Entity and presents them in a grid view, showing the name and description of each type.		
Pre-condition:	Project types must exist in the database.		
Trigger:	The Admin navigates to the Project Types management section.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The Admin selects Project Types from the management menu or Admin dashboard.		**Step 2:** The system loads the project type list component and extracts the data from the API (“GET api/ProjectTypes”).
			Step 3: The backend (“ProjectTypesController ”) executes a SQL Read

			Query to fetch all records from the database (“ProjectType” Entity). [ALT]
			Step 4: The system returns a collection of project types to the frontend.
			Step 5: The system renders the Project Type Management screen with the following controls: • A Project Types title. • A Search bar to filter types by name. • A list of project type cards or rows, each displaying the Name and Description. • Action buttons for Edit and Delete on each row. • An Add New Type button.
Alternate courses:	[ALT] Step 3: No project types are found in the database. The system displays an empty state message: “No project types found. Create a new mission category to get started.”		
Conclusion:	The user successfully views the configured project categories.		
Post-condition:	The user can proceed to update or delete specific project types.		
Business Rules:	• Secure Access		

Implementation constraints and specifications:	• JWT tokens must be securely processed and handled purely in memory and local storage on the client.
Assumptions:	• None
Open issues:	• None

Table 32: 7.6 View Project Type Technical Use Case Narrative

System Name: Integrated NGO Management System		
AUTHOR:	Alexander Steven	**Date:** 28 April 2025 **Version:** 1.0
USE CASE NAME:	Update Project Type	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓
USE CASE ID:	7.7	
PRIORITY:	High	
SOURCE:	EPI-USE	
Primary Business Actor (PBA):	System Admin	
Primary System Actor (PSA):	• None	
Other Participating Actors:	• None	
Other Interested Stakeholders:	None	
Description:	This use case describes the process of modifying the details of an existing project category. The user chooses to modify a project type from the management list which triggers a call to the system API (“GET api/ProjectTypes/{id}”). The user updates the name and description through the Angular frontend modal. Once the user saves the changes, the system dispatches the updated object to the backend using the .NET 8.0 framework via the API (“PUT api/ProjectTypes/{id}”). The backend updates the	

<table>
<tr><td></td><td colspan="3">“EntityState” of the “ProjectType” Entity to modified within the database context (“AppDbContext”) and persists the changes to the SQL database.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">The project type must exist in the database.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The Admin selects the “Edit” option next to a project type in the management list.</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: Step 1: The Admin clicks the Edit button on a specific project type row.</td><td></td><td>Step 2: The system loads the Update Project Type modal using the Angular frontend and populates the controls with existing data retrieved from the API (“GET api/ProjectTypes/{id}”).</td></tr>
<tr><td></td><td></td><td>Step 3: The system displays the following controls within the modal:
• An Update Project Type title.
• A Name label with a text input field containing the current name.
• A Description label with a textarea field containing the current description.
• A Save Changes button.
• A Cancel button.</td></tr>
<tr><td>Step 4: The User modifies the name or description</td><td></td><td>Step 5: The system dispatches the updated</td></tr>
</table>

	fields and clicks the Save Changes button. [ALT]		data to the backend API ("PUT api/ProjectTypes/{id}") built on the .NET 8.0 framework.
			Step 6: The backend ("ProjectTypesController") receives the payload and marks the "ProjectType" Entity as modified in the database context ("AppDbContext").
			Step 7: The system executes "SaveChangesAsync()" and returns an HTTP 204 No Content response upon success. [ALT]
			Step 8: The system closes the modal and refreshes the management list to display the updated information.
Alternate courses:	[ALT] Step 4: The user clicks the Cancel button. The system closes the modal without saving any changes and the record remains in its original state.		
	[ALT] Step 7: The system fails to save the changes due to a database concurrency conflict. An error message is displayed: "The record you attempted to edit was modified by another user."		
Conclusion:	The project type details are successfully updated.		
Post-condition:	The updated project type name and description are visible in the management list and the project wizard.		

Business Rules:	• Secure Access
Implementation constraints and specifications:	• JWT tokens must be securely processed and handled purely in memory and local storage on the client.
Assumptions:	• None
Open issues:	• None

Table 33: 7.7 Update Project Type Technical Use Case Narrative

System Name: Integrated NGO Management System		
AUTHOR:	Alexander Steven	**Date:** 28 April 2025 **Version:** 1.0
USE CASE NAME:	Deactivate Project Type	**USE CASE TYPE** Business Requirements: □ System Analysis: □ System Design: ✓
USE CASE ID:	7.8	
PRIORITY:	High	
SOURCE:	EPI-USE	
Primary Business Actor (PBA):	System Admin	
Primary System Actor (PSA):	• None	
Other Participating Actors:	• None	
Other Interested Stakeholders:	Users in the project	
Description:	This use case describes the administrative removal of a project category from the system. The user chooses to remove a project type from the list which triggers a request to the system API (“DELETE api/ProjectTypes/{id}”) using the .NET 8.0 framework. The backend identifies the specific record in the “ProjectType” Entity and removes it from the database context (“AppDbContext”). Once the change is committed to the SQL database, the project type is no longer available for selection during project creation.	
Pre-condition:	The project type must exist and should not be currently assigned to any active projects to avoid referential integrity errors.	

Trigger:	The Admin selects the "Delete" or "Deactivate" option for a project type in the management list.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The Admin clicks the Delete button next to a project type entry.		**Step 2:** The system displays a confirmation modal using the Angular frontend with the following controls: • A Confirm Deletion title. • A message asking if the user is sure they want to remove the project type. • A Delete button. • A Cancel button.
	Step 3: The Admin clicks the Delete button. [ALT]		**Step 4**: The system dispatches a request to the backend API ("DELETE api/ProjectTypes/{id}") developed with the .NET 8.0 framework.
			Step 5: The backend ("ProjectTypesController") locates the entry in the "ProjectType" Entity via the database context ("AppDbContext").
			Step 6: The system removes the record from the context and executes "SaveChangesAsync()" to update the SQL database.

			[ALT]
			Step 7: Upon a successful deletion, the API returns an HTTP 204 No Content response.
			Step 8: The system closes the modal and removes the entry from the management list.
Alternate courses:	[ALT] Step 3: The user clicks the Cancel button. The confirmation modal closes and no data is sent to the API.		
	[ALT] Step 6: The system prevents deletion because the project type is linked to existing project records. An error message is displayed: "Cannot delete project type as it is currently in use by one or more projects."		
Conclusion:	The project type is successfully removed from the system.		
Post-condition:	The project type no longer appears in the management list or the project creation wizard or on the project type list.		
Business Rules:	• Secure Access		
Implementation constraints and specifications:	• JWT tokens must be securely processed and handled purely in memory and local storage on the client.		
Assumptions:	• None		
Open issues:	• None		

Table 34: 7.8 Deactivate Project Type Technical Use Case Narrative

<table>
<tr><th colspan="4">System Name: Integrated NGO Management System</th></tr>
<tr><td>Author:</td><td>Amahle Khoza</td><td colspan="2">Date: 22 April 2026
Version: 1.0</td></tr>
<tr><td>Use case name:</td><td>Add Indemnity 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>7.9</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">• 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">Administrator</td></tr>
<tr><td>Description:</td><td colspan="3">This use case allows the Super Admin to add a new indemnity form to the system by uploading a PDF document and capturing its related details. The Angular frontend presents a reactive form allowing the capture of metadata and a PDF file upload. Upon submission, the system transmits a multipart/form-data request to the ASP.NET Core API endpoint. The backend processes the file using the IFormFile interface, validates the file type, and persists the file path alongside metadata in the IndemnityForm entity via Entity Framework Core. The system ensures uniqueness constraints and enforces business rules before committing the transaction to the SQL Server database. The form becomes part of the project documentation structure and can be linked to a project during the capturing indemnity process.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">• The Super Admin must be logged into the system.
• The Super Admin must have permission to manage indemnity forms.
• The indemnity form must be available as a valid PDF document.
• The form name must not already exist in the system.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The Super Admin selects the “Add Indemnity Form” option.</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 navigates to the indemnity management interface</td><td></td><td>Step 2: The Angular Router loads the IndemnityFormComponent.</td></tr>
</table>

System Name: Integrated NGO Management System			
	via the Angular frontend.		The system renders a reactive form containing: • Form Name input field • Description input field • File upload control (PDF only) • Submit button
	Step 3: The Super Admin enters the form name, description, and uploads a PDF file, then selects submit. [ALT]		**Step 4:** The Angular service constructs a multipart/form-data payload and sends a POST request to: POST /api/IndemnityForms
			Step 5: The API controller receives the request. The system performs validation: • Confirms file is not null • Validates file extension (pdf) Validates form name uniqueness via EF Core query. [ALT]
			Step 6: The system processes the uploaded file: • Generates a unique filename using GUID • Stores the file in the server directory (e.g. /indemnity-forms/) Construct a file path string. [ALT]
			Step 7: The system creates a new IndemnityForm entity:

System Name: Integrated NGO Management System			
			• IndemnityFormId (PK) • FormName • Description • FilePath • CreatedAt • IsActive
			Step 8: The system persists the entity.
			Step 9: The API returns HTTP 200 OK with the created object.
			Step 10: The frontend displays a success toast notification.
Alternate courses:	**[ALT] Step 3:** The Super Admin uploads a non-PDF file. The system rejects the file and returns HTTP 400 Bad Request with message: "Only PDF files are allowed."		
	[ALT] Step 5: A duplicate form name is detected. The system aborts the process and returns HTTP 400 with message: "An indemnity form with this name already exists."		
	[ALT] Step 6: File storage fails. The system returns HTTP 500 Internal Server Error, and no database record is created.		
Conclusion:	The indemnity form is successfully uploaded and made available for project use.		
Post-condition:	• A new indemnity form record exists in the system. • The uploaded PDF is stored and linked to the indemnity form record. • The form status is active.		
Business rules:	• Only a Super Admin may add indemnity forms. • Only PDF documents may be uploaded. • Each indemnity form name must be unique. • Newly added forms must be active by default.		
Implementation constraints and specifications:	• The system must support secure PDF upload. • The file path must be stored in the database.		
Assumptions:	• The serve has available storage space. • The project documentation structure has already been defined.		
Open issues:	• Should uploaded forms require approval before becoming active?		

Table 27:7.9 Add Indemnity Form Technical Use Case Narrative

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 22 April 2026 **Version: 1.0**	
Use case name:	Update Indemnity Form	**USE CASE TYPE** Business Requirements: □ System Analysis: □ ✓ System Design: ✓	
Use case id:	7.10		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Administrator		
Description:	This use case allows the Super Admin to update an existing indemnity form's detail or replace an PDF document. This is necessary when the wording, legal requirements, project conditions, or document details change. The update ensures that the system continues to use the most accurate and relevant version of the indemnity form.		
Pre-condition:	• The Super Admin must be logged into the system. • The indemnity form must already exist, • The Super Admin must have permission to update indemnity forms. • If replacing then document, the new file must be a PDF.		
Trigger:	The Super Admin selects an existing indemnity form and chooses "Update Indemnity Form."		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: The Super Admin navigates to the indemnity form list and selects a form to update.		**Step 2:** The Angular Router loads the UpdateIndemnityFormComponent. The system retrieves the selected form via: GET /api/IndemnityForms/{id}
			Step 3: The system populates a reactive form with:

System Name: Integrated NGO Management System			
			• Form Name • Description • Existing file reference Optional file upload control
	Step 4: The Super Admin modifies the form details and optionally uploads a new PDF file. [ALT]		**Step 5:** The Angular service constructs a multipart/form-data payload and sends: PUT /api/IndemnityForms/{id}
			Step 6: The API controller retrieves the existing record using EF Core. [ALT]
			Step 7: The system validates: • Form name uniqueness • File format (if provided) • Required fields [ALT]
			Step 8: If a new file is uploaded: • Generate GUID filename • Replace stored file path • Save file to server directory [ALT]
			Step 9: The system updates the entity fields and executes.
			Step 10: The API returns HTTP 200 OK.
			Step 11: The frontend displays a success notification.
Alternate courses:	**[ALT] Step 4:** Invalid file format uploaded. System returns HTTP 400 Bad Request (“Only PDF files allowed”)		
	[ALT] Step 6: Indemnity form not found. System returns HTTP 404 Not Found		
	[ALT] Step 7: Duplicate form name. System returns HTTP 400 Bad Request		
	[ALT] Step 8: File upload failure. System returns HTTP 500 Internal Server Error		
Conclusion:	The indemnity form is successfully uploaded and made available for project use.		
Post-condition:	• A new indemnity form record exists in the system. • The uploaded PDF is stored and linked to the indemnity form record. • The form status is active.		

System Name: Integrated NGO Management System	
Business rules:	• Only a Super Admin may update indemnity forms. • Form names must remain unique. • Only PDF replacement files are allowed. • A form linked to active projects should not be replaced without confirmation.
Implementation constraints and specifications:	• The system must validate all updated fields before saving. • Database changes and file replacement should be handled carefully to avoid inconsistency.
Assumptions:	• The replacement PDF has been reviewed and approved. • The system has access to the existing stored file.
Open issues:	• Should users be notified when an indemnity form changes?

Table 28: 7.10 Update Indemnity Form Technical Use Case Narrative

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 22 April 2026 **Version: 1.0**	
Use case name:	View Indemnity Form	**USE CASE TYPE**	
Use case id:	7.11	Business Requirements: □	
Priority:	High	System Analysis: □ ✓	
Source:	EPI-USE	System Design: ✓	
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Administrator		
Description:	This use case allows authorised users to view indemnity forms stored in the system. The Super Admin may view all forms for management purposes, while other authorised users may view active forms relevant to their projects.		
Pre-condition:	• The user must be logged into the system. • At least one indemnity form must exist. • The user must have permission to view indemnity forms. • The uploaded PDF file must still be available in storage.		
Trigger:	The user selects “View Indemnity Form” or opens the indemnity form list.		
Typical Course of Events:	**Actor Action**	**System Response**	
			Step 2: Angular component loads and triggers: GET /api/IndemnityForms [ALT]
	Step 1: The user navigates to the indemnity form section.		**Step 3:** API executes EF Core query.
			Step 4: The system returns a list of forms.
			Step 7: The system retrieves file path and opens PDF. [ALT]

System Name: Integrated NGO Management System			
	Step 6: The user selects a form to view.		**Step 2:** Angular component loads and triggers: GET /api/IndemnityForms [ALT]
Alternate courses:	**[ALT] Step 2:** No forms exist. System returns empty list		
	[ALT] Step 7: File missing. System returns error message		
Conclusion:	The user successfully views the indemnity form details and, where available, the uploaded PDF.		
Post-condition:	• No data is changed. • The selected indemnity form remains available in the system. • The user has accessed the relevant document.		
Business rules:	• Users may only view forms they are authorised to access. • Inactive forms should not be visible to general users. • Super Admins may view both active and inactive forms for management purposes. • A form must have a valid file path before it can be opened.		
Implementation constraints and specifications:	• The system must prevent unauthorised direct access to restricted documents. • The frontend must support viewing or downloading PDF files.		
Assumptions:	• The document was uploaded successfully. • The user has a stable connection to load the file. • Permissions are correctly configured.		
Open issues:	• Should the system record when a user views a form? • Should inactive forms be hidden completely or only marked as inactive?		

Table 29: 7.11 View Indemnity Form Technical Use Case Narrative

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 22 April 2026 **Version: 1.0**	
Use case name:	Remove Indemnity Form	**USE CASE TYPE** Business Requirements: □ System Analysis: □ ✓ System Design: ✓	
Use case id:	7.12		
Priority:	High		
Source:	EPI-USE		
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Administrator		
Description:	This use case allows the Super Admin to remove an indemnity form from active use. Instead of permanently deleting the document, the system should preferably deactivate it so that historical project records remain intact. This protects data integrity, especially where the indemnity form may already be linked to existing projects.		
Pre-condition:	• The Super Admin must be logged into the system. • The indemnity form must exist. • The Super Admin must have permission to remove indemnity forms. • The system must check whether the form is linked to any projects.		
Trigger:	The Super Admin selects “Remove Indemnity Form.”		
Typical Course of Events:	**Actor Action**	**System Response**	
			Step 2: System retrieves form via EF Core. [ALT]
	Step 1: The Super Admin selects a form.		**Step 3:** System checks for project links
			Step 5: System updates: form.IsActive = false;
	Step 4: The Super Admin confirms removal. [ALT]		**Step 6:** System saves changes.

System Name: Integrated NGO Management System			
			Step 7: System returns HTTP 200 OK.
			Step 2: System retrieves form via EF Core. [ALT]
Alternate courses:	**[ALT] Step 2:** Form not found. HTTP 404		
	[ALT] Step 4: User cancels. Process terminates		
Conclusion:	The indemnity form is removed from active use without compromising historical records.		
Post-condition:	• The indemnity form status is inactive. • The form can no longer be attached to new projects.		
Business rules:	• Only a Super Admin may remove indemnity forms. • Forms linked to projects should not be permanently deleted. • Inactive forms must not be available for new captures.		
Implementation constraints and specifications:	• The user interface should clearly distinguish inactive forms. • Soft delete should be implemented using an active or inactive status field. • The system must check dependencies before removal.		
Assumptions:	• Inactive records are excluded from normal user selection lists.		
Open issues:	• Should inactive forms be restorable?		

Table 30: 7.12 Remove Indemnity Form Technical Use Case Narrative

System Name: Integrated NGO Management System			
Author:	Amahle Khoza	**Date:** 22 April 2026 **Version: 1.0**	
Use case name:	Capture Indemnity Form	**USE CASE TYPE**	
Use case id:	7.13	Business Requirements: □	
Priority:	High	System Analysis: □ ✓	
Source:	EPI-USE	System Design: ✓	
Primary business actor (PBA):	• Super Admin		
Primary system actor (PSA):	• None		
Other participating actors:	• None		
Other interested stakeholders:	Administrator		
Description:	This use case allows the Super Admin to capture an indemnity form against a specific project by linking an existing active indemnity form to that project. This ensures that each project can be associated with the correct legal or consent documentation required for participation, compliance, or project activities. Unlike adding a form, which uploads the document, capturing a form means assigning or attaching an existing form to a project.		
Pre-condition:	• The Super Admin must be logged into the system. • The Super Admin must have permission to manage project documentation. • The project must exist and be active. • At least one active indemnity form must exist. • The selected indemnity form must not already be linked to the selected project.		
Trigger:	The Super Admin selects “Capture Indemnity Form” from a project.		
Typical Course of Events:	**Actor Action**	**System Response**	
		Manual Action	**Automated Action**
	Step 1: Super Admin selects a project.		**Step 2:** System retrieves project details. [ALT]
			Step 3: System retrieves active forms. [ALT]

System Name: Integrated NGO Management System			
			Step 4: System displays selectable forms.
			Step 5: Super Admin selects form and confirms.
			Step 6: System validates: • Project exists • Form exists • No duplicate link [ALT]
			Step 7: System creates: ProjectIndemnityForm: • ProjectId • IndemnityFormId • CapturedAt
			Step 8: System saves via EF Core.
			Step 9: Returns HTTP 200 OK.
Alternate courses:	**[ALT] Step 2:** Project not found. HTTP 404		
	[ALT] Step 3: No forms available. Prompt to add form		
	[ALT] Step 6: Duplicate capture. HTTP 400		
Conclusion:	The selected indemnity form is successfully captured against the project and becomes part of that project's documentation.		
Post-condition:	• A ProjectIndemnityForm record exists. • The project is linked to the selected indemnity form. • The captured form can be viewed as part of the project documentation.		
Business rules:	• Only a Super Admin may capture indemnity forms for projects. • Only active indemnity forms may be captured. • Only active projects may be linked to indemnity forms. • A project cannot have the same indemnity form captured more than once. • Capture does not create a new form; it links an existing form to a project.		
Implementation constraints and specifications:	• A linking table should be used to connect Project and IndemnityForm.		

System Name: Integrated NGO Management System	
	• A unique constraint should prevent duplicate project-form combinations. • The system must validate both ProjectId and IndemnityFormId before saving.
Assumptions:	• Projects are already created before forms are captured. • The correct indemnity form has already been uploaded.
Open issues:	• Should a project be allowed to have multiple indemnity forms? • Should users linked to the project be notified when a form is captured?

Table 31: 7.13 Capture Indemnity Form Technical Use Case Narrative

2.4.5 Message Subsystem Technical Narratives

<table>
<tr><th colspan="4">INTEGRATED NGO MANAGEMENT SYSTEM</th></tr>
<tr><td>AUTHOR:</td><td colspan="2">Christopher Bailey</td><td>Date: 1 May 2026
Version: 1</td></tr>
<tr><td>USE CASE NAME:</td><td colspan="2">Create Message</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>USE CASE ID:</td><td colspan="2">11.1</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">User</td></tr>
<tr><td>Primary System Actor (PSA):</td><td colspan="3">• None</td></tr>
<tr><td>Other Participating Actors:</td><td colspan="3">• Group Members (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 the event in which a user creates and sends a message on a group. The user will navigate to the messages tab after which the system retrieves all the groups the user is currently in. The user will select a group and the system will retrieve all the messages for that group from the Firebase Firestore database. The user will write their message and send it to the group. The use case ends once the new message is sent and stored in the Firebase Firestore database.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">User is logged into the system and on their account.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The user requests to send a message on a group.</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>
</table>

	Step 1: The user requests to send a message on a group.		
	Step 2: The user navigates to and selects the Messages tab.		**Step 3:** The system invokes use case 11.2 (View Messages)
	Step 4: The user writes their message in the message input field and clicks the Send button. [ALT]		**Step 5:** The Angular frontend creates a messages object and sends the message to the Firebase Firestore database using the Chat service. The message is saved in the appropriate groups message collection with the following details: • senderId • senderName • text • timestamp
			Step 6: All clients subscribed to the group's messages collection receive the new message instantly via the collectionData() observable, and the chat UI updates automatically.
Alternate courses:	[ALT] Step 4: The user decides not to send a message and exits the Messages tab. Use case terminates.		
Conclusion:	The user's message is saved in the Firebase Firestore database.		
Post-condition:	The user's message is successfully saved and appears for other users to view.		

Business Rules:	• Users can only send messages on groups they are a part of.
Implementation constraints and specifications:	• None
Assumptions:	• None
Open issues:	• None

Table 32: 11.1 Create Message Technical Use Case Narrative

<table>
<tr><th colspan="4">INTEGRATED NGO MANAGEMENT SYSTEM</th></tr>
<tr><td>AUTHOR:</td><td colspan="2">Christopher Bailey</td><td>Date: 1 May 2026
Version: 1</td></tr>
<tr><td>USE CASE NAME:</td><td colspan="2">View Message</td><td rowspan="4">USE CASE TYPE
Business Requirements: □
System Analysis: □
System Design: ✓</td></tr>
<tr><td>USE CASE ID:</td><td colspan="2">11.2</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">User</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">Other users on the group.</td></tr>
<tr><td>Description:</td><td colspan="3">This use case describes the event in which a user views messages on a group. The user will navigate to the messages tab after which the system retrieves all the groups the user is currently in. The user will select a group and the system will retrieve all the messages for that group from the Firebase Firestore database. The use case ends once all the messages for the selected group are displayed for the user.</td></tr>
<tr><td>Pre-condition:</td><td colspan="3">User is logged into the system and on their account.</td></tr>
<tr><td>Trigger:</td><td colspan="3">The user requests to view a message on a group.</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 view a message on a group.</td><td></td><td></td></tr>
</table>

	Step 2: The user navigates to and selects the Messages tab.		**Step 3:** The Angular frontend calls the UserMessageGroup service to send a GET request to the C# Web API backend Group controller through the GetUserGroups endpoint. The backend retrieves the user's groups from the following entities: Project Entity: • ProjectId (PK) • ProjectName Group Entity: • GroupId (PK) • ProjectId (FK) • GroupName • ActiveStatus UserGroup Entity: • UserId (PK) • GroupId (FK) [ALT]
			Step 4: The system loads and displays the Messages screen using the Angular frontend with the following controls: A screen divided into two main sections, a sidebar and a chat area. The sidebar section:

			• Displaying a list of projects. • Each project is shown with a project title. • Under each project, a list of groups is displayed. • Each group item includes a chat icon and the group name. • Clicking on a group opens the corresponding chat. • The selected group is visually highlighted as active. A chat section that changes based on selection: • When no group is selected, an empty state is displayed. • A message indicating No conversation selected. • A prompt instructing the user to select a group from the sidebar to start chatting. • When a group is selected, a chat interface is displayed.
	Step 4: The user clicks the group they wish to message. [ALT]		**Step 5:** When a group is selected the Angular frontend calls the Chat service to retrieve

			messages from the Firebase Firestore database.
			Step 6: The Angular frontend displays the messages for the selected group in the chat area: • A header showing the group name. • A messages area displaying a list of chat messages. • Messages are styled differently depending on whether they are sent by the current user or others. Each message includes: • The sender's name (shown only for messages from other users). • A message bubble containing the message text. An input section at the bottom of the chat. • A text input field with a placeholder prompting the user to type a message.

			• The ability to send a message by pressing Enter or clicking the Send button.
Alternate courses:	[ALT] Step 3: The system fails to retrieve the user's groups. The system will displays an error message based on the status code.		
	[ALT] Step 4: The user does not select a group and exits the messages tab. Use case terminates.		
Conclusion:	The groups messages are displayed.		
Post-condition:	The user can view messages from the selected group.		
Business Rules:	• Users can only view messages in groups they are a part of.		
Implementation constraints and specifications:	• None		
Assumptions:	• None		
Open issues:	• None		

Table 33: 11.2 View Message Technical Use Case Narrative

2.5 Technical Context Diagram

2.5.1 User Subsystem Context Diagram

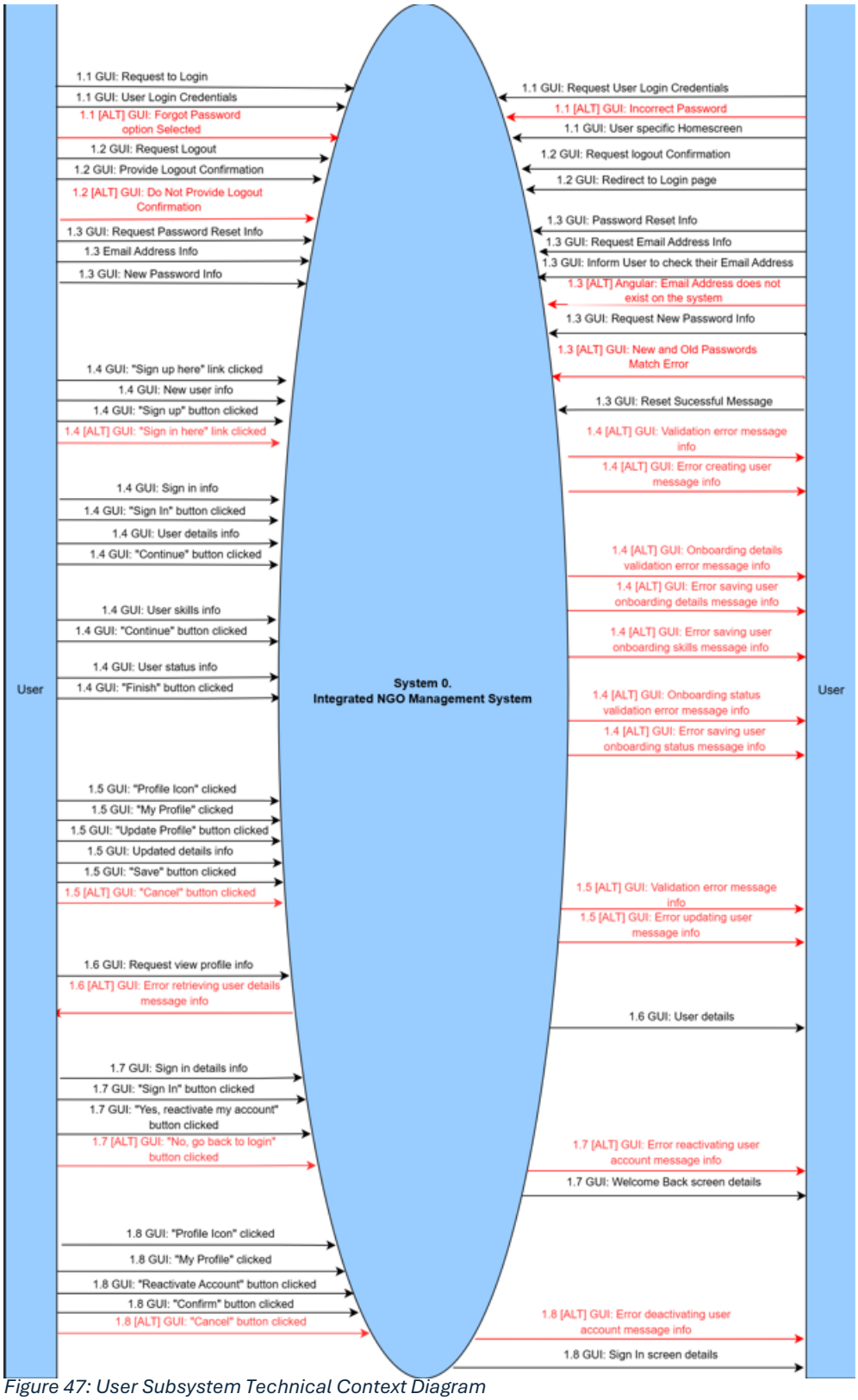


Figure 47: User Subsystem Technical Context Diagram

2.5.2 Admin Subsystem Context Diagram

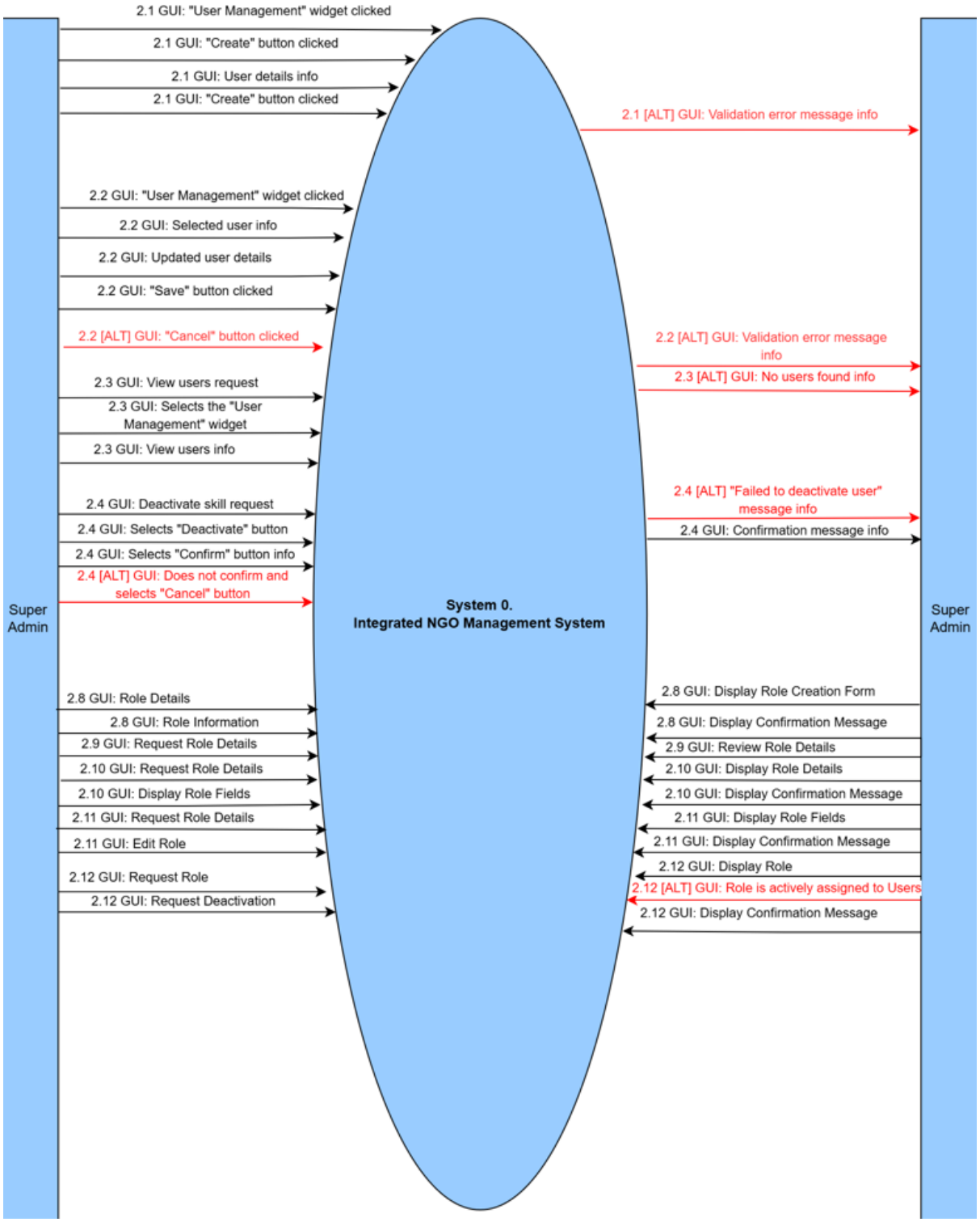


Figure 48: Admin Subsystem Technical Context Diagram

2.5.3 Group Subsystem Context Diagram

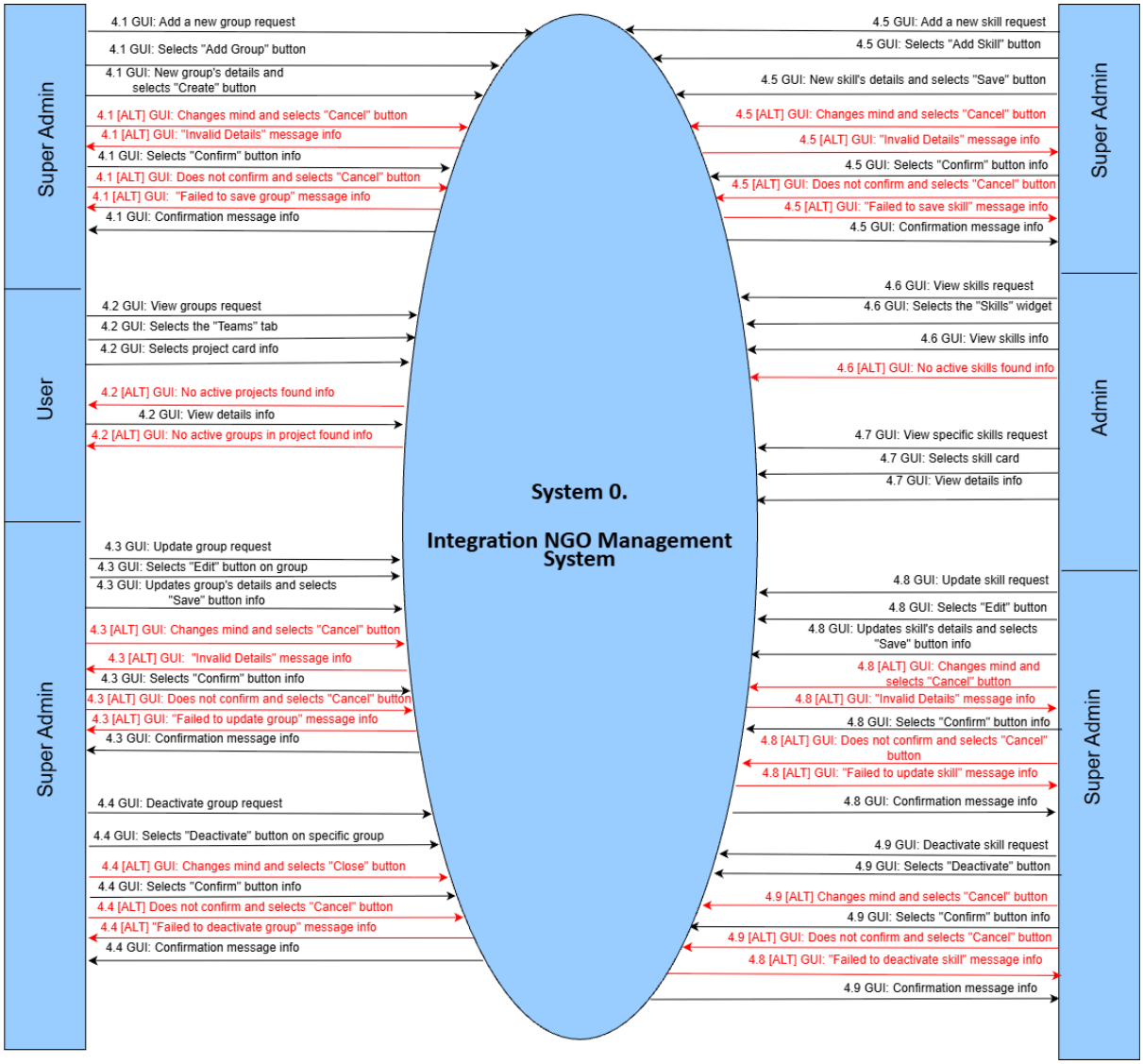


Figure 49: Group Subsystem Technical Context Diagram

2.5.4 Project Subsystem Context Diagram

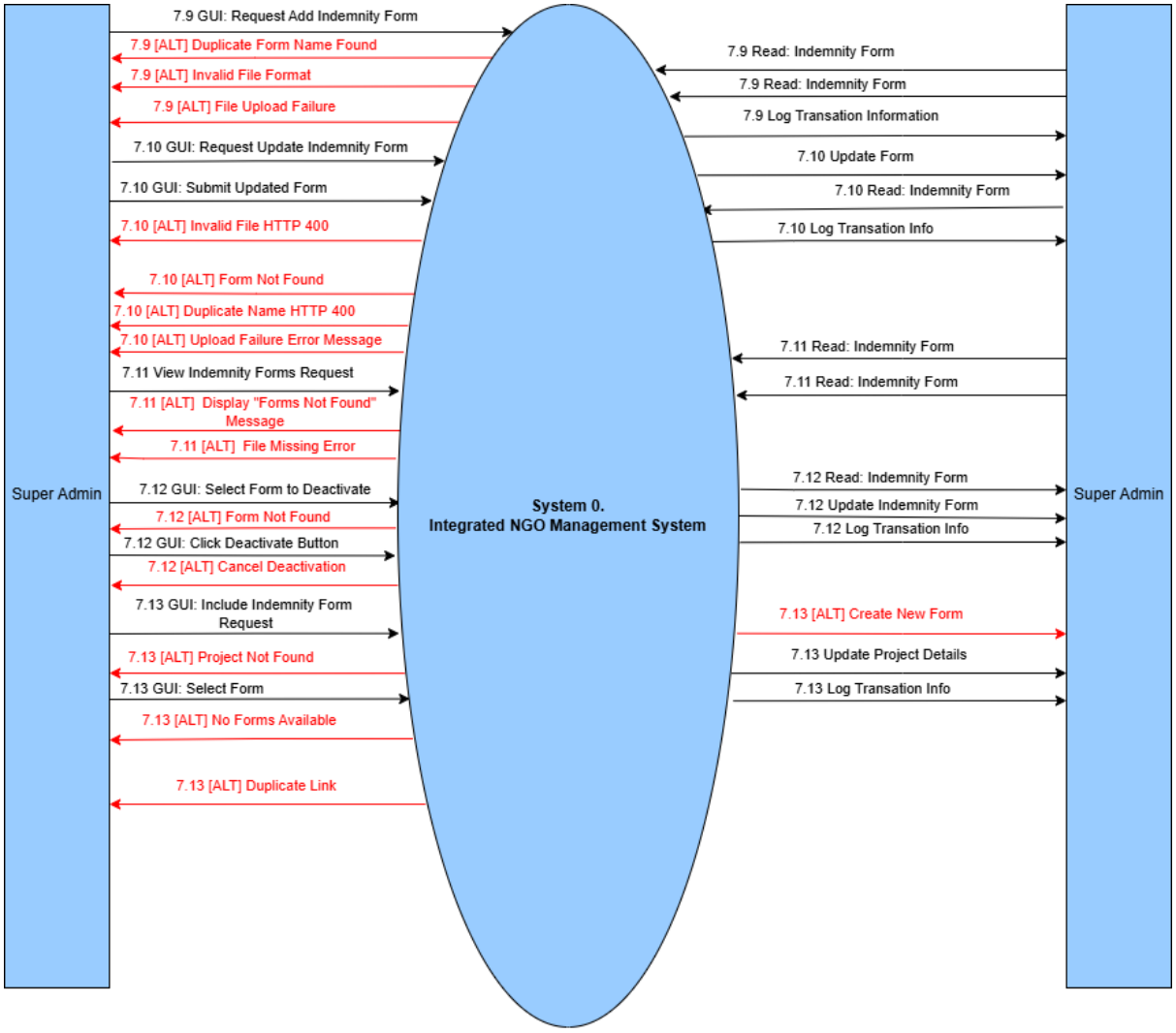


Figure 50: Project Subsystem Technical Context Diagram

2.5.5 Message Subsystem Context Diagram

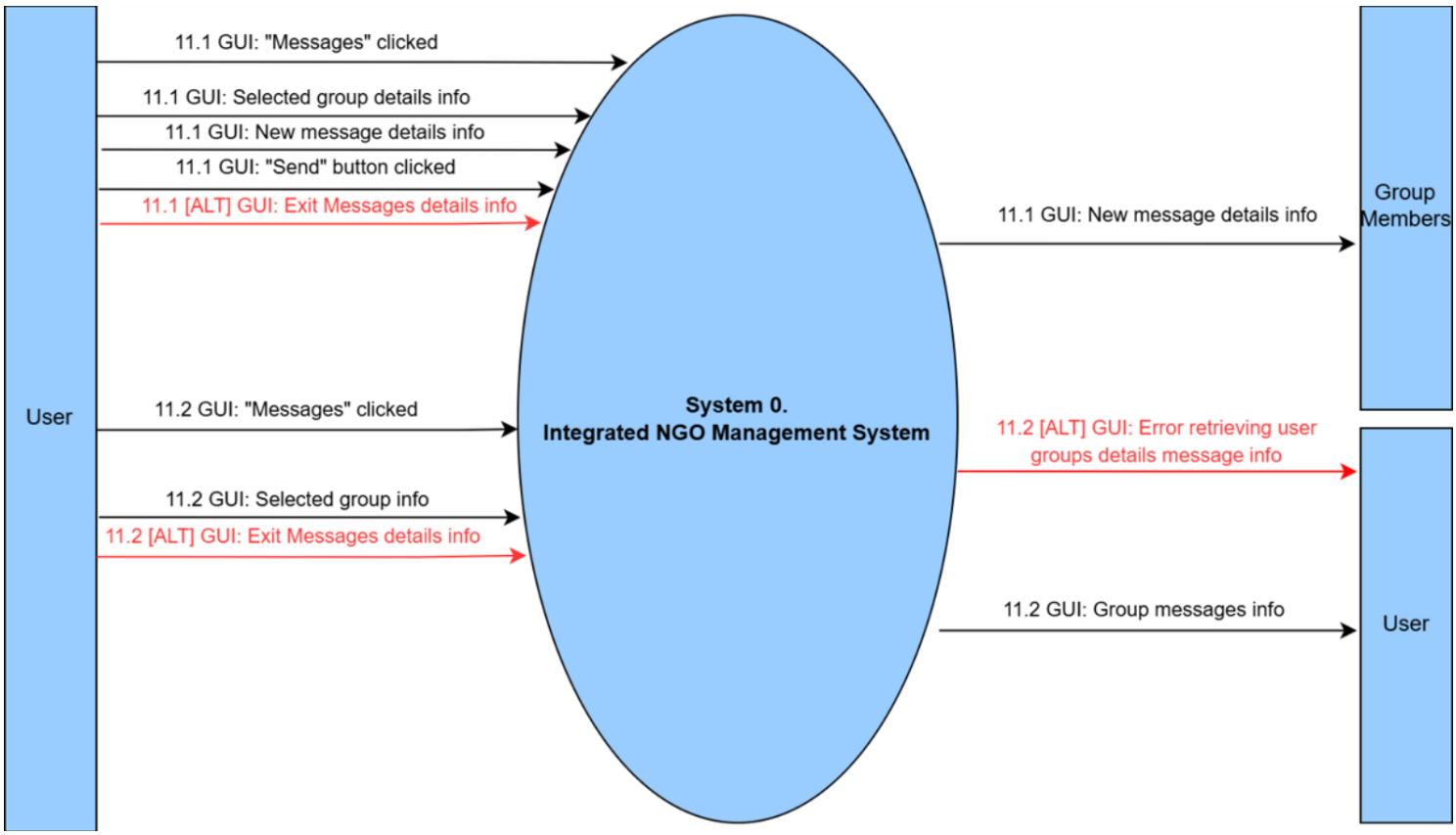


Figure 51: Message Subsystem Technical Context Diagram

2.6 Technical Primitive Level Diagram

2.6.1 User Subsystem Primitive Diagrams

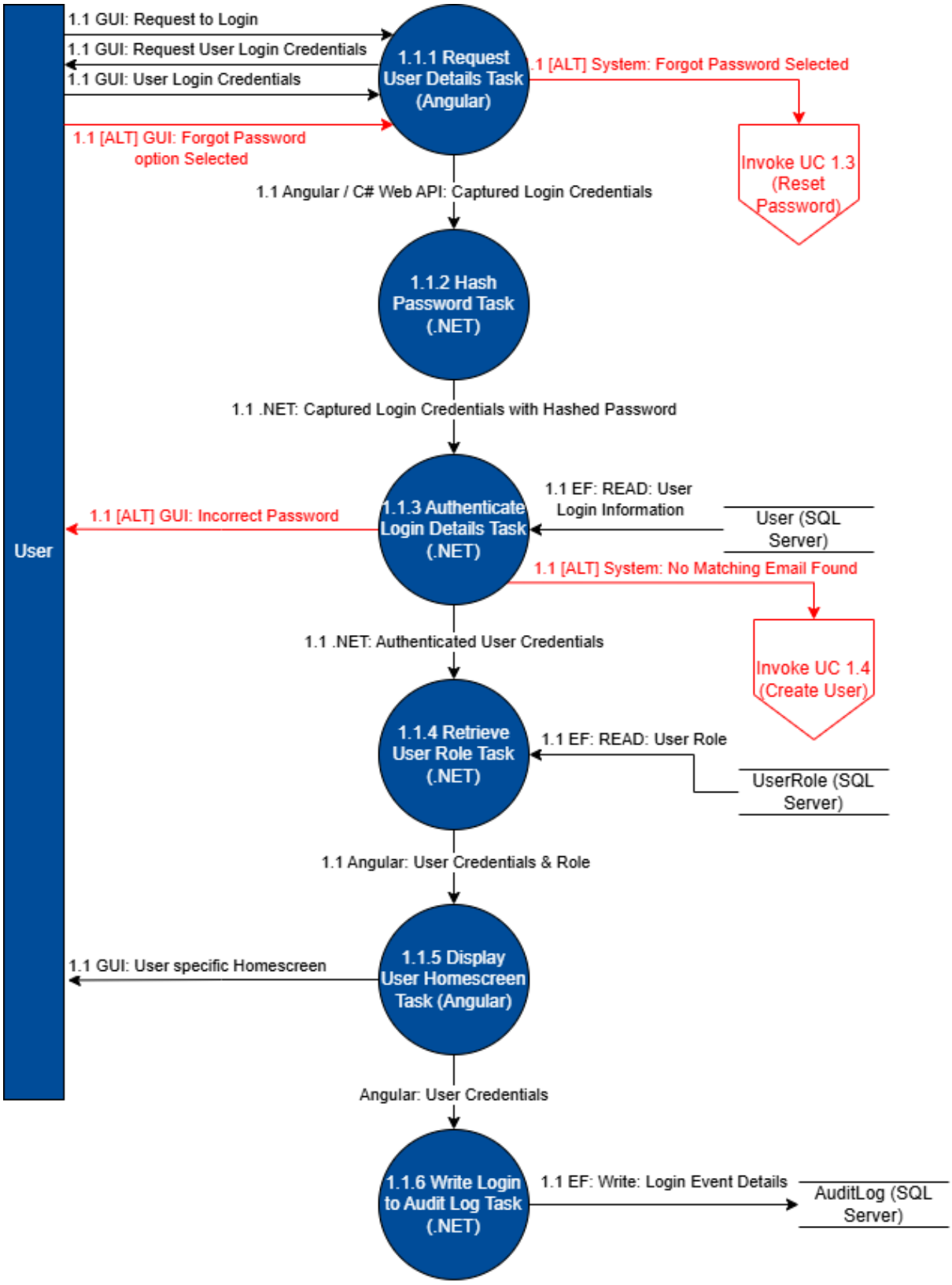


Figure 52: 1.1 User Login Technical Primitive Diagram

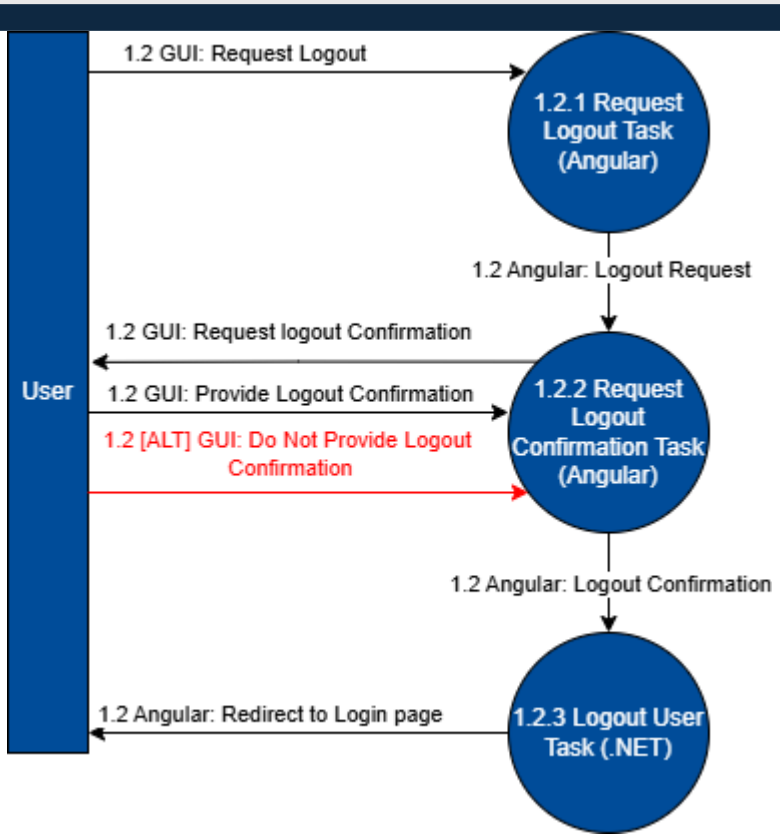


Figure 53: 1.2 User Logout Technical Primitive Diagram

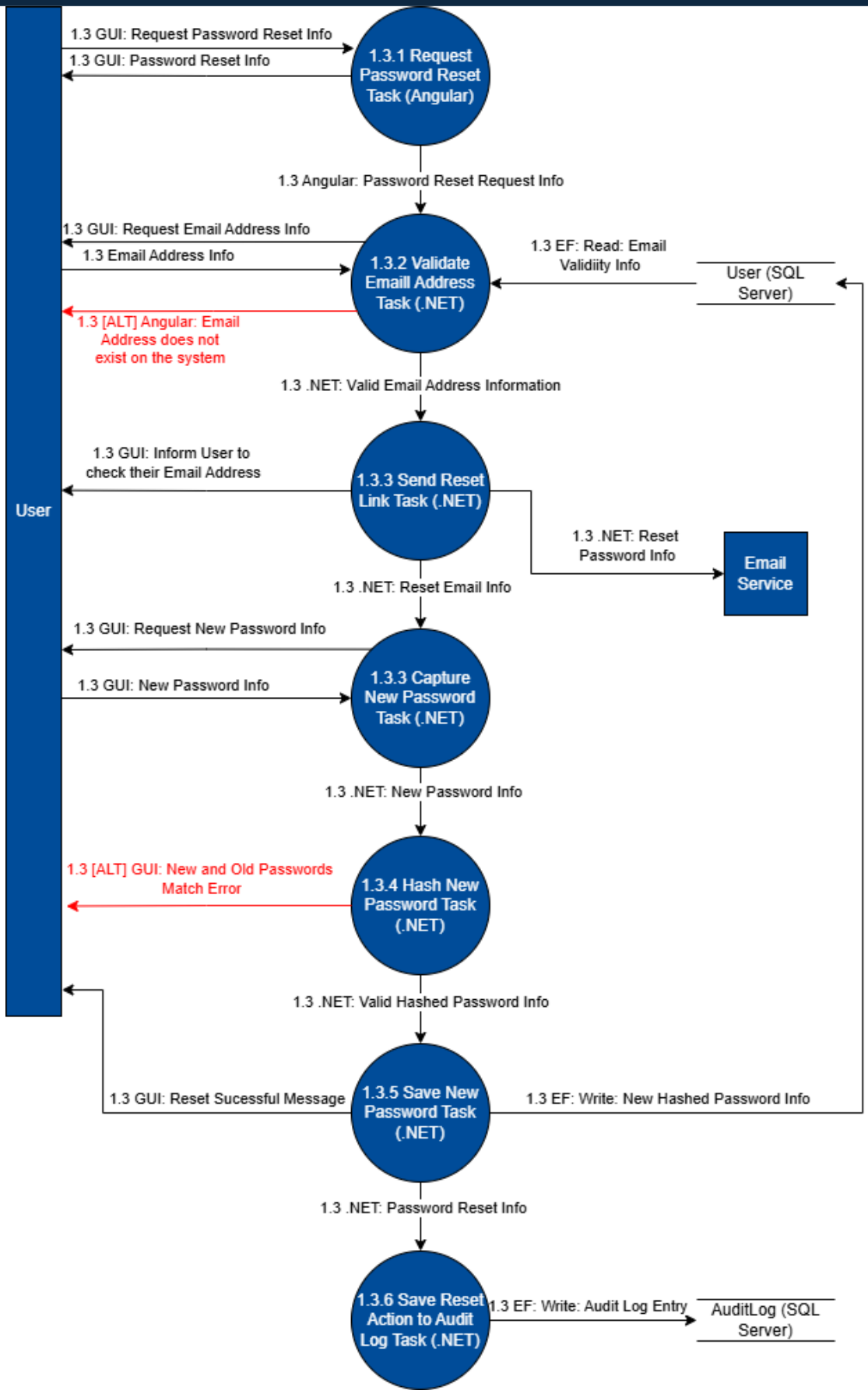


Figure 54: 1.3 Reset Password Technical Primitive Diagram

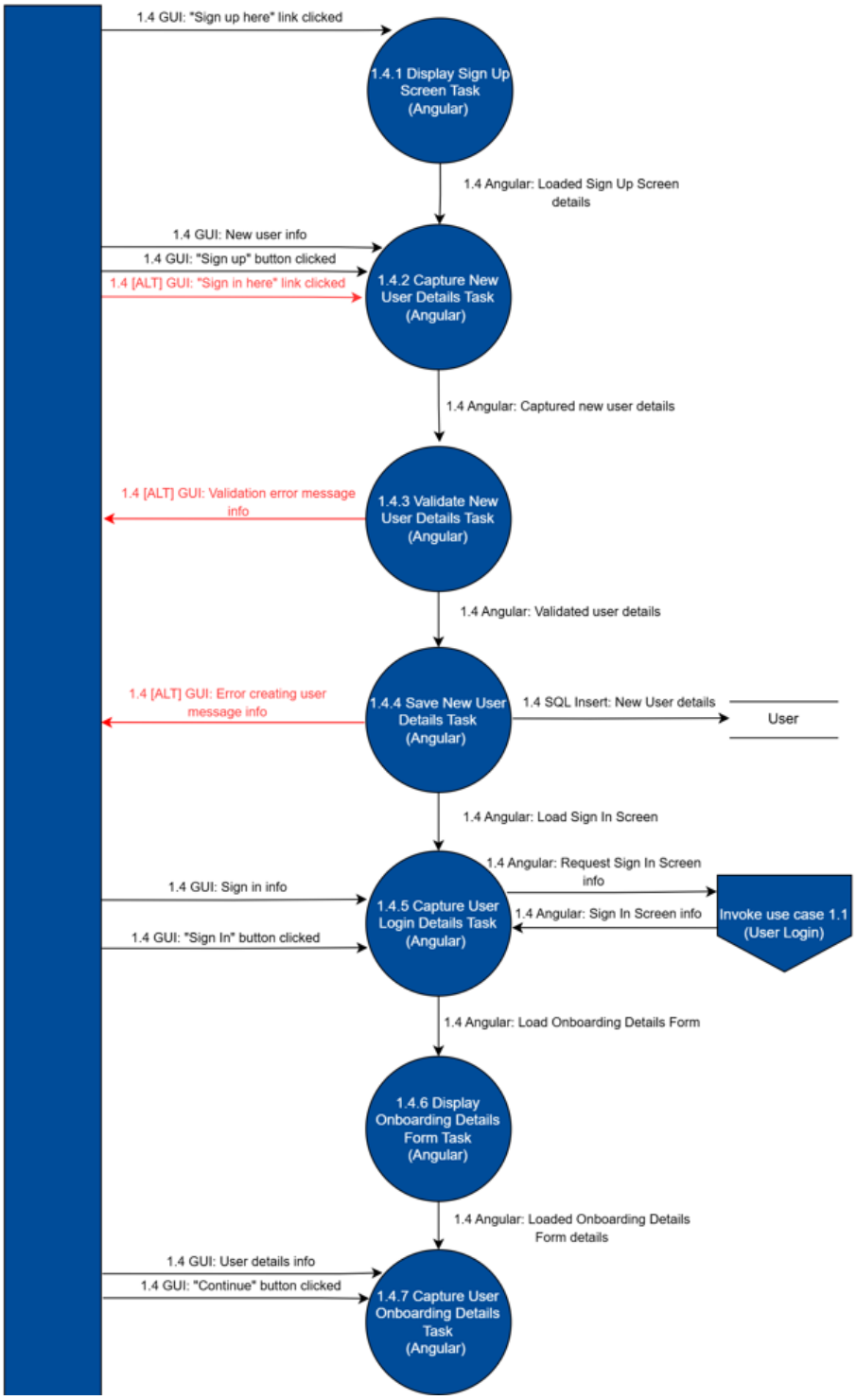


Figure 55: 1.4 Create User Technical Primitive Diagram

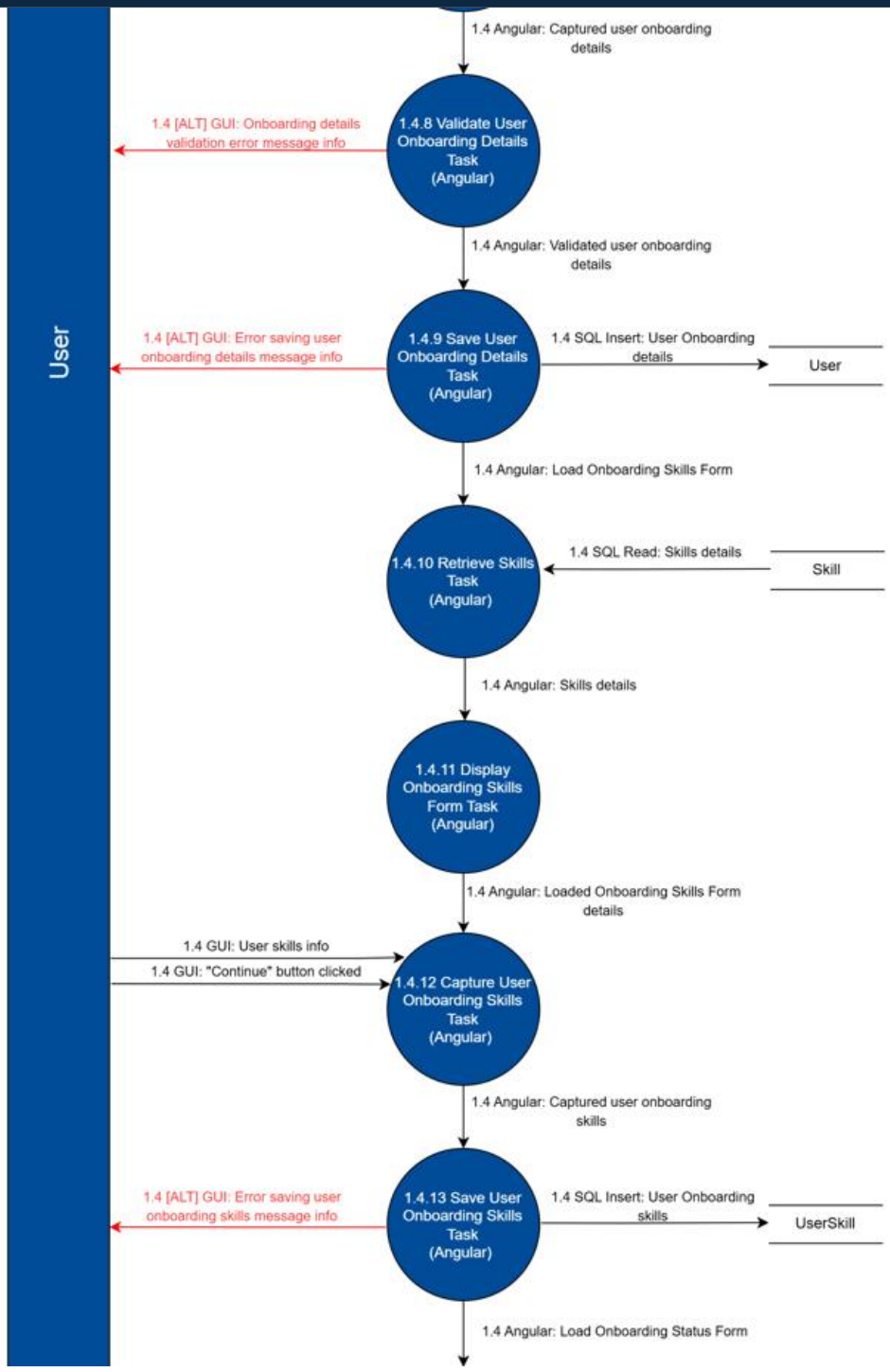


Figure 56: 1.4 Create User Technical Primitive Diagram (Continued)

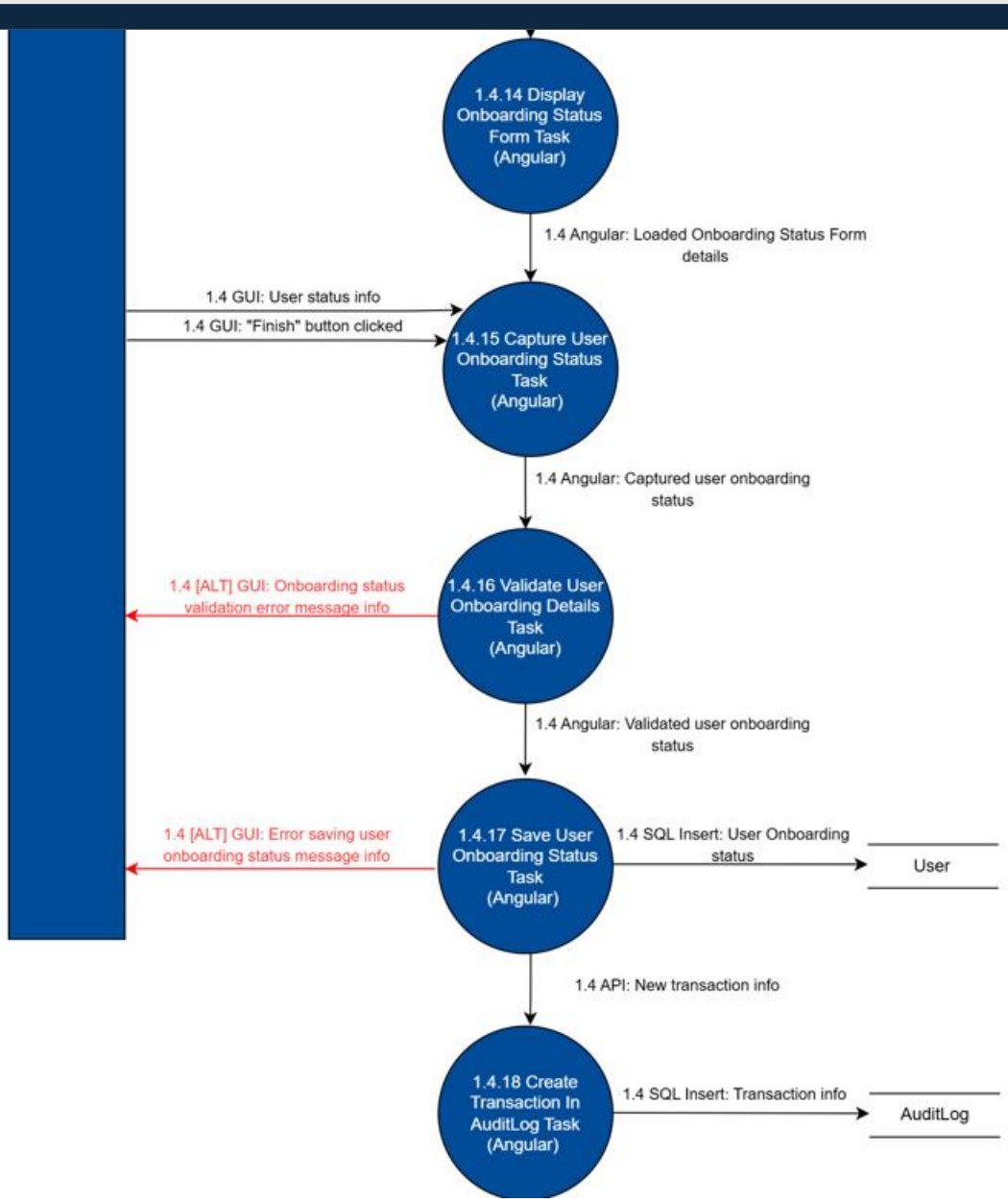


Figure 57: 1.4 Create User Technical Primitive Diagram (Continued)

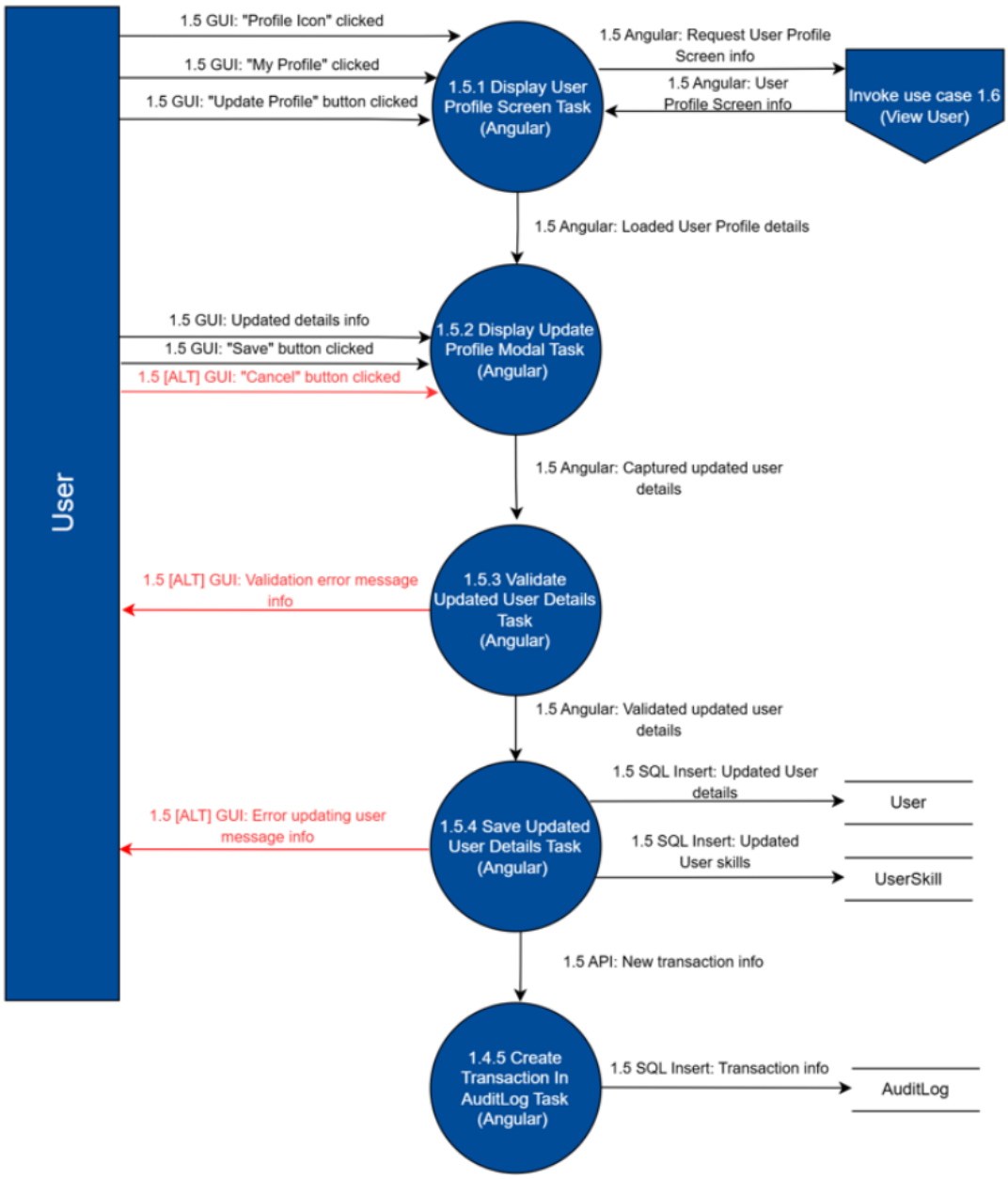


Figure 58: 1.5 Update User Technical Primitive Diagram

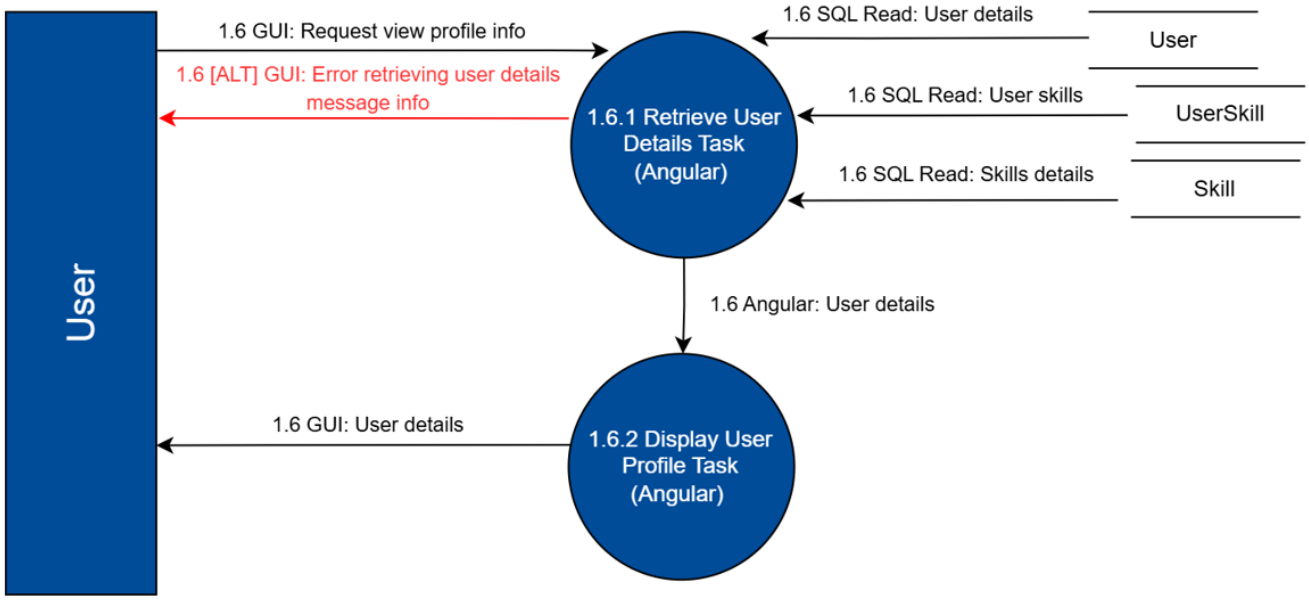


Figure 59: 1.6 View User Technical Primitive Diagram

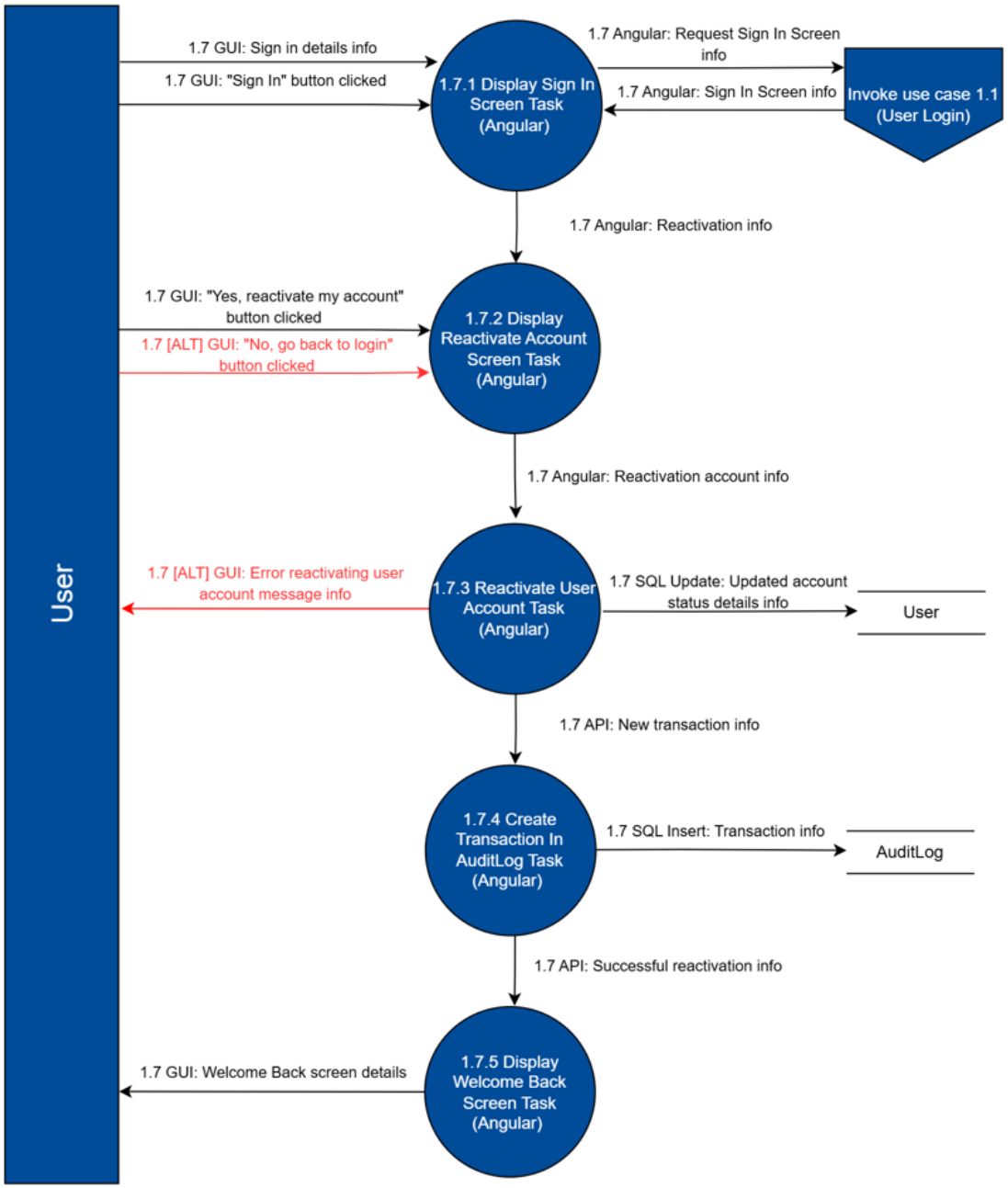


Figure 60: 1.7 Reactivate User Technical Primitive Diagram

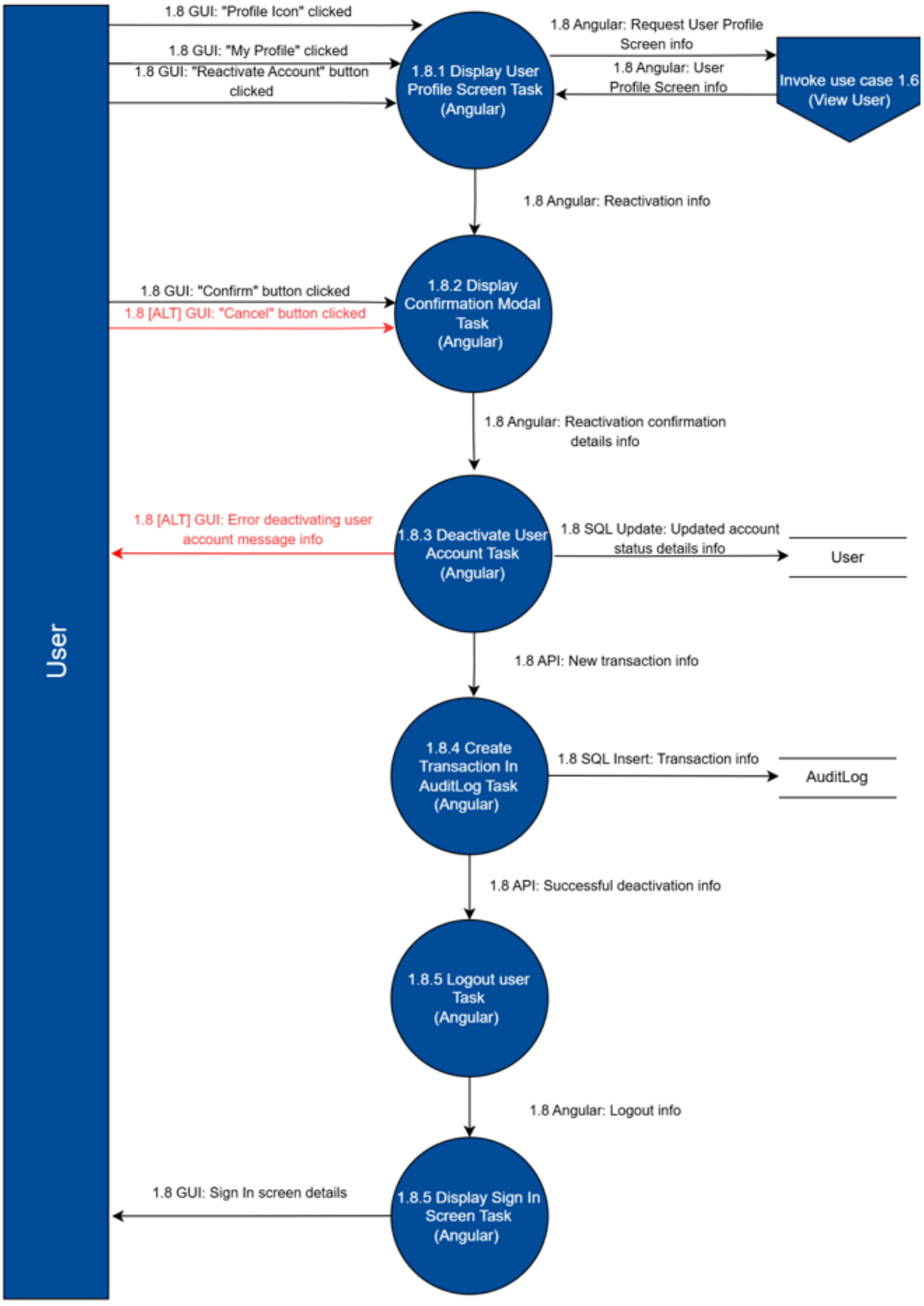


Figure 61: 1.8 Deactivate User Technical Primitive Diagram

2.6.2 Admin Subsystem Primitive Diagrams

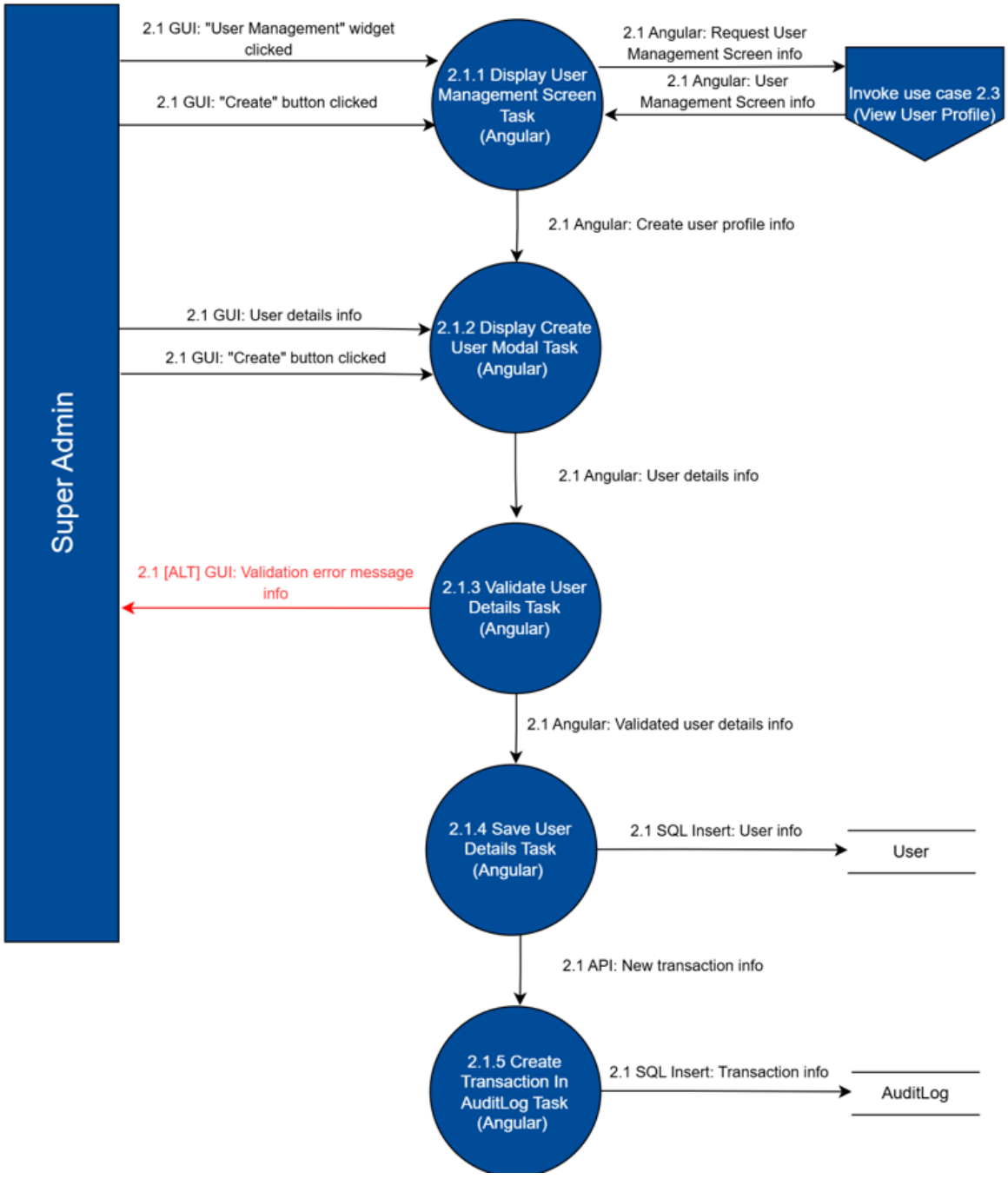


Figure 62: 2.1 Create User Profile Technical Primitive Diagram

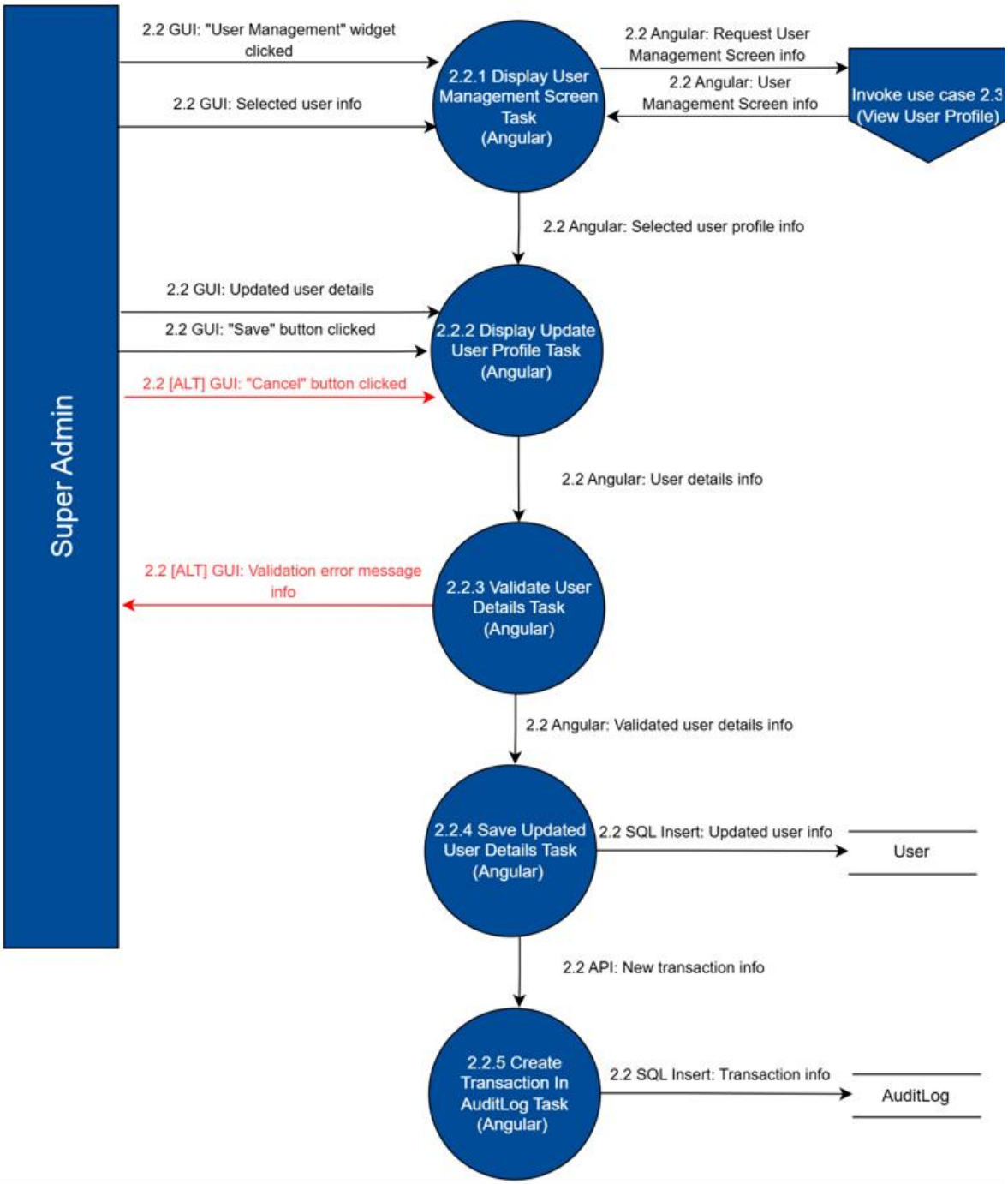


Figure 63: 2.2 Update User Profile Technical Primitive Diagram

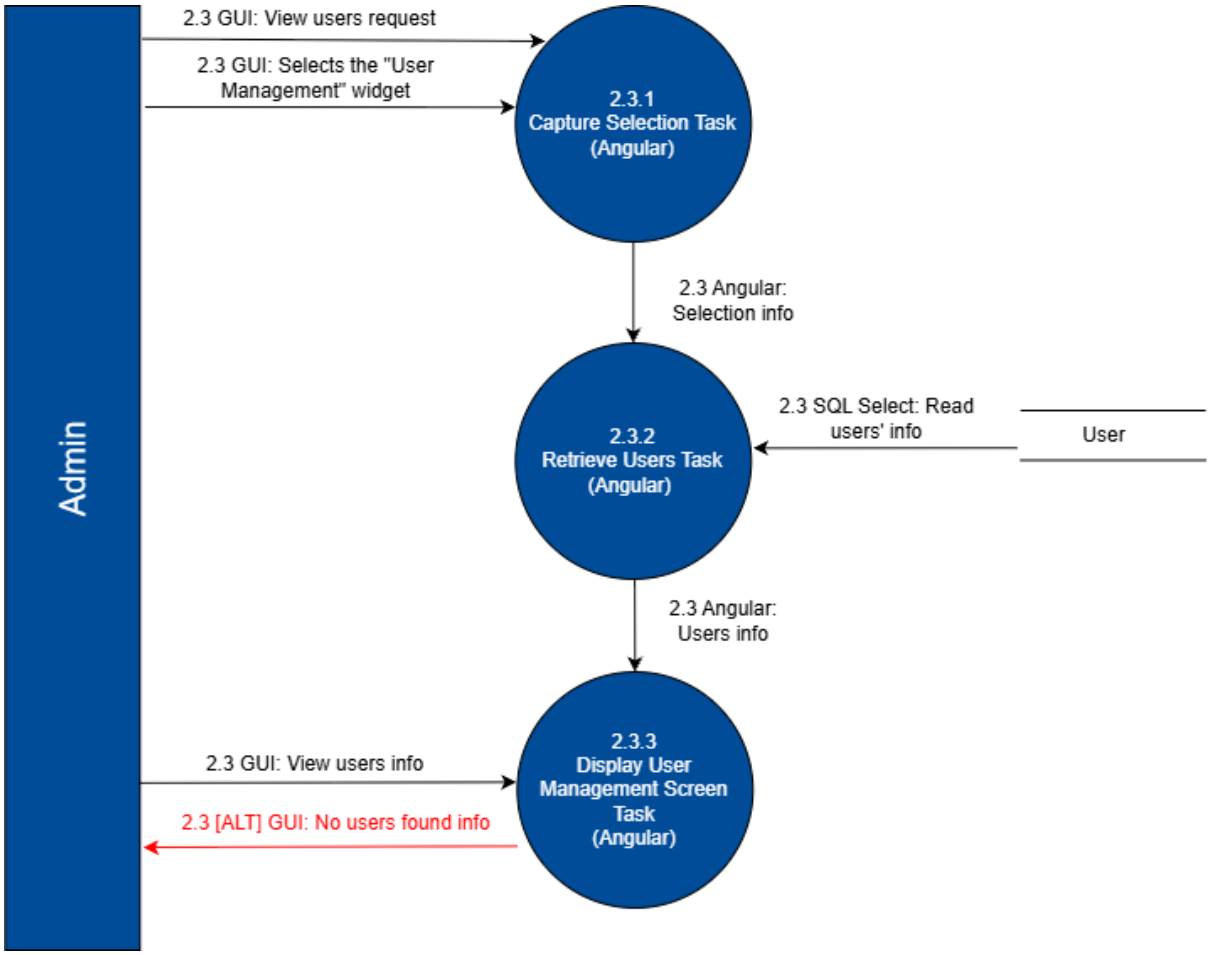


Figure 64: 2.3 View User Profile Technical Primitive Diagram

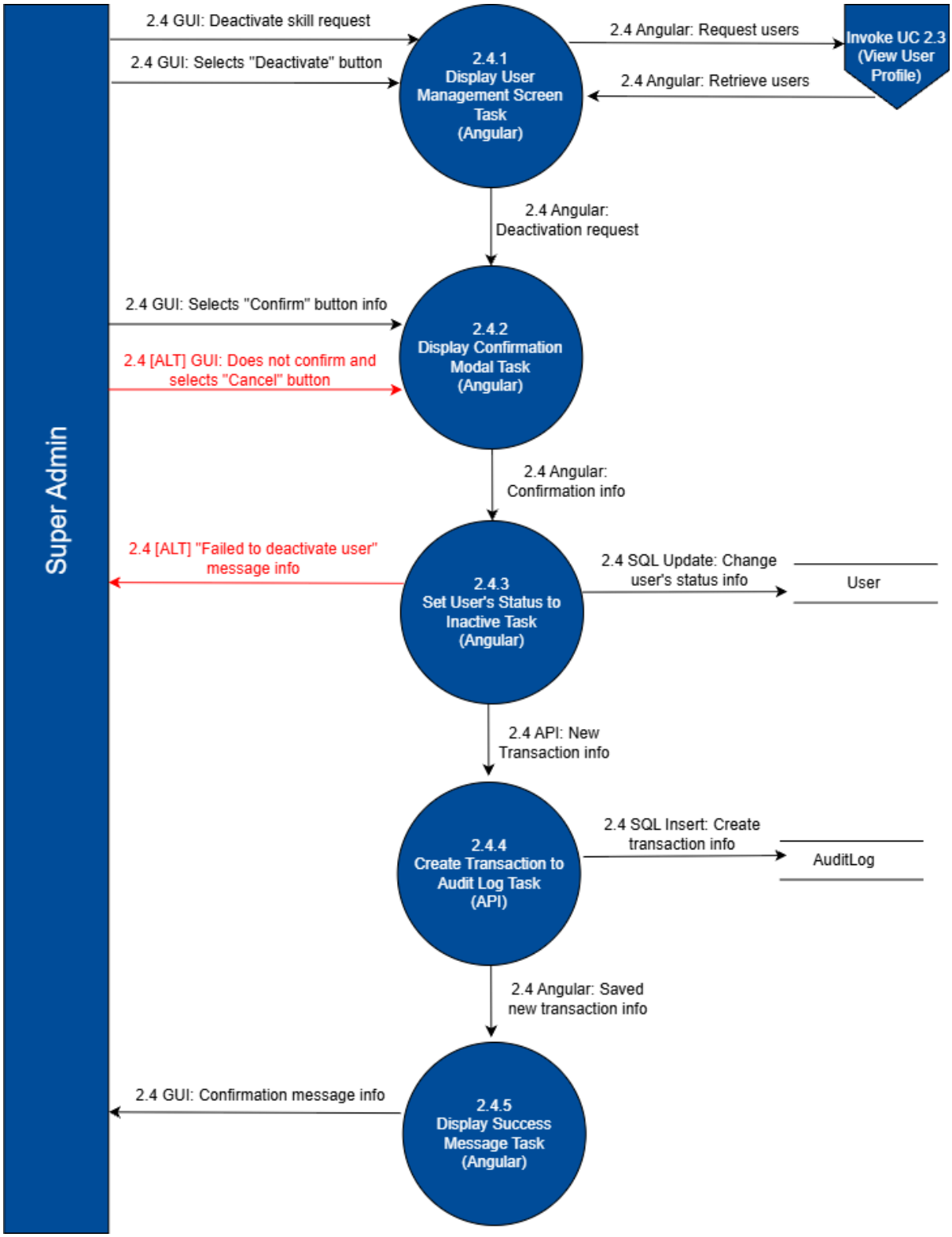


Figure 65: 2.4 Deactivate User Technical Primitive Diagram

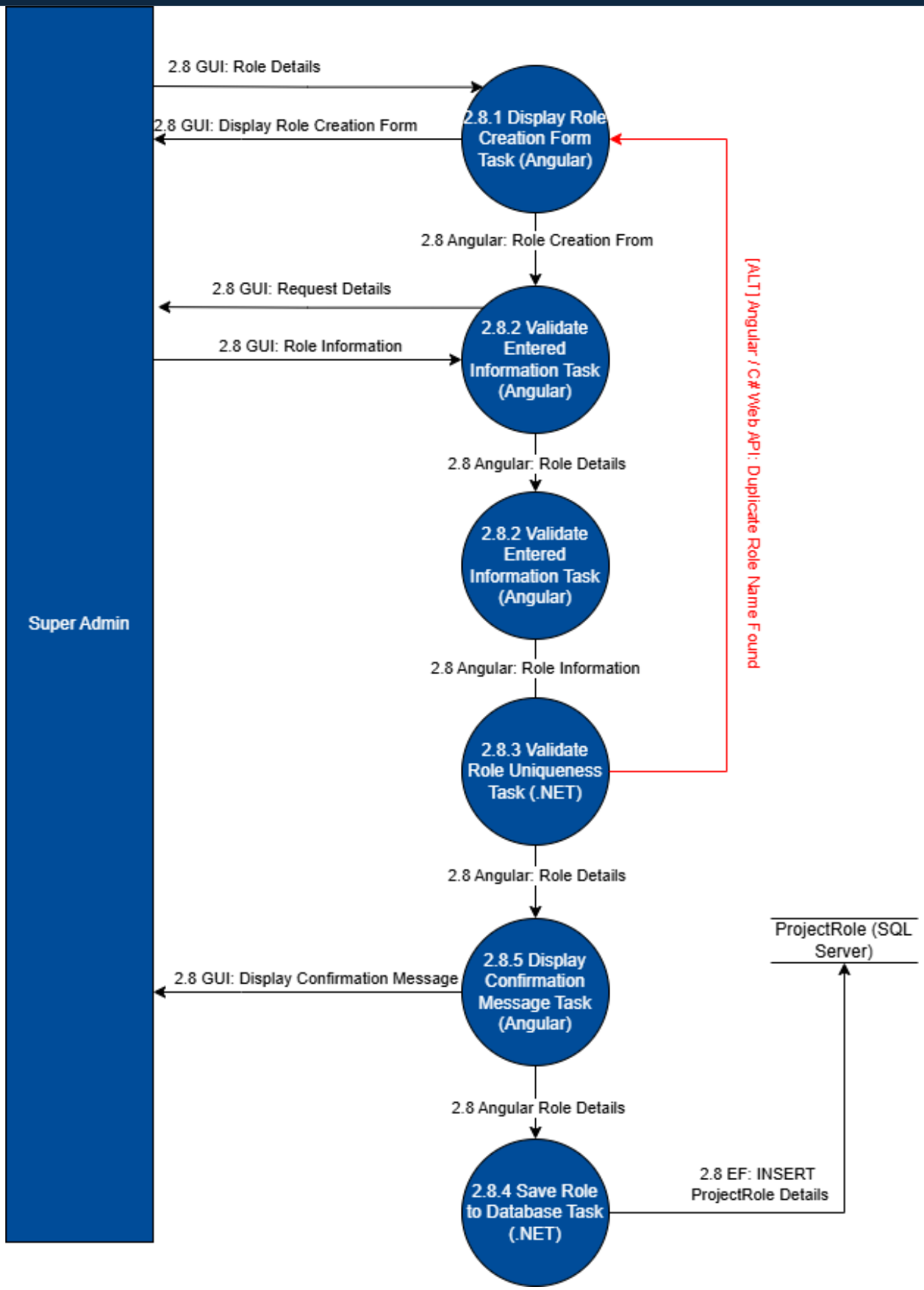


Figure 66: 2.8 Create Role Technical Primitive Diagram

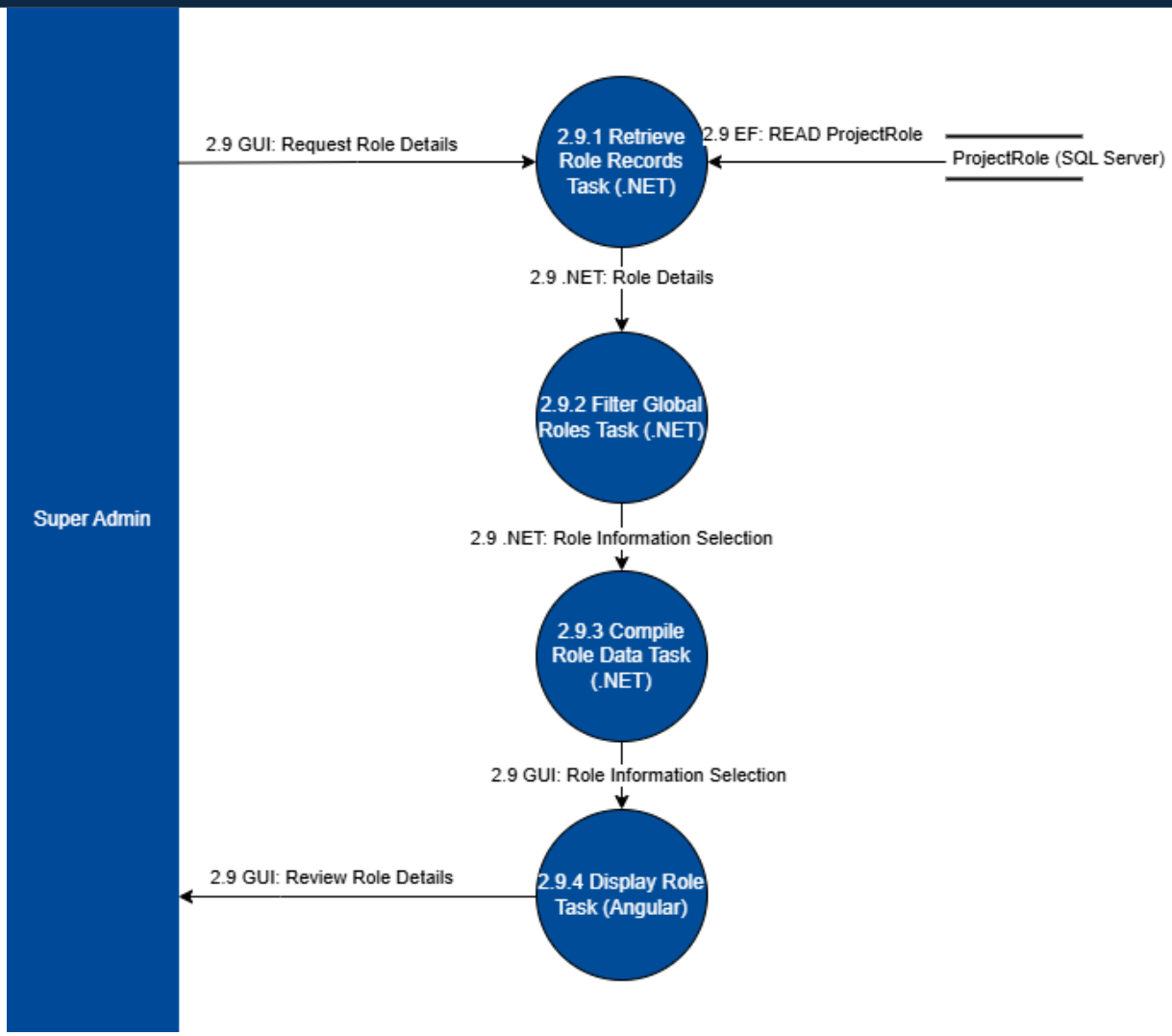


Figure 67:2.9 View Role Technical Primitive Diagram

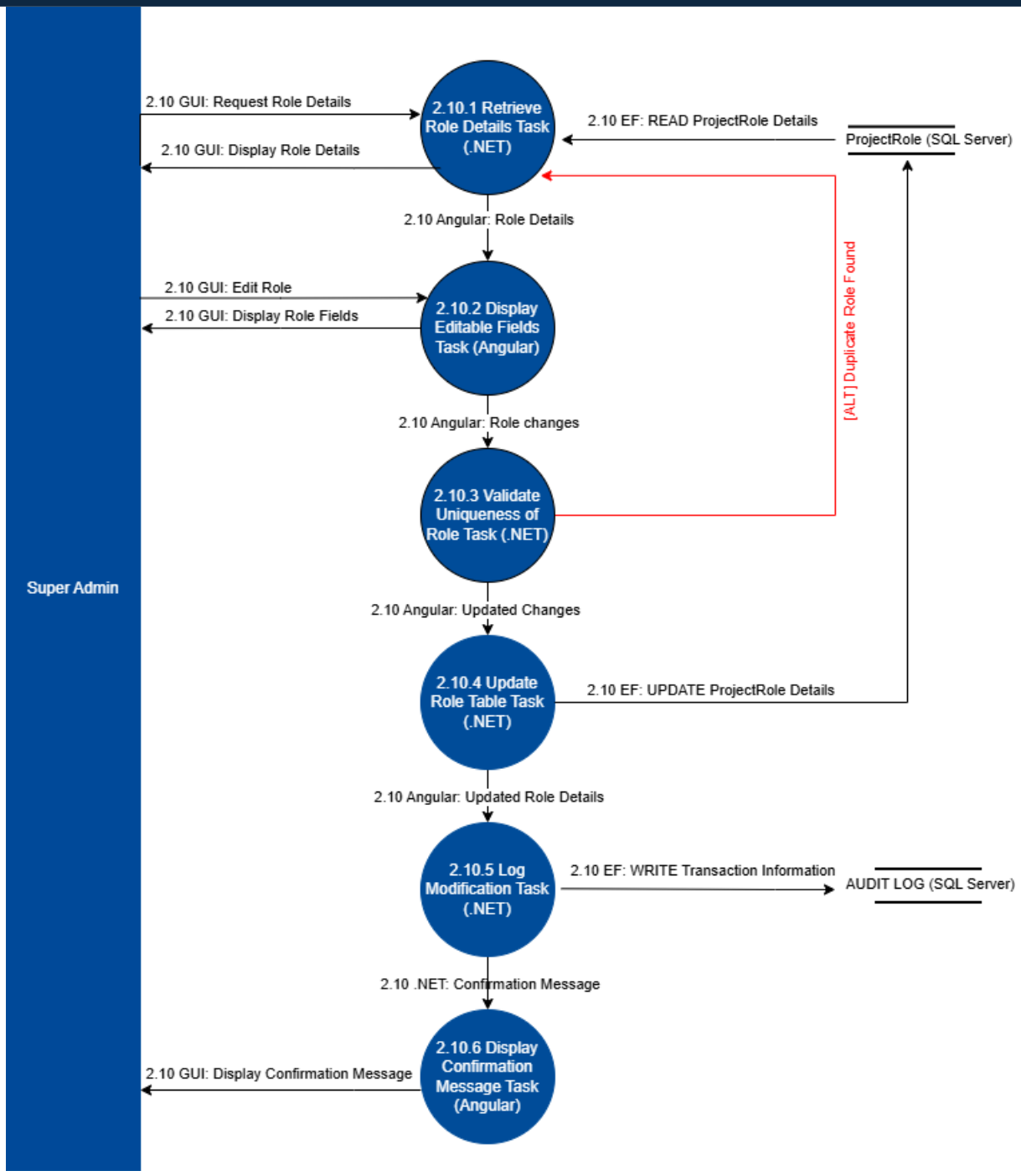


Figure 68: 2.10 Update Role Technical Primitive Diagram

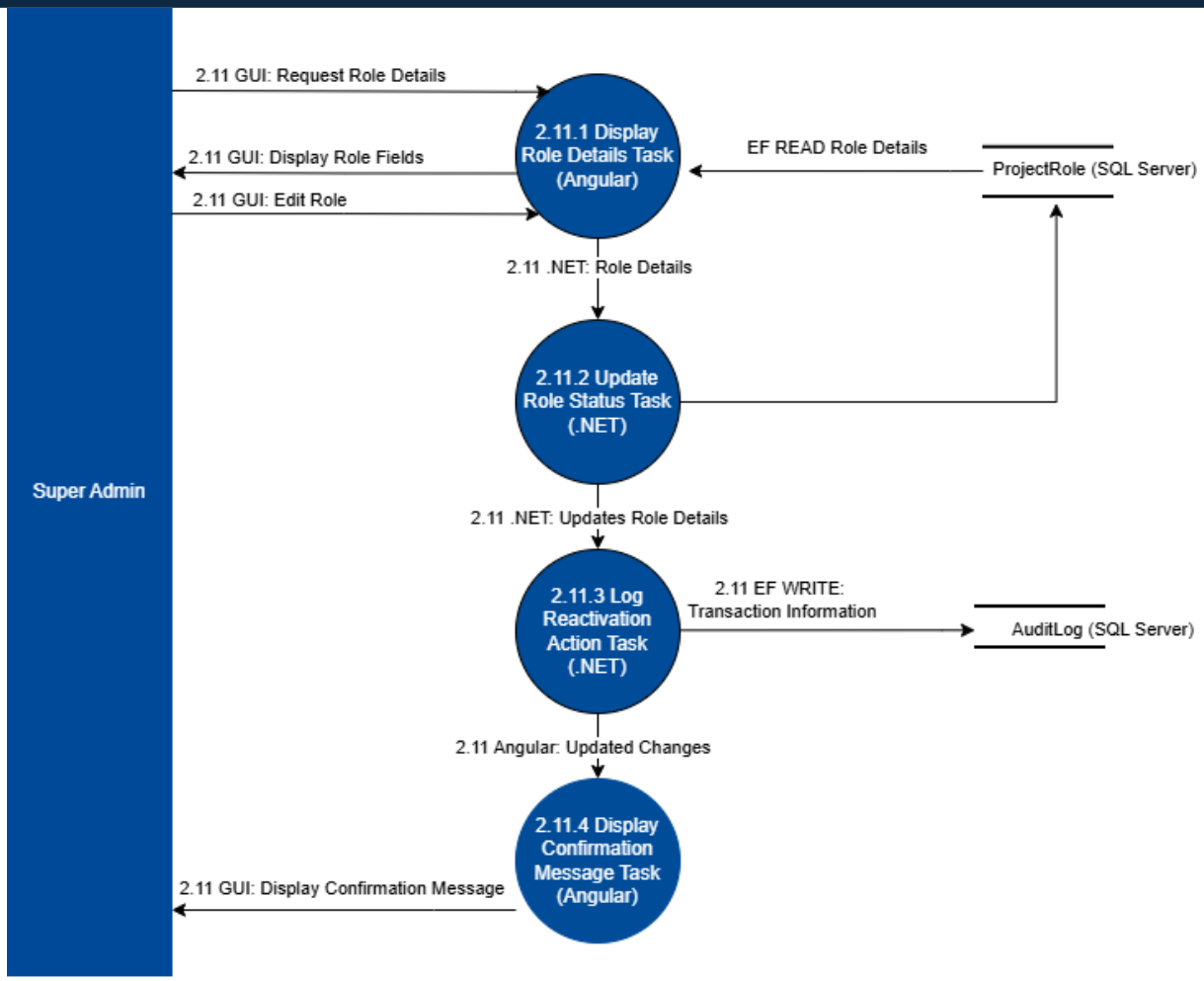


Figure 69: 2.11 Reactivate Role Technical Primitive Diagram

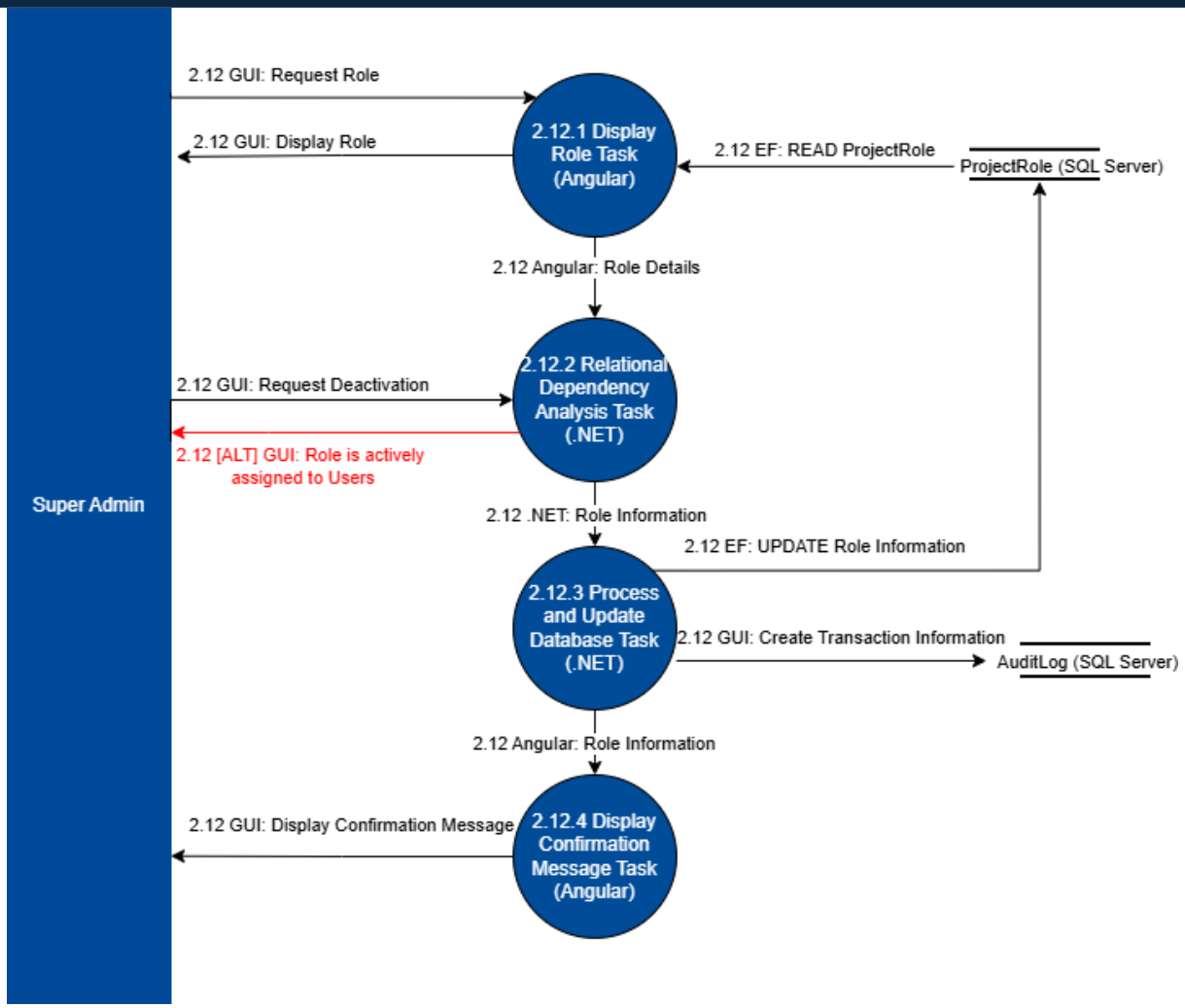


Figure 70:2.12 Deactivate Role Technical Primitive Diagram

2.6.3 Group Subsystem Primitive Diagrams

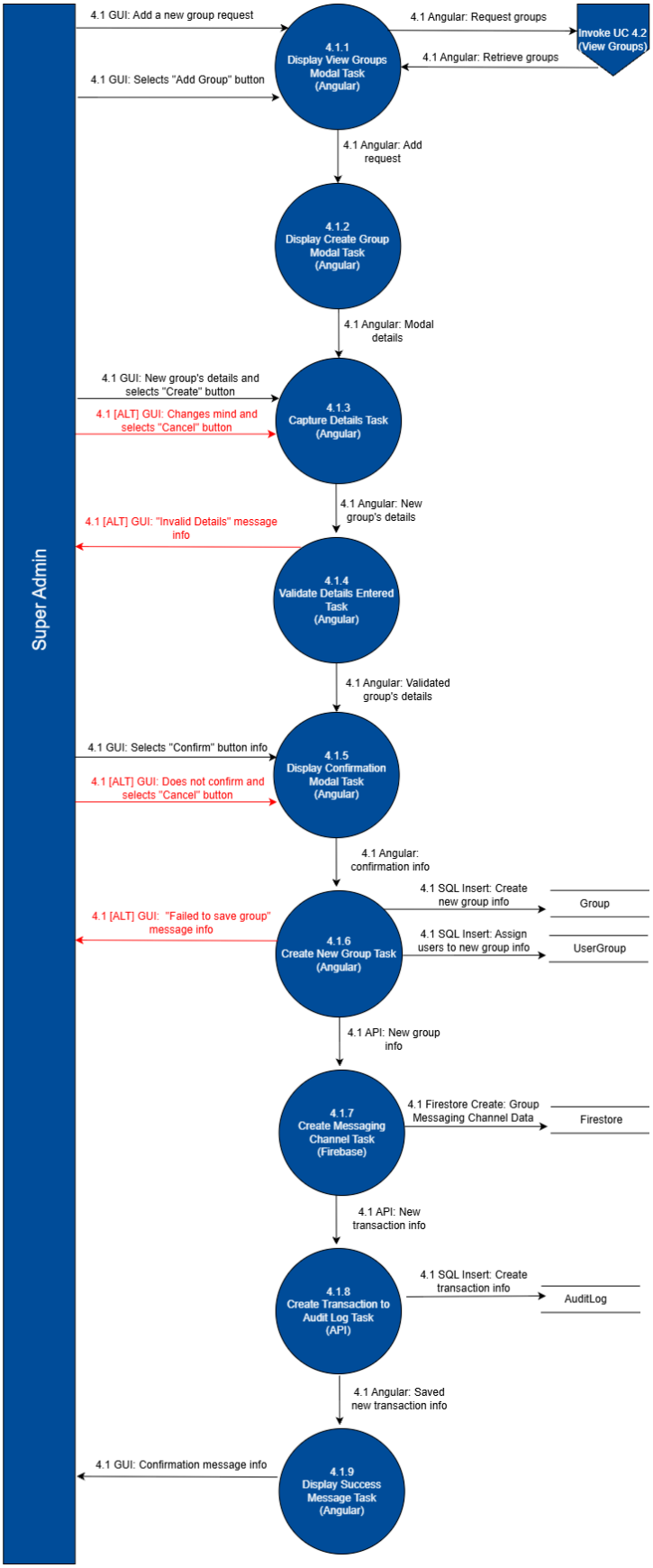


Figure 71: 4.1 Create Group Technical Primitive Diagram

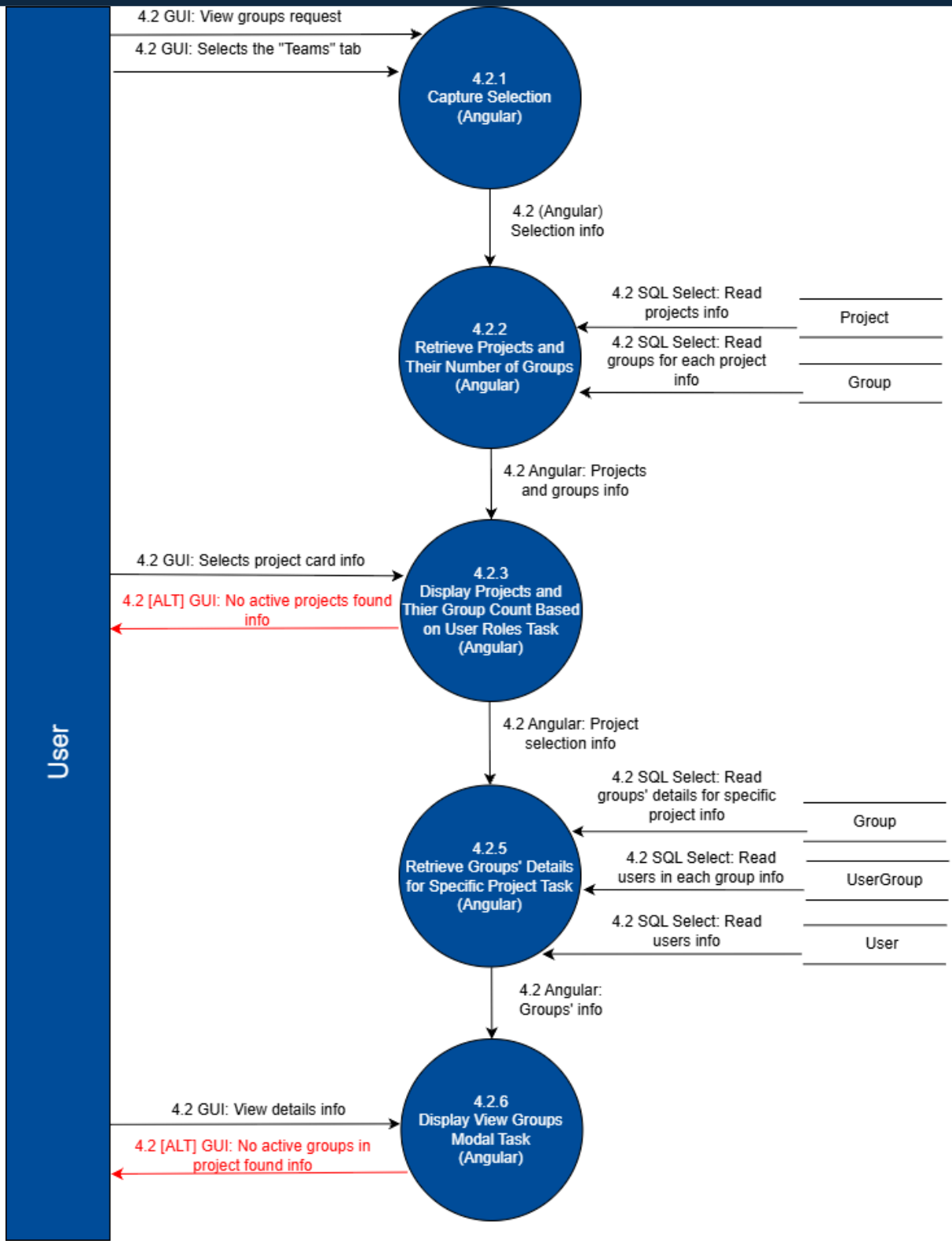


Figure 72: 4.2 View Groups Technical Primitive Diagram

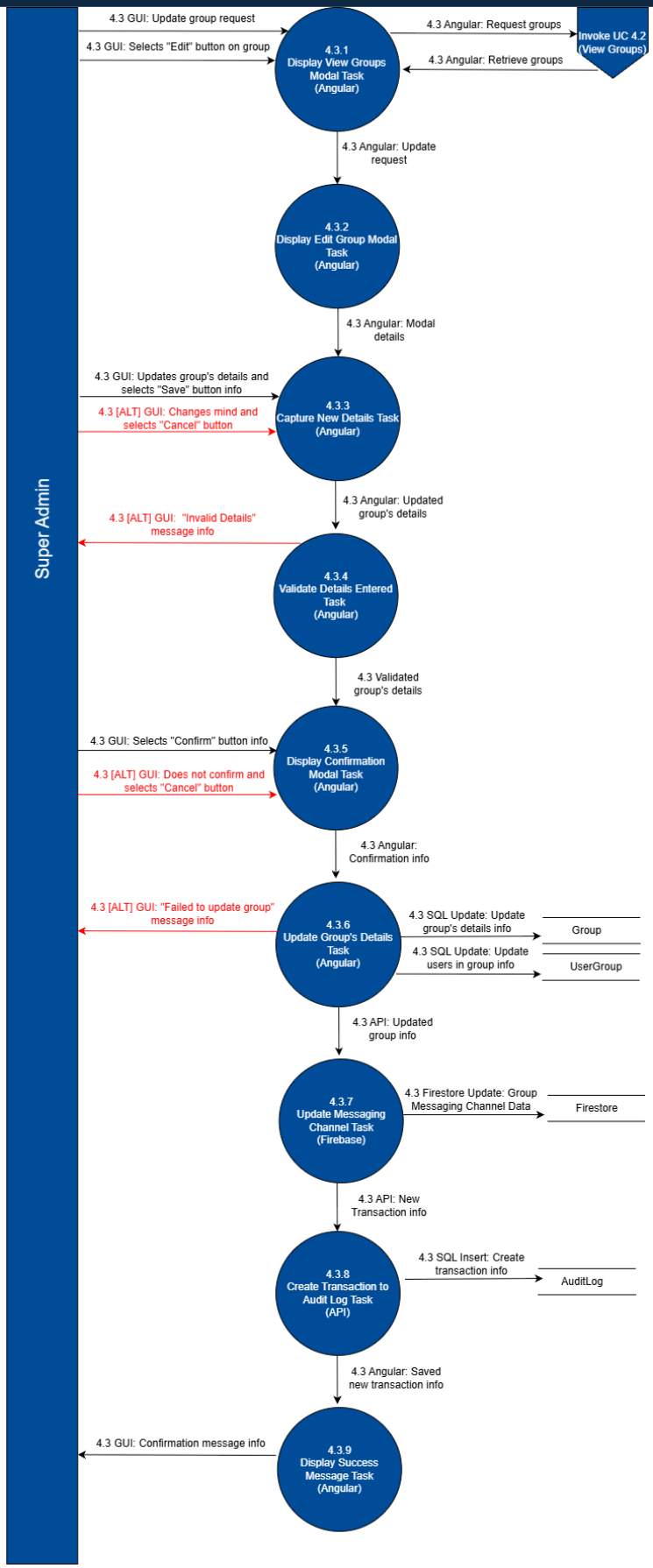


Figure 73: 4.3 Update Group Technical Primitive Diagram

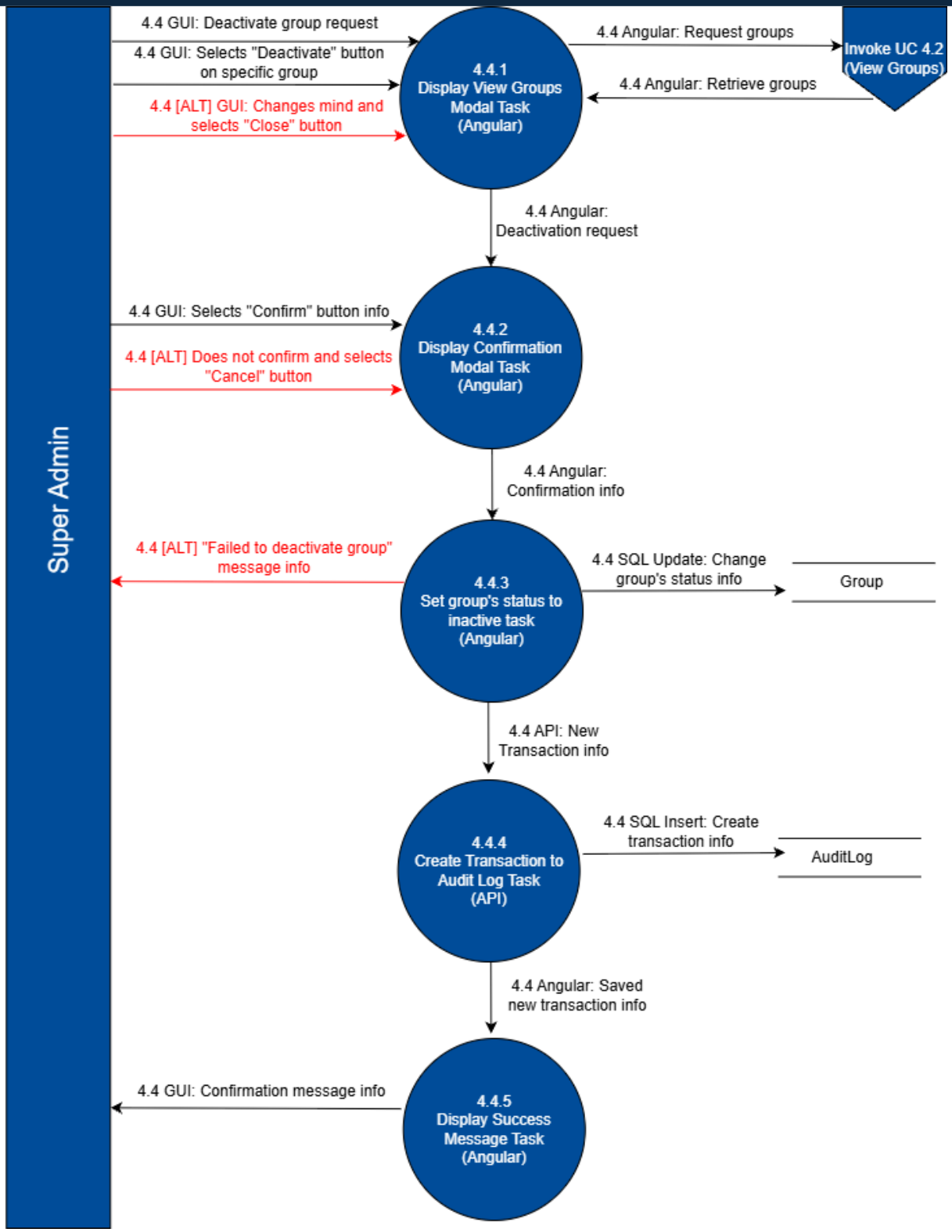


Figure 74: 4.4 Deactivate Group Technical Primitive Diagram

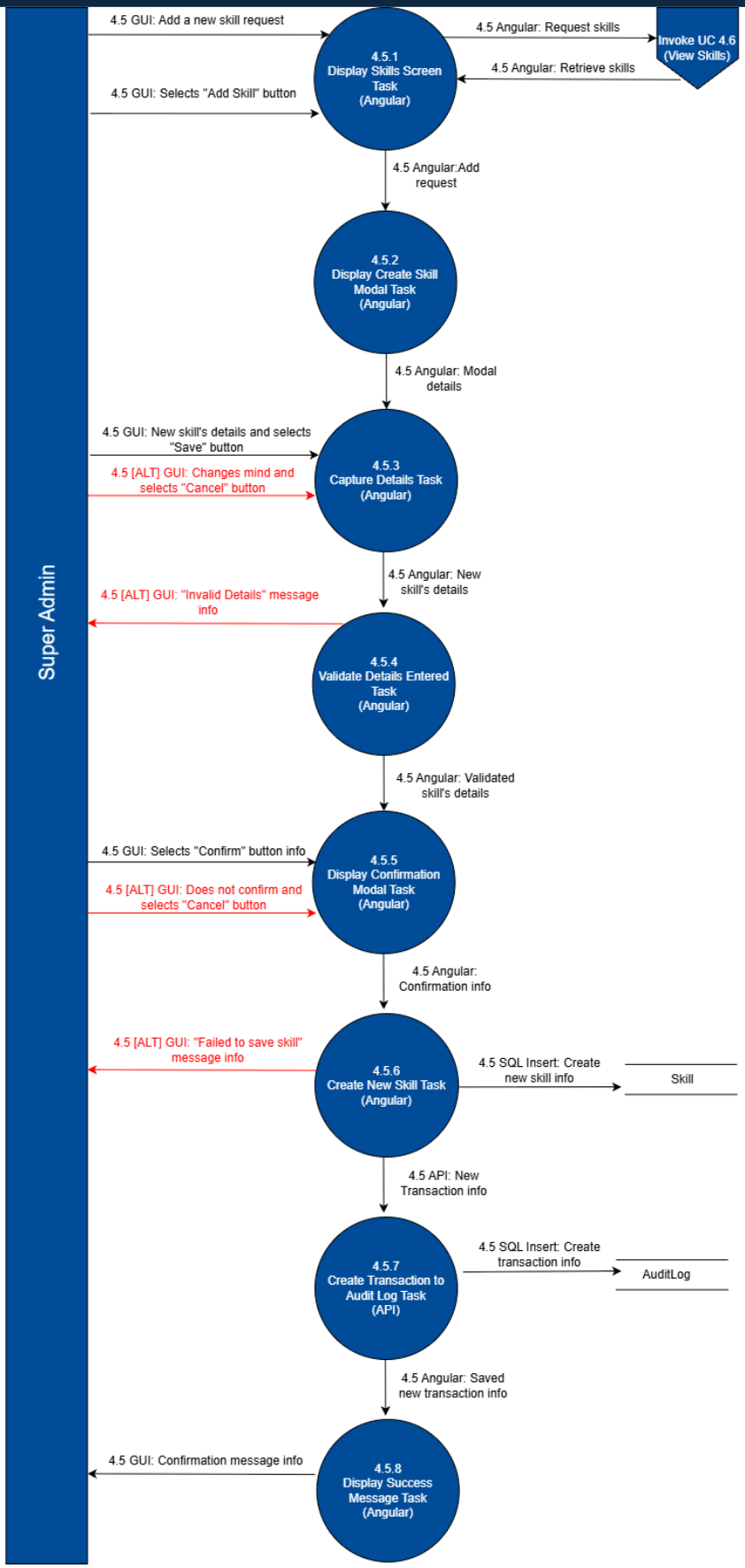


Figure 75: 4.5 Create Skill Technical Primitive Diagram

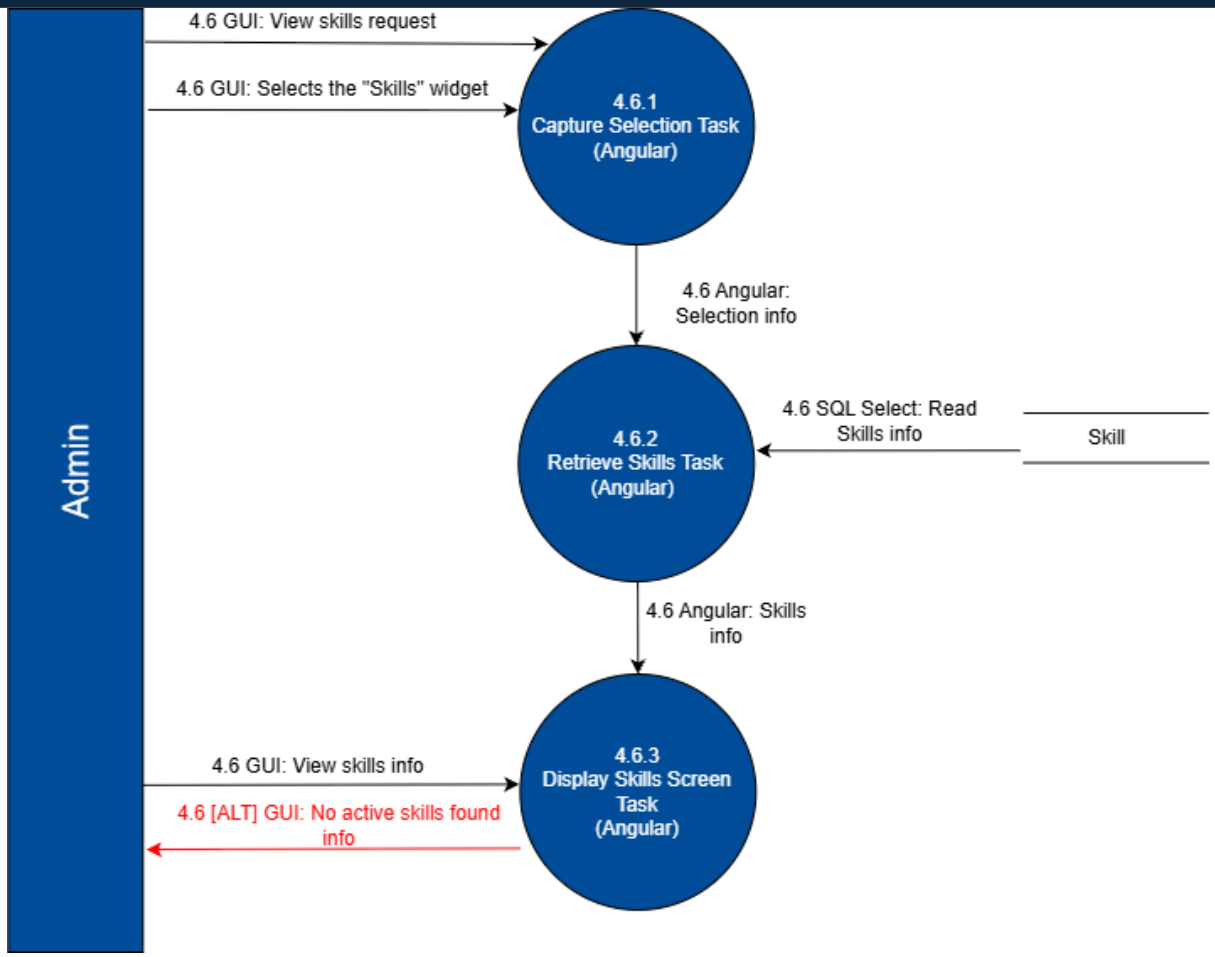


Figure 76: 4.6 View Skills Technical Primitive Diagram

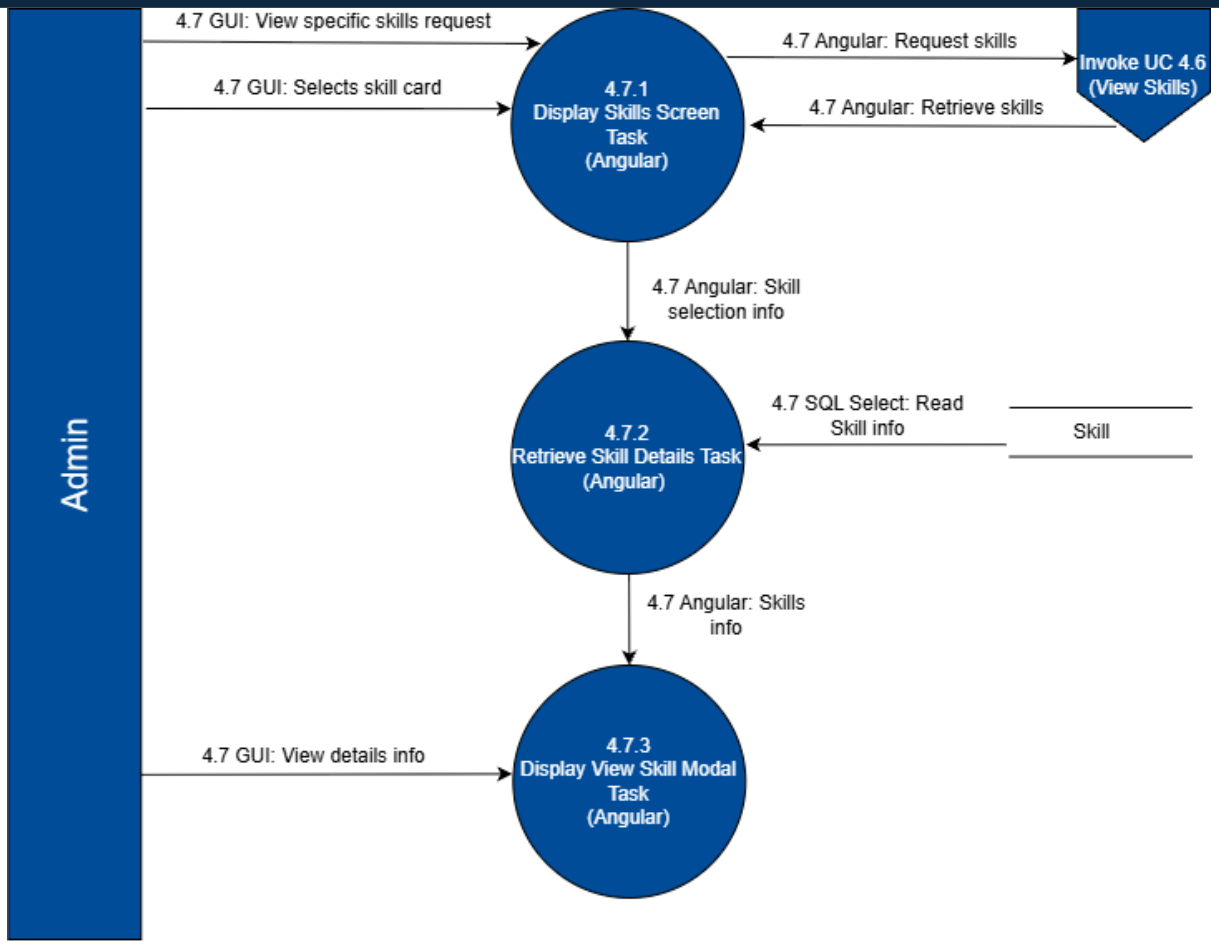


Figure 77: 4.7 View Skill Technical Primitive Diagram

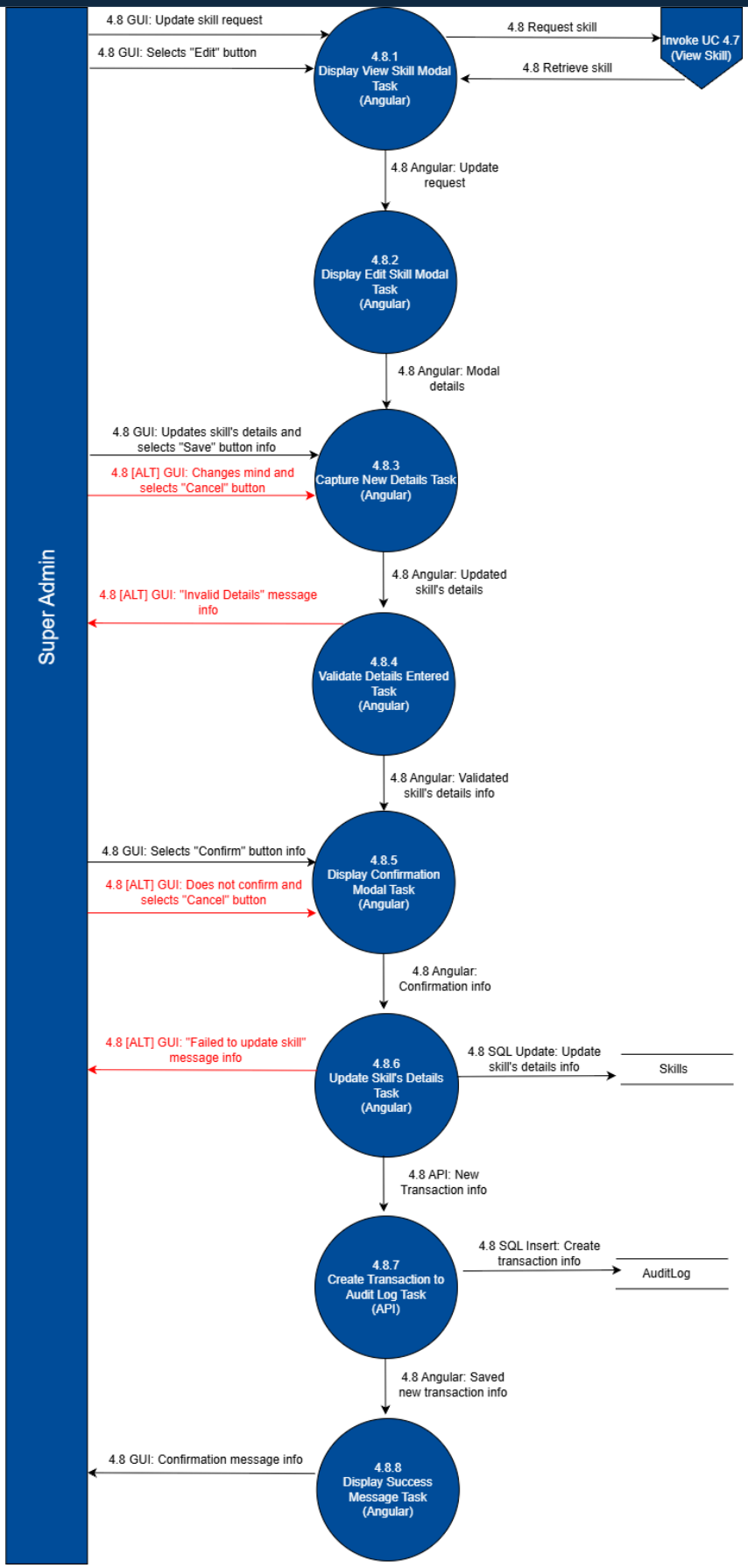


Figure 78: 4.8 Update Skill Technical Primitive Diagram

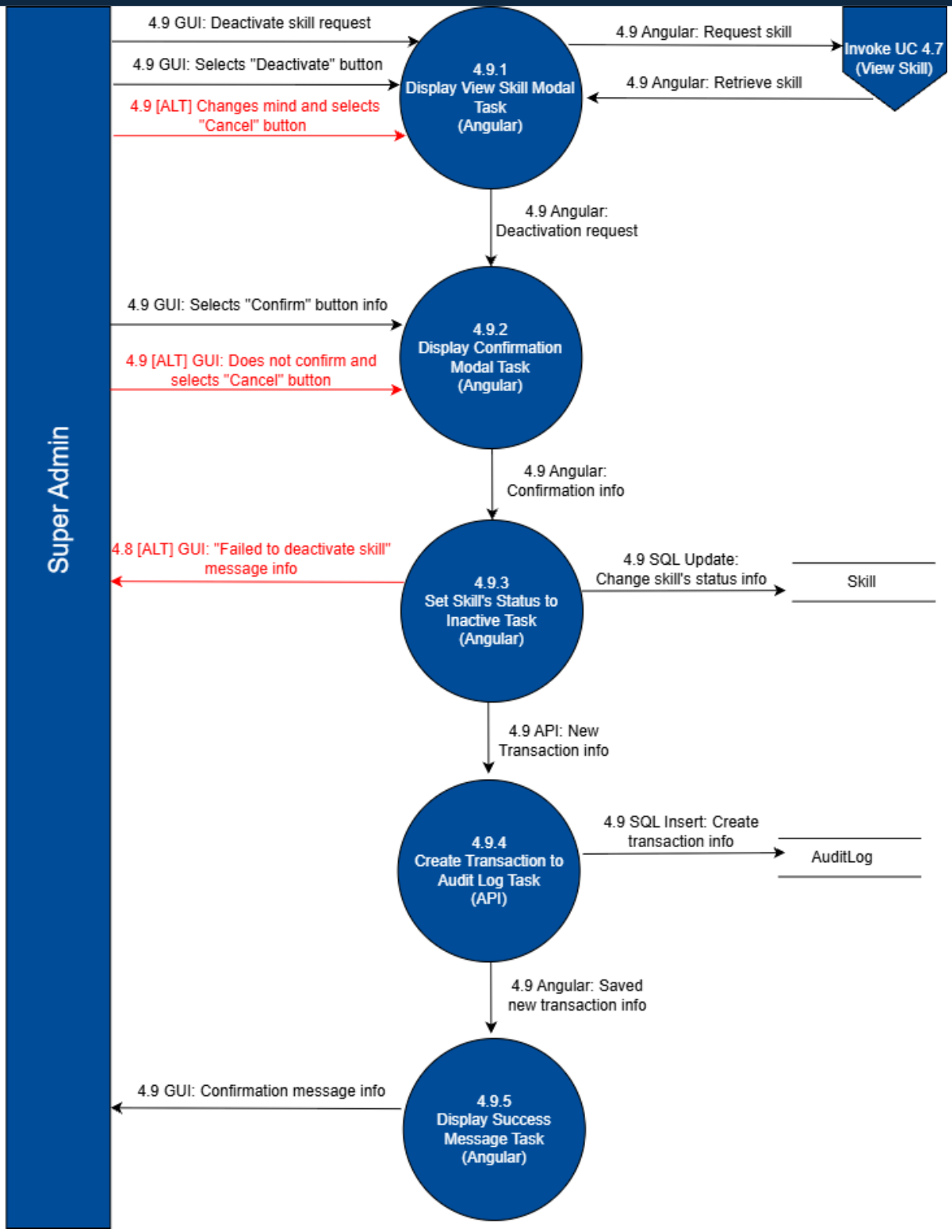


Figure 79: 4.9 Deactivate Skill Technical Primitive Diagram

2.6.4 Project Subsystem Primitive Diagrams

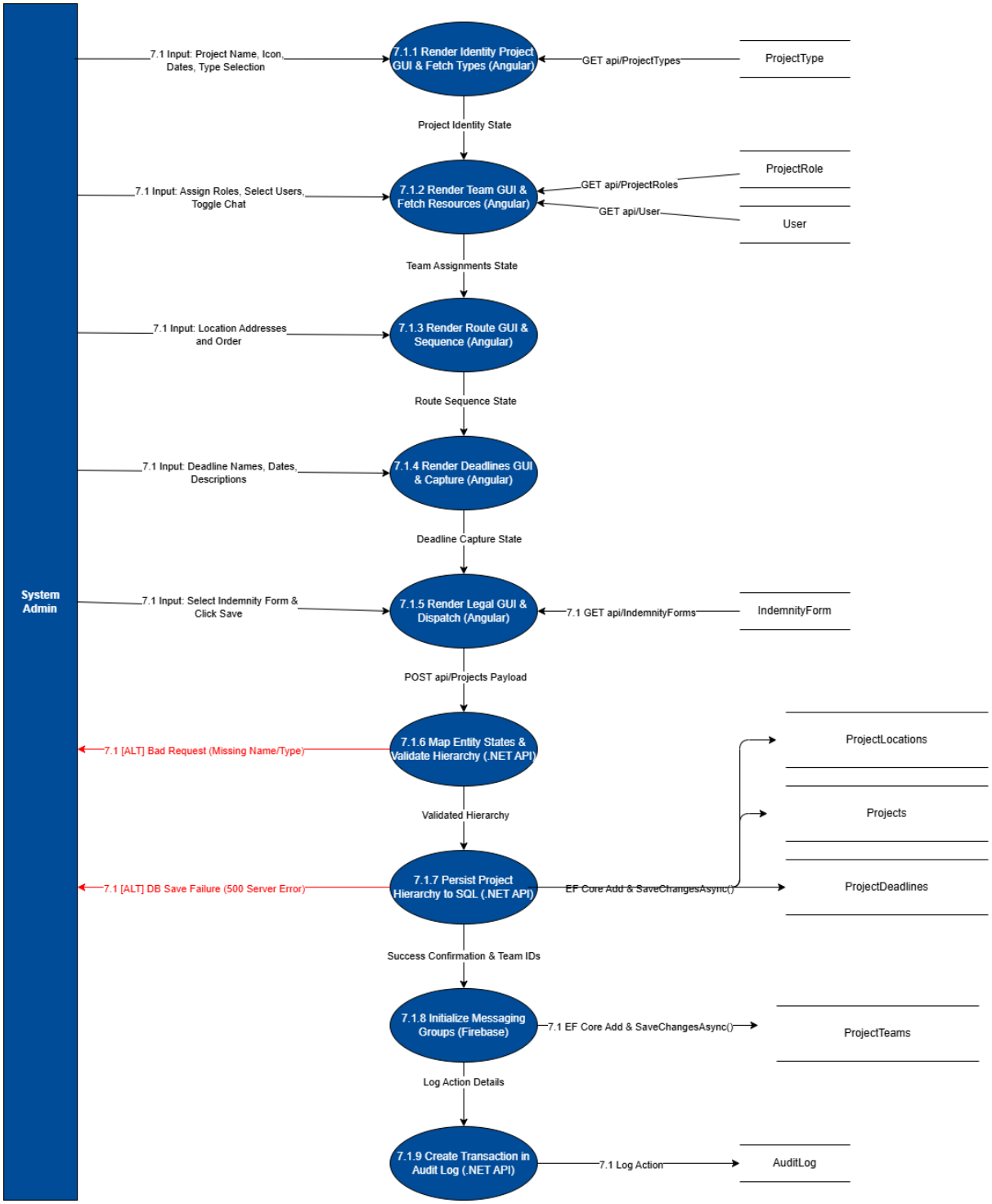


Figure 57: 7.1 Create Project Technical Primitive Diagram

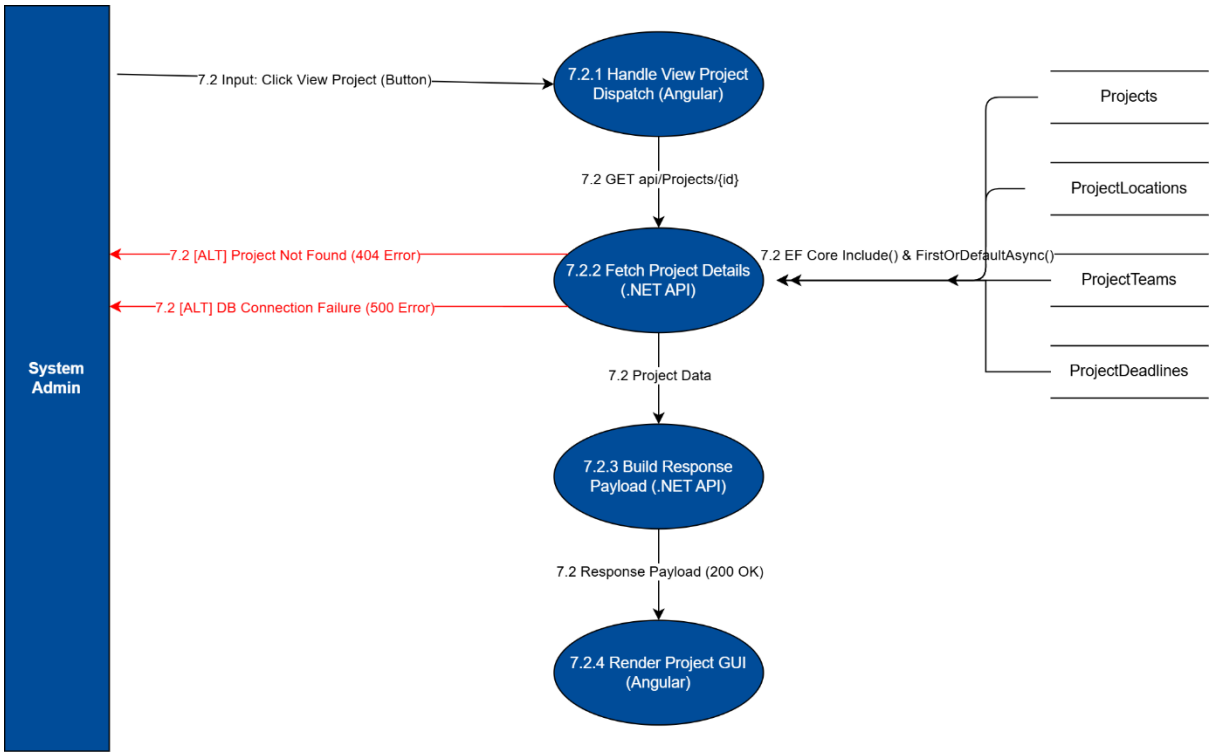


Figure 58: 7.2 View Project Technical Primitive Diagram

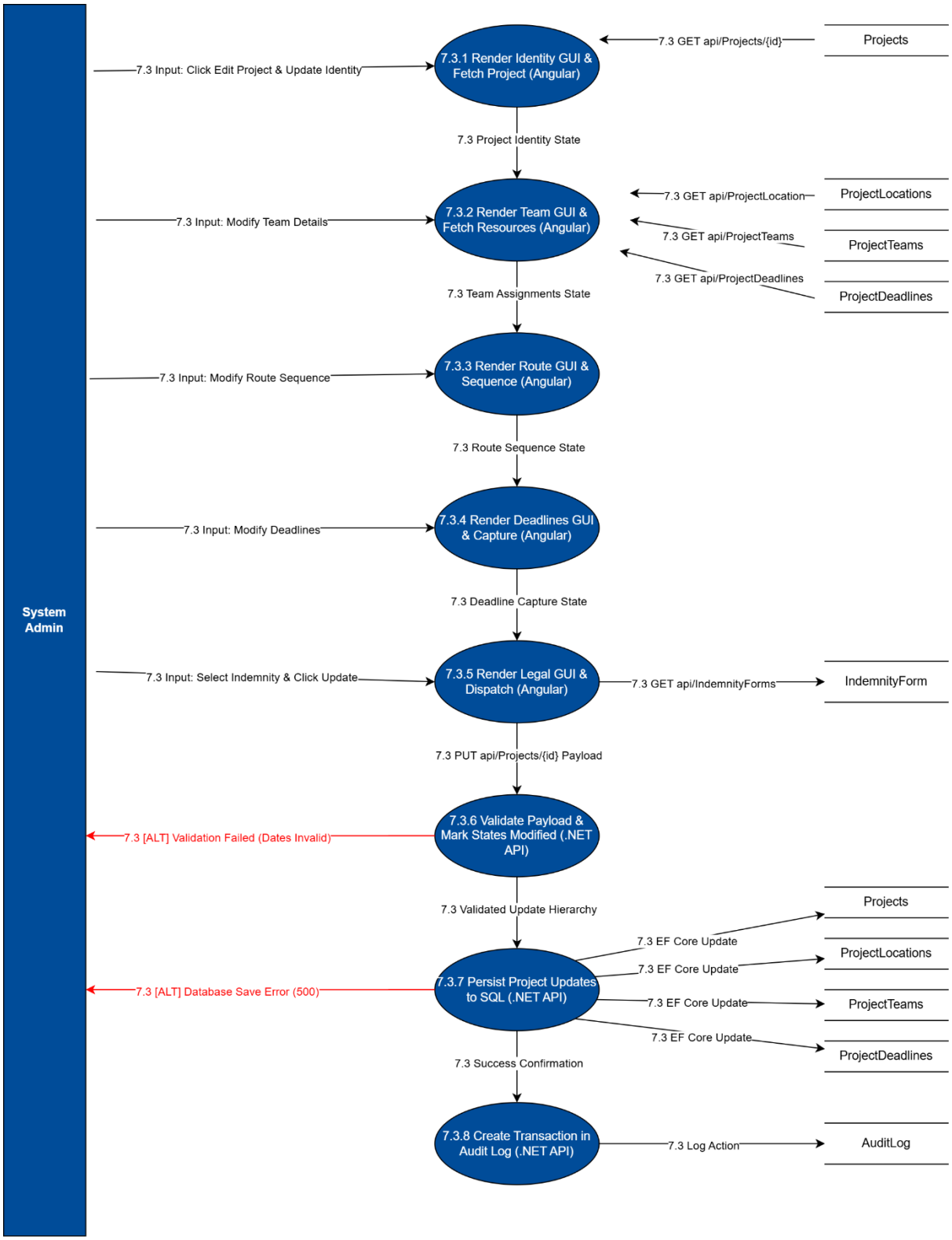


Figure 59: 7.3 Update Project Technical Primitive Diagram

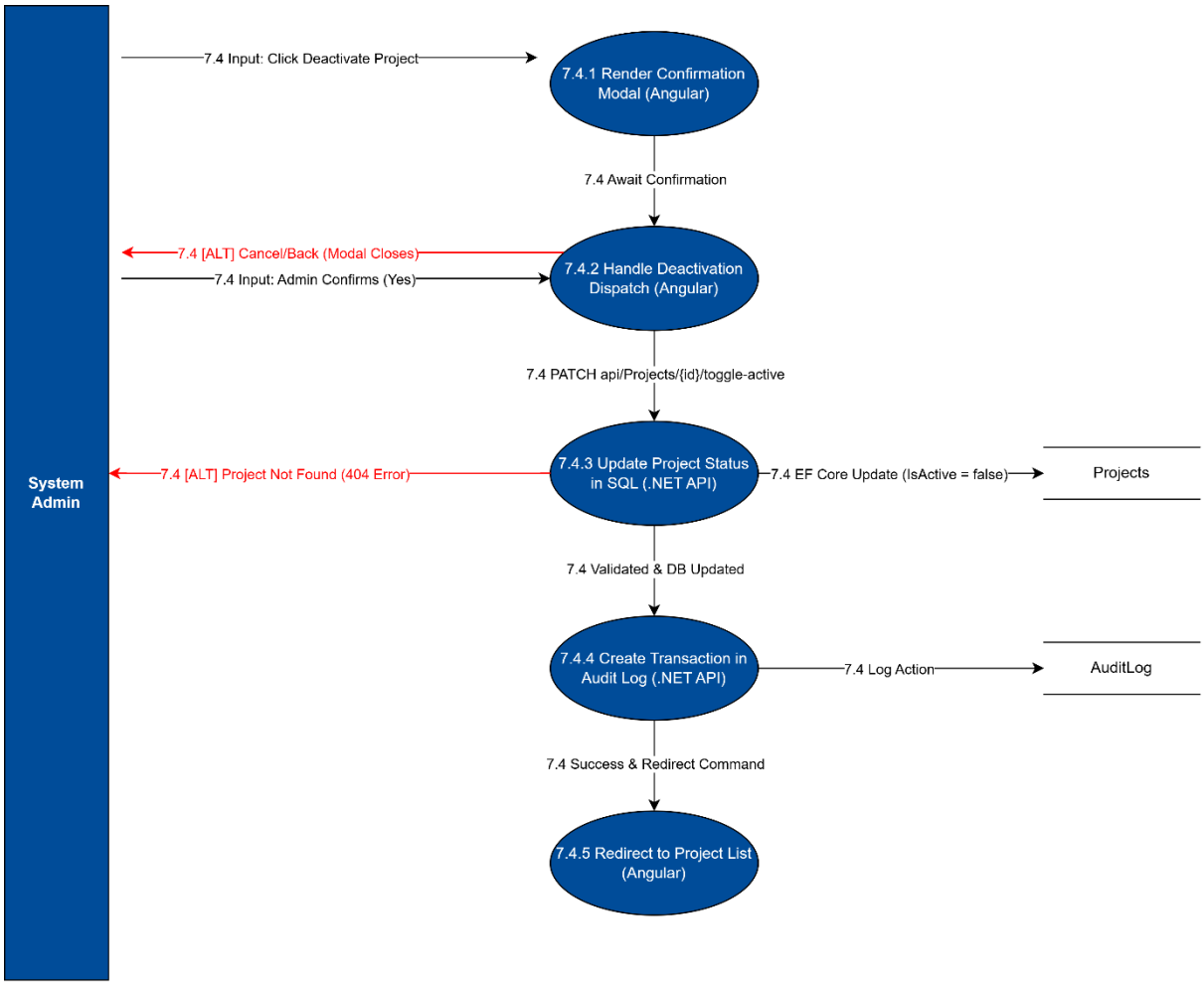


Figure 60: 7.4 Deactivate Project Technical Primitive Diagram

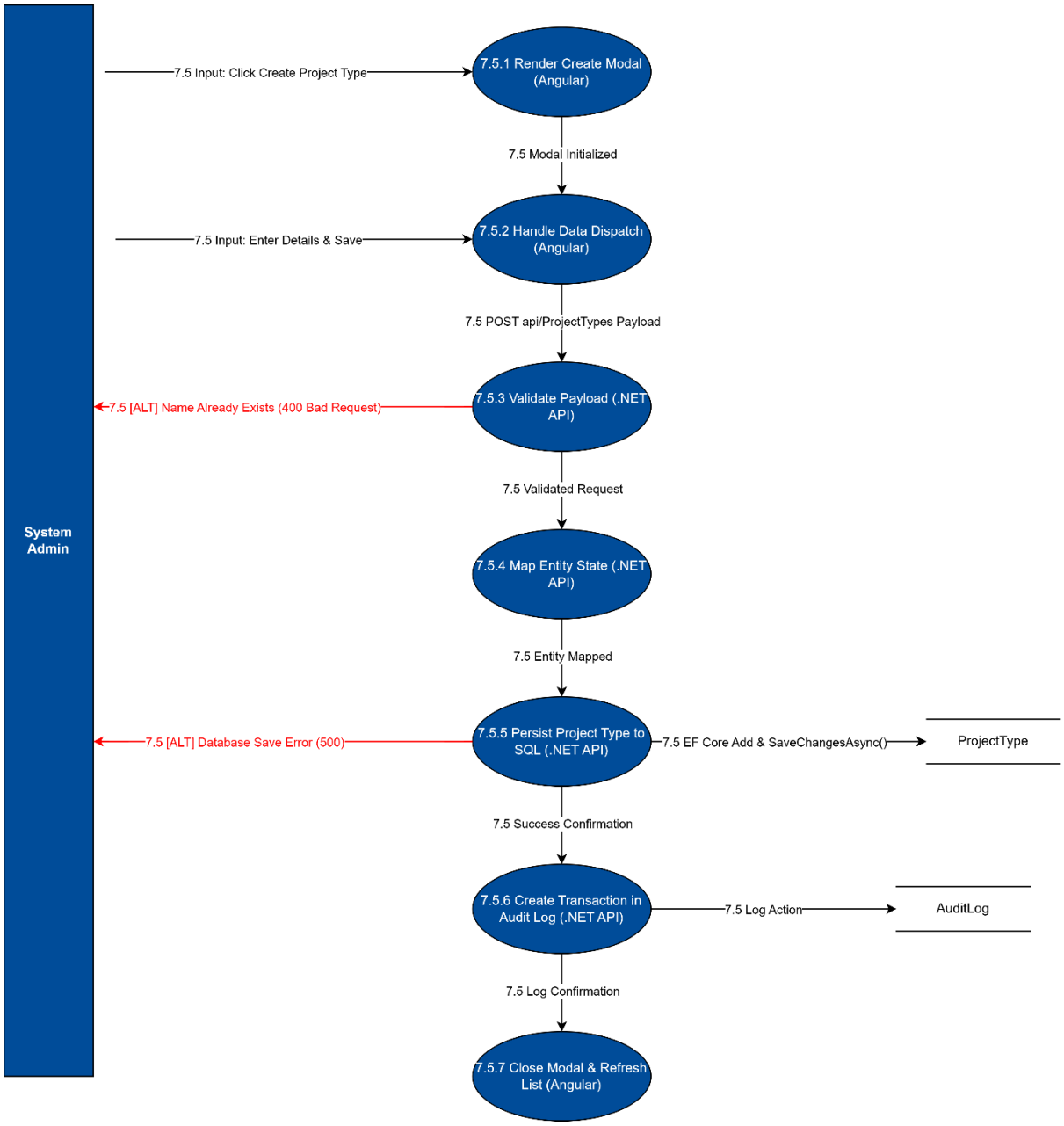


Figure 61: 7.5 Create Project Type Technical Primitive Diagram

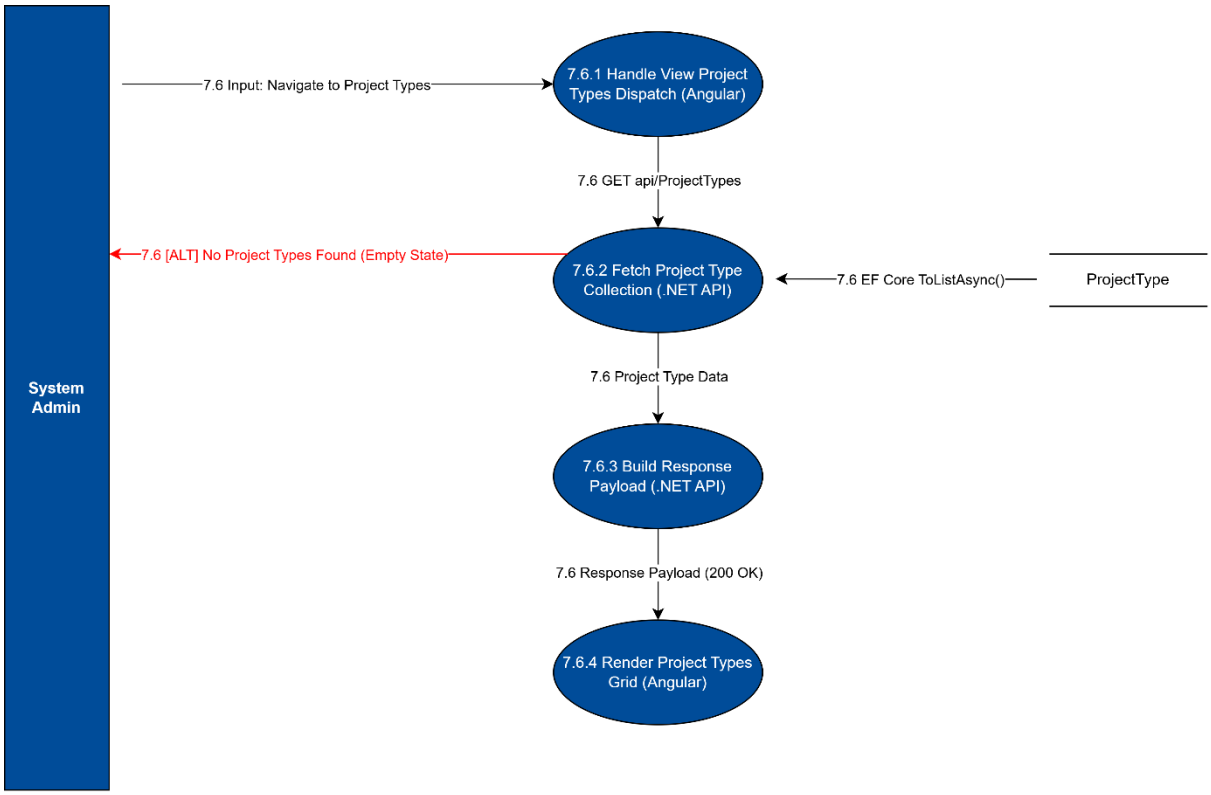


Figure 62: 7.6 View Project Types Technical Primitive Diagram

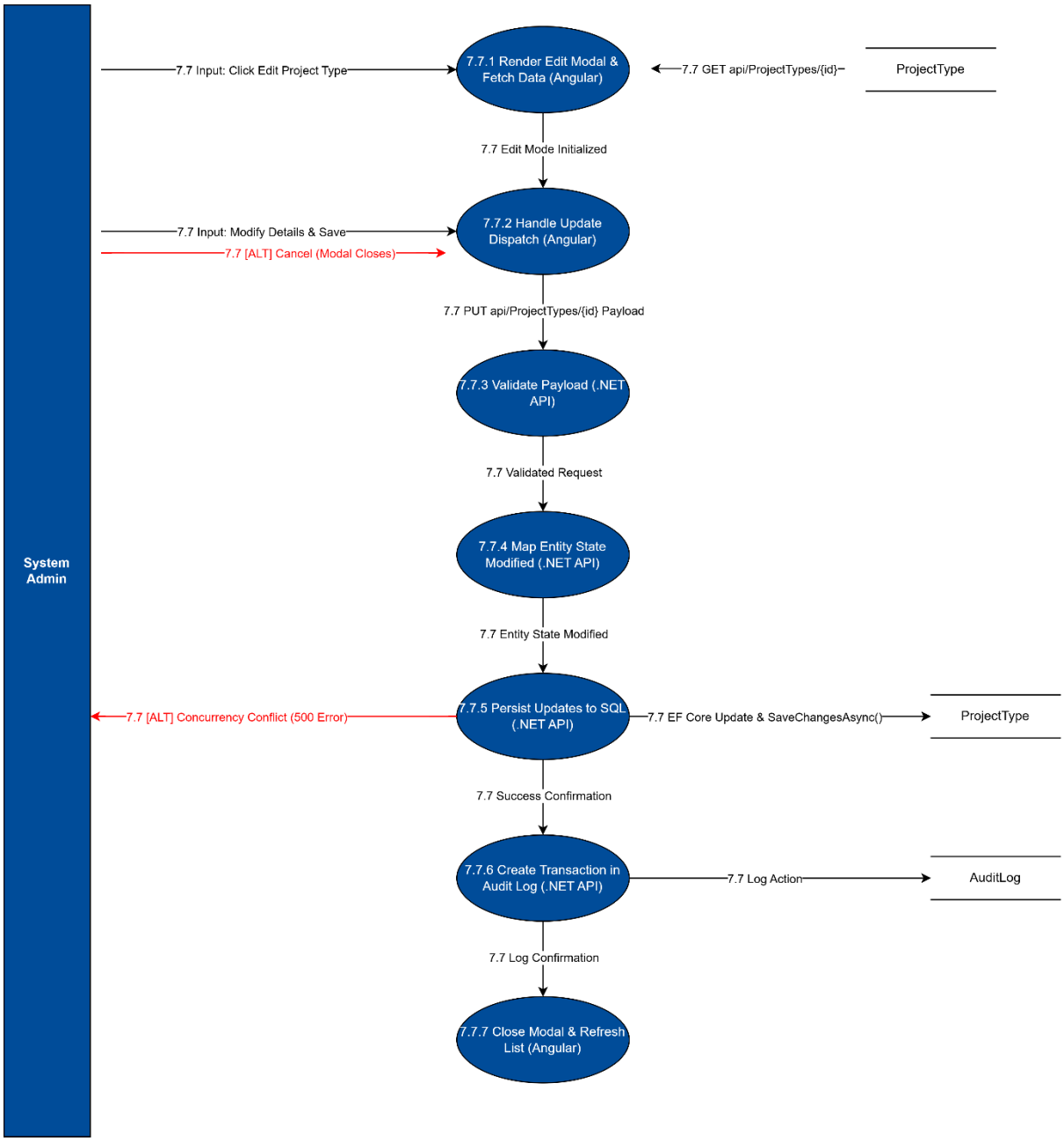


Figure 63: 7.7 Update Project Type Technical Primitive Diagram

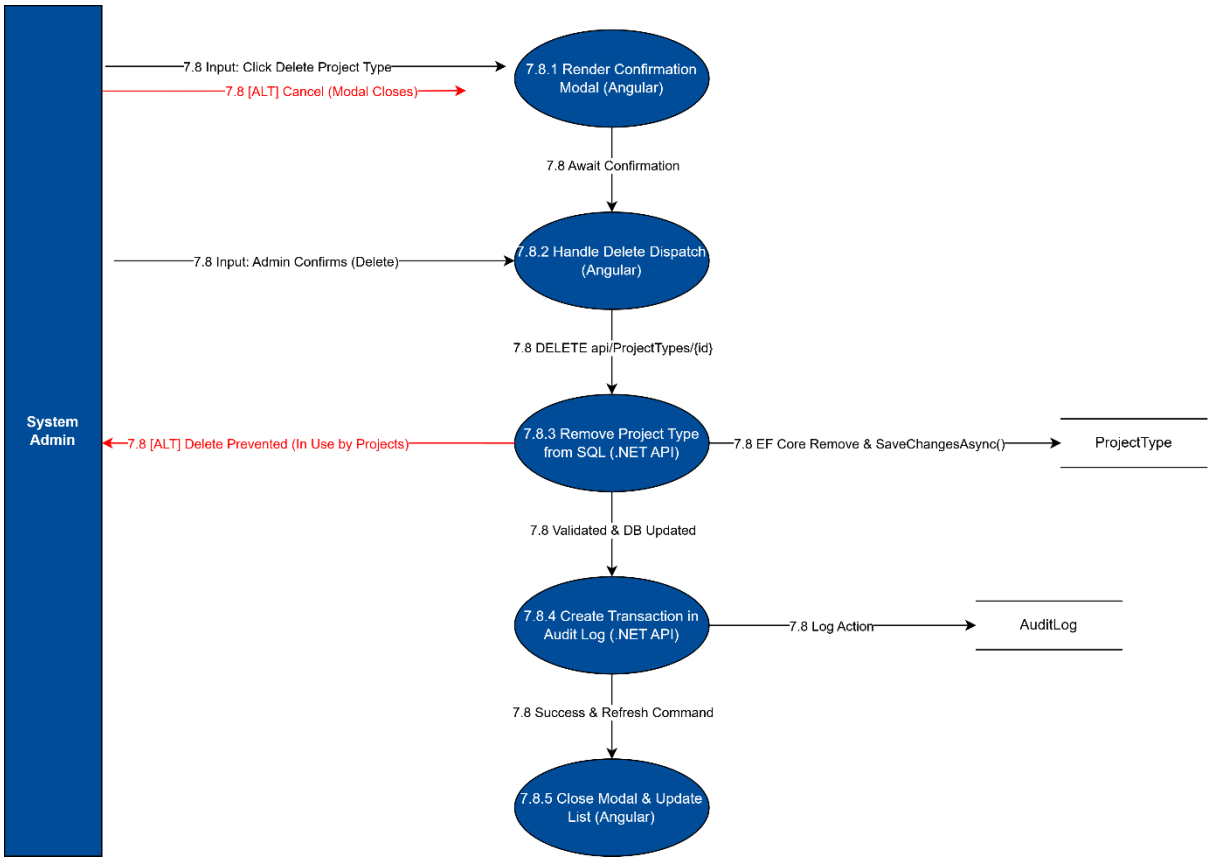


Figure 64: 7.8 Deactivate Project Type Technical Primitive Diagram

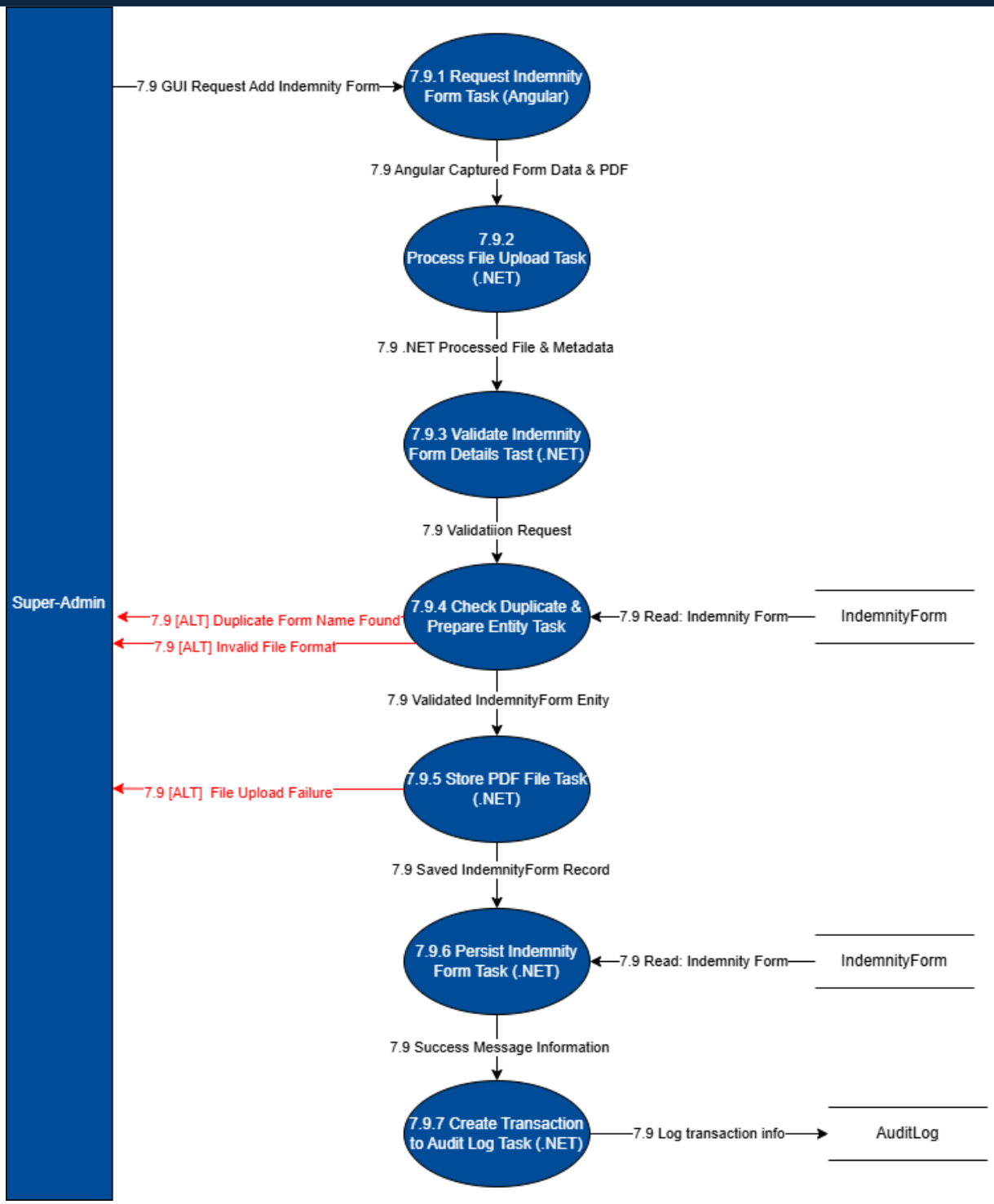


Figure 80: Add Indemnity Form Technical Primitive Diagram

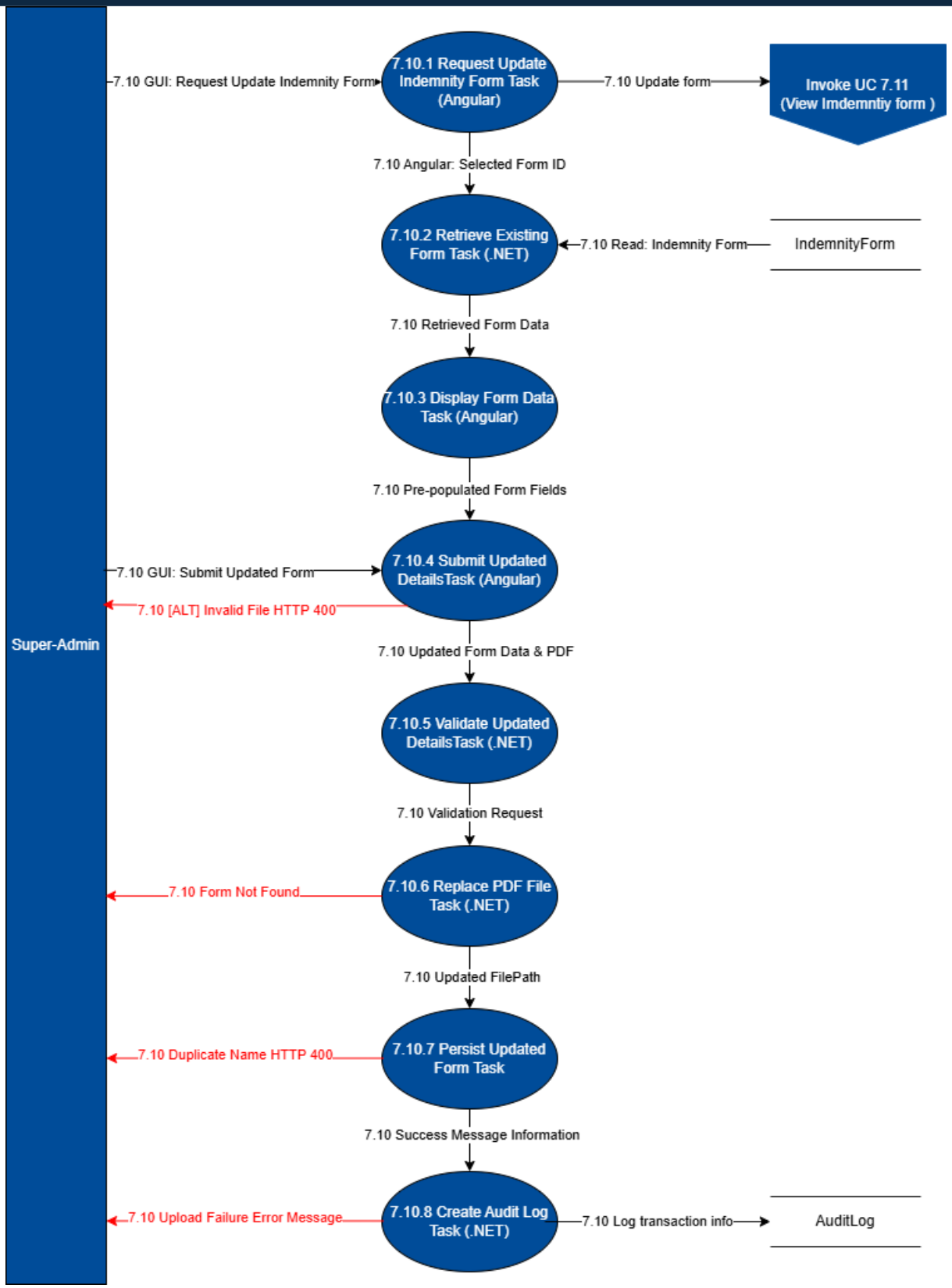


Figure 81: Update Indemnity Form Technical Primitive Diagram

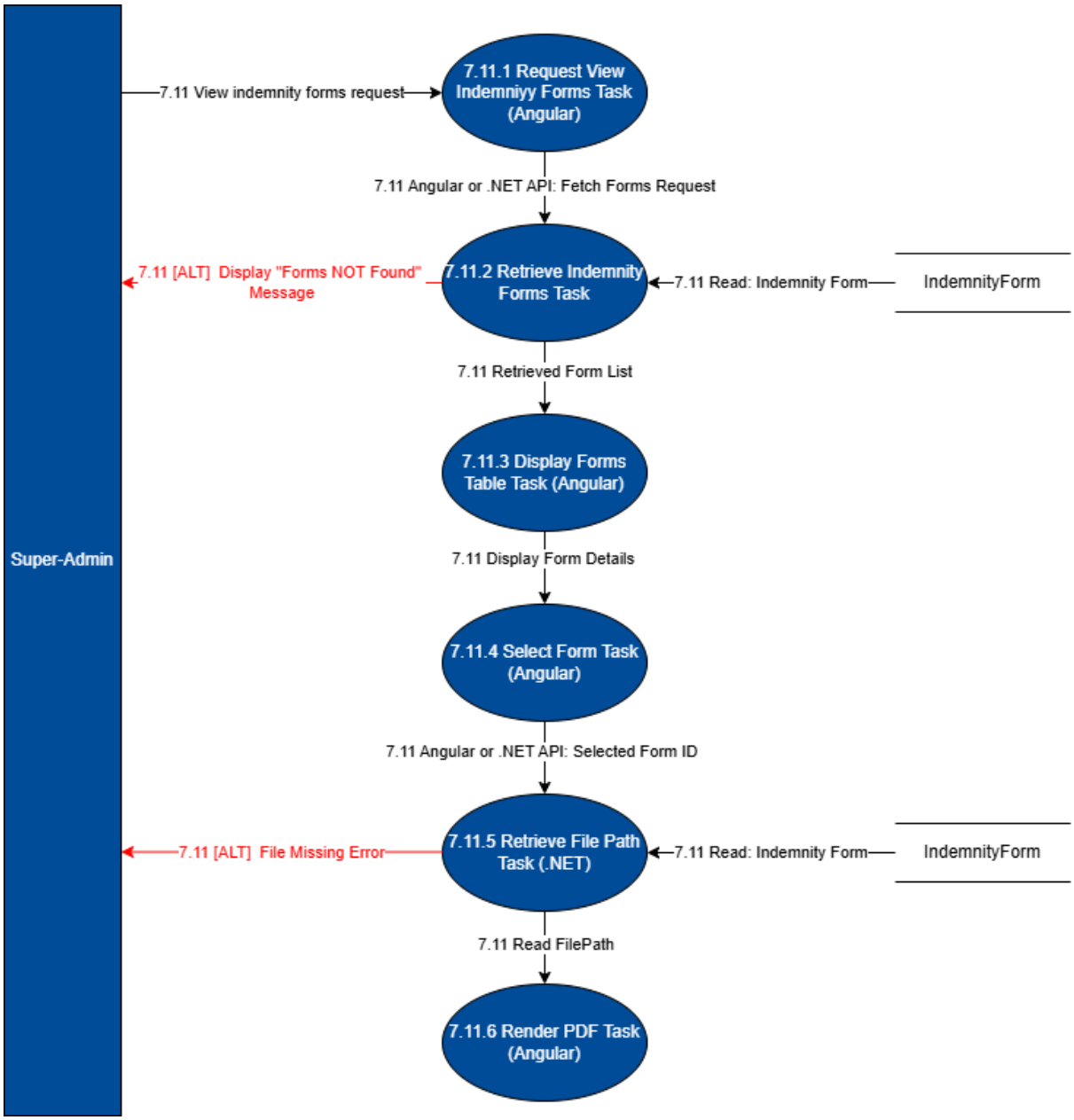


Figure 82: View Indemnity Form Technical Primitive Diagram

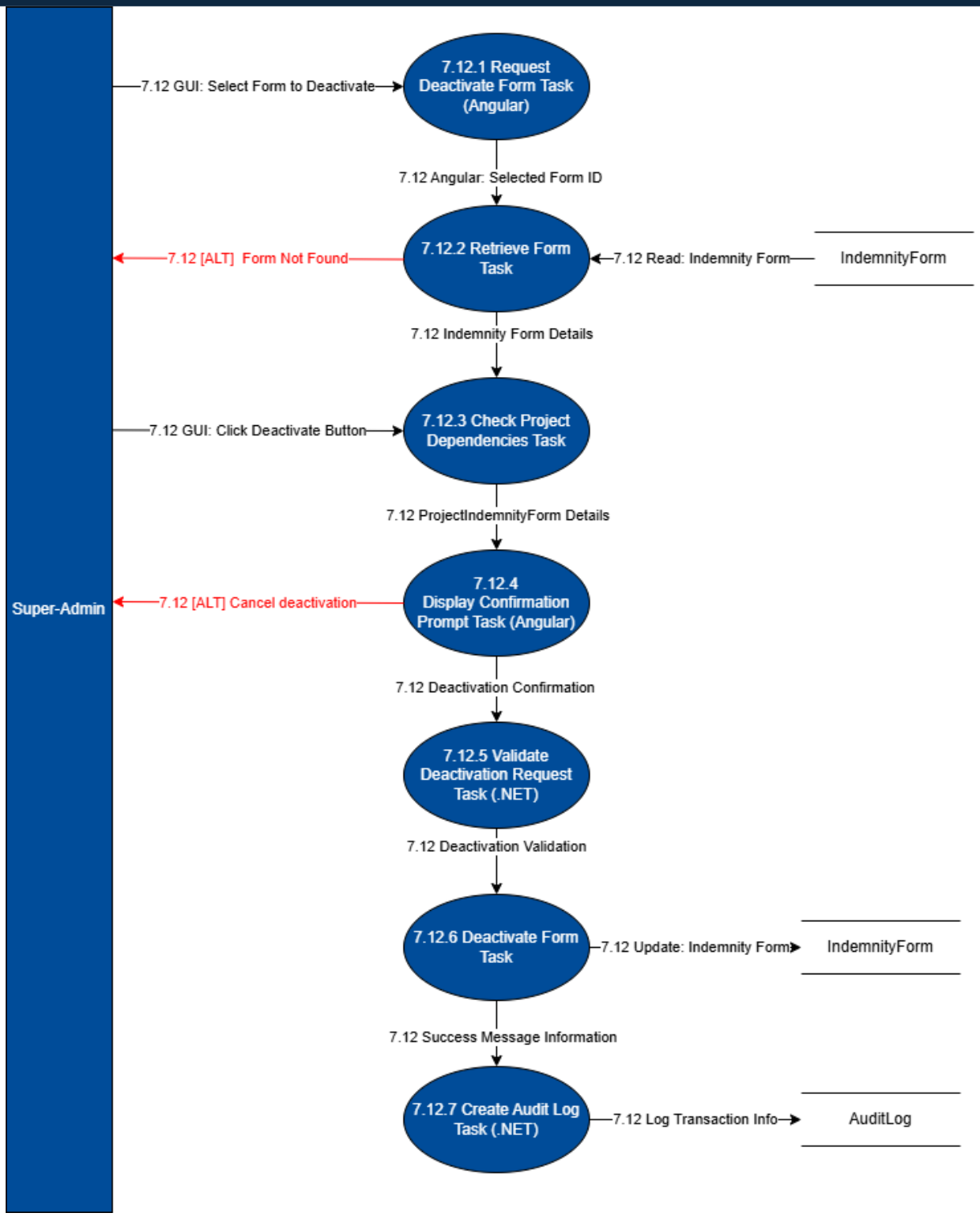


Figure 83: Remove Indemnity Form Technical Primitive Diagram

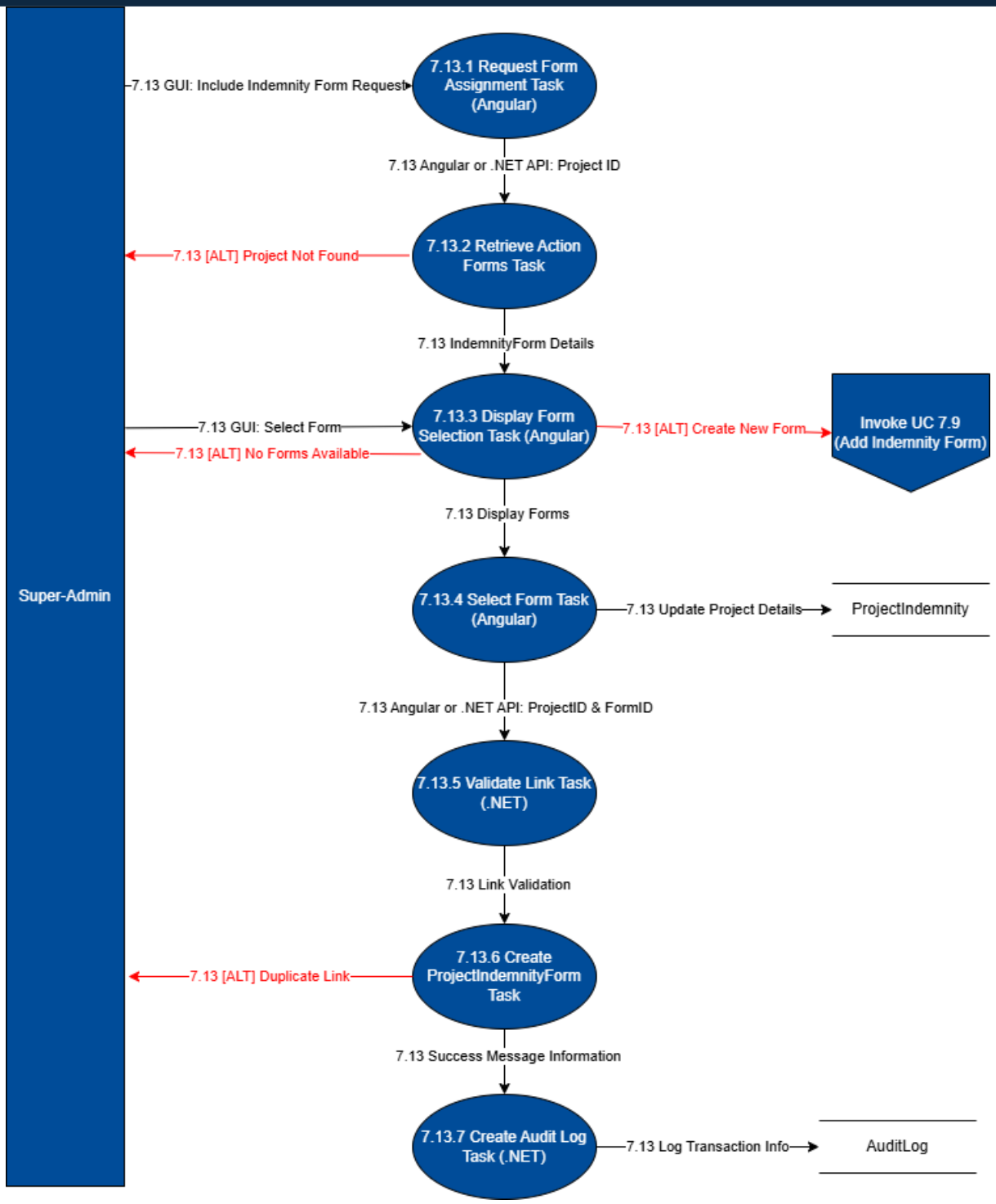


Figure 84: Capture Indemnity Form Technical Primitive Diagram

2.6.4 Message Subsystem Primitive Diagrams

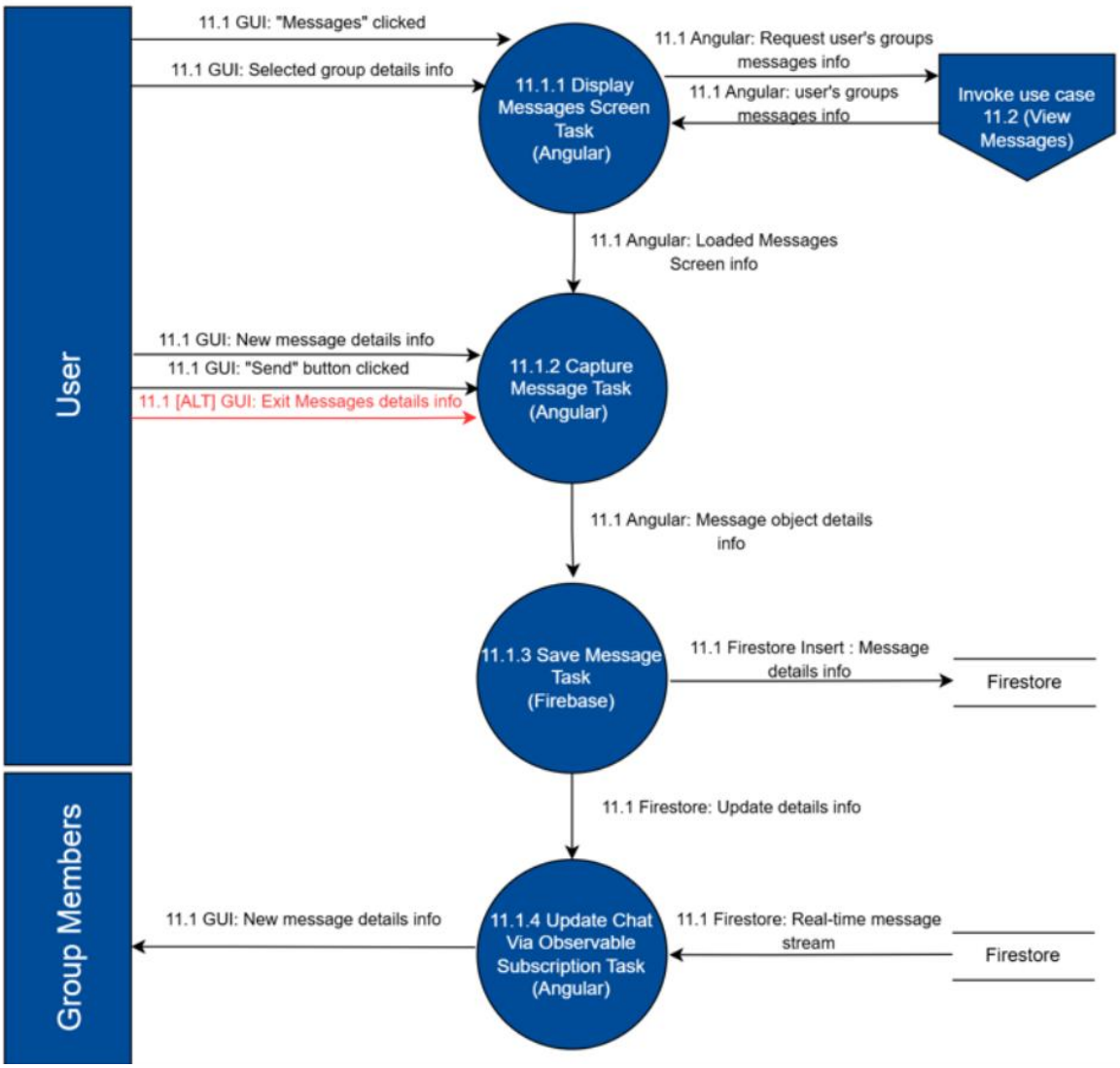


Figure 85: 11.1 Create Message Technical Primitive Diagram

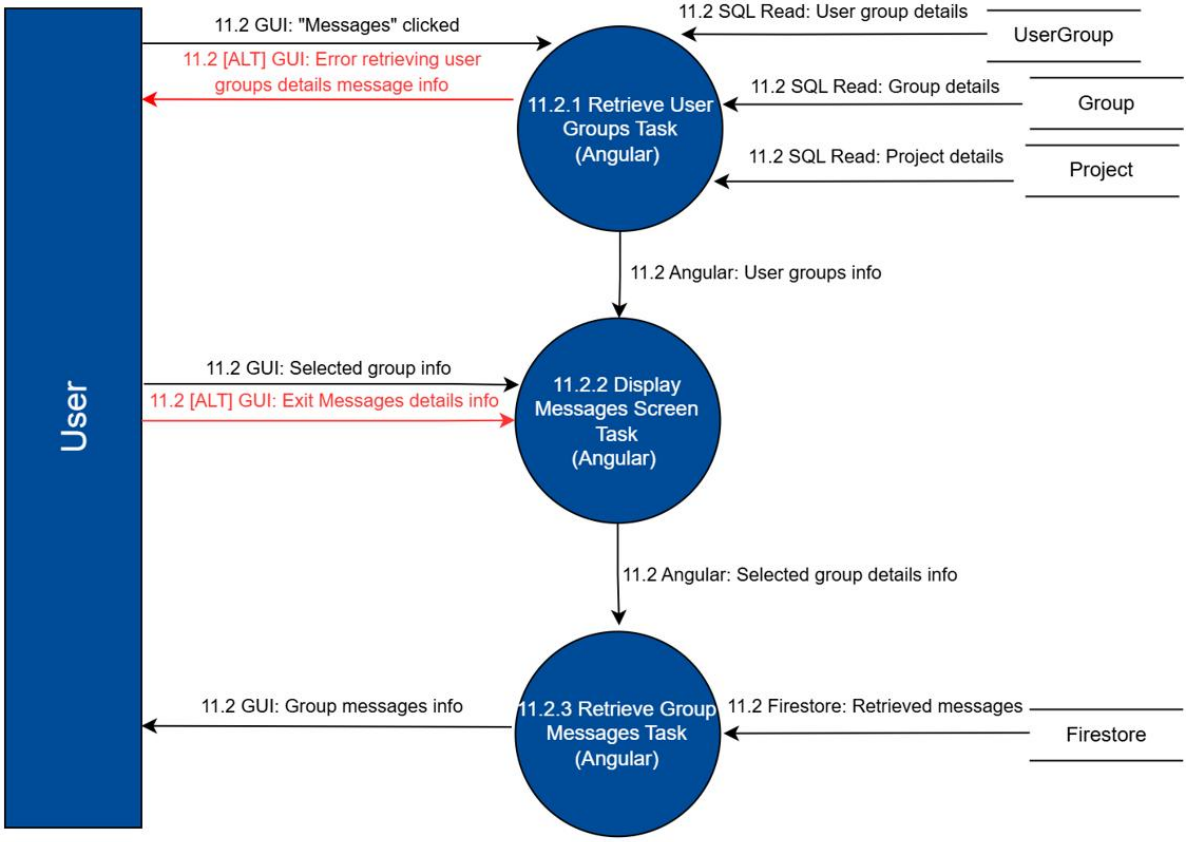


Figure 86: 11.2 View Message Technical Primitive Diagram

2.7 Sequence Diagram

2.7.1 User Subsystem Sequence Diagrams

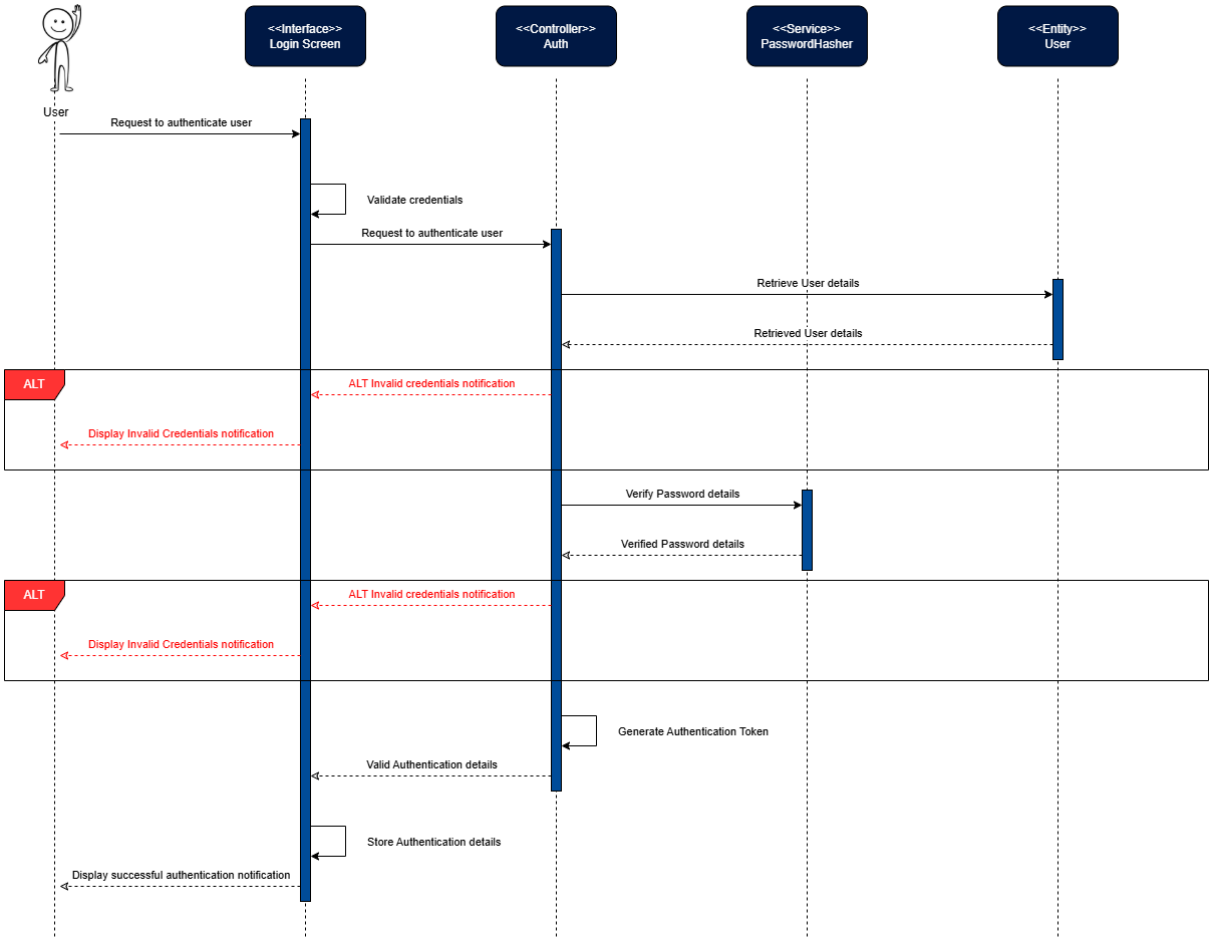


Figure 87: 1.1 User Login Sequence Diagram

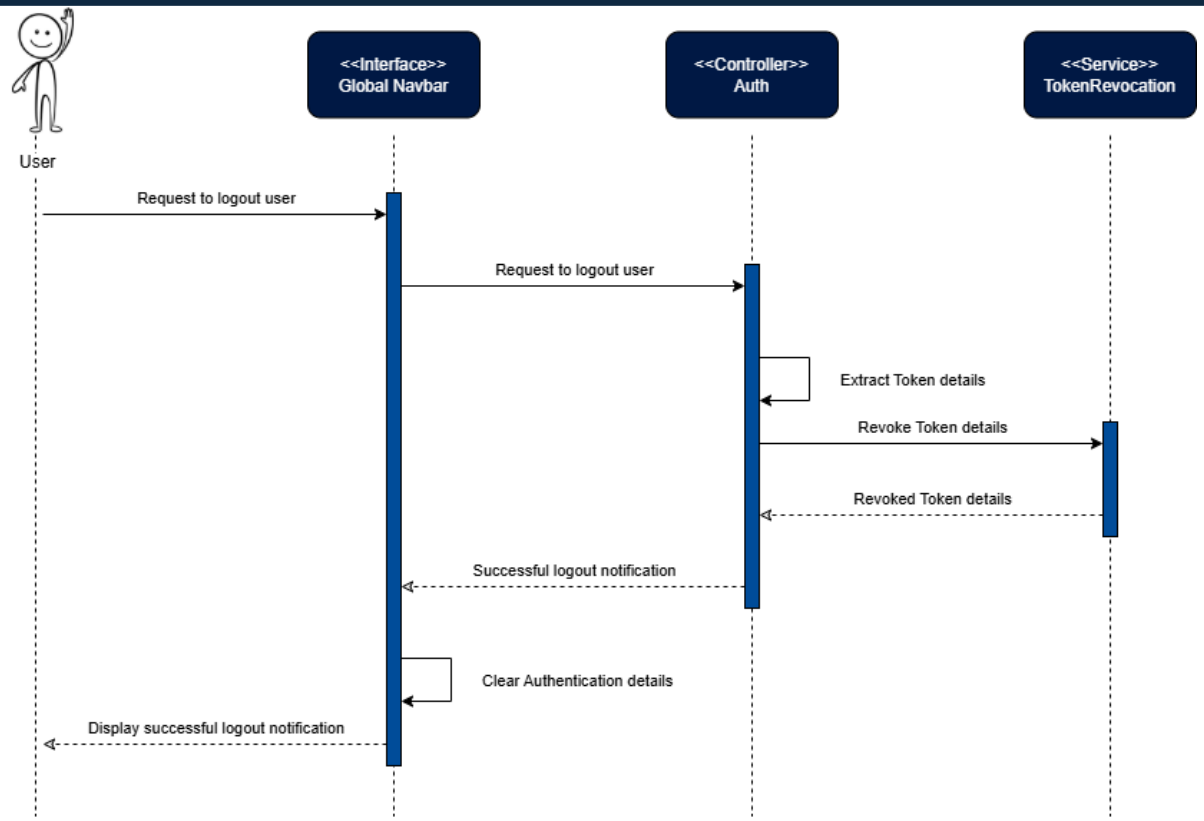


Figure 88: 1.2 User Logout Sequence Diagram

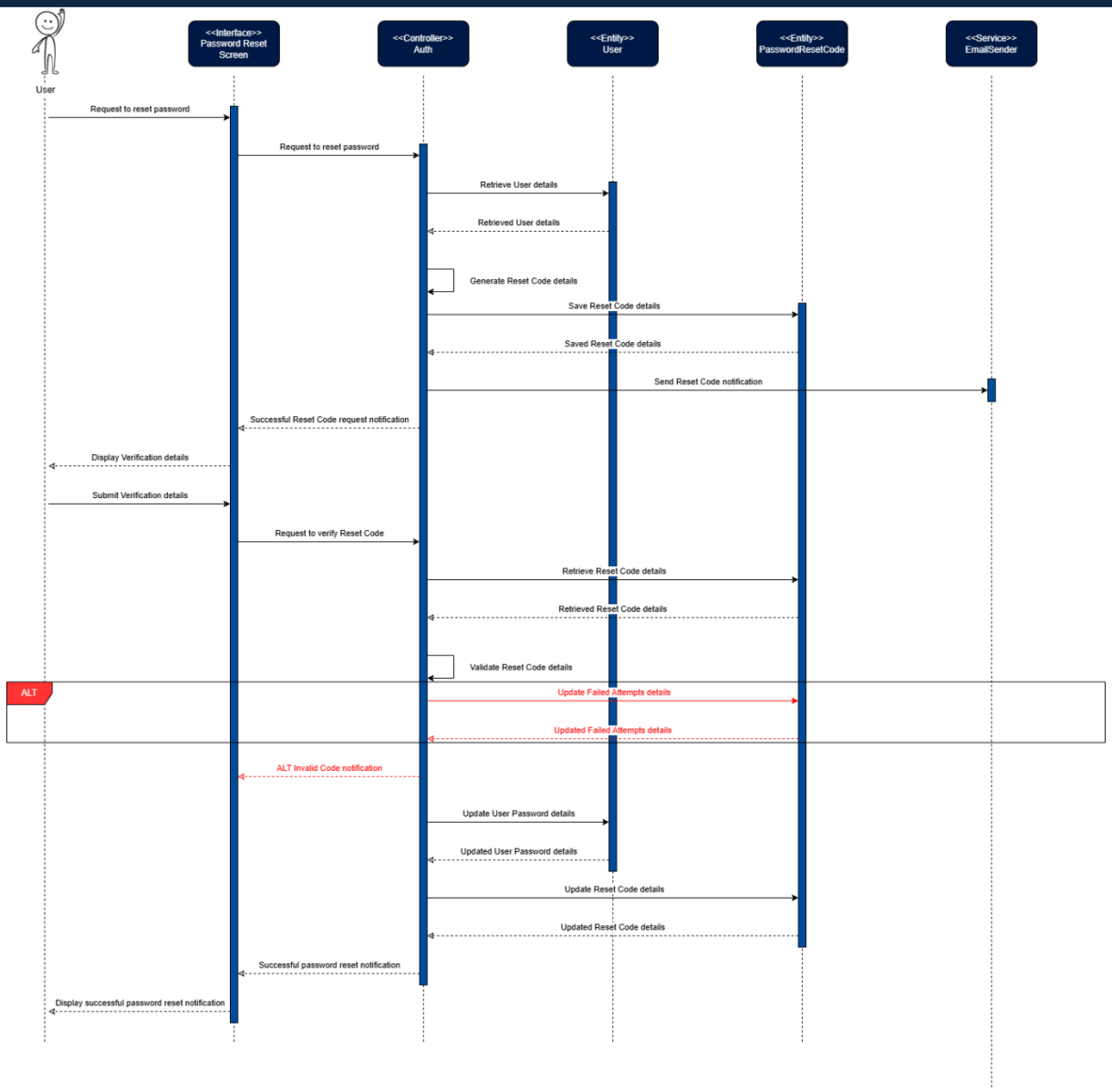


Figure 89: 1.3 Password Reset Sequence Diagram

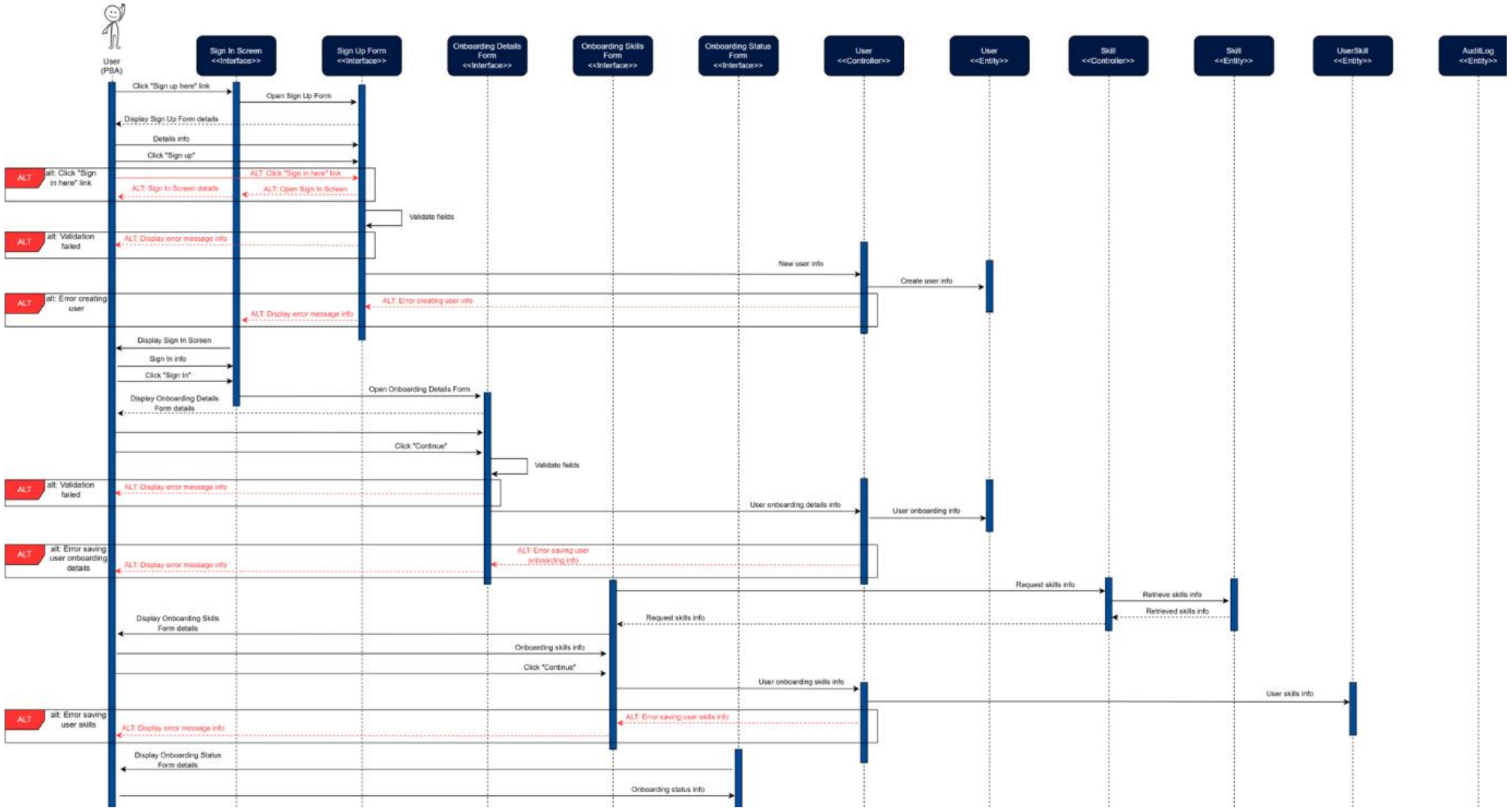


Figure 90: 1.4 Create User Sequence Diagram

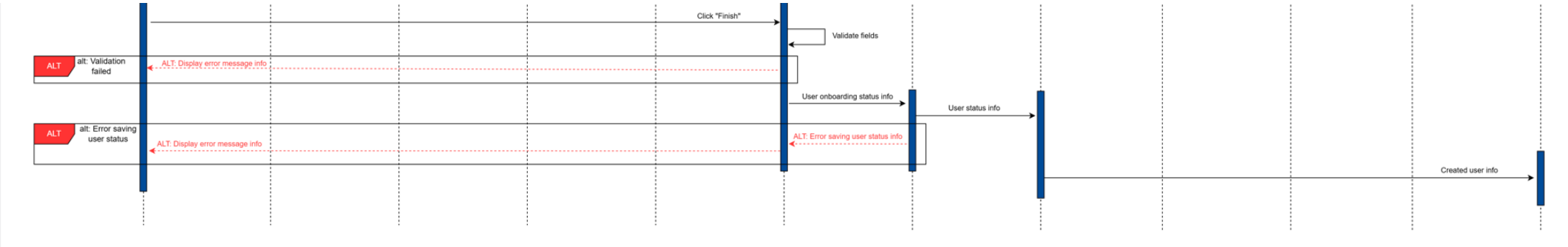


Figure 91: 1.4 Create User Sequence Diagram (Continued)

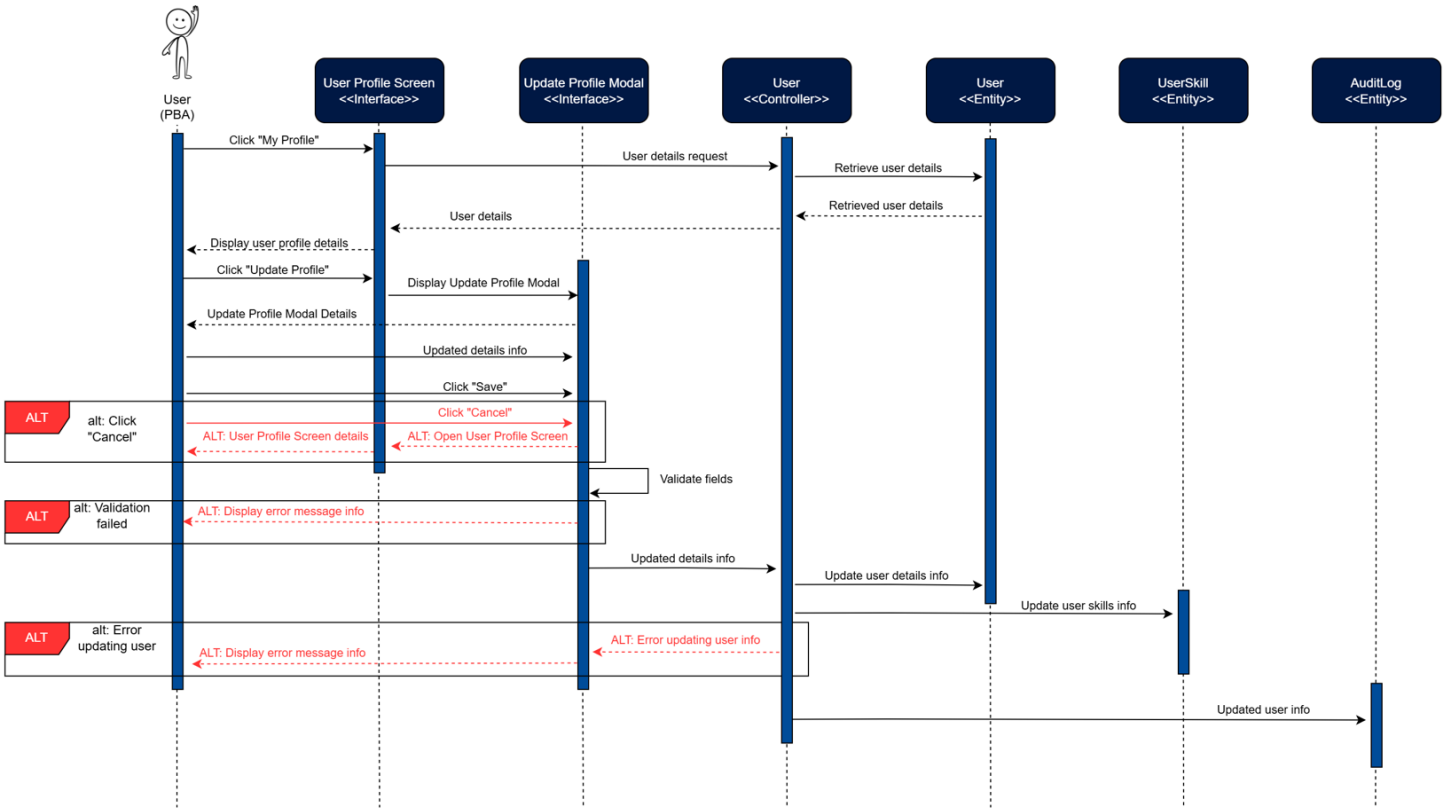


Figure 92: 1.5 Update User Sequence Diagram

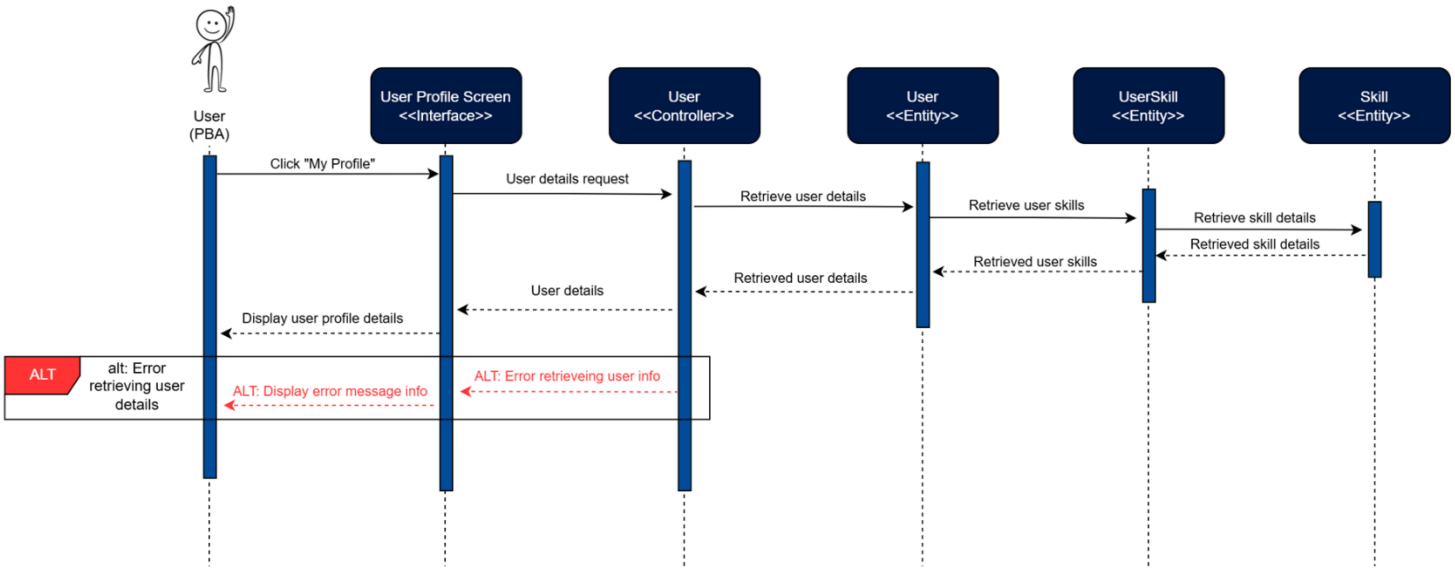


Figure 93: 1.6 View User Sequence Diagram

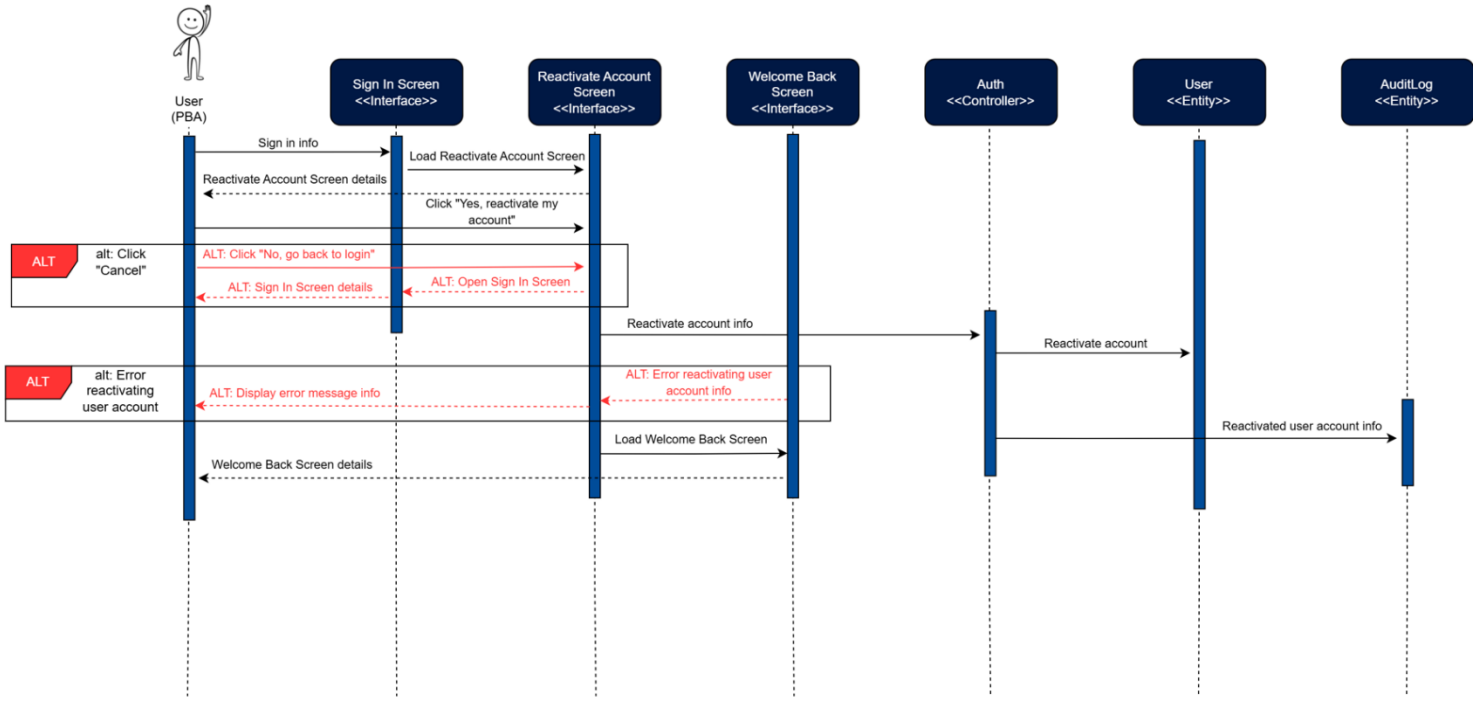


Figure 94: 1.7 Reactivate User Sequence Diagram

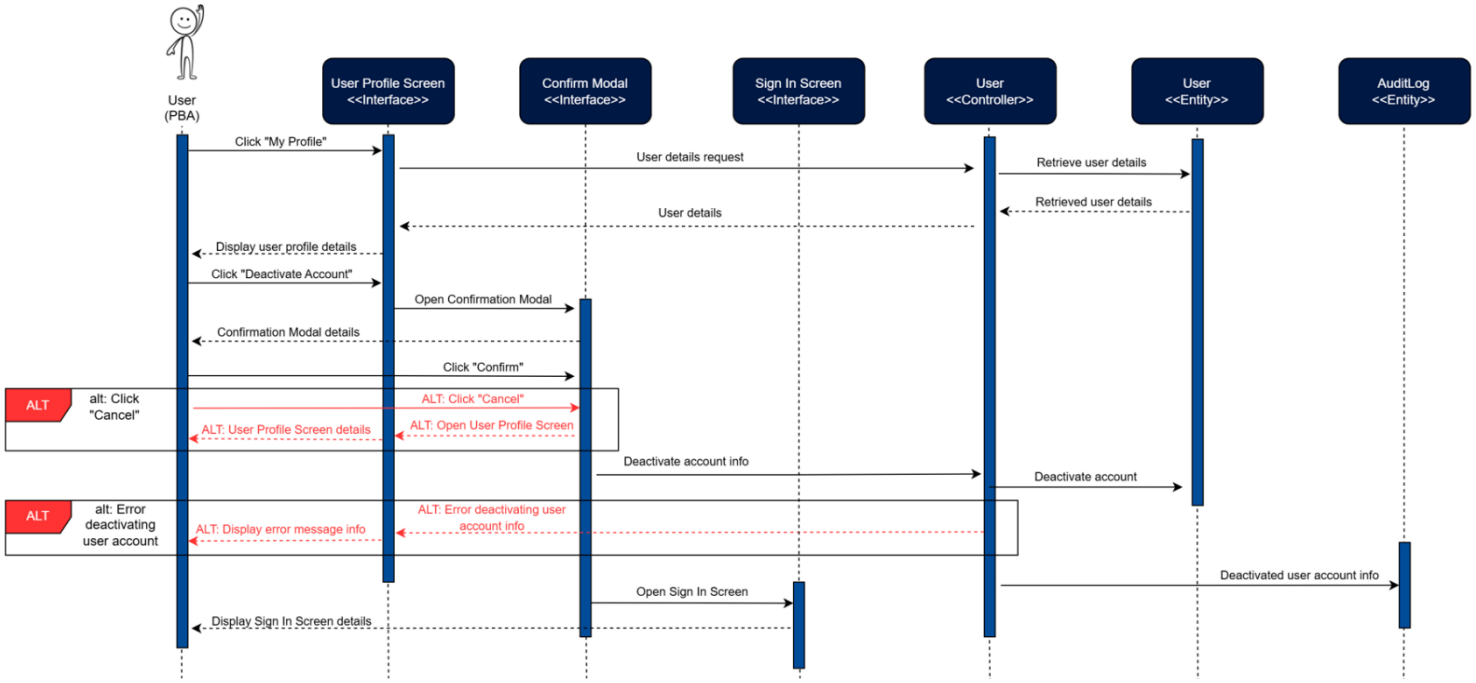


Figure 95: 1.8 Deactivate User Sequence Diagram

2.7.2 Admin Subsystem Sequence Diagrams

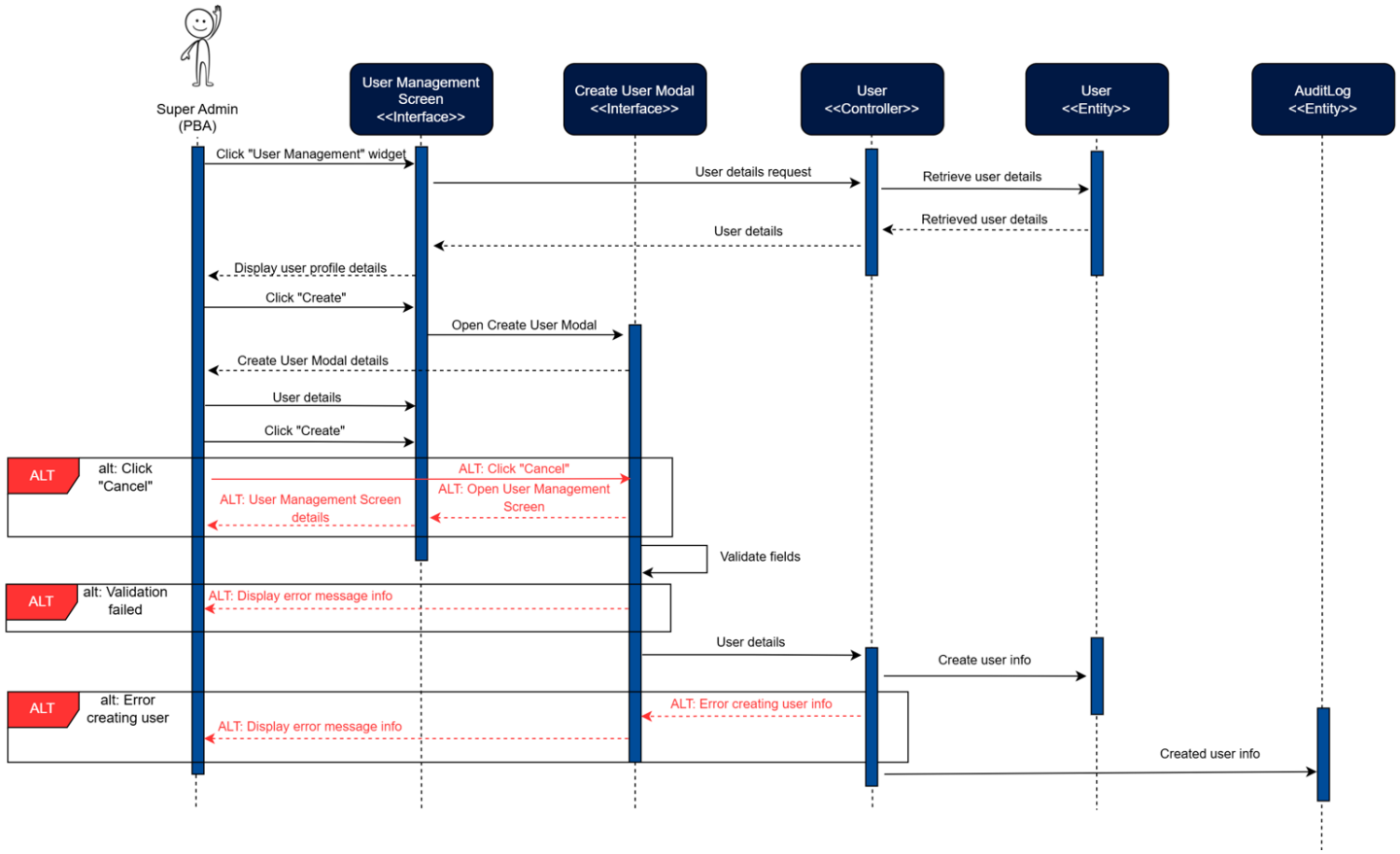


Figure 96: 2.1 Create User Profile Sequence Diagram

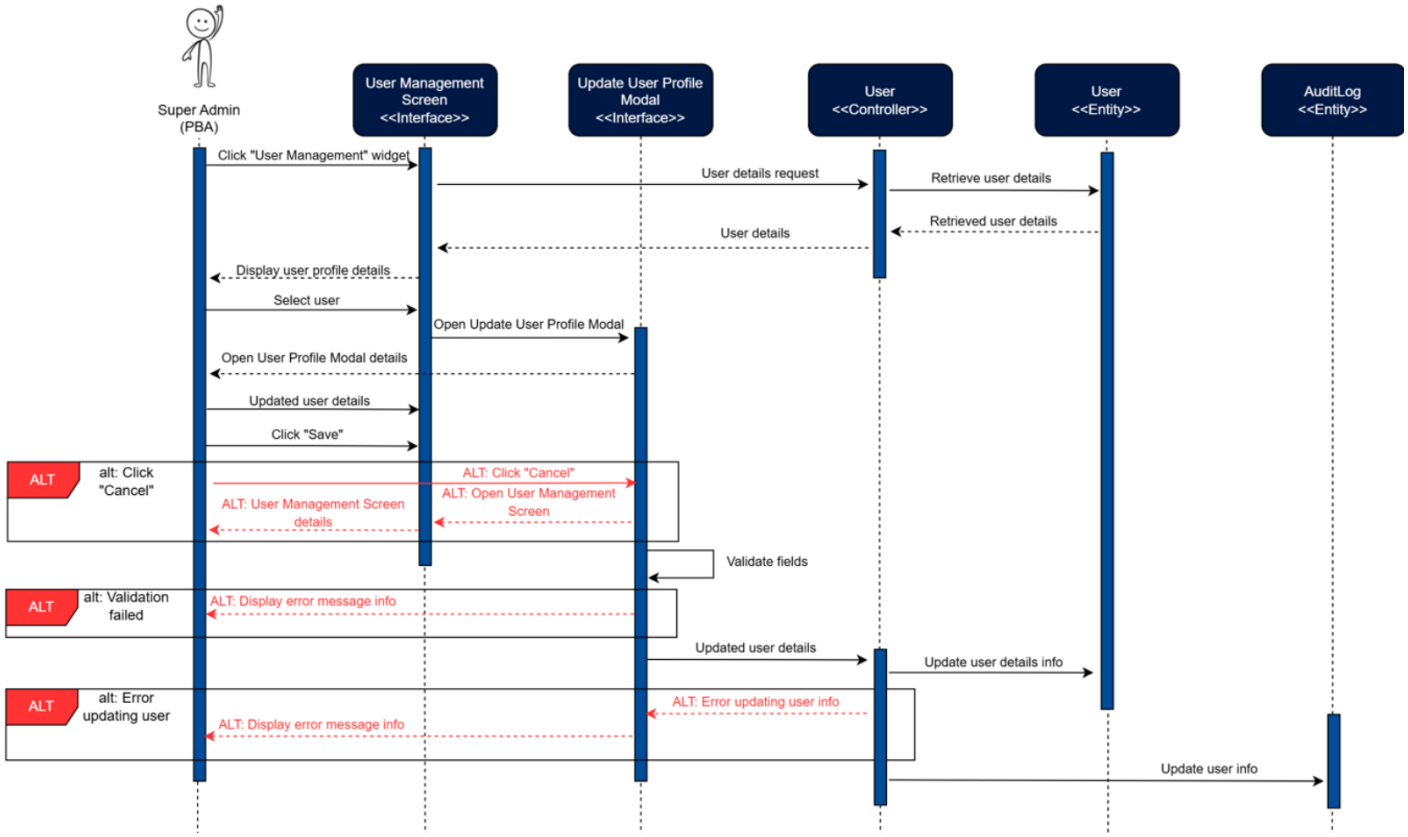


Figure 97: 2.2 Update User Profile Sequence Diagram

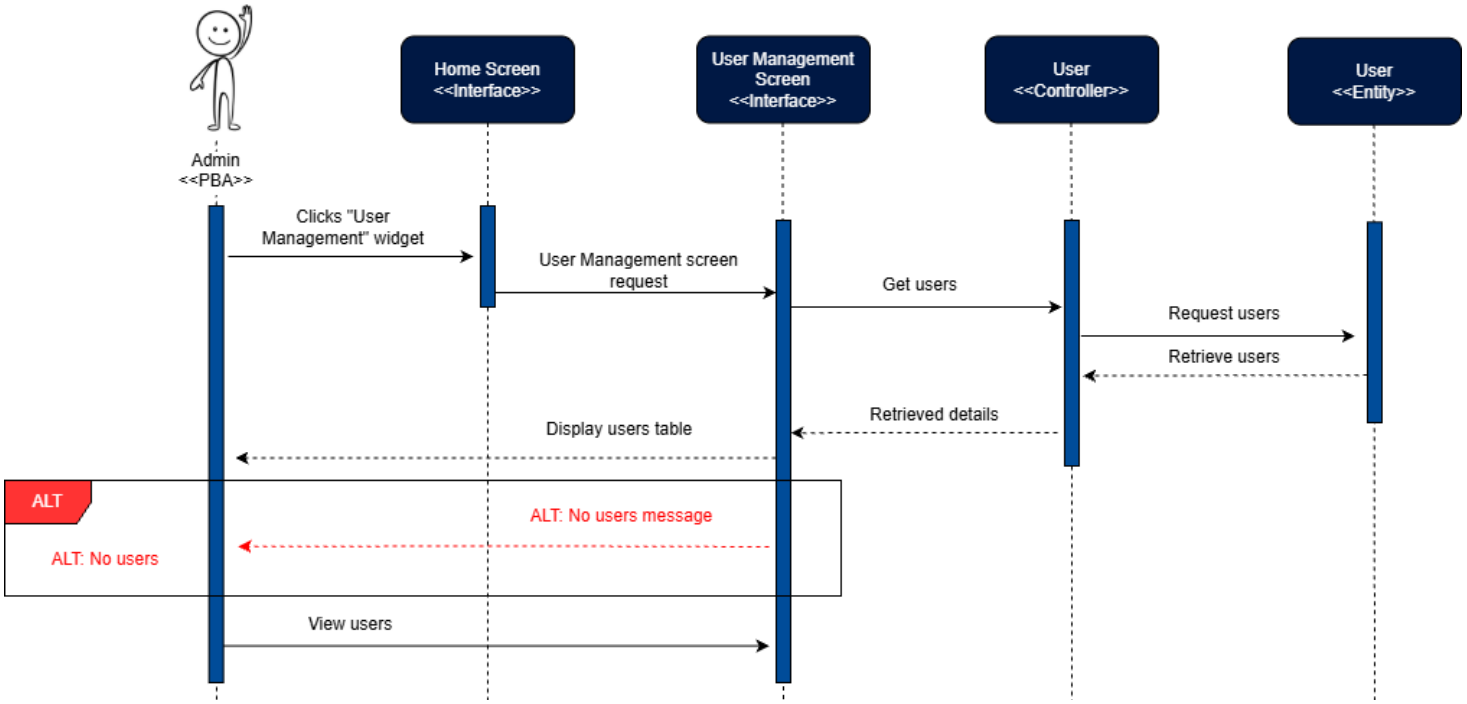


Figure 98: 2.3 View User Profile Sequence Diagram

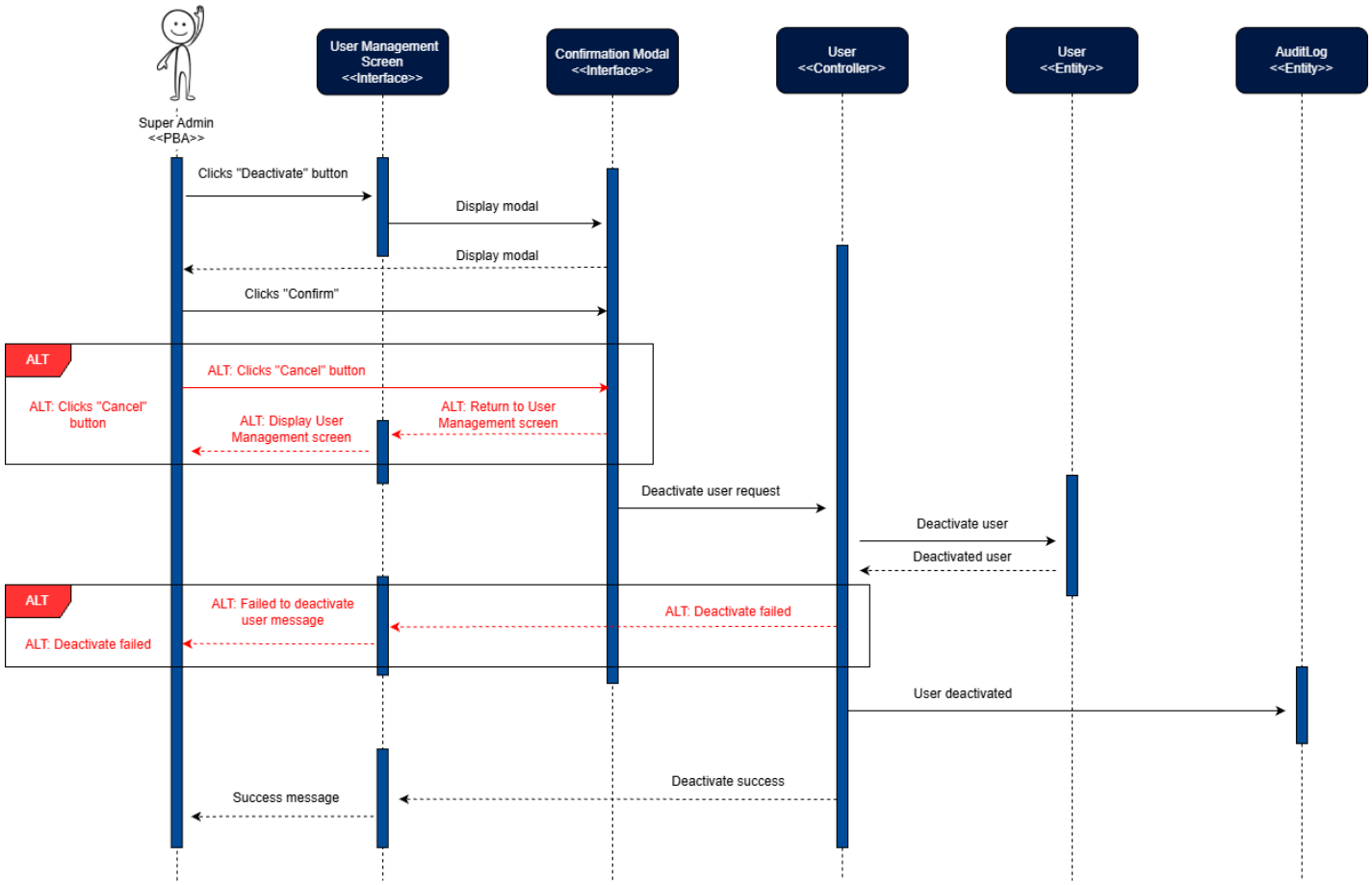


Figure 99: 2.4 Deactivate User Sequence Diagram

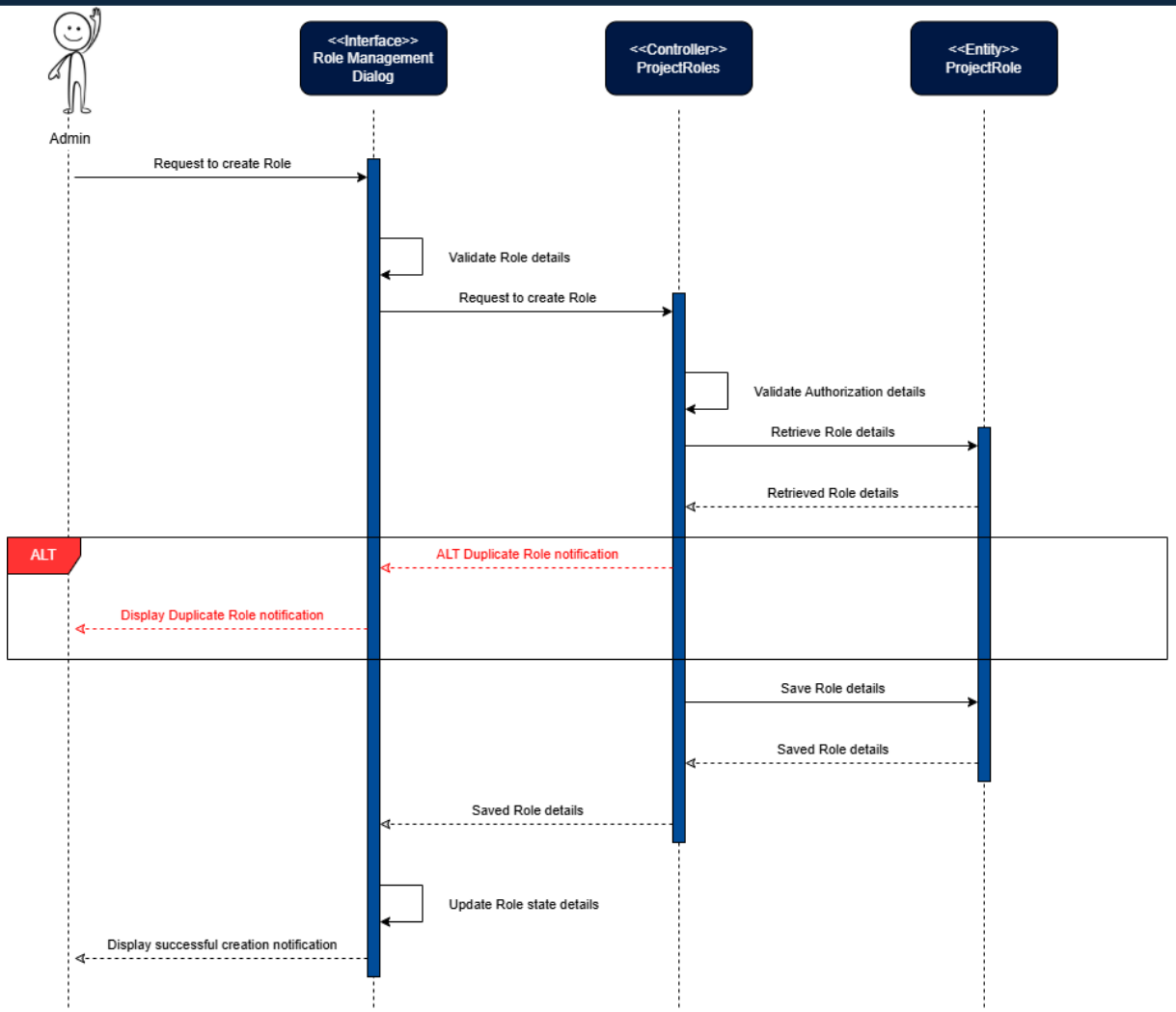


Figure 100:2.8 Create Role Sequence Diagram

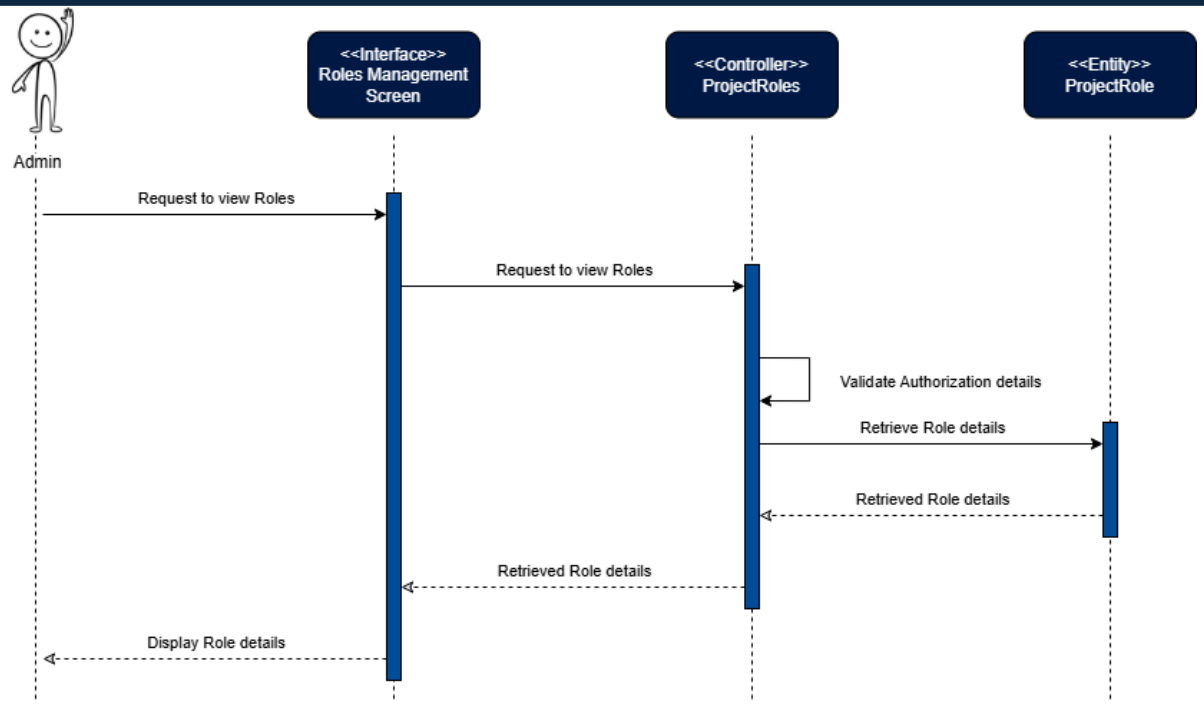


Figure 101: 2.9 View Role Sequence Diagram

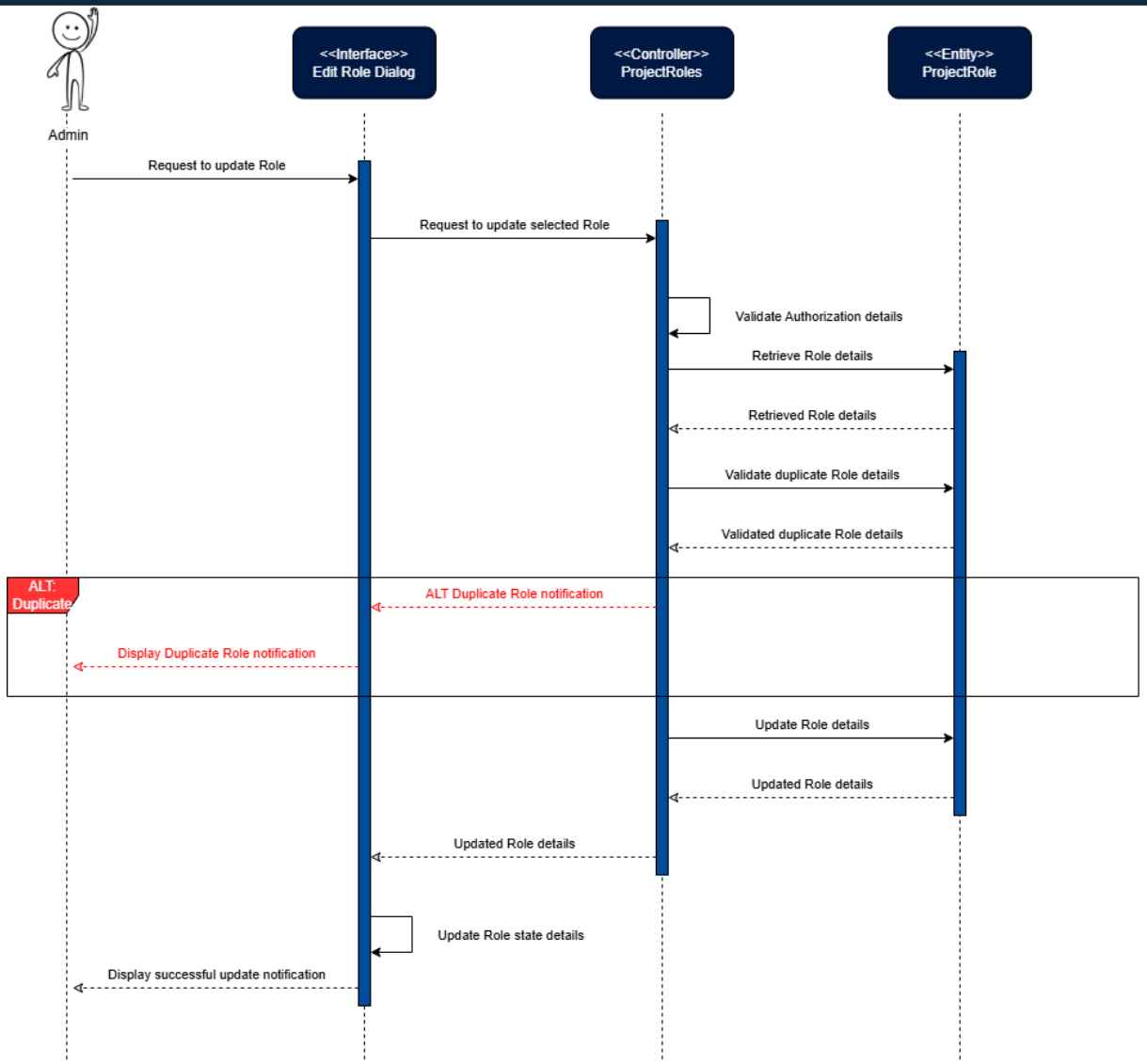


Figure 102: 2.10 Update Role Sequence Diagram

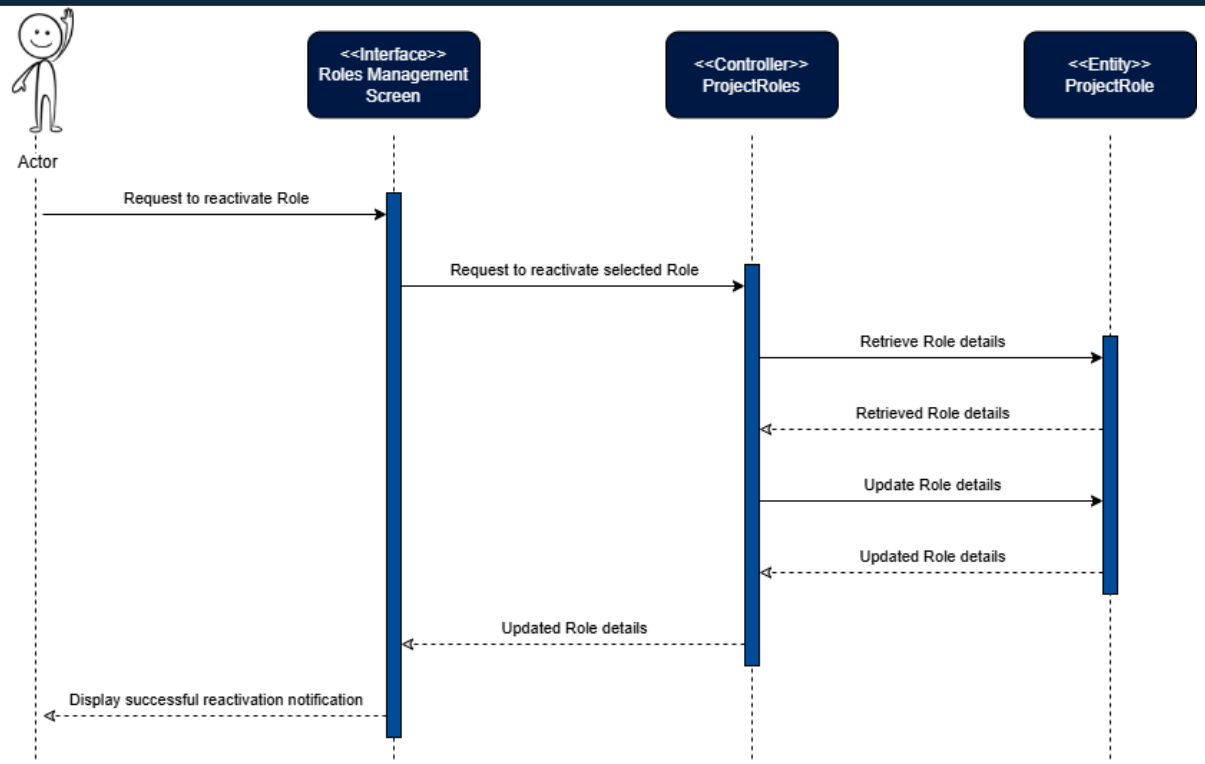


Figure 103: 2.11 Reactivate Role Sequence Diagram

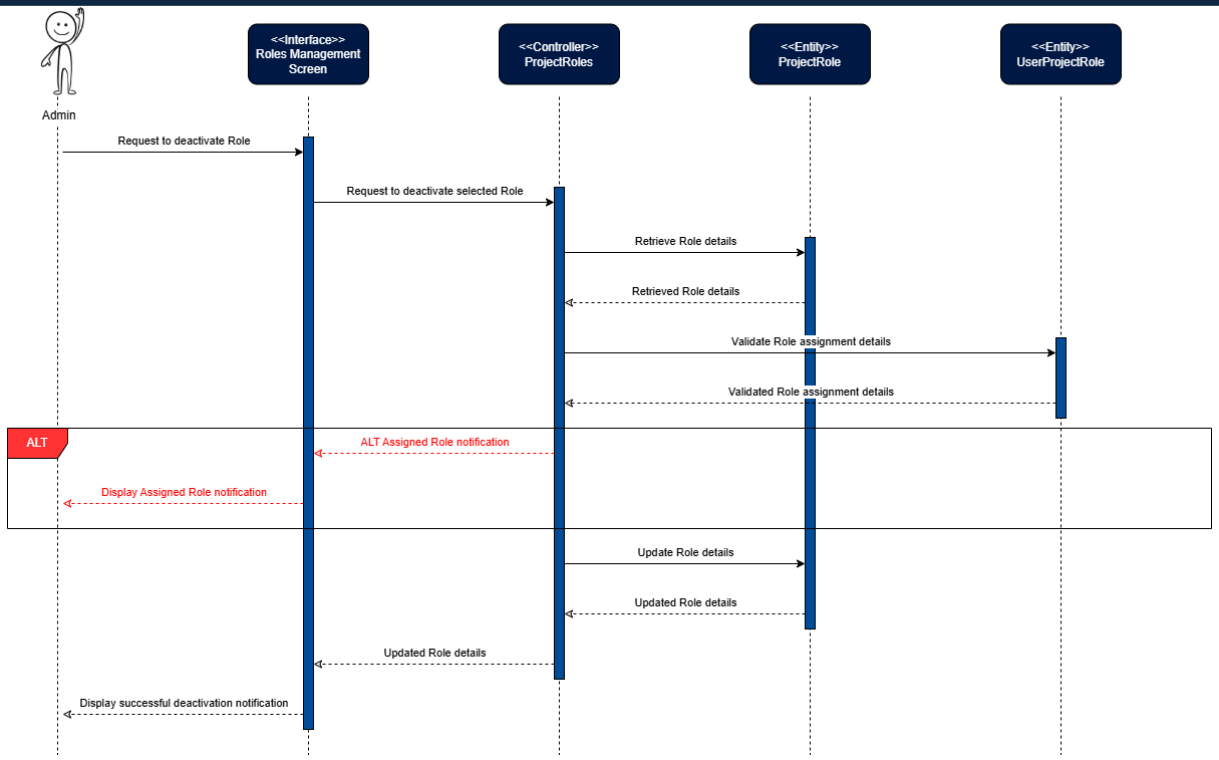


Figure 104: 2.12 Deactivate Role Sequence Diagram

2.7.3 Group Subsystem Sequence Diagrams

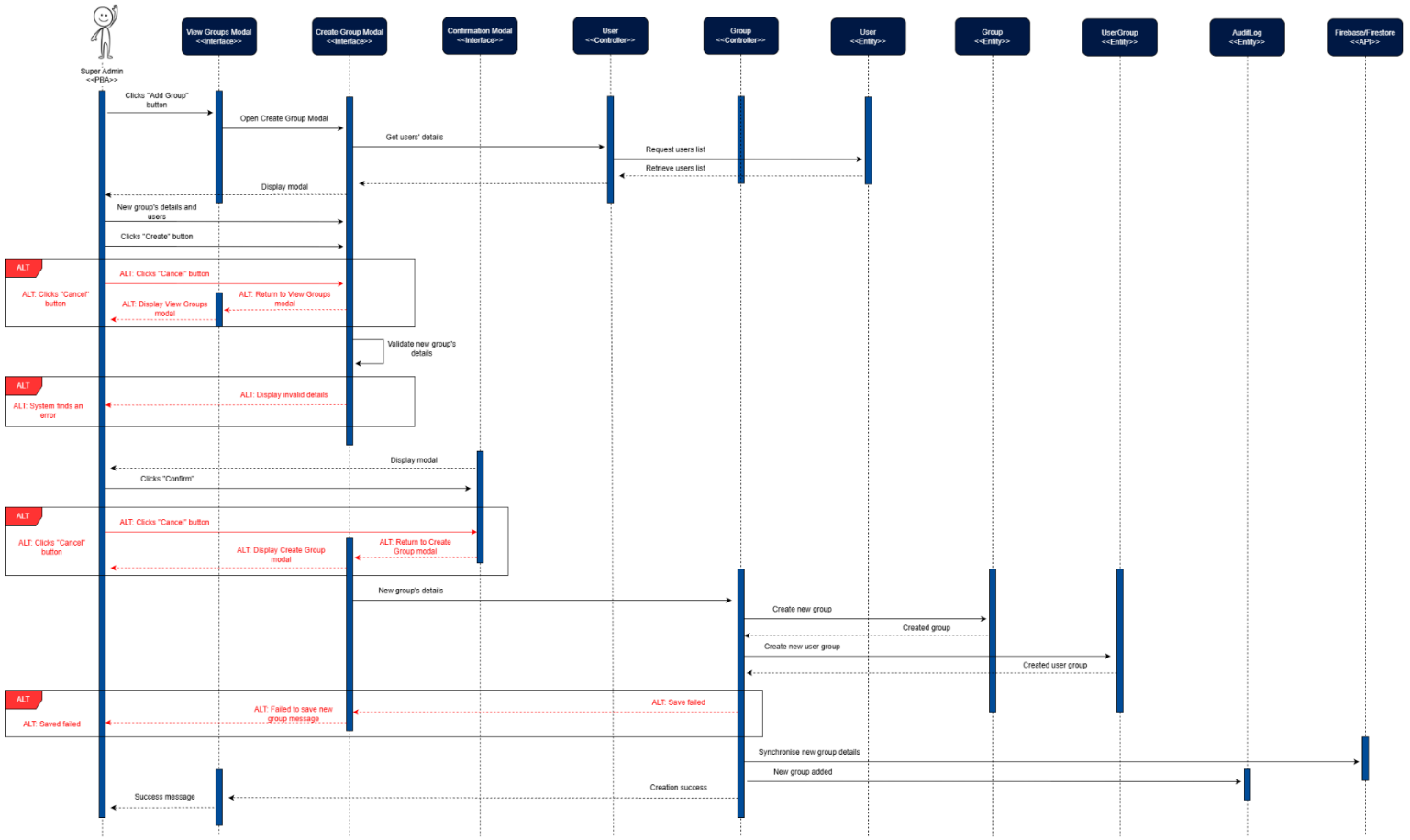


Figure 105: 4.1 Create Group Sequence Diagram

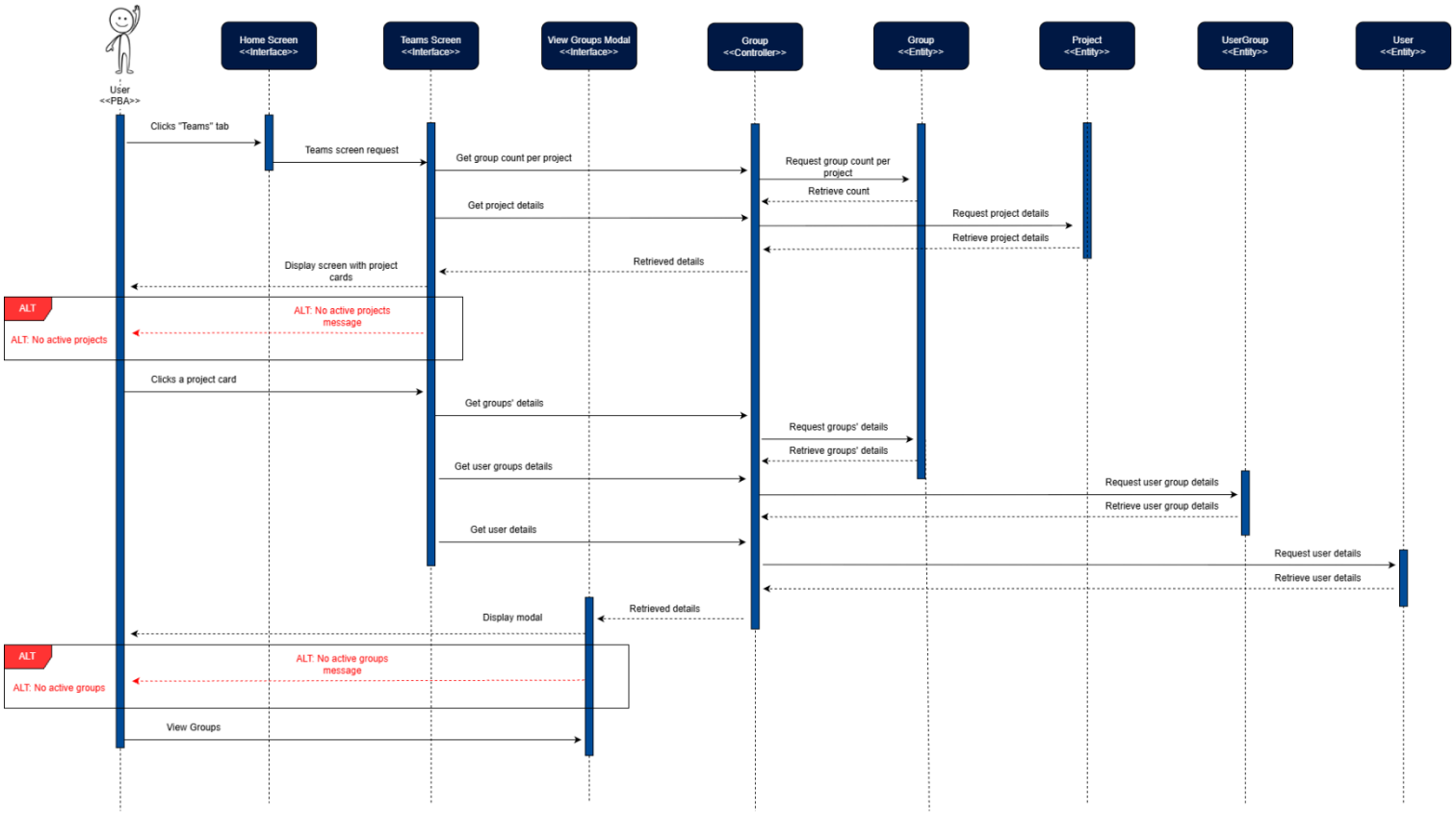


Figure 106: 4.2 View Groups Sequence Diagram

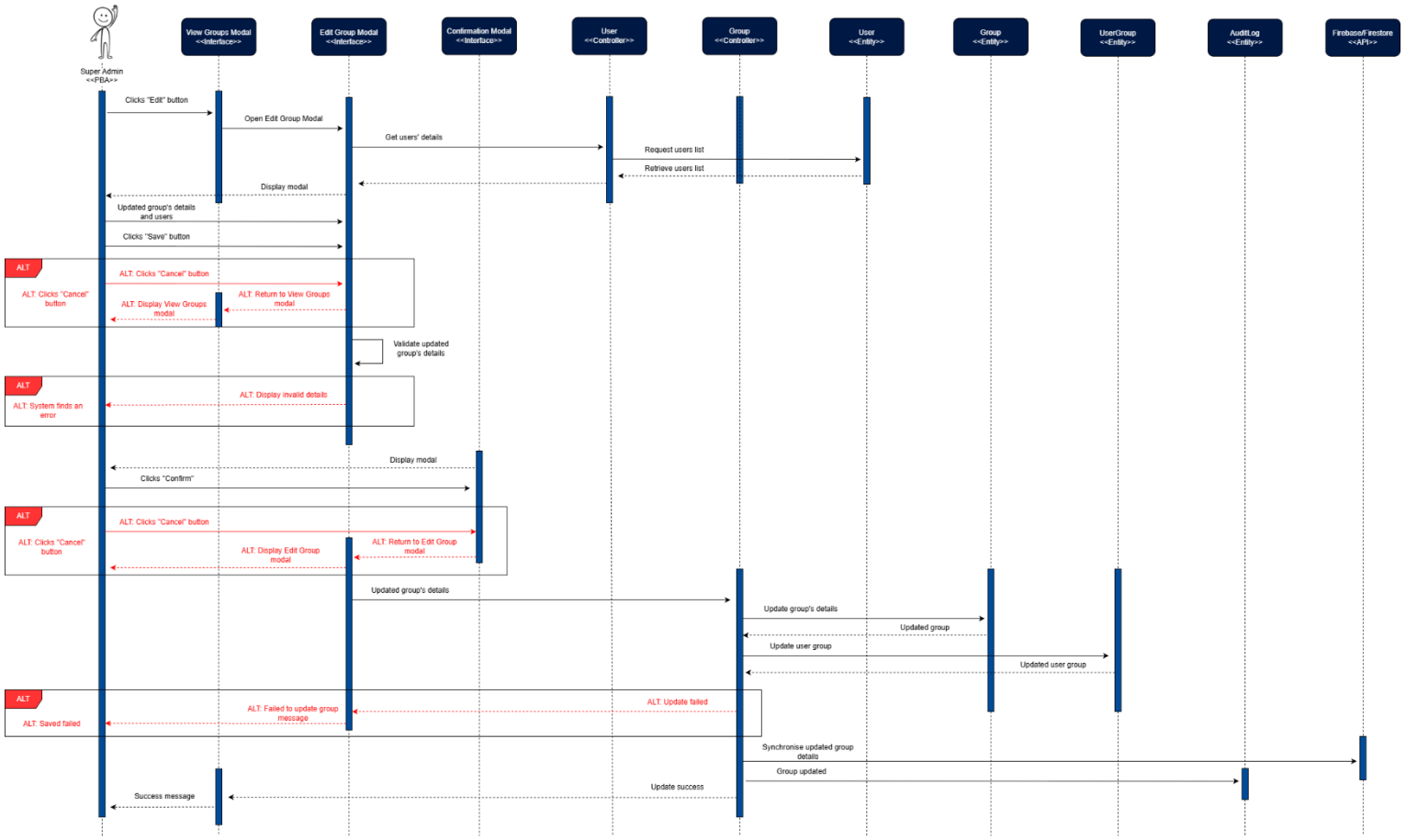


Figure 107: 4.3 Update Group Sequence Diagram

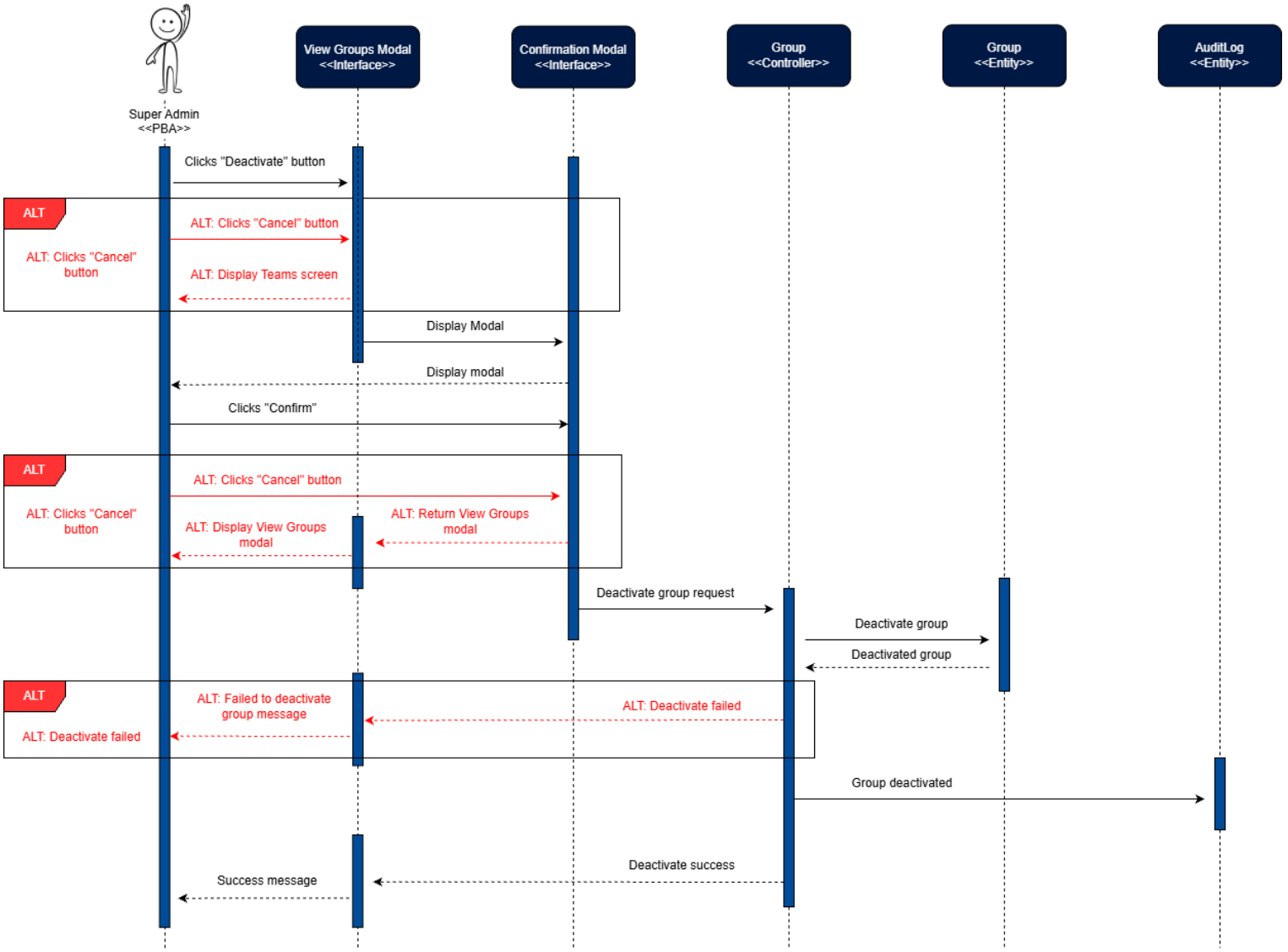


Figure 108: 4.4 Deactivate Group Sequence Diagram

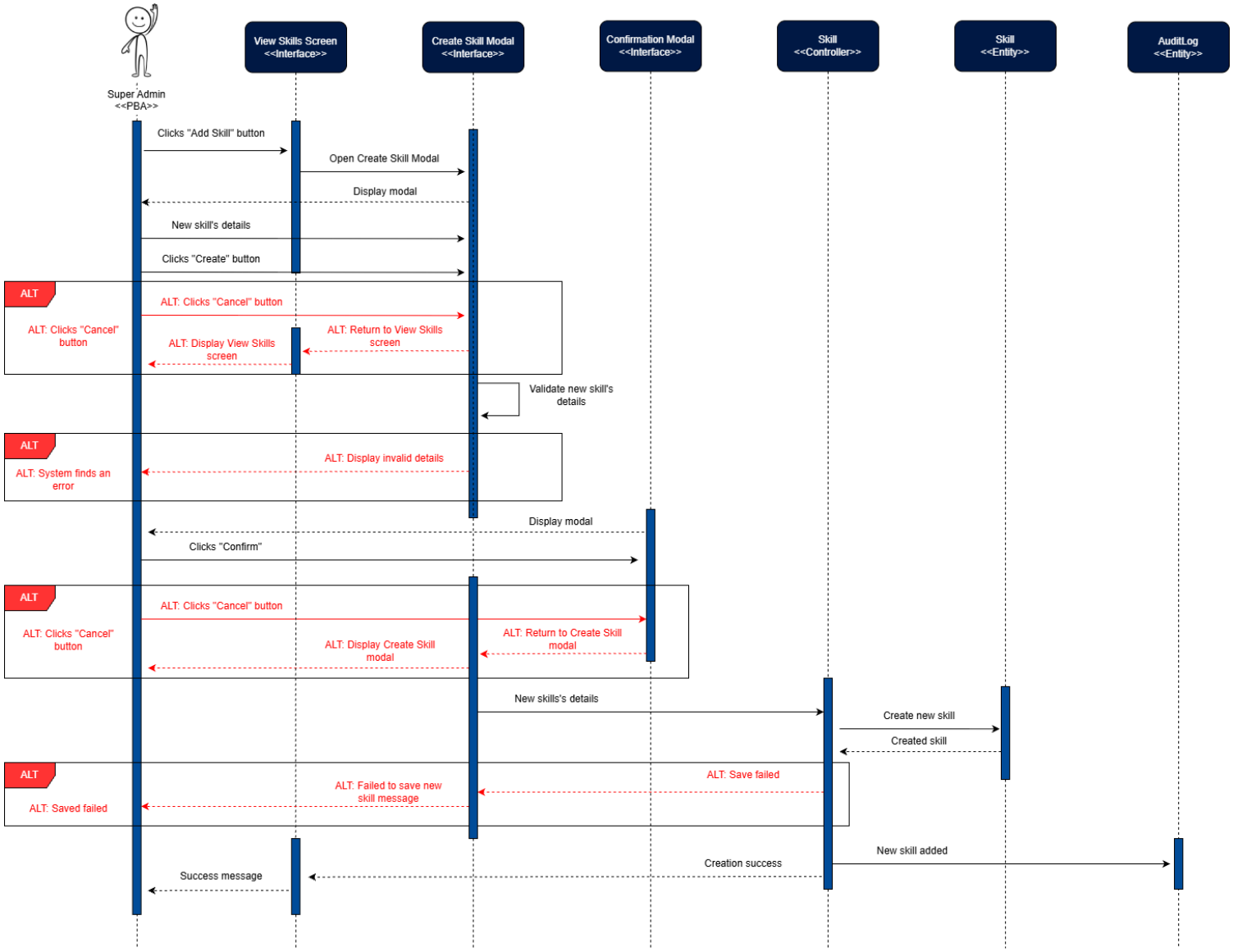


Figure 109: 4.5 Create Skill Sequence Diagram

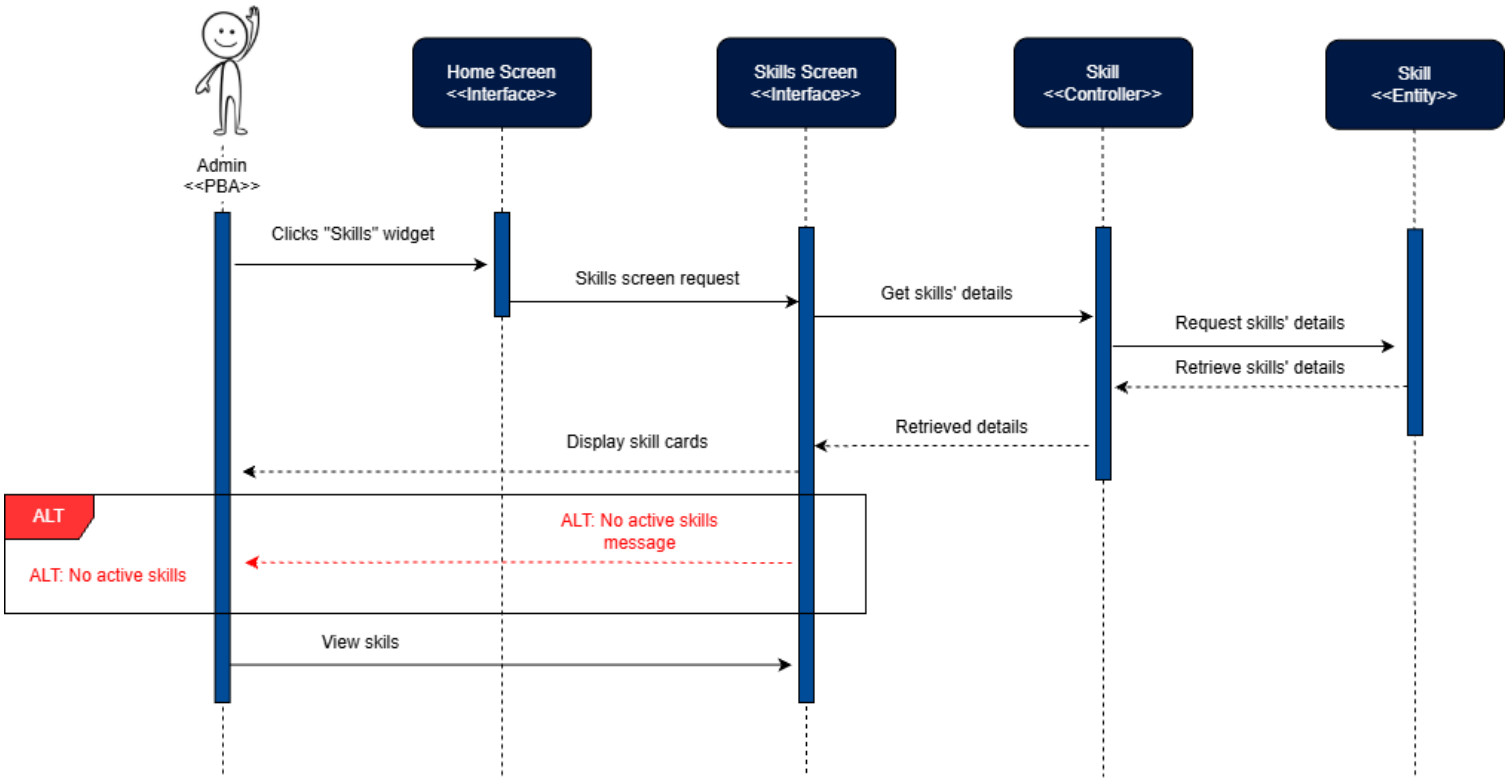


Figure 111: 4.6 View Skills Sequence Diagram

Figure 110: 4.6 View Skills Sequence Diagram

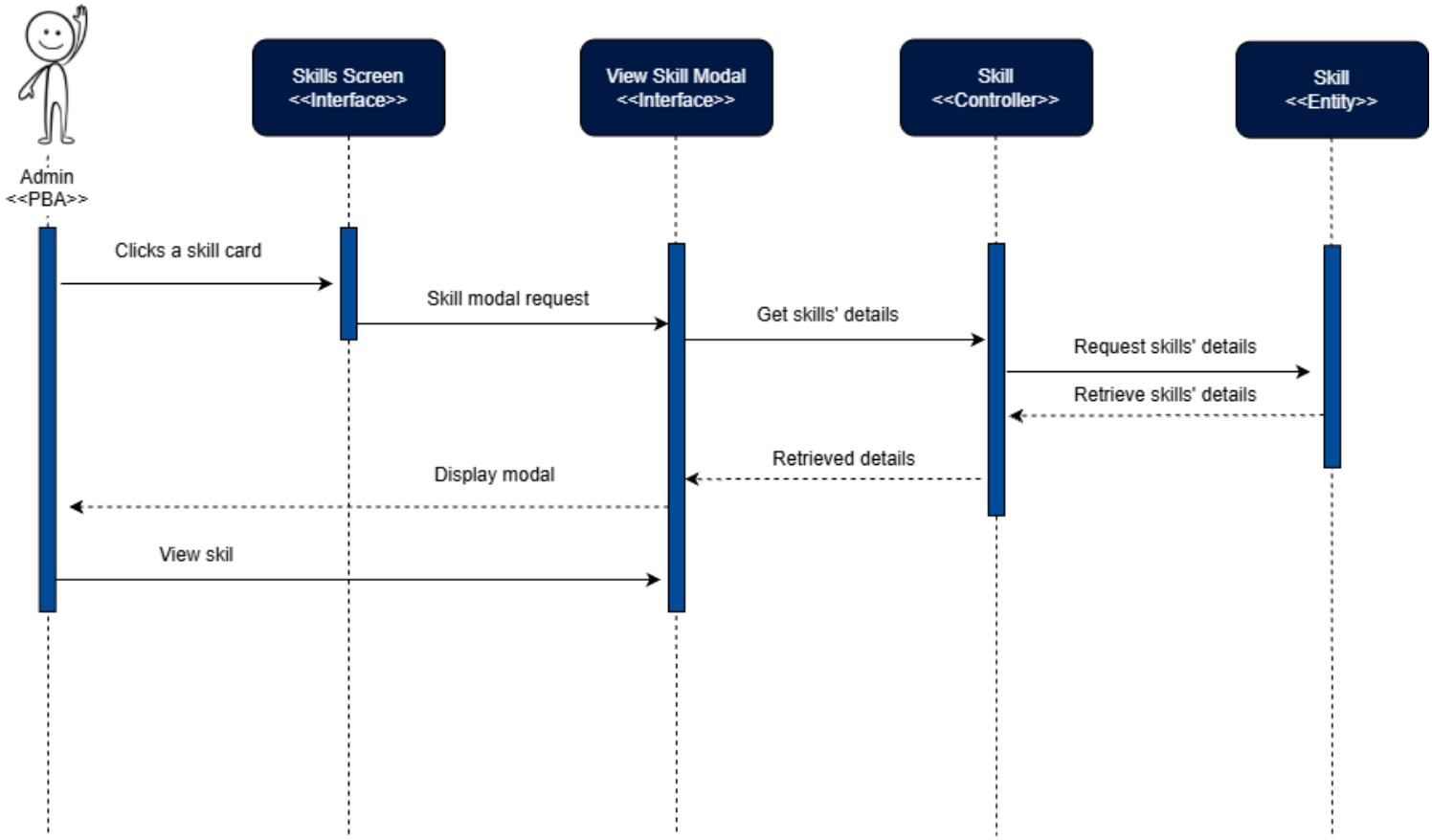


Figure 112: 4.7 View Skill Sequence Diagram

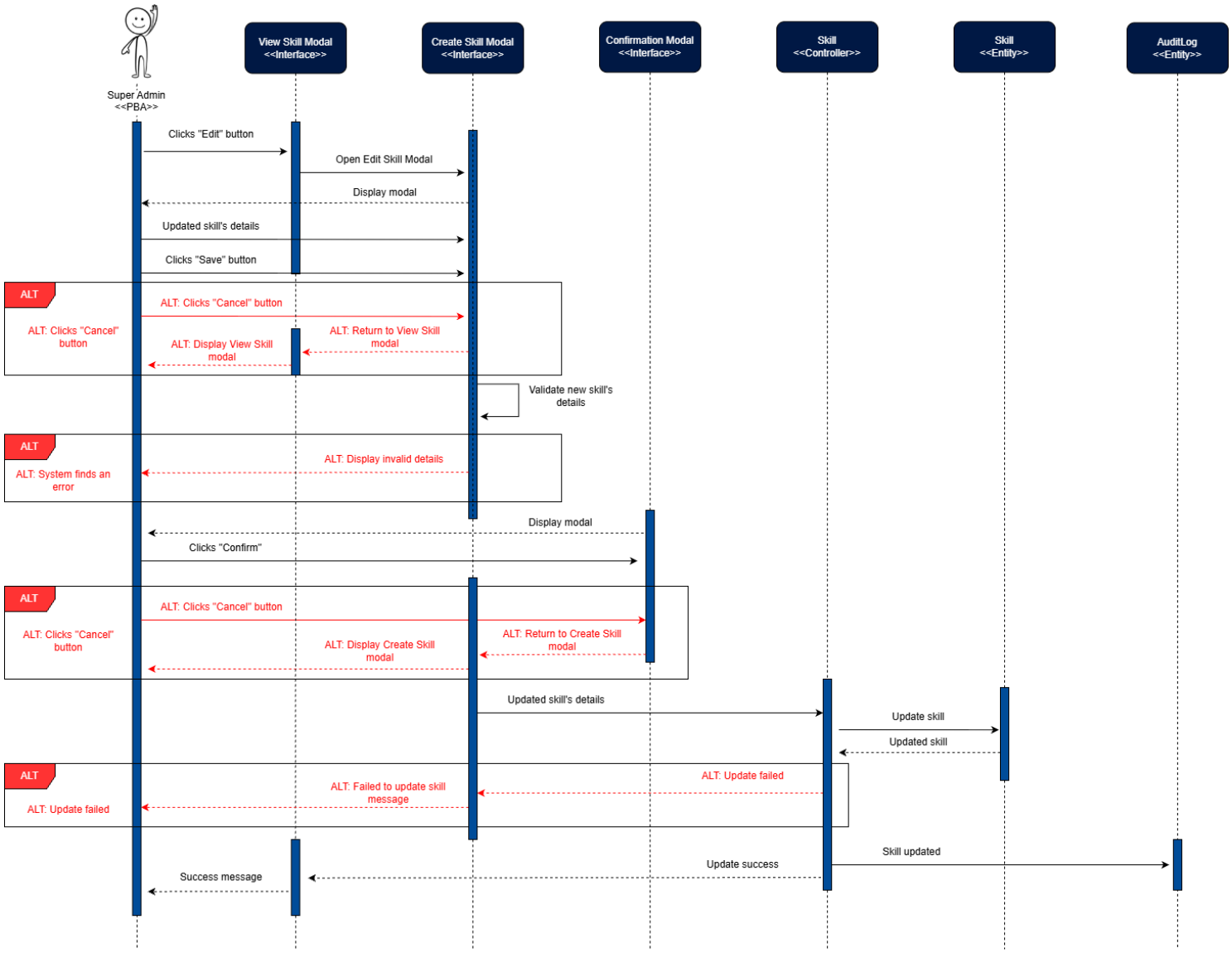


Figure 113: 4.8 Update Skill Sequence Diagram

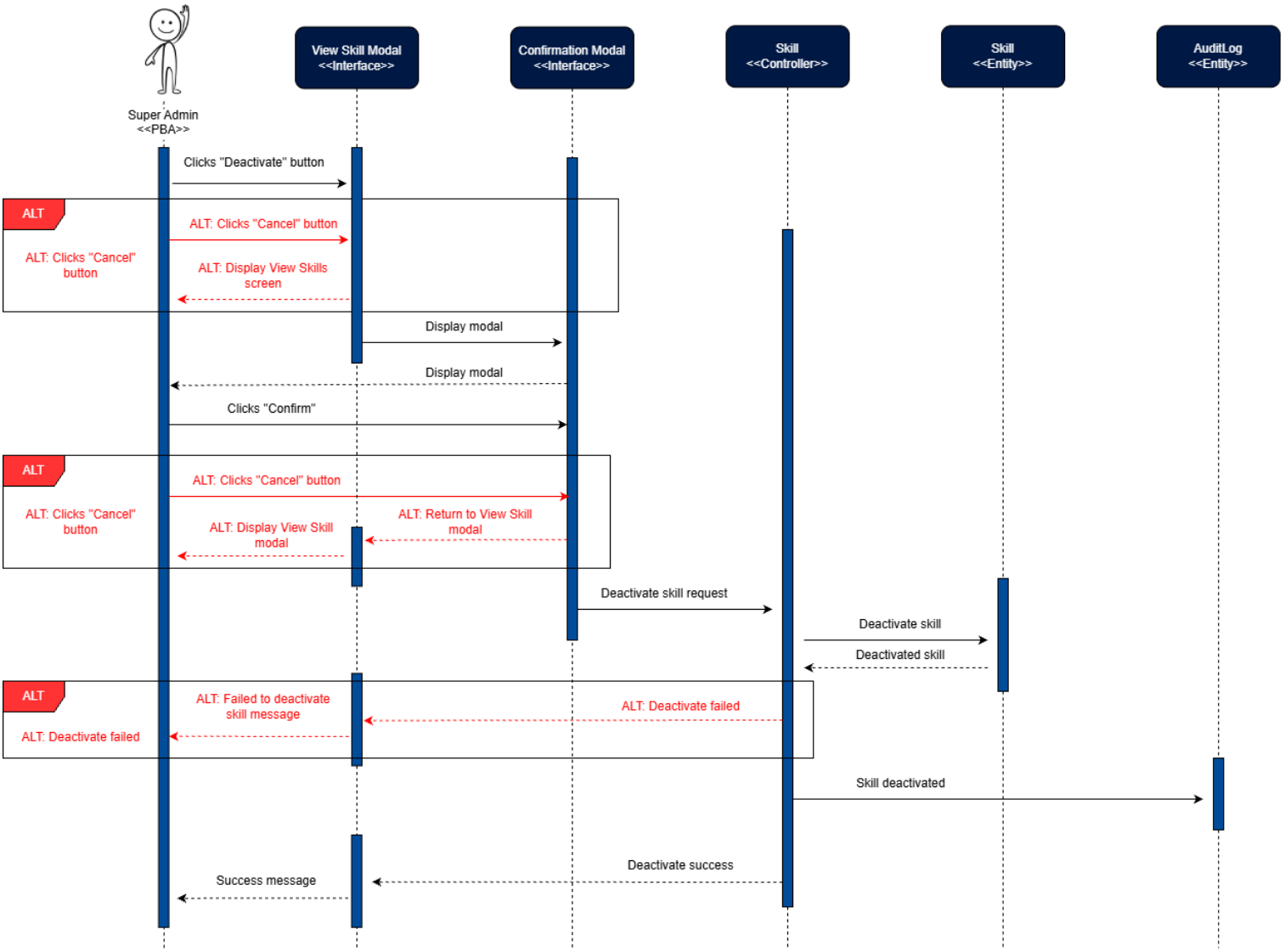


Figure 114: 4.9 Deactivate Skill Sequence Diagram

2.7.4 Project Subsystem Sequence Diagrams

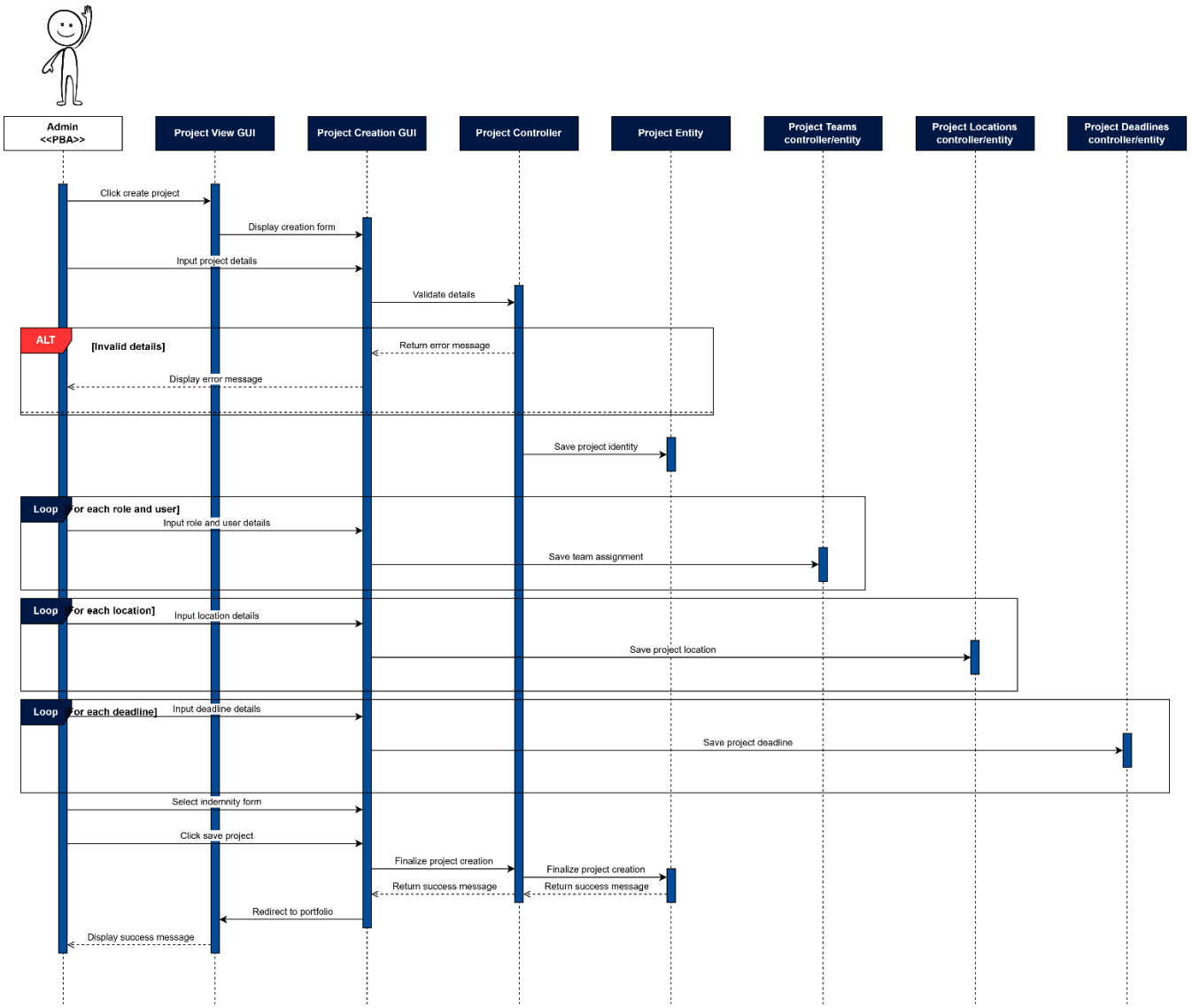


Figure 92: 7.1 Create Project Sequence Diagram

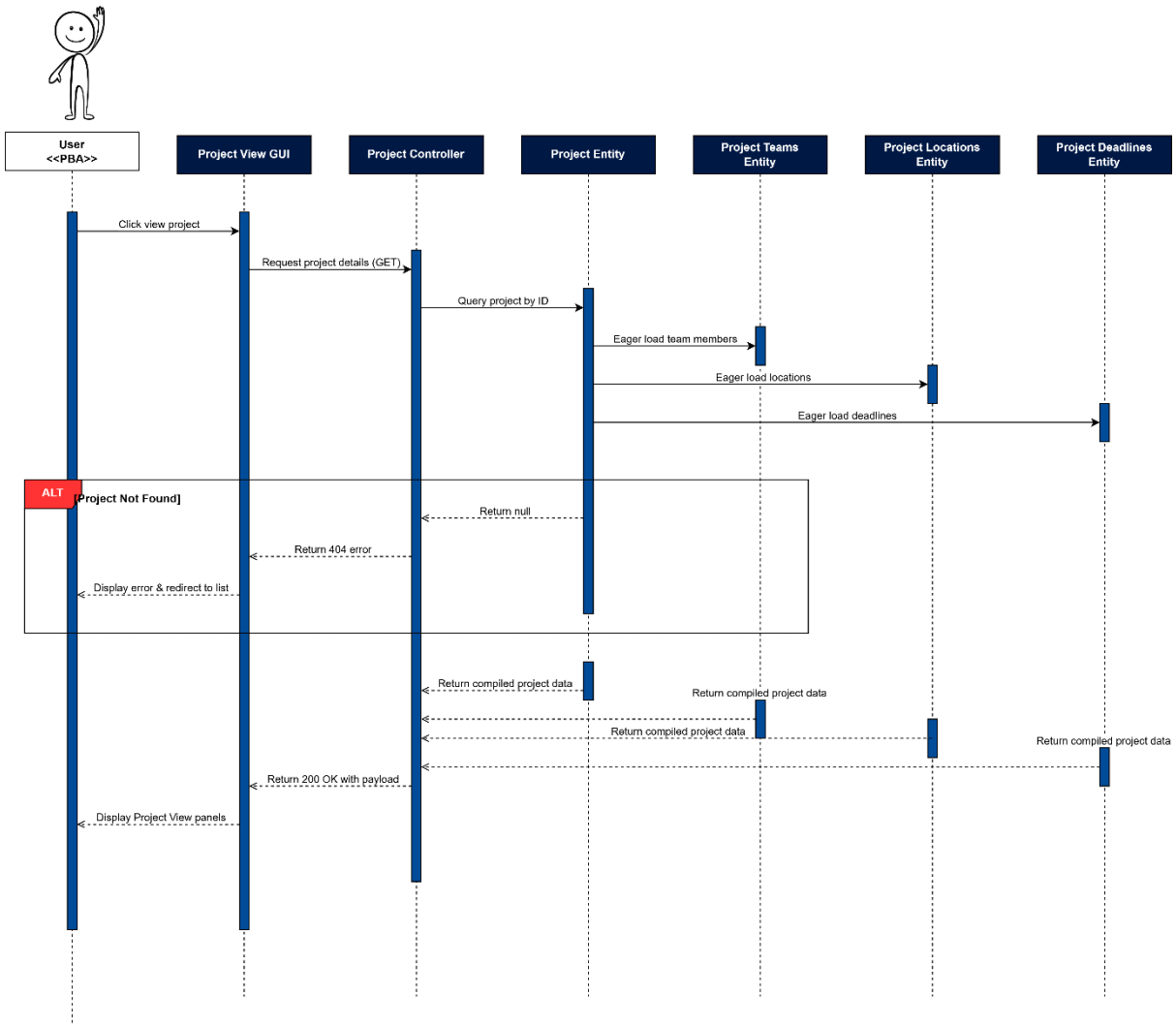


Figure 93: 7.2 View Project Sequence Diagram

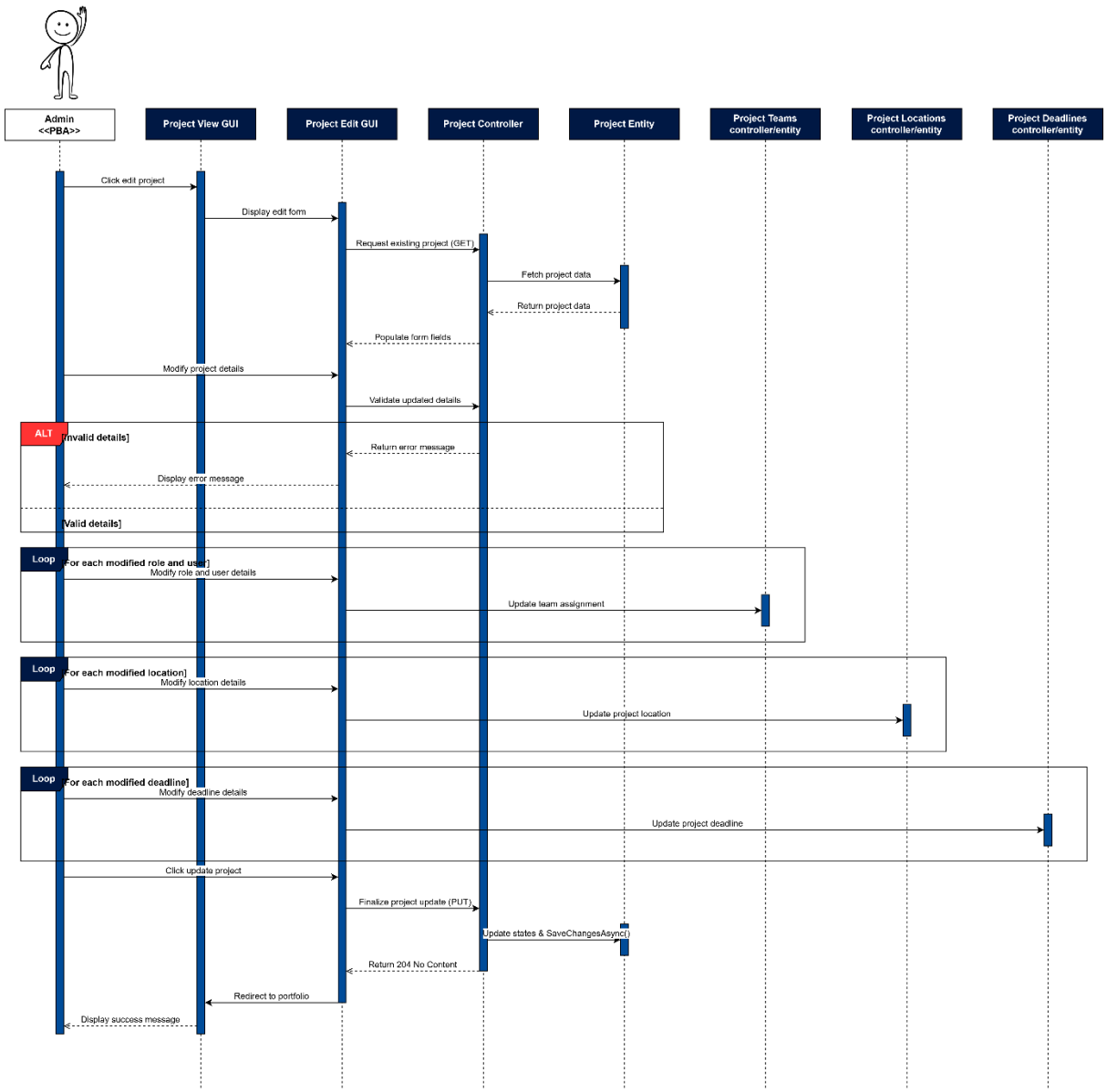


Figure 94: 7.3 Update Project Sequence Diagram

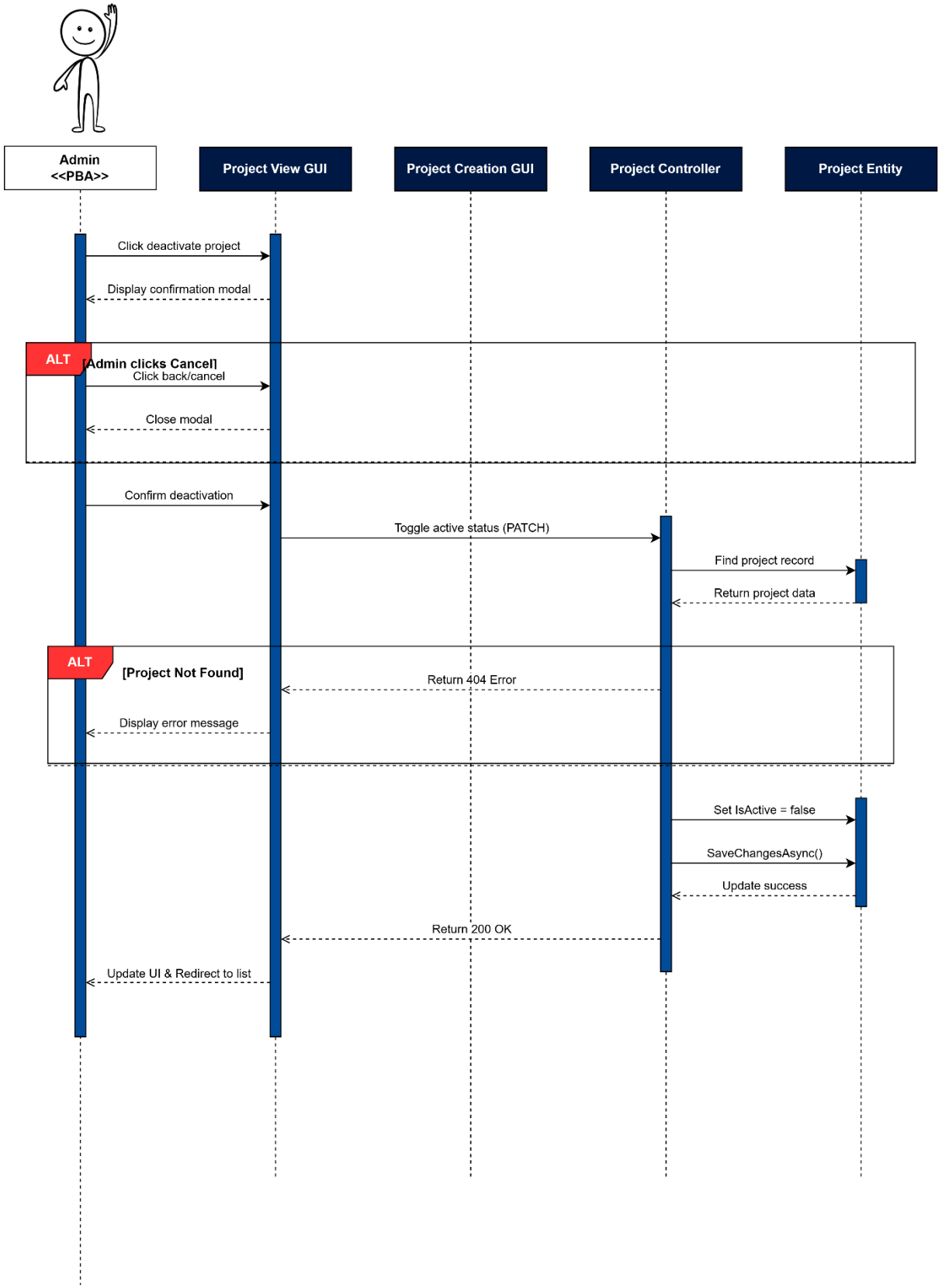


Figure 95: 7.4 Deactivate Project Sequence Diagram

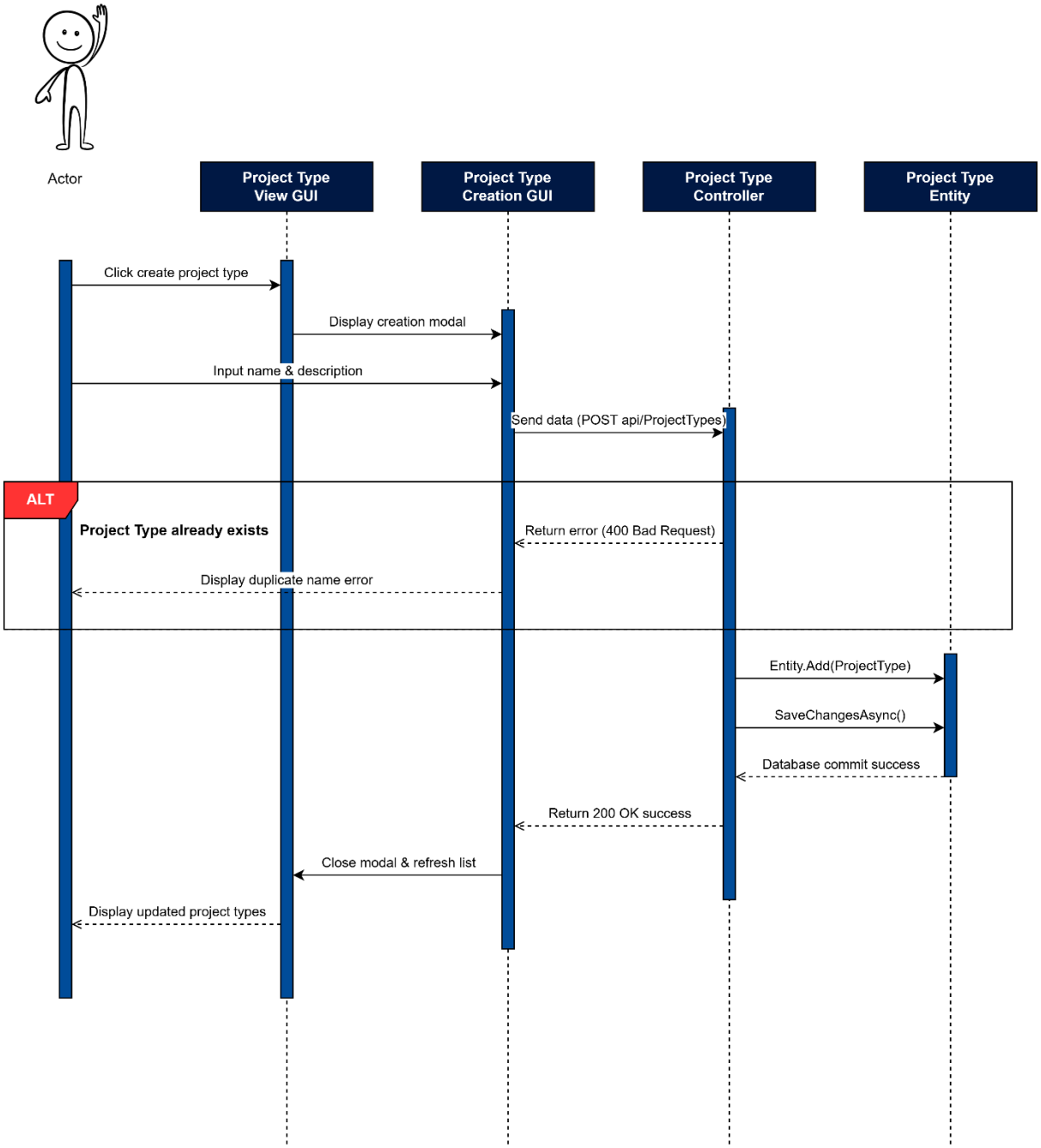


Figure 96: 7.5 Create Project Type Sequence Diagram

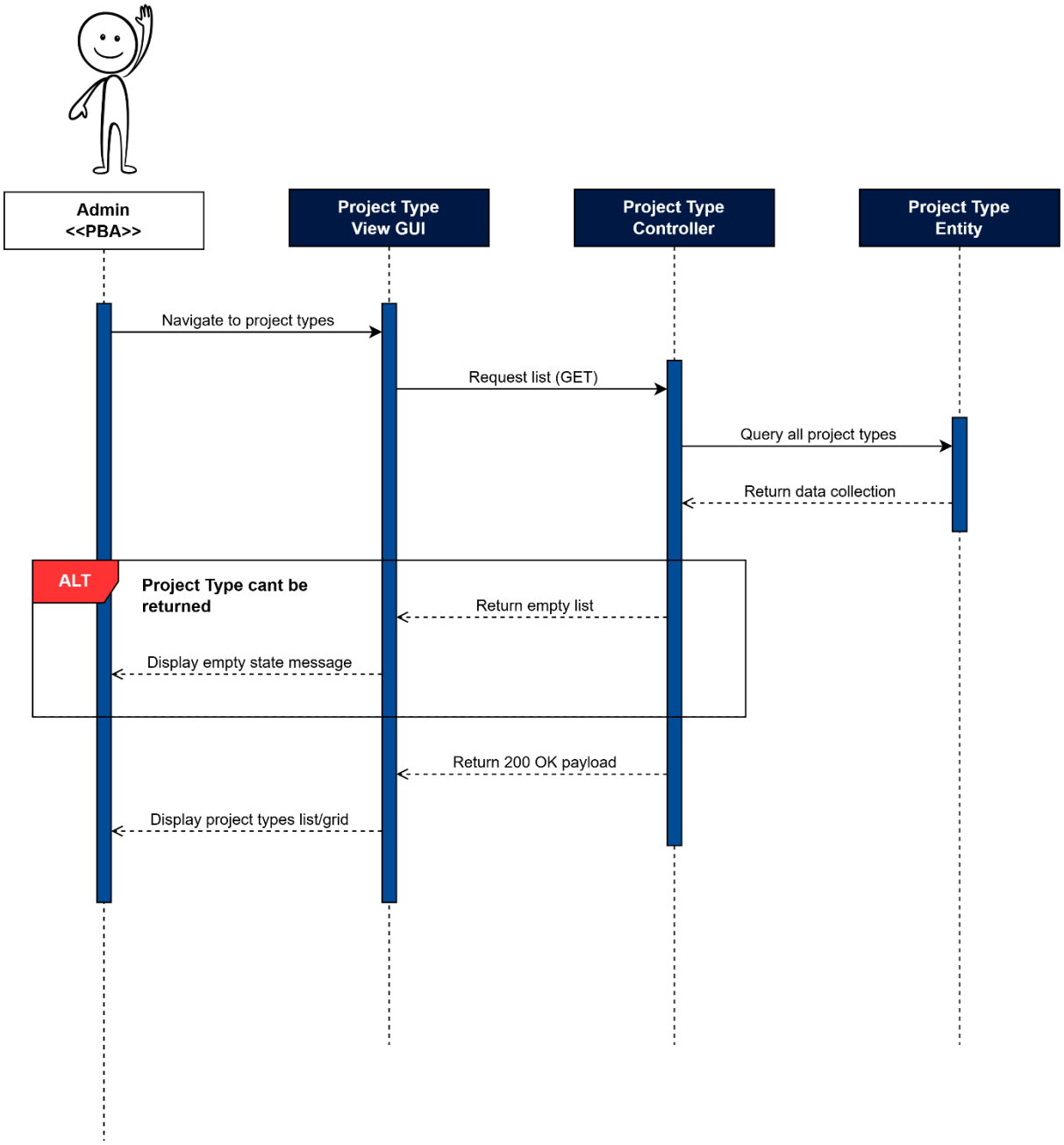


Figure 97: 7.6 View Project Type Sequence Diagram

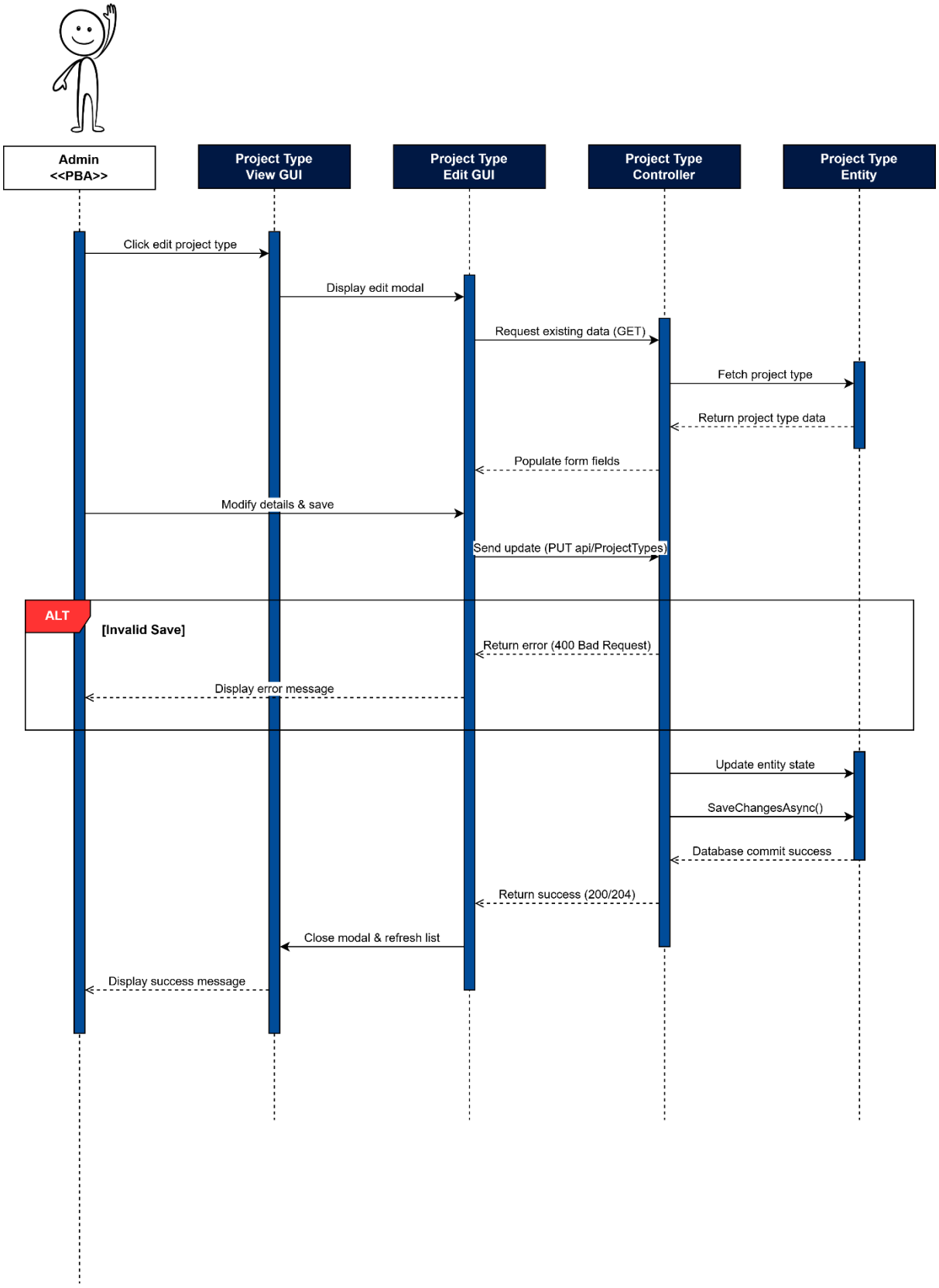


Figure 98: 7.7 Update Project Type Sequence Diagram

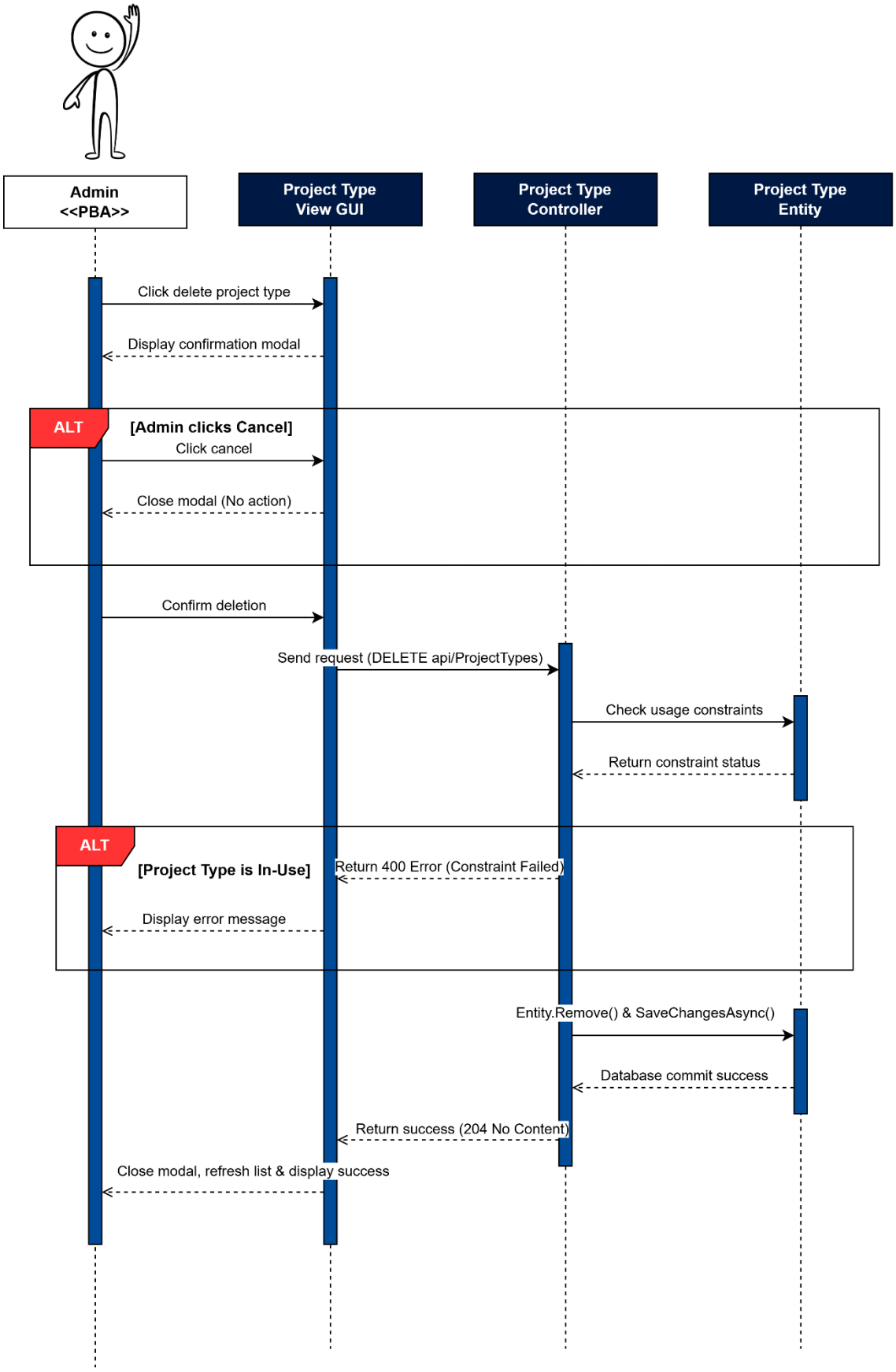


Figure 99: 7.8 Deactivate Project Type Sequence Diagram

2.7.5 Message Subsystem Sequence Diagrams

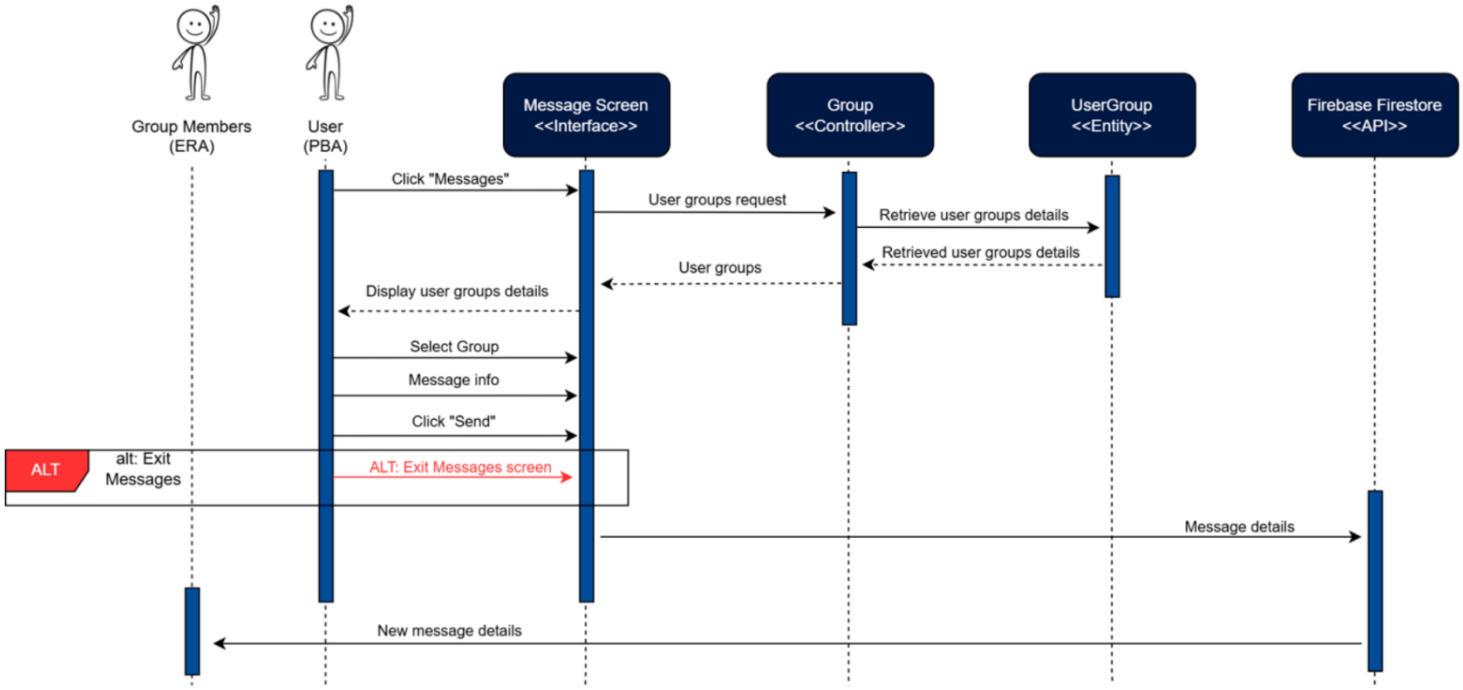


Figure 115: 11.1 Create Message Sequence Diagram

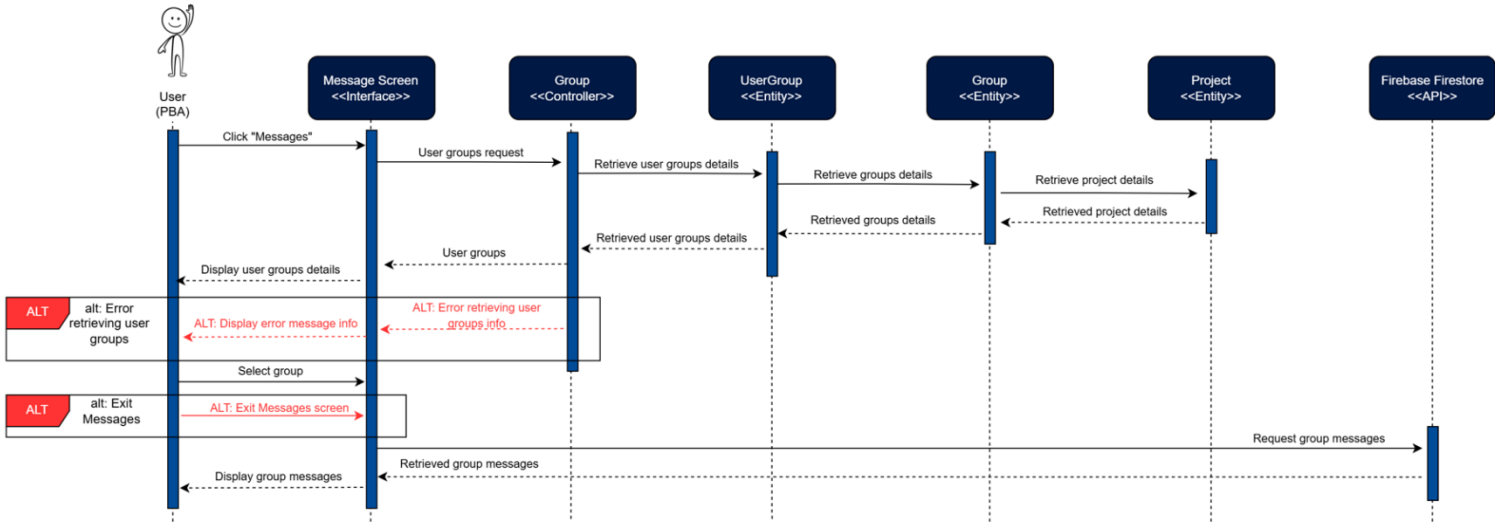


Figure 116: 11.2 View Message Sequence Diagram

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 Create Role

2.8.2.2.9 View Role

2.8.2.2.10 Update Role

2.8.2.2.11 Reactivate Role

2.8.2.2.12 Deactivate Role

2.8.2.2.13 Update Role Permission

2.8.2.2.14 View Role Permission

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

2.8.2.3.6 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 Feedback

2.8.2.7.21 View Impact Feedback

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 34: 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 35: 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.

Functional Requirement	Explanation
Business rules applicable to this requirement	• Secure Access. • Ability to recover Account.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 36: 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 37: 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 38: 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 39: 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 40: 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 41: 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 42: 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 43: 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 44: 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 45: 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 46: 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 47: 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 48: 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 49: 2.7 Remove FAQ – Description

Functional Requirement	Explanation
Requirement number:	2.8
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:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 50: 2.8 Create Role – Description

Functional Requirement	Explanation
Requirement number:	2.9
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:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 51: 2.9 View Role – Description

Functional Requirement	Explanation
Requirement number:	2.10
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:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 52: 2.10 Update Role – Description

Functional Requirement	Explanation
Requirement number:	2.11
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:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 53:2.11 Reactivate Role – Description

Functional Requirement	Explanation
Requirement number:	2.12
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:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 54: 2.12 Deactivate Role – Description

Functional Requirement	Explanation
Requirement number:	2.13
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:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 55: 2.13 Update Role Permission – Description

Functional Requirement	Explanation
Requirement number:	2.14
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:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 56: 2.14 View Role Permission – Description

2.8.3.3 Inventory Subsystem

Functional Requirement	Explanation
Requirement number:	3.1
Requirement name (use case name):	Receive Inventory Item
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:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 57: 3.1 Receive Inventory Item – Description

Functional Requirement	Explanation
Requirement number:	3.2
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 receive inventory.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 58: 3.2 View Inventory – Description

Functional Requirement	Explanation
Requirement number:	3.3
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 receive inventory.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 59: 3.3 Write-Off Inventory - Description

Functional Requirement	Explanation
Requirement number:	3.4
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 receive perform a stock take.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 60: 3.4 Allocate Inventory – Description

Functional Requirement	Explanation
Requirement number:	3.5
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:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 61: 3.5 Create Inventory Item Type – Description

Functional Requirement	Explanation
Requirement number:	3.6
Requirement name (use case name):	View Inventory Item Type
Requirement short description:	A user must be able to view an inventory item type.
Requirement detailed description and constraints:	A user must be able to view an inventory item type. The system will retrieve and display all the inventory item types on the system before the user selects an inventory item type to view. Information including name, description and security level must be displayed. **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 an inventory item type.
Revision date and Revision number:	2026-04-09 Version 3.0
Criticality/Priority:	High

Table 62: 3.6 View 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:	High

Table 63:3.7 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:	High

Table 64: 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:	High

Table 65: 3.9 Deactivate 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 66: 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 67:4.2 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 68: 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 69: 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 70: 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 71: 4.6 View ~~Skill~~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 72: 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 73: 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 74: 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 75 :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 76: 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 77: 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 78 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 79: 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 80: 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 81: 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 82 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 83: 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 84: 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 85: 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 86: 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 87: 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 88: 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 89: 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 90: 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 91: 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 92: 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 93: 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 94: 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 95: 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 96: 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 97: 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 98: 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 99: 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 100: 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 101: 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 102: 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 103: 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 104: 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 105: 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 106: 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 107: 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 108: 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 109: 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 110: 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 111: 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 112: 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 113: 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 114: 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 115: 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 116: 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 117: 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 118: 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 119: 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 120: 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 121: 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 122: 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 123: 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 124: 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 125: 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 126: 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 127: 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 128: 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 129: 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 130: 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 131: 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:	High

Table 132: 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:	High

Table 133:9.2 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:	High

Table 134: 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:	High

Table 135: 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:	High

Table 136: 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 137: 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 138: 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 139: 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 140: 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 141: 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 142: 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 143: 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 144: 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 145: 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 146: 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 147: 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 148: 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 149: 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 150: 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 151: 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 152: 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 153: 13.1 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 154: 14.1 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 155: 14.2 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 156: 14.3 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 157: 15.1 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 158: 15.2 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 159: 15.3 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 160: 15.4 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 161: 15.5 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 162: 15.6 Remove An Expense – Description

2.8.3 Functional Requirement List Detailed Changes Description

2.8.3.1 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 163: User Subsystem Modifications

2.8.3.2 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 Create Role	2.8 Create Role	-
2.9 View Role	2.9 View Role	-
2.10 Update Role	2.10 Update Role	-
2.11 Reactivate Role	2.11 Reactivate Role	-
2.12 Deactivate Role	2.12 Deactivate Role	-
2.13 Update Role Permission	2.13 Update Role Permission	-
2.14 View Role Permission	2.14 View Role Permission	-

Table 164: Admin Subsystem Modifications

2.8.3.3 Inventory Subsystem

Inventory Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
3.1 Receive Inventory Item	3.1 Receive Inventory Item	-
3.2 View Inventory	3.2 View Inventory	-
3.3 Write Off Inventory	3.3 Write Off Inventory	-
3.4 Inventory Stock Take	[REMOVED]	Moved to warehouse subsystem as inventory

Inventory Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
		stock takes are done on a warehouse.
3.5 Allocate Inventory	3.4 Allocate Inventory	Numbering change.
3.6 Create Inventory Item Type	3.5 Create Inventory Item Type	Numbering change.
3.7 View Inventory Item Type	3.6 View Inventory Item Type	Numbering change.
3.8 View Inventory Item Types	3.7 View Inventory Item Types	Numbering change.
3.9 Update Inventory Item Type	3.8 Update Inventory Item Type	Numbering change.
3.10 Deactivate Inventory Item Type	3.9 Deactivate Inventory Item Type	Numbering change.

Table 165 Inventory Subsystem Modifications

2.8.3.4 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 Update Skill	4.7 Update Skill	-
4.8 Deactivate Skill	4.8 Deactivate Skill	-

Table 166 Group Subsystem Modifications

2.8.3.5 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	Moved from Inventory Subsystem to Warehouse Subsystem.

Table 167: Warehouse Subsystem Modifications

2.8.3.6 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 168 Vendor Subsystem Modifications

2.8.3.7 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	-
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	Clarity between impact and feedback
7.17 View Impact Feedback Form	7.17 View Impact Form	Clarity between impact and feedback

Project Management Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
7.18 Update Impact Feedback Form	7.18 Update Impact Form	Clarity between impact and feedback
7.19 Deactivate Impact Feedback Form	7.19 Deactivate Impact Form	Clarity between impact and feedback
7.20 Capture Impact Feedback	7.20 Capture Impact Form	Clarity between impact and feedback
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 169 Vendor Subsystem Modifications

2.8.3.8 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 170 K9 Subsystem Modifications

2.8.3.9 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 171 Rhino Subsystem Modifications

2.8.3.10 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 172 Elephant Subsystem Modifications

2.8.3.11 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 173 Message Subsystem Modifications

2.8.3.12 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	-

Table 174: Report Subsystem Modifications

2.8.3.13 Machine Learning Subsystem

Machine Learning Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
13.1 View Machine Learning Dashboard	12.10 View Machine Learning Dashboard	Moved to the Report Subsystem because a subsystem can't have one use case.

Table 175: Machine Learning Subsystem Modifications

2.8.3.14 System Subsystem

System Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
14.1 Backup System	13.1 Backup System	Numbering change.
14.2 Restore System	13.2 Restore System	Numbering change.
14.3 View Audit Log	13.3 View Audit Log	Numbering change.

Table 176: System Subsystem Modifications

2.8.4.15 Budget Subsystem

Budget Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
15.1 Create Project Budget	[REMOVED]	A project budget will be created when a project is created.
15.2 View Project Budget	14.1 View Project Budget	Numbering change.
15.3 View Project Budgets	14.2 View Project Budgets	Numbering change.

Budget Subsystem		
Use Case Name	*Modification Change*	*Reason for Change*
15.4 Update Project Budget	[REMOVED]	A project's budget can be updated while updating a project.
15.5 Log An Expense	14.3 Log An Expense	Numbering change.
15.6 Update Expense	14.4 Update Expense	Numbering change.
15.7 View Expense	14.5 View Expense	Numbering change.
15.8 Remove Expense	14.6 Remove Expense	Numbering change.

Table 177 Budget Subsystem Modifications

2.8.5 Conclusion

Section 2.8 of the document outlined the requirements for the proposed system. It contained an updated functional requirements list and their descriptions. We also included the new changes made to the functional requirements and the reasoning behind the changes.

2.9 Technical Use Case Diagram

2.9.1 User Subsystem

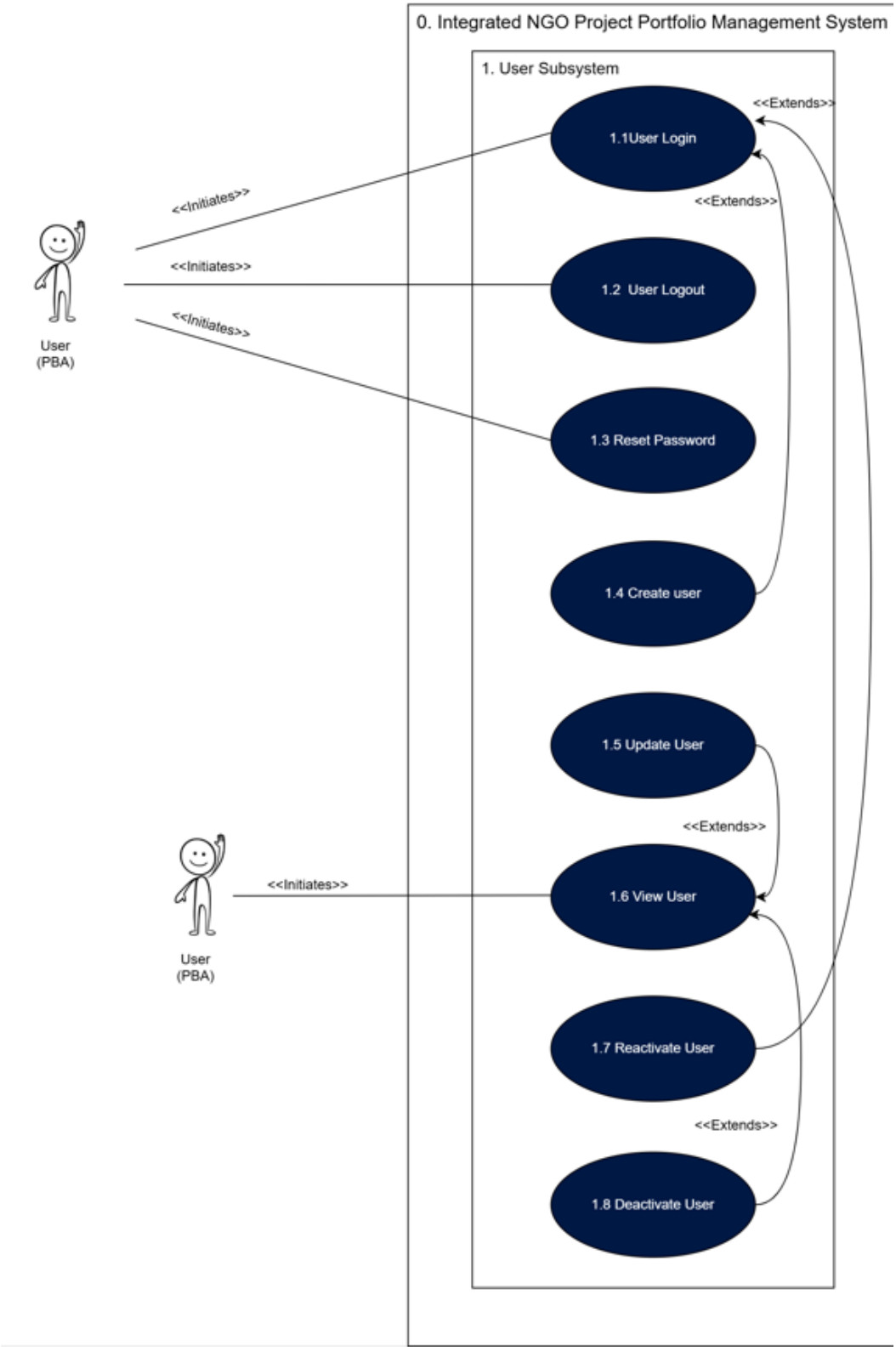


Figure 117: User Subsystem Technical Use Case Diagram

2.9.2 Admin Subsystem

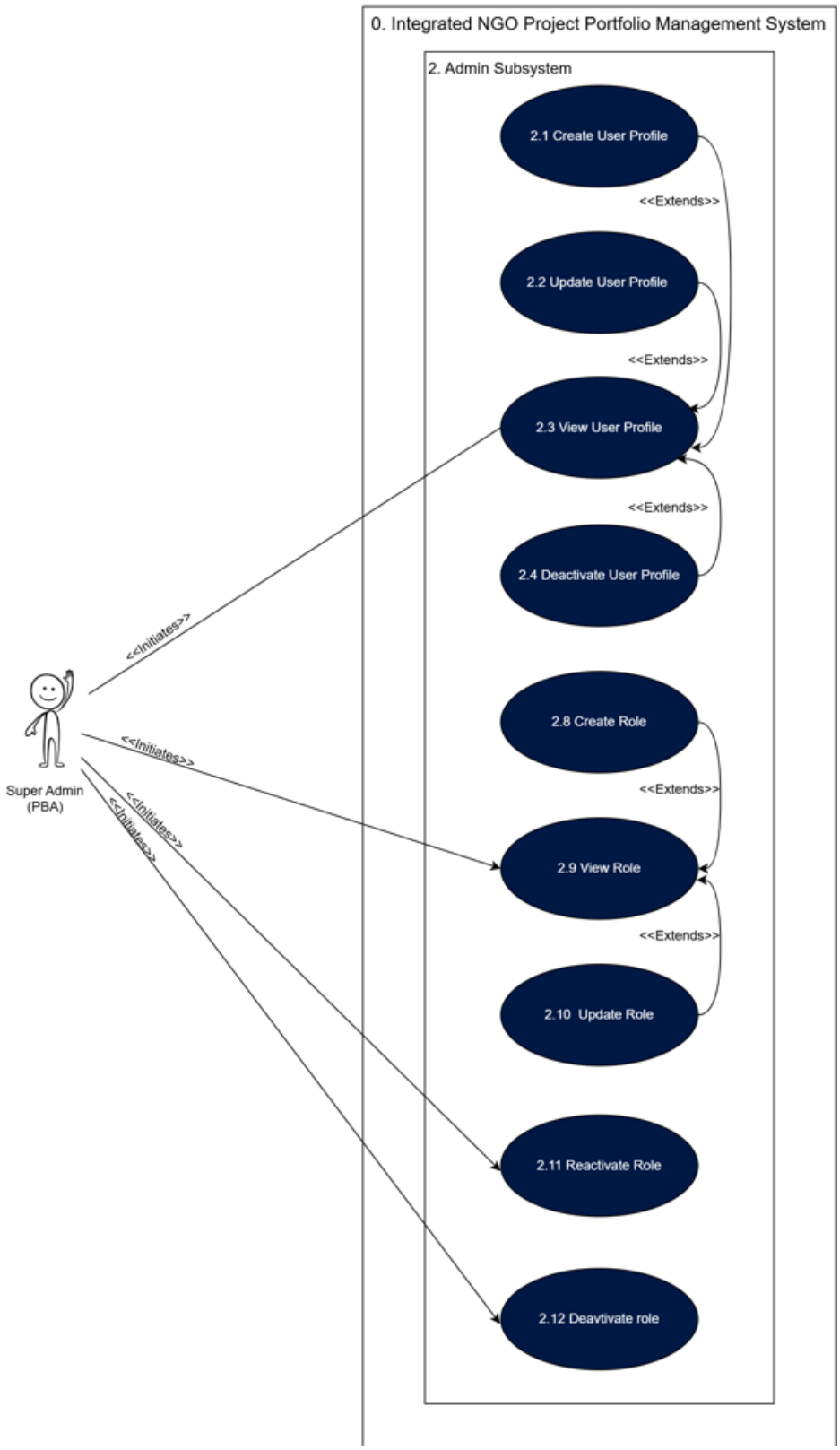


Figure 118: Admin Subsystem Technical Use Case Diagram

2.9.3 Group Subsystem

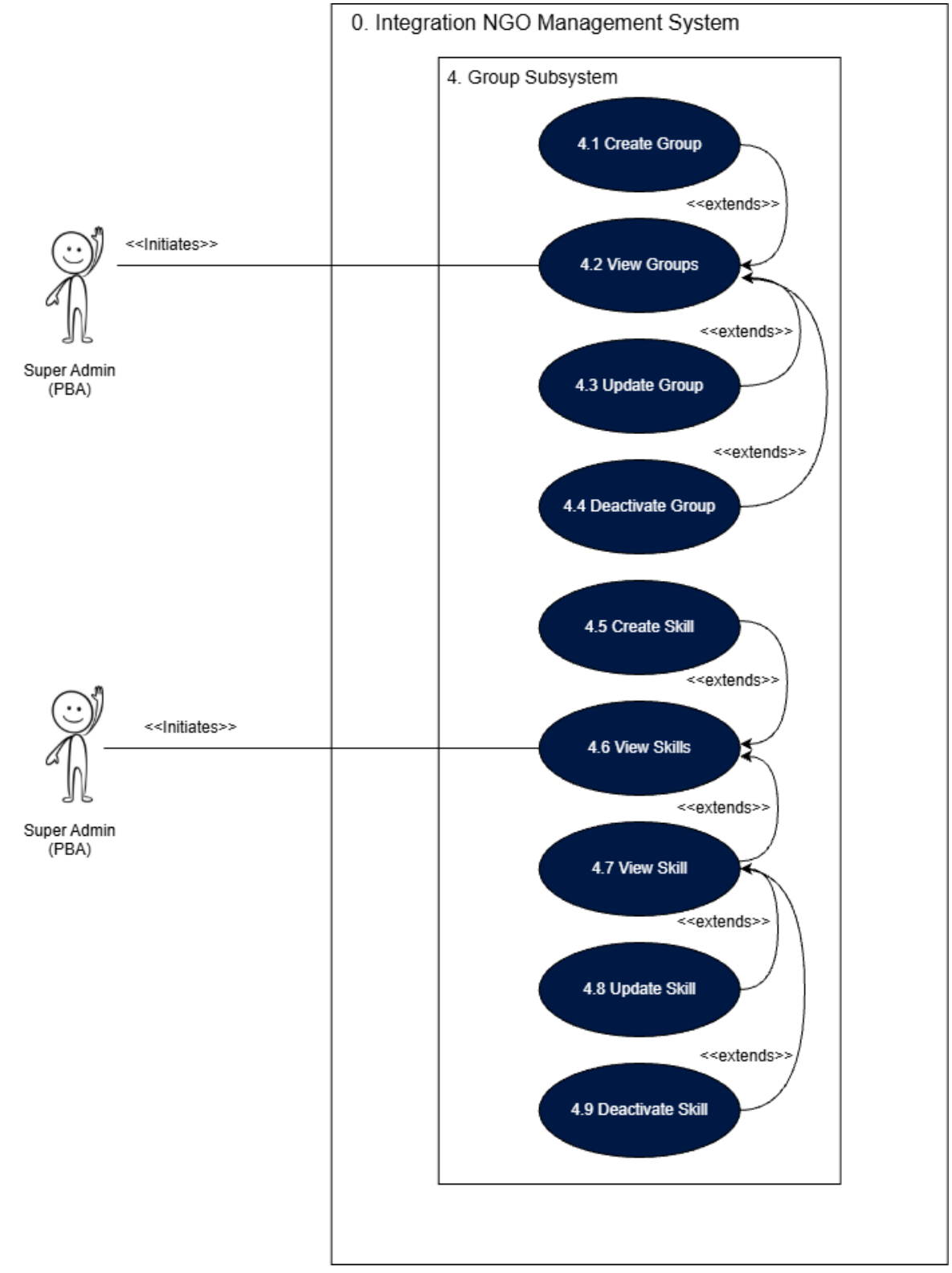


Figure 119: Group Subsystem Technical Use Case Diagram

2.9.4 Project Management Subsystem

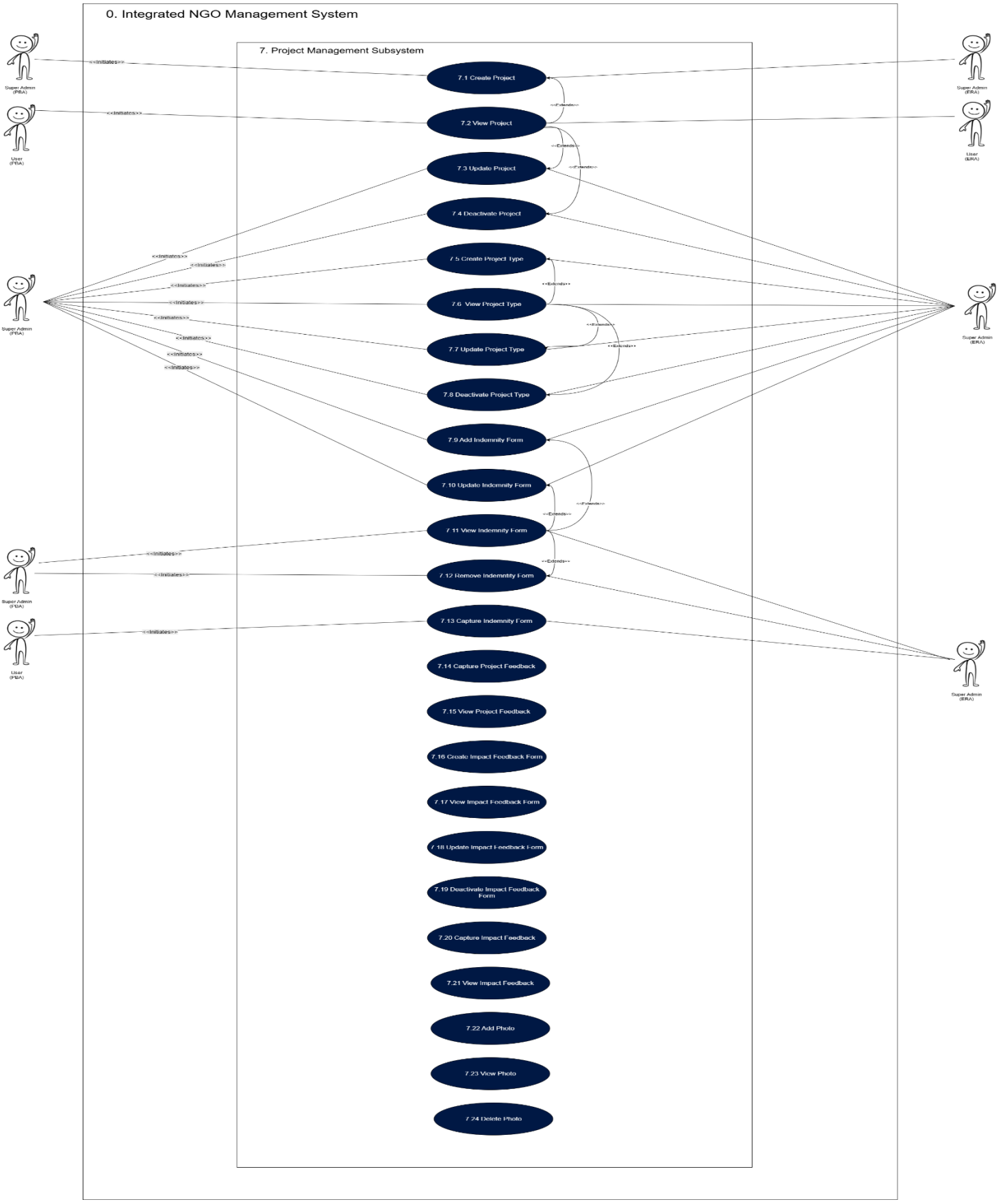


Figure 120: Project Subsystem Technical Use Case Diagram

2.9.5 Message Subsystem

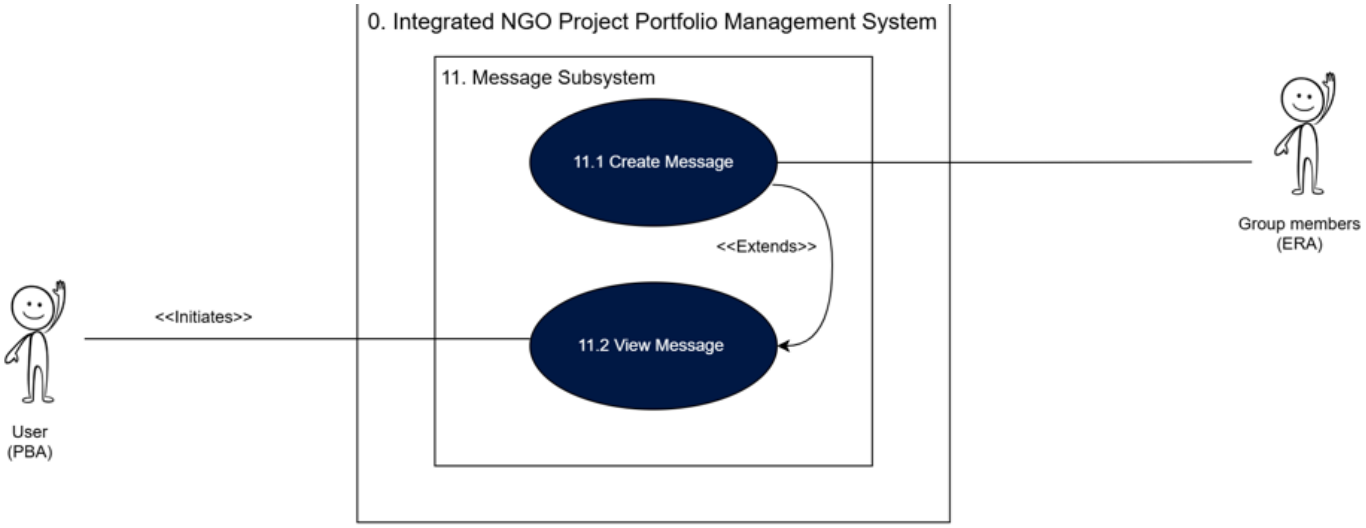


Figure 121: Message Subsystem Technical Use Case Diagram

2.10 Technical Entity Relationship Diagram

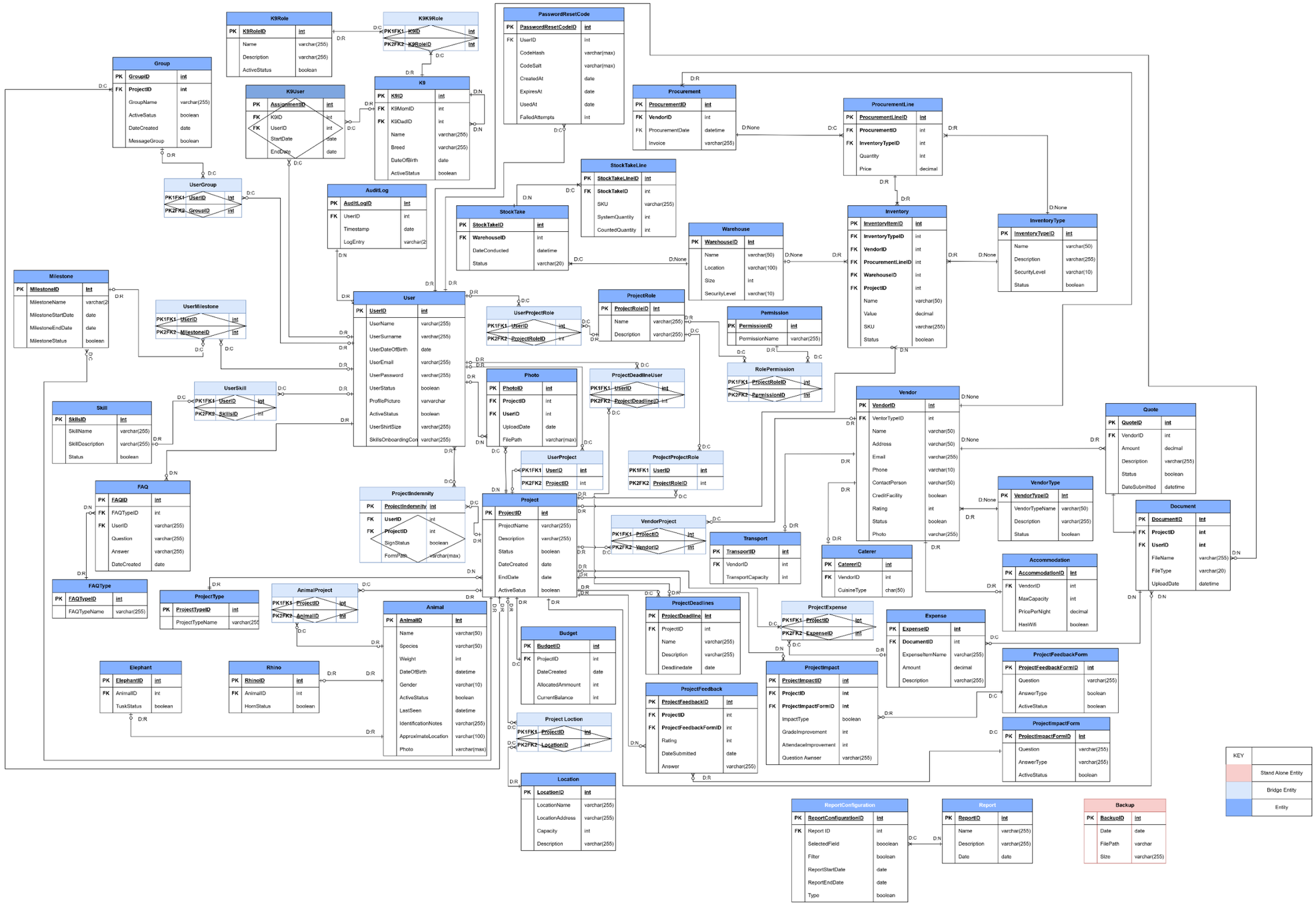


Figure 122: Technical Entity Relationship Diagram

2.11 State Diagram

2.11.1 Projects Controller

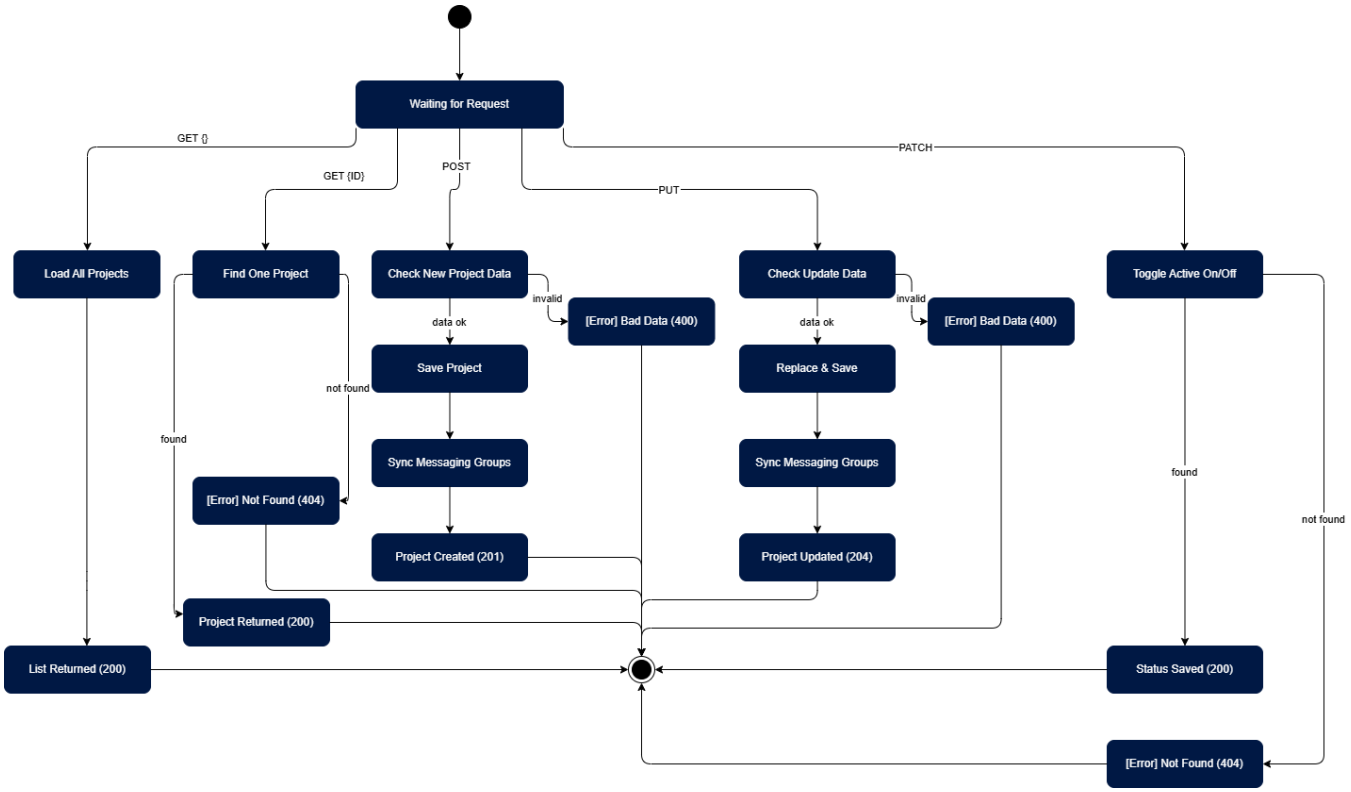


Figure 123: Project Controller State Diagram

2.11.2 Auth Controller

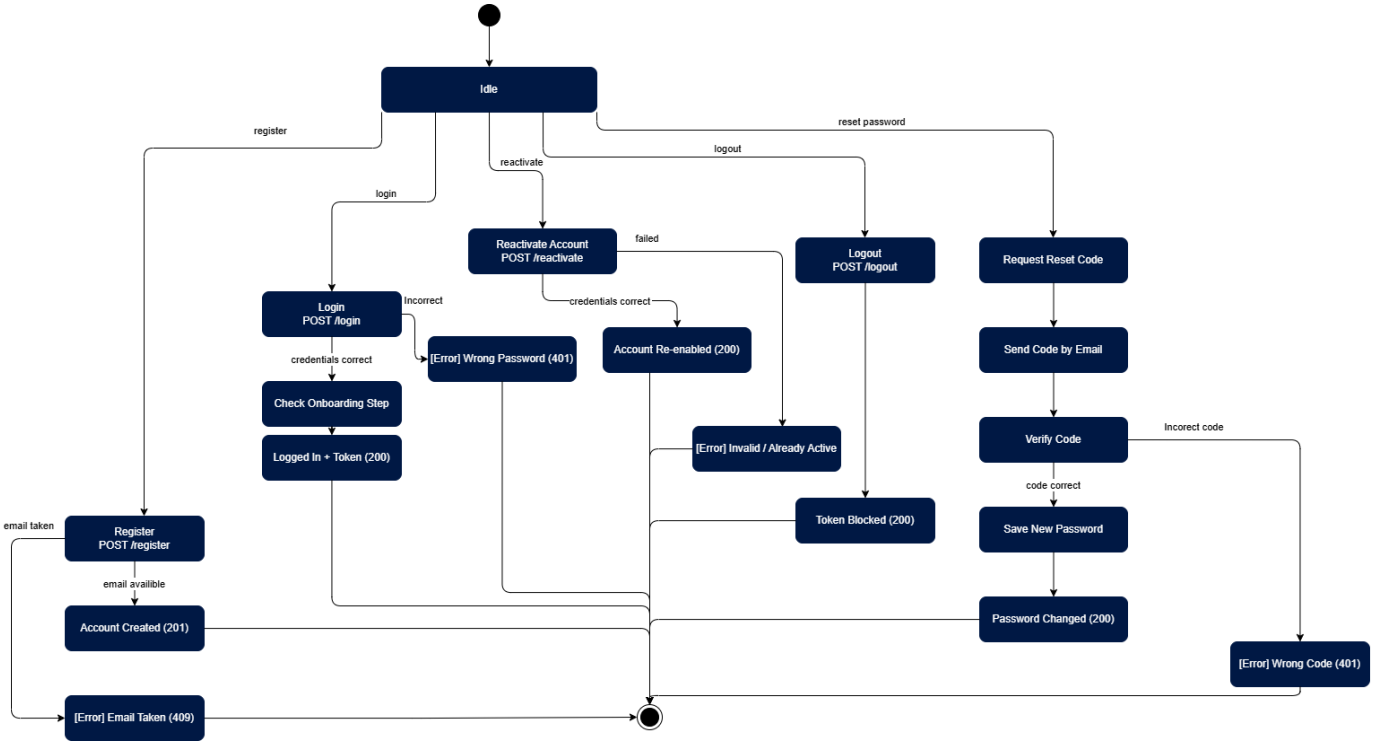


Figure 124: Auth Controller State Diagram

2.11.3 User Controller

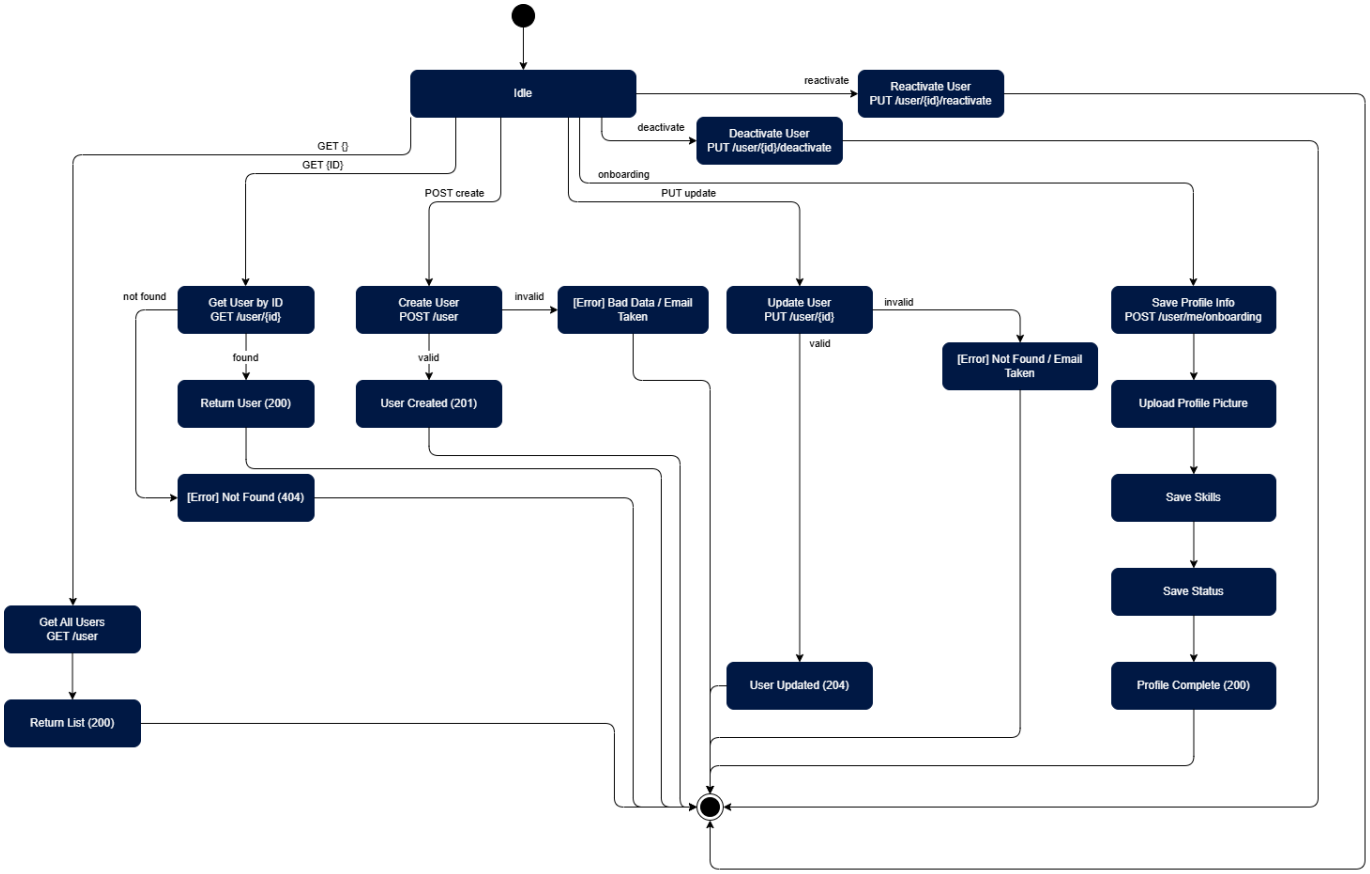


Figure 125: User Controller State Diagram

2.11.4 Project Role Repository

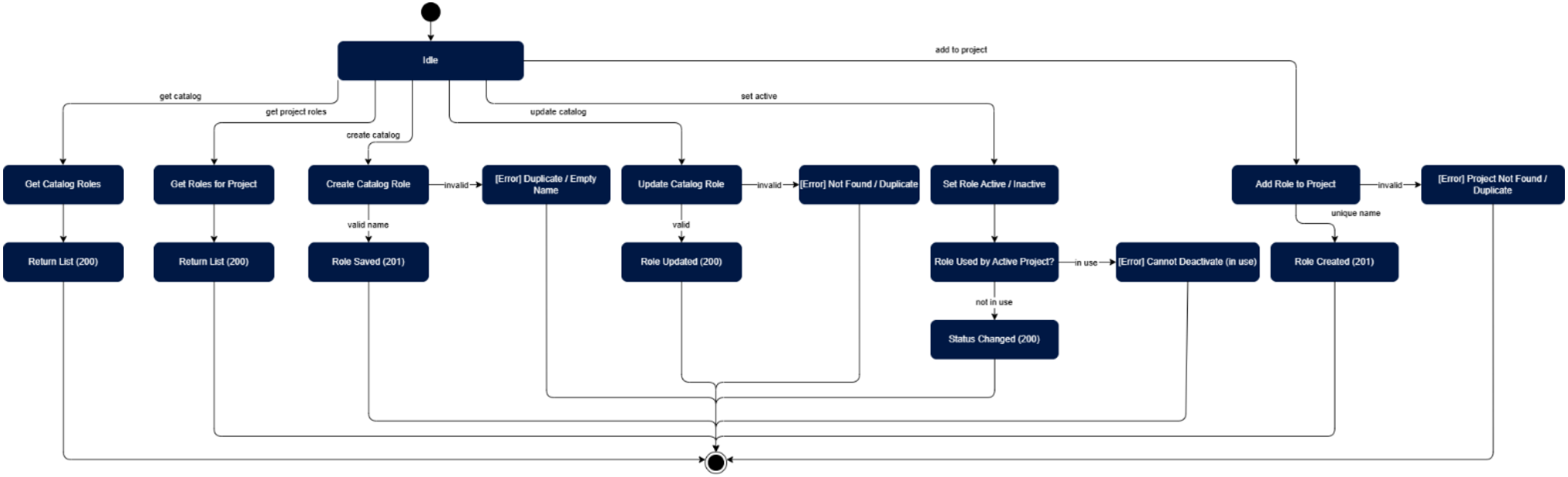


Figure 126: Project Role Repository State Diagram

2.11.5 Group Repository

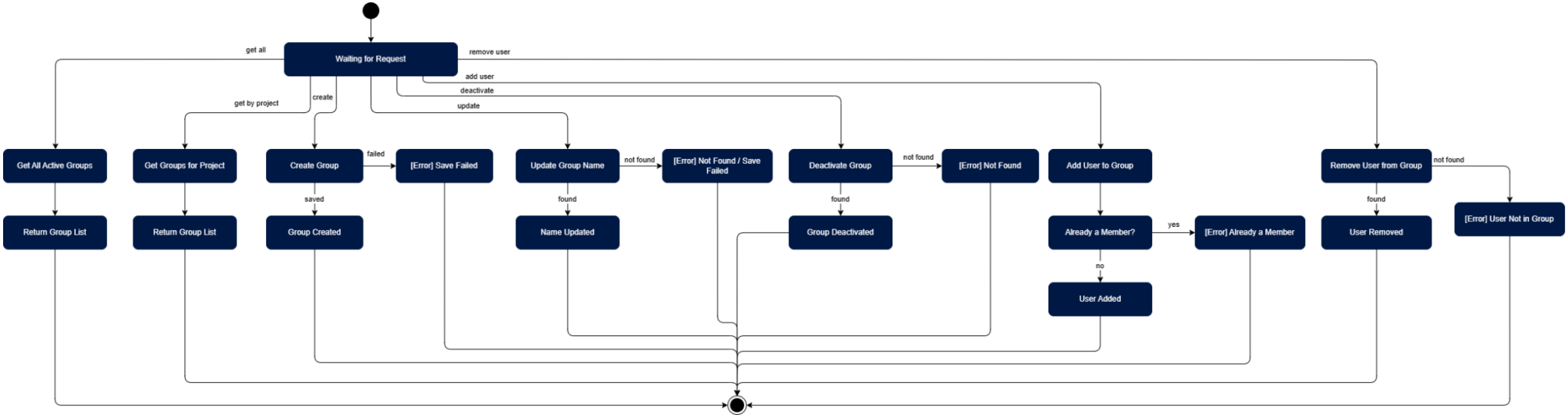


Figure 127: Group Repository State Diagram

3. Document Conclusion

The documentation presented showcased the screen designs for 40 uses assigned to Iteration 4. 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, a link to our GitHub, and the meeting and project planning for Iteration 5.

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: Dané Coetzee

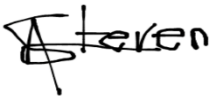
Date: 04 May 2026

5. Sign-off by Team

Team member:

Name: Alexander Steven

Date: 3 May 2025

Signature: 

Team member:

Name: Amahle Khoza

Date: 3 May 2025

Signature: 

Team member:

Name: Christopher Bailey

Date: 3 May 2025

Signature: 

Team member:

Name: Nicholas Bailey

Date: 3 May 2025

Signature: 

Team member:

Name: Stephan Bredenhann

Date: 3 May 2025

Signature: 