

ITERATION 7

TEAM 35



NGO PROJECT PORTFOLIO MANAGEMENT SYSTEM

This document defines the hardware and software that is required to operate the Integrated NGO Management System developed for EPI-USE and groupelephant.com, the technical specifications which are the different technologies used to develop the system, and the network layout.

OUR CLIENT:



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 Figures.....	3
1. Document Introduction	4
2. Hardware and Software Requirements	5
2.1 Introduction.....	5
2.2 Hardware	6
2.2.1 PC / Laptop	6
2.2.2 Tablet	7
2.2.3 Mobile (Smartphone)	7
2.2.4 Printer	9
2.3 Application Specific Software Specifications	10
2.3.1 Visual Studio 2022	10
2.3.2 Visual Studio Code	10
2.3.3 Microsoft SQL Server + Entity Framework Core 8.....	11
2.3.4 Angular Framework (v19)	11
2.3.5 Component-Scoped CSS (Design Tokens).....	13
2.3.6 JWT (JSON Web Tokens).....	13
2.3.7 Chart.js.....	13
2.3.8 Audit Log Service	14
2.3.9 Resend	14
2.3.10 Firebase / Google Cloud Firestore	14
2.3.11 Leaflet, Leaflet.heat & Google Maps API.....	14
2.3.12 QuestPDF & ClosedXML.....	15
2.3.13 Python ML Microservice	15
2.3.14 DeepSeek AI.....	15
2.4 Conclusion	16
3. Network Layout.....	17
3.1 Introduction.....	17
3.2 Network Diagram	18
3.3 Conclusion	19
4. Document Conclusion	20
5. Sign-off by Client	21
6. Sign-off by Team	22

Table of Figures

Figure 1: EPI-USE ERP network layout: client devices, hosted frontend/API, database, ML microservice and third-party integrations.....	18
--	----

1. Document Introduction

This document outlines the hardware and software requirements required for the implementation and operation of the proposed system solution. The hardware requirements section identifies the devices and peripheral equipment necessary for users to access and interact with the system and to support specific system functionalities. The software requirements section details the development and runtime technologies used to implement the system, as well as the external applications, frameworks, and libraries required to support its functionality and ensure effective system operation for EPI-USE and groupelephant.com.

2. Hardware and Software Requirements

2.1 Introduction

In the proposed system for EPI-USE and groupelephant.com, we will be using a combination of hardware, software and external services to achieve the desire functionality.

2.2 Hardware

2.2.1 PC / Laptop

Primary device for administrators, project managers, finance/procurement staff and system administrators

A desktop or laptop is the primary way administrators, finance/procurement staff and project managers use the system: managing users, roles and permissions; running the Project Hub; building custom and structured reports (with PDF/Excel export); reviewing the ML dashboard and audit logs; and administering database backups. It is used purely as a web client and there is no desktop install.

Minimum Requirements

- ☐ **CPU:** Dual-core x86 or ARM processor
- ☐ **Memory:** 4 GB RAM
- ☐ **Storage:** 2 GB free disk space (browser cache and downloaded reports)
- ☐ **Operating System:** Windows 10, macOS 11, or a modern Linux distribution
- ☐ **Display:** 13-inch, 1366 x 768 resolution
- ☐ **Browser:** Latest Chrome, Edge or Firefox — Angular 19 targets evergreen, standards-compliant browsers
- ☐ **Connectivity:** Broadband internet via Wi-Fi or Ethernet

Recommended Requirements

- ☐ **CPU:** Quad-core (Intel i5 / AMD Ryzen 5 or better)
- ☐ **Memory:** 8 GB RAM or more
- ☐ **Storage:** SSD with 10 GB free space
- ☐ **Display:** 15-inch or larger, 1920 x 1080 resolution or higher
- ☐ **Browser:** Latest Chrome or Edge, for best Chart.js and Leaflet map rendering performance
- ☐ **Connectivity:** Stable broadband (10 Mbps or faster) needed for map tiles, PDF/Excel report downloads and the real-time chat

2.2.2 Tablet

Used by warehouse and stock-take staff moving between physical storage locations

A tablet lets warehouse and inventory staff walk the physical storage areas while capturing stock-take counts, receiving procurement deliveries against purchase orders, and looking up inventory and warehouse records without returning to a desk.

Minimum Requirements

- ☐ **CPU:** Quad-core processor
- ☐ **Memory:** 2 GB RAM
- ☐ **Storage:** 16 GB, with 1 GB free for the browser cache
- ☐ **Display:** 8-inch, 1280 x 800 resolution
- ☐ **Operating System:** Android 10 or iOS 15
- ☐ **Camera:** 5 MP, for photographing stock/delivery condition
- ☐ **Connectivity:** Wi-Fi (mandatory)
- ☐ **Battery Life:** 6 hours minimum

Recommended Requirements

- ☐ **CPU:** Octa-core processor
- ☐ **Memory:** 4 GB RAM or more
- ☐ **Storage:** 32 GB or higher
- ☐ **Display:** 10-12 inches, 1920 x 1200 resolution or higher
- ☐ **Operating System:** Latest Android or iOS version
- ☐ **Camera:** 12 MP or higher
- ☐ **Connectivity:** Wi-Fi and LTE (optional, for warehouses without reliable Wi-Fi)
- ☐ **Battery Life:** 10 hours or more

2.2.3 Mobile (Smartphone)

Used by field staff such as K9 handlers, wildlife (rhino/elephant) monitoring units and project staff completing forms on site

A smartphone is used away from the office: K9 handlers and wildlife monitoring staff check animal and project assignments and view the rhino/elephant location heat-maps in the field; project staff complete impact forms and indemnity forms on site.

Minimum Requirements

- ☐ **Device:** Modern smartphone running Android 10 or iOS 15

- ☐ **CPU:** Quad-core ARM-based processor
- ☐ **Memory:** 2 GB RAM
- ☐ **Storage:** 200 MB free for the browser cache
- ☐ **Display:** 4.7 inches, 1280 x 720 resolution
- ☐ **Camera:** 8 megapixels, for wildlife/incident photo capture
- ☐ **Network Compatibility:** 4G, compatible with the carrier's network

Recommended Requirements

- ☐ **Device:** Recent smartphone (released within the last 2-3 years)
- ☐ **CPU:** Multi-core ARM-based processor (octa-core)
- ☐ **Memory:** 4 GB RAM or more
- ☐ **Display:** 5.5 inches or larger, 1920 x 1080 resolution or higher
- ☐ **Camera:** 12 megapixels or higher
- ☐ **Network Compatibility:** 4G/5G
- ☐ **Battery Life:** 3,000 mAh or higher

2.2.4 Printer

Used to print system-generated reports, exports and compliance forms

The printer is used to produce hard copies of the PDF reports rendered by QuestPDF (structured, date-ranged reports and the AI-parsed custom reports), the Excel exports produced by ClosedXML for expenses, and physical copies of indemnity forms and permits where a wet signature or filed hard copy is required for compliance.

Minimum Requirements

- ☐ **Type:** Standard inkjet or laser multifunction printer
- ☐ **Resolution:** 300 x 300 DPI
- ☐ **Connectivity:** USB or network (Wi-Fi / Ethernet)
- ☐ **Paper Support:** A4
- ☐ **Functions:** Print (PDF report printing is the core requirement)

Recommended Requirements

- ☐ **Type:** Network laser multifunction printer (MFP)
- ☐ **Resolution:** 600 x 600 DPI or higher
- ☐ **Duplex:** Automatic duplex, for multi-page structured reports
- ☐ **Connectivity:** Wi-Fi and Ethernet
- ☐ **Functions:** Print, Scan — scanning lets signed indemnity forms/permits be filed back into the system as an attachment

2.3 Application Specific Software Specifications

2.3.1 Visual Studio 2022

Backend IDE - ASP.NET Core 8 Web API

Visual Studio 2022 is used to write, debug and build the backend, EPI-USE_ERP_API, an ASP.NET Core 8 (net8.0) Web API. It was chosen for its first-class .NET debugging, integrated NuGet package management, and built-in Git support.

What the Web API does

The Web API is the single point of access to the system's data and business logic. It exposes around 40 RESTful controllers covering Users, Roles/Permissions, Projects (with a dedicated Project Overview "hub" endpoint), Wildlife (Animals, K9, Rhino, Elephant), Warehouse/Inventory/Procurement/Stock-Take, Expenses/Project Financials, Vendors, FAQ, Impact Forms, Indemnity Forms, Permits, Reports/Structured Reports, Predictions (ML) and Backups, following the standard REST verbs:

- ☐ **GET** – retrieve data, e.g. a project's financial summary or a list of vendors
- ☐ **POST** – create data, e.g. register a new procurement order or submit an impact form
- ☐ **PUT** – update data, e.g. edit a project's role assignments
- ☐ **DELETE** – remove data, e.g. a stock-take line item

Architecturally the API follows Controller → Service → Repository → Entity Framework Core → SQL Server, with cross-cutting services layered in for authentication (JWT), authorisation (a permission-based RBAC model), auditing, email, backups, and the ML/AI integrations described below.

2.3.2 Visual Studio Code

Frontend IDE — Angular 19 application

VS Code is the primary editor for EPI-USE_ERP_FrontEnd, the Angular 19 single-page application. It was chosen for its fast TypeScript/Angular tooling, integrated terminal (used to run ng serve, ng build and ng test), and its Git integration.

Developers use VS Code to build Angular standalone components, wire them to the Web API over HttpClient, and manage application state and routing. The Angular CLI (@angular/cli 19.2) drives day-to-day tasks: ng serves for local development against <https://localhost:7066>, and ng build for the production bundle deployed to Firebase Hosting.

2.3.3 Microsoft SQL Server + Entity Framework Core 8

System of record for every subsystem

SQL Server is the single, centralised relational database behind every modules such as users, roles/permissions, projects, wildlife (animals/K9/rhino/elephant), warehouses, inventory, procurement, stock-takes, expenses, vendors, FAQs, impact/indemnity forms, permits, audit logs and ML training data.

The API talks to it exclusively through Entity Framework Core 8 (Microsoft.EntityFrameworkCore.SqlServer), which maps each table to a C# entity, so the codebase works with objects rather than hand-written SQL for day-to-day CRUD.

Referential integrity

Relationships are enforced with real foreign keys, for example Inventory.VendorId and Procurement.VendorId, Expense.VendorId/ProcurementId, and the K9–Animal 1:1 relationship so that cost, inventory and wildlife data stay consistent as they are edited across different subsystems.

2.3.4 Angular Framework (v19)

Frontend single-page application

Angular 19 (TypeScript-based, maintained by Google) is the foundation of the frontend. The application is built from standalone components, Angular's router for client-side navigation between subsystems (Projects, Wildlife, Warehouse, Reports, Admin, etc.), and injectable services that wrap HttpClient calls to the Web API and attach the JWT to every request.

Key Angular ecosystem packages in use

- ☐ **@angular/fire 19:** Angular bindings for Firebase (used for the real-time group chat)
- ☐ **@fullcalendar/angular 6.1:** calendar views for project deadlines and scheduling
- ☐ **chart.js 4.5:** dashboard and reporting charts
- ☐ **leaflet 1.9 + leaflet.heat:** wildlife and vendor location maps and heat-maps
- ☐ **jwt-decode 4:** reading claims out of the JWT on the client for role-aware UI
- ☐ **rxjs 7.8:** reactive state and HTTP streams

System Requirements (build/dev machine)

- ☐ Node.js compatible with Angular 19 and the Angular CLI 19.2 toolchain
- ☐ 4 GB RAM, 4 GB free storage, admin privileges to install global CLI tools

Common Angular CLI commands used to build the application

- ☐ ng serve — local development server
- ☐ ng generate component <name> — scaffold a new standalone component
- ☐ ng build — production bundle, deployed to Firebase Hosting
- ☐ ng test — Karma/Jasmine unit tests

2.3.5 Component-Scoped CSS (Design Tokens)

Styling approach as there is no Bootstrap/Tailwind dependency

Unlike a Bootstrap-based UI, the frontend does not depend on a third-party CSS framework. Instead, `src/styles.css` defines a global design-token system (CSS custom properties for colour roles such as `--primary`, `--surface`, `--on-surface`, `--outline`, in a Material-inspired palette), the Manrope Google Font, and shared Leaflet map styling; each component then supplies its own scoped `.css` file for layout.

This keeps bundle size down and styling consistent without pulling in unused framework components, at the cost of writing more layout CSS by hand per component.

2.3.6 JWT (JSON Web Tokens)

Authentication and authorisation

Authentication is implemented with `Microsoft.AspNetCore.Authentication.JwtBearer`. On login the API issues a signed JWT (HMAC-SHA, via a configured secret key, issuer `EPI-USE-ERP-API` and audience `EPI-USE-ERP-Client`) containing the user's identity claims. The Angular client stores the token, decodes it client-side with `jwt-decode` to drive role-aware UI, and attaches it as an `Authorization: Bearer` header on every API call.

Authorization uses a global fallback policy that requires an authenticated user by default — endpoints must opt out explicitly with `[AllowAnonymous]` (used only for login, registration, password reset and account reactivation) rather than opting in, so a controller can never be accidentally left open. On top of that sits a permission-based RBAC model: application-wide `AppRoles` plus project-scoped `SystemRoles`, checked per-request by `IPermissionService`.

Tokens also support server-side revocation (`TokenRevocationService` checks the token's `jti` claim on every validated request) and configurable session lifetimes (1, 15, 60, 480 or 1440 minutes), so a session can be forced to expire before the JWT's own expiry.

2.3.7 Chart.js

Dashboards and reporting

Chart.js 4.5 renders the charts used across the Dashboard, Project Financials, structured reports, and the ML Prediction views using the bar, line and pie chart elements driven by data the Angular services pull from the API. It was chosen for its light weight, wide chart-type support and straightforward integration with Angular via plain HTML5 canvas, without needing a heavier charting suite.

2.3.8 Audit Log Service

Traceability and compliance for sensitive actions

Every significant action such as permission changes, backup/restore operations, and other administrative events is written to an append-style audit log (IAuditLogRepository / AuditLogsController) backed by SQL Server. Combined with the RBAC model, this gives administrators a queryable, attributable trail of who did what and when, which the system exposes through its own Audit Logs screen rather than relying on server log files.

2.3.9 Resend

Transactional email API

Outbound emails such as password-reset codes are sent through Resend, a transactional email API, via EmailSenderService. The API key and sender address are supplied through .NET user secrets/environment variables rather than committed configuration. Resend was used instead of an SMTP library because it needs no mail server of its own and gives simple, auditable delivery of a small number of transactional email types.

2.3.10 Firebase / Google Cloud Firestore

Real-time group chat

Google.Cloud.Firestore on the backend and the Firebase JS SDK (via angular/fire) on the frontend together power the system's real-time group chat (ChatController / FirestoreService, Firebase project epi-use-messaging-65b19). Chat groups are created and referenced from the relational database (linked to a Project), but the messages themselves live in Firestore so the browser can subscribe to new messages in real time without polling the ASP.NET Core API.

2.3.11 Leaflet, Leaflet.heat & Google Maps API

Geospatial mapping

Leaflet 1.9 (with the leaflet.heat plugin) renders the Rhino and Elephant location maps and proximity/heat-maps directly in the browser using OpenStreetMap-style tiles, while a Google Maps API key supports geocoding of vendor, project and warehouse addresses into latitude/longitude. Together these back the vendor-proximity recommendation feature (ranking vendors by haversine distance from a project's geocoded locations) and the wildlife-monitoring maps.

2.3.12 QuestPDF & ClosedXML

Report and export generation

QuestPDF renders the system's PDF output: the pre-structured, date-ranged reports (project, financial, inventory, wildlife) and the branded letterhead used across them (configured via ReportBrandingOptions, reading a logo and organisation name from configuration). ClosedXML generates Excel exports, currently used for expense data via ExpensesController. QuestPDF is used under its free Community licence, which the application declares in code (QuestPDF.Settings.License) before the first document is rendered.

2.3.13 Python ML Microservice

FastAPI + scikit-learn prediction service

A small Python microservice (ml-service/app.py, built on FastAPI 0.115 and scikit-learn 1.5) sits alongside the .NET API and powers the Machine Learning Dashboard. It is deliberately domain-agnostic: the .NET side (TrainingDataService, PredictionService) does all the feature engineering and only sends the Python service a numeric feature matrix and target vector. The service fits a model (logistic regression for classification, ridge regression for continuous targets, cross-validated with k-fold/leave-one-out depending on sample size) and returns predictions and diagnostics.

The .NET API talks to it over plain HTTP via a typed HttpClient (IModelServiceClient), configured to the internal address <http://127.0.0.1:8000>.

2.3.14 DeepSeek AI

LLM-assisted natural-language report parsing

The custom report builder accepts a natural-language description of the report a user wants and uses the DeepSeek chat API (model deepseek-v4-flash, called via a dedicated IDeepSeekChatClient) to parse it into a structured report definitions including domain, fields, filters and operators, which is then validated against IReportFieldVocabularyService (the actual distinct values present in the database, e.g. valid provinces or shirt sizes) before being run. This keeps the AI's output constrained to value the report engine can actually query.

2.4 Conclusion

This section of the document concluded the hardware and application software for the Integrated NGO Management System for EPI-USE and groupelephant.com.

3. Network Layout

3.1 Introduction

This section of the document details the network diagram for the Integrated NGO Management System for EPI-USE and groupelephant.com.

3.2 Network Diagram

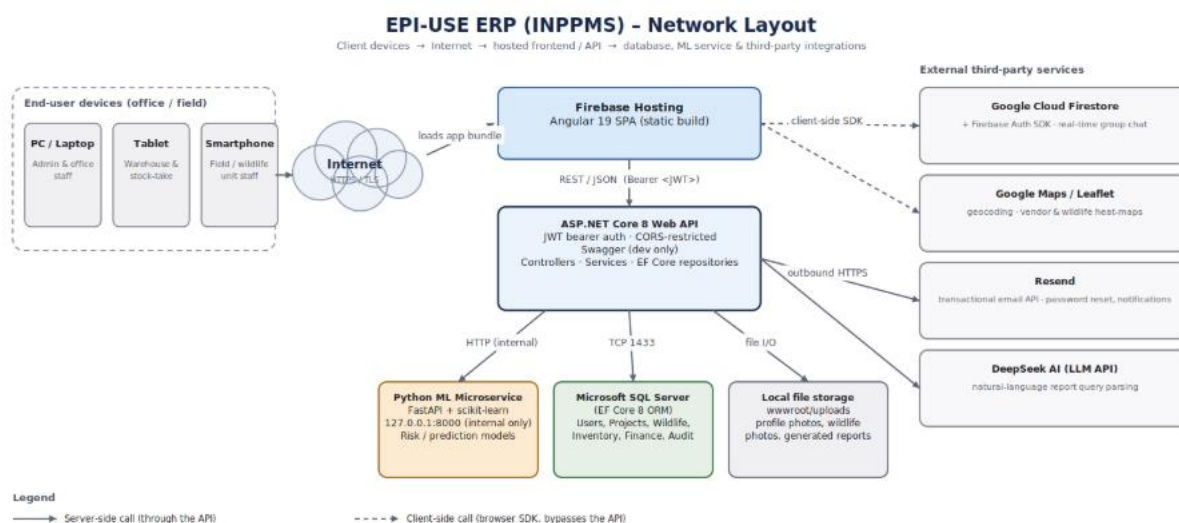


Figure 1: EPI-USE ERP network layout: client devices, hosted frontend/API, database, ML microservice and third-party integrations.

Key points shown in the diagram:

- ❑ **Frontend hosting:** the compiled Angular 19 application is a static build served from Firebase Hosting; the browser downloads it once, then talks to the API directly for data.
- ❑ **API hosting:** the ASP.NET Core 8 Web API is hosted separately from the frontend. Its CORS policy (AllowFrontend) is configuration-driven, so a deployed frontend origin can be permitted without a code change; it falls back to the local Angular dev server (<http://localhost:4200>) when unset.
- ❑ **Authentication in transit:** every API request after login carries a JWT in the Authorization header; requests are also rejected while a database restore is in progress (maintenance mode returns 503).
- ❑ **Database:** the API reaches Microsoft SQL Server over TCP 1433 exclusively through Entity Framework Core; nothing else talks to the database directly.
- ❑ **Internal-only ML service:** the Python FastAPI microservice is only reachable from the API host itself (127.0.0.1:8000), not from the internet or the browser.
- ❑ **Server-side external calls:** the API makes outbound HTTPS calls to Resend (email) and DeepSeek (AI report parsing) on the server side, where API keys stay out of the browser.
- ❑ **Client-side external calls:** the browser talks directly to Google Cloud Firestore/Firebase Auth (chat) and Google Maps/Leaflet tile servers using their JavaScript SDKs, bypassing the ASP.NET Core API entirely for those features.

3.3 Conclusion

This concludes the network diagram for the Integrated NGO Management System for EPI-USE and groupelephant.com.

4. Document Conclusion

This document detailed the hardware and software that would be needed to operate the system which was developed for EPI-USE and groupelephant.com, the technical specifications with the different technologies that were used to develop the system as well as the Network Layout for a visual display of all the technologies used to fully utilise the system.

5. 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: 11/08/2026

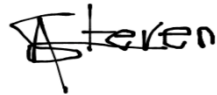
6. Sign-off by Team

Team member:

Name: Alexander Steven

Date: 16 August 2026

Signature:



Team member:

Name: Amahle Khoza

Date: 16 August 2026

Signature:



Team member:

Name: Christopher Bailey

Date: 16 August 2026

Signature:

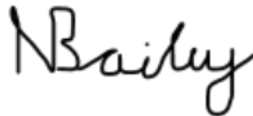


Team member:

Name: Nicholas Bailey

Date: 16 August 2026

Signature:



Team member:

Name: Stephan Bredenhann

Date: 16 August 2026

Signature:

