REPUBLIQUE DU CAMEROUN	REPUBLIC OF CAMEROON
PAIX – TRAVAIL – PATRIE	PEACE – WORK – FATHERLAND
MINISTERE DE L'ENSEIGNEMENT SUPERIEURE	MINISTRY OF HIGHER EDUCATION
UNIVERSITE DE BUEA	UNIVERSITY OF BUEA

FACULTY OF ENGINEERING AND TECHNOLOGY

PO BOX 63

BUEA, SOUTH WEST REGION

DEPARTMENT OF COMPUTER ENGINEERING

COURSE TITLE/CODE: Internet Programming (J2EE) and Mobile Programming – CEF440

COURSE INSTRUCTOR: Dr. NKEMENI Valery

REPORT FOR THE SYSTEM MODELING AND DESIGN (TASK 4)

Presented by **GROUP 13**:

AFAYI MUNSFU OBASE NGALA – FE22A136

ATANKEU TCHAKOUTE ANGE N. – FE22A158

BETNESSI GRACE BLESSING – FE22A446

CHESSEU NJIPDI TERTULLIEN – FE22A181

JAMES ARISTIDE MASSANGO – FE22A226

May 2025

ACADEMIC YEAR: 2024/2025

ABSTRACT

The "Design and Implementation of a Mobile App for the Collection of Quality of Experience (QoE) Data from Mobile Network Subscribers" project addresses persistent challenges faced by mobile network users in Cameroon, including unreliable connectivity, slow internet speeds, and ineffective feedback channels. This document presents a comprehensive system design using Unified Modeling Language (UML) diagrams—Context, Data Flow, Use Case, Sequence, Class, and Deployment Diagrams—to model the app's architecture, interactions, and workflows. The proposed solution enables subscribers to report issues, provide feedback, and monitor network performance, while operators gain actionable insights through an analytics dashboard. Emphasizing privacy, scalability, and usability, the system leverages Flutter for cross-platform development, Firebase for backend services, and SQLite for offline support. The design adheres to MVC principles and incorporates key design patterns to ensure robustness and maintainability.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION

1.1 PROJECT BACKGROUND

Mobile network subscribers in Cameroon face persistent challenges with service quality including unreliable connectivity, weak signals, slow internet speeds, call drop, lack of transparency and ineffective feedback channels. These pain points degrade the Quality of Experience of users and hinder operator's ability to prioritize infrastructure improvements.

In response to these various challenges, the project titled "Design and Implementation of a mobile app for the collection of Quality of Experience (QoE) data from mobile network subscribers" aims at bridging the gap between subscribers' experiences and operator responses through a mobile application. The mobile application will be able to collect QoE data, empower the users and support operators.

1.2 PURPOSE OF THE DOCUMENT

This document serves as a comprehensive guide to the design and modeling of the QoE mobile application. It presents a structured approach to system design through Unified Modeling Language (UML) diagrams, including Context, Dataflow, Use Case, Sequence, Class, and Deployment Diagrams. These diagrams collectively model the system's interactions, data flow, user behaviors, internal structure, and physical deployment. The document ensures traceability between design elements and project requirements, facilitating development, stakeholder validation, and academic evaluation. It also documents design decisions, constraints, and considerations to provide a clear roadmap for implementing the system.

1.3 SCOPE OF THE PROJECT

The project consists in designing and implementing a mobile app for collecting QoE data from the mobile network subscribers. The mobile app will enable issue reporting, feedback surveys, metric collection, anomaly detection, operator analytics, authentication and offline support.

We need to take note that the QoE data will be used to generate insights for mobile network operators through an analytics dashboard, allowing for better service planning, rapid issue resolution, and transparent communication with their subscribers. The app also emphasizes low resource consumption, privacy, offline-first design, and user accessibility, particularly for low-literacy users.

1.4 INTENDED AUDIENCE

This section intends to recall the intended audience of this project which can also be referred to as the stakeholders of the project. They include: mobile network subscribers, network operator, project team (designers and developers of the system) and lecturer (evaluator of the project).

1.5 DEFINITIONS & ABBREVIATION

The purpose of this section is to ensure consistent terminology across diagrams, clarify technical jargon for non-technical stakeholders and provide adequate references for SRS alignment.

Abbreviations

- **QoE**: Quality of Experience, combining user satisfaction and technical metrics.
- **SRS**: Software Requirements Specification.
- **UML**: Unified Modeling Language for system design visualization.
- **GPS**: Global Positioning System, used for anonymized location data.
- **FR**: Functional Requirement.
- **NFR**: Non-Functional Requirement.
- **UI**: User Interface.
- **UX**: User Experience.
- **MVP**: Minimum Viable Product.

Definitions

- **QoE**: A measure of user satisfaction with mobile network services, combining objective metrics and subjective feedback.
- **Objective QoE data**: Quantifiable network metrics collected automatically from the mobile network subscribers' device.
- **Subjective QoE data**: This refers to user-reported feedback that can be done through feedback surveys.
- **Heatmap**: Visual representation of network issues aggregated by geographical location.
- **Background service**: This refers to an app component running discretely to collect network performance metrics.

2.0 SYSTEM OVERVIEW

The purpose of this section of the document is to provide a high-level description of the system’s architecture, stakeholders, and functionality.

2.1 SYSTEM DESCRIPTION

The QoE mobile application is designed to enhance mobile network service quality in Cameroon by collecting and analyzing Quality of Experience (QoE) data from subscribers. The system enables users to report network issues (e.g., slow speeds) and provide feedback via surveys, while automatically logging technical metrics like signal strength and latency. Anonymized data is processed to generate actionable insights for network operators, presented through interactive dashboards and heatmaps. The app operates on Android and iOS devices, with a cloud-based backend for real-time data management and offline support for unreliable network conditions.

2.2 KEY FEATURES & CONSTRAINTS

We can briefly outline the system’s core features in modules. These modules include:

- User reporting and feedback
- Automated network monitoring
- User experience management
- Operator analytics
- Authentication and security

Talking about the system’s constraints, the system adheres to the following constraints:

- **Privacy**: The system shall anonymize sensitive data & secure data transmission.
- **Scalability**: The system shall support concurrent usage by 25+ users.
- **Usability**: The system shall provide a simple and intuitive UI for users.
- **Data Integrity**: The system shall ensure proper accuracy in data transmission.
- **Performance**: Background operations shall make minimal battery/data usage.

2.3 TECHNOLOGY STACK

The system will be leverage based on the following technologies:

- **Frontend & some logics:** Flutter for cross-platform development.
- **Backend**: Firebase for authentication, real-time database, and analytics.
- **Local storage:** SQLite for offline data caching.
- **Figma:** For UI/UX design.
- **GitHub actions:** For continuous Integration/Deployment (CI/CD).

3.0 MODELING AND DESIGN METHODOLOGY/APPROACH

3.1 MODELING METHODOLOGY

The modeling and design phase serves as the architectural blueprint of the mobile application, translating the requirements gathered during the requirement gathering phase into a visual, structured representation of the system's behavior, components, and deployment. This approach ensures that development proceeds with clarity, consistency, and technical precision. We adopted an iterative and user-centric design process. The modelling and design of the mobile app was done using the **Unified Modeling Language (UML)**, a standardized framework for visualizing and documenting system architecture, behavior, and interactions.
The modeling process involves iterative refinement, starting with high-level system boundaries and progressing to detailed structural and behavioral designs. This approach facilitates collaboration among the project team, supports academic evaluation, and provides a clear blueprint for the development phase.

3.2 UML DIAGRAM SELECTION

Six (6) UML diagrams were assigned to comprehensively model the system, each addressing specific aspects of the design. We need to note that each of these diagrams offers a unique perspective of the system and serves a specific purpose in guiding implementation. These diagrams include: **context diagram, data flow diagram (DFD), use case diagram, sequence diagram, class diagram** and **deployment diagram**.

Table 1: UML Diagram for the system modeling and design

Diagrams	Purpose
Context diagram	Defines system's boundaries and interactions with external entities, providing a high-level overview of the system's data flow.
Data flow diagram	Illustrate the flow of data by describing internal system processes, data sources/sinks, external entities and movement of information. It tells how QoE data is collected, stored & analyzed.
Use case diagram	It captures the functional interactions between system actors (example: network subscribers) and system functionalities (example: issue reporting).
Sequence diagram	Models the flow of events or time-ordered interactions for key use cases, highlighting object interactions and system behavior.
Class diagram	Represents the system's static structure or object-oriented structure, including classes, attributes, operations and relationships.
Deployment diagram	Represents the physical distribution of system components across devices - hardware and software components.

3.3 DESIGN PRINCIPLE

The design of the system adheres to the following principles to ensure robustness, maintainability, and user satisfaction.

- **Modularity**: Classes and components have single responsibilities, reducing complexity and enabling independent updates.
- **Abstraction**: Interfaces and abstract classes promote extensibility, allowing future feature additions like AI diagnostics.
- **Encapsulation**: Private attributes and methods protect data integrity and enforce privacy constraints.
- **Loose Coupling**: Components interact via well-defined interfaces, enhancing scalability for 25+ concurrent users.
- **Usability**: The simple and intuitive interface and offline support ensure accessibility for Cameroon's diverse user base.
- **Performance**: Background operations are optimized to consume ≤3% battery per hour, meeting efficiency requirements.

These principles were implemented using the Model-View-Controller (MVC) architectural pattern, which organizes the application's logic, user interface, and user interactions into distinct roles, streamlining development and maintenance. The **Model-View-Controller (MVC)** architectural pattern is a design framework that separates an application into three interconnected components: **Model**, **View**, and **Controller**. This separation of concerns enhances modularity, maintainability, and testability, making it well-suited for developing structured and scalable applications. It's in the regard of defining an MVP that we decided to choose the MVC architecture.

3.4 DESIGN PATTERNS

The following design patterns are employed to address specific challenges because they support scalability, maintainability, and efficient data handling, and align with the system's technical constraints:

- **Singleton**: Ensures a single instance of the authentication manager, streamlining user and operator login processes.
- **Repository**: Abstracts data access (e.g., for Network Metric storage), decoupling business logic from Firebase and SQLite storage.
- **Observer**: Manages real-time notifications using Flutter's state management, ensuring responsive user updates.

3.5 TOOLS

The modeling and design process leverage the following software tools: draw.io and **StarUML.**

4.0 SYSTEM DIAGRAM

4.1 CONTEXT DIAGRAM

4.1.1 Purpose

The **Context Diagram** provides a **high-level overview** of the QoE (Quality of Experience) Mobile App System, illustrating **key interactions** between **external entities** (users, operators, devices, authentication providers) and the **internal system components** (Mobile App, Backend Server, Operator Dashboard).

4.1.2 Importance of this Diagram.

- **Clarifies System Scope:** Shows what's inside (Mobile App, Backend Dashboard) vs outside (Users, Operators, Auth Providers)
- **Identifies Critical Interactions:** Highlights how data moves between users, devices, and the backend.
- **Ensures Requirement Coverage:** Confirms all key features (feedback, monitoring, notifications) are included.
- **Supports Development and Testing:** Guide developers in building the correct data flows and helps testers verify end-to-end processes (example: login -> feedback -> analytics).

4.1.3 Context Diagram Visualization

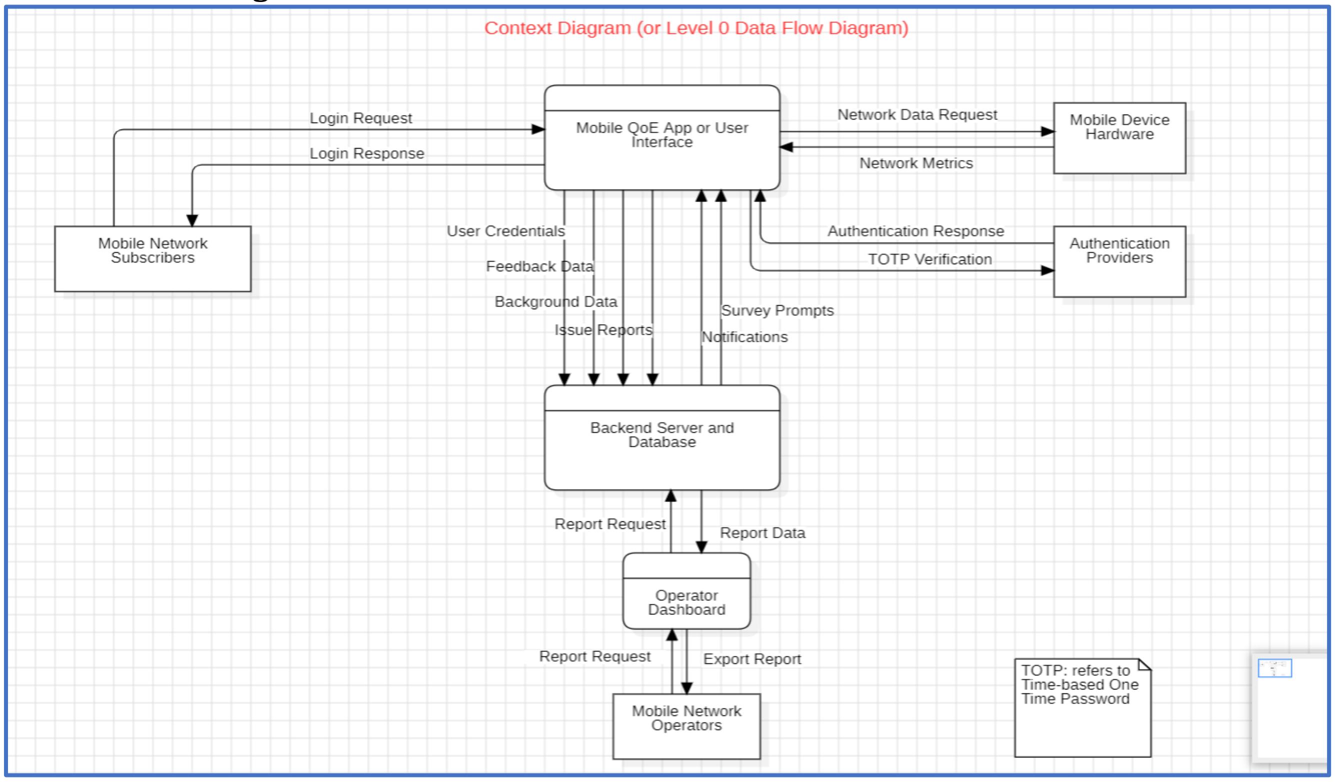

Figure 1: Context diagram (or Level 0 Data Flow Diagram) visualization

4.1.4 Context Diagram Description

A. Key Components of the Diagram.

I. External Entities (Out of the System Boundary)

These are the **sources and destinations** of data that interact with the system but are not part of it.

1. Mobile Network Subscribers (Users)

Role: Primary users who submit feedback, report issues, and receive notifications.

Key Interaction:

i. **Login Request:** Sends credentials (mobile number) to the Mobile App.

ii. **Feedback Data:** Submits surveys and manual issue reports.

iii. **Receives:** Survey prompts, notifications, and network status updates.

2. Mobile Network Operators (Admins)

Role: Administrators who analyze network performance and manage reports.

Key Interaction:

i. **Report Request:** Requests analytics from the Operator Dashboard.

ii. **Export Report:** Downloads reports (CSV/PDF) for analysis.

3. Mobile Device Hardware

Role: Provides real-time network metrics (signal strength, speed, GPS).

Key Interaction:

i. **Network Metrics:** Sends automated background data to the Mobile App.

4. Authentication Providers (example: Firebase Auth)

Role: Handles user verification via **TOTP (Time-based One-Time Password)**.

Key Interaction:

i. **TOTP Verification:** Validates login requests and sends an authentication response.

II. Internal System Components (Inside the System Boundary)

These represent the **core modules** of the QoE Mobile App System.

1. Mobile App (User Interface)

Role: The frontend application used by subscribers.

Key Functions:

i. User Authentication: Receives login requests, verifies via TOTP.

ii. Feedback Collection: Captures survey responses and issue reports

iii. Background Data Collection: Gathers network metrics from the device.

iv. Displays: Network status and notifications.

2. Backend Server and Database

Role: Stores and processes all collected data

Key Functions:

i. Stores: User credentials, feedback, network metrics.

ii. Triggers: Survey prompts and notifications.

iii. Sends Data: To the Operator Dashboard for analytics.

3. Operator Dashboard

Role: Admin interface for analyzing network performance.

Key Functions:

i. Generates Reports: Based on aggregated

ii. Exports Reports: Allows operators to download CSV/PDF files.

B. Key Data Flows in the Diagram

Table 2: Data flows in the system and its descriptions

Data flow	Source → Destination	Description
Login request	Subscriber → Mobile App	User initiates login with mobile number
TOTP verification	Mobile App → Auth Provider	App requests OTP validation
Authentication response	Auth Provider → Mobile App	Confirms successful login
User credentials	Mobile App → Backend Server	Stores verified user details
Feedback data	Subscriber → Mobile App → Backend	User submits surveys/reports
Background data	Device → Mobile App → Backend	Automated network metrics collection
Network metrics	Device → Mobile App	Real-time signal/speed data
Survey prompts	Backend → Mobile App → Subscriber	Triggers feedback reminders
Notifications	Backend → Mobile App → Subscriber	Alerts for issues/resolutions
Report request	Operator → Dashboard → Backend	Request analytics data
Report data	Backend → Dashboard → Operator	Send aggregated reports
Export report	Dashboard → Operator	Download reports (CSV/PDF)

C. How does this Diagram align with Project Requirements?

The **Level 0 DFD** maps directly to the **functional requirements** from the document:

- **User Authentication (TOTP Login)** → Ensures secure access.
- **Feedback Collection (Surveys & Reports)** → Captures user-reported issues.
- **Background Monitoring (Network Metrics)** → Automates data logging.
- **Real-Time Notifications** → Keeps users informed.
- **Operator Dashboard (Analytics & Reports)** → Helps admins improve network quality.

Note: The context diagram is the level 0 Data Flow Diagram (DFD), so in the next section of this document, we will start the design of the Data Flow Diagram but at level 1.

4.2 DATA FLOW DIAGRAM (DFD)

4.2.1 What is a Data Flow Diagram (DFD)?

This is a modeling diagram used to define the system's boundaries and provides a clear graphical representation of how data moves through the entire system. It is used to complement the use case diagram by defining the flow of data in user scenarios and stories developed from this use case diagram there by offering a more intuitive and accessible view.

4.2.2 Importance of DFDs and its relevance to the QoE Mobile App

1. Visualize Data Flow: DFD helps to create a mind-map of how data moves through the QoE Mobile App. For example, it illustrates how user feedback flows from the user's mobile device, through the app, to the backend server and is stored in the database. This visual representation makes it easier to understand the sequence of events and data transformations within the system.

2. Establish System Boundaries: DFDs help define the boundaries of the QoE Mobile App system.

3. Identifying System Processes: DFDs break down the System (QoE Mobile App) into key processes, Level 1 DFD, for instance, details processes like User Authentication, Feedback Collection, Issue Reporting, Background Monitoring, Data Aggregation, and Analytics & Visualization, making it easier to grasp the system's functionalities.

4. Identifying Data Stores: DFDs explicitly shows where data is stored. In the QoE Mobile App, data stores like the User Profile, Feedback DB, Network Metrics Log, and Location Data Store are clearly identified, helping to understand how data is organized and accessed.

5. Communicating with Stakeholders: DFDs serve as a communication tool for the QoE Mobile App project. They help developers, operators and other stakeholders understand the system's data flow, ensuring everyone is on the same page regarding requirements and design.

Other importance of DFDs include; **Analyzing and Designing Systems, helps in System's Documentation, provide support for other modelling techniques (**like the user stories development**).**

4.2.3 Design Considerations for DFDs in the QoE Mobile App Project

- **Clarity and Simplicity:** DFDs are designed to be relatively clear, with processes and data flows labeled. Decomposing the system into Level o and Level 1 diagrams helps manage complexity with Level 1 Diagram giving more details about system processes.
- **Consistent Symbols and Notation:** DFDs should use a consistent notation, which is essential for avoiding confusion.
- **Meaningful Labels:** Always give clear, simple and descriptive labels to any DFD component (Data store, Data flow, Entity or Processes).
- **Data Conservation:** DFDs should show how data is transformed and moved through the system, highlighting how data inputs are processed and result in specific outputs.
- **Balance between Detail and Complexity:** When designing DFDs, you have a clear understanding on how you will balance details and complexity, presenting only essential components to best fit how data moves through the system in the mind of stakeholders.
- **Focus on data Flow, Not Control Flow:** DFDs should rather focus on how data changes states when it moves through different stages of the system rather than the sequence of control.
- **Numbering and Referencing:** Data Stores and Processes should be labeled when designing DFDs, this is to ease readability and understanding.

4.2.4 Purpose of the Level 1 DFD

In this section, we described the **Level 1 DFD** which is essential to understand the core **change of state** and **flow of data** in the system**.**

This Diagram uses the Level 0 DFD as context to come out with more detail information about the system like key data stores, other internal entities, system's processes and data flows relating all these components. Suitable to visualize the change of state of data in various system's components.

4.2.5 Level 1 DFD Visualization

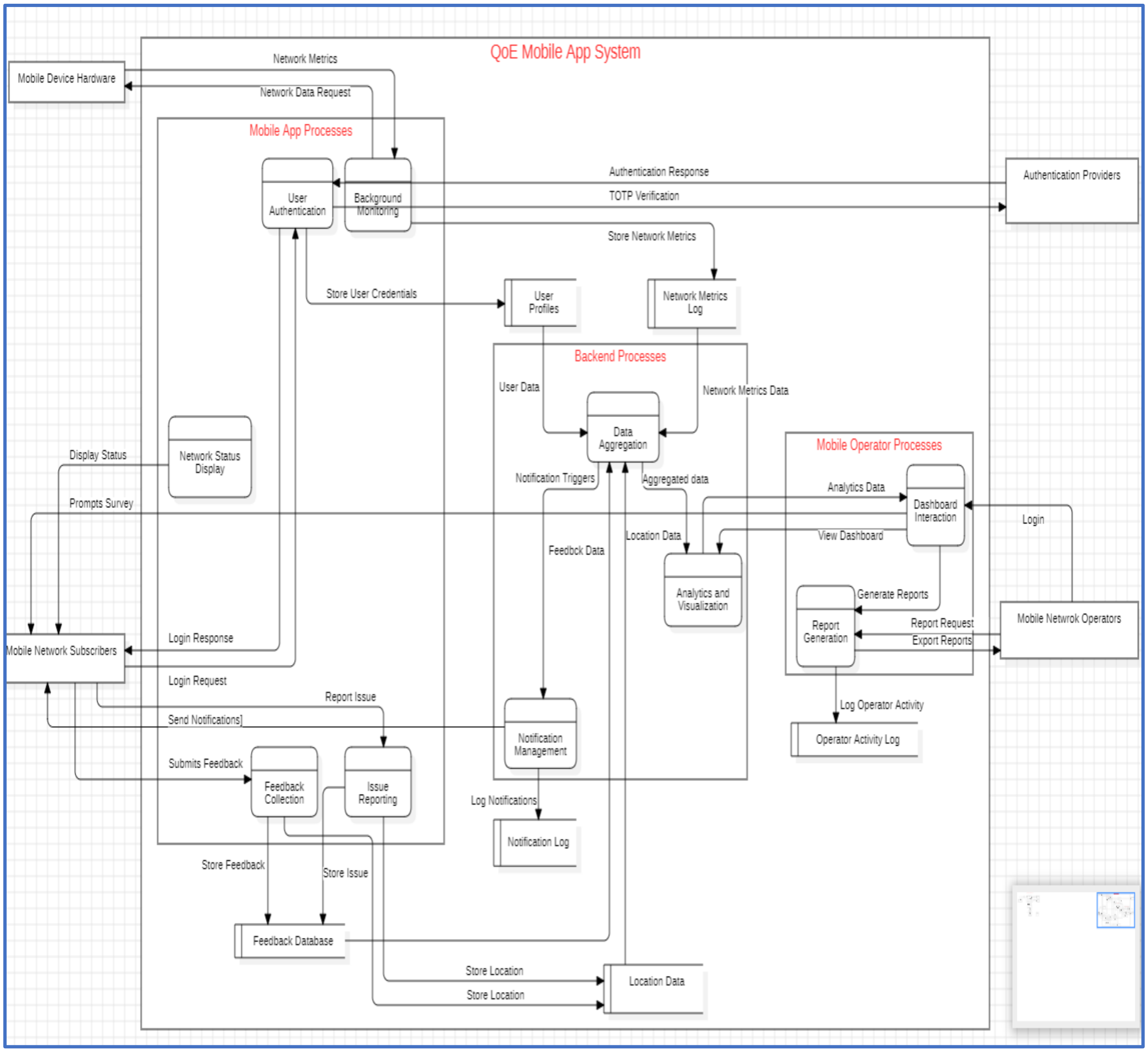

Figure 2: Level 1 Data Flow Diagram (DFD) visualization

4.2.6 Level 1 DFD Description

This diagram provides a decomposed view of the system (obtained from **the context Diagram**), showing how data moves between internal processes, data stores, and external entities.

A. Overall System Boundaries

The entire diagram is enclosed within a large rectangle labeled "QoE Mobile App System", clearly defining the scope of the system.

I. External Entities (out of the system boundary)

- Mobile Device Hardware: This entity provides Network Metrics data to the system and receives Network Data Request from the system.

- **Mobile Network Subscribers (Users):** These users initiate Login Request and receive Login Response, Display Status, and Prompt Survey from the system. They also send Submits Feedback and Report Issue to the system, and receive Send Notifications from it.

- **Authentication Providers:** This external service sends Authentication Response and TOTP Verification data to the system. The system also sends User Credentials to it.

- **Mobile Network Operators:** These operators send Login requests to the system and receive View Dashboard and Export Reports from it. They also send Report Request to the system and receive Generated Reports from it.

II. Internal System Components (within the app's system boundary)

1. Mobile App Processes

➢ **User Authentication:** Receives Login Request from Mobile Network Subscribers and Authentication Response from Authentication Providers. It sends User Credentials to Authentication Providers and Store User Credentials to User Profiles data store. It also generates Login Response back to Mobile Network Subscribers.

➢ **Background Monitoring:** Receives Network Metrics from Mobile Device Hardware and sends Store Network Metrics to Network Metrics Log data store. It also sends Network Data Request to Mobile Device Hardware.

➢ **Network Status Display:** Receives Display Status from Mobile Network Subscribers (this flow seems reversed, typically the system *sends* display status to users). It also receives data from Mobile App Processes (the specific source isn't explicitly labeled but implies internal data from monitoring or feedback).

➢ **Feedback Collection:** Receives Prompt Survey from Mobile Network Subscribers (this flow seems reversed, typically the system *sends* a prompt). It sends Submits Feedback to Mobile Network Subscribers (also seems reversed, users *submit* feedback). It also sends Store Feedback to Feedback Database.

➢ **Issue Reporting:** Receives Report Issue from Mobile Network Subscribers (this flow seems reversed, users *report* issues). It sends Store Issue to Feedback Database.

2. Backend Processes

This section illustrates the server-side logic and data management.

- **Data Aggregation:** Receives User Data (likely from User Profiles and Feedback Database), Network Metrics Data (from Network Metrics Log), and Location Data (from Location Data store). It produces Aggregated Data.
- **Analytics and Visualization:** Receives Aggregated Data from Data Aggregation and Location Data from Location Data store. It sends Analytics Data" to Dashboard Interaction.
- **Notification Management:** Receives Notification Triggers (likely from Data Aggregation or Analytics and Visualization) and sends Log Notifications to Notification Log data store. It also sends Send Notifications to Mobile Network Subscribers.

3. Mobile Operator Processes

This section shows the functionalities that are accessed by network operators.

- **Dashboard Interaction:** Receives Login from Mobile Network Operators and Analytics Data from Analytics and Visualization. It sends View Dashboard" to Mobile Network Operators. It also sends Report Request to Report Generation and receives Generated Reports from it. It also sends Log Operator Activity to Operator Activity Log.
- **Report Generation:** Receives Report Request from Dashboard Interaction and sends Generated Reports and Export Reports to Mobile Network Operators.

B. Data Stores (internal to the System)

- **User Profiles:** Stores Store User Credentials from User Authentication and provides User Data to Data Aggregation.
- **Network Metrics Log:** Stores Store Network Metrics from Background Monitoring and provides Network Metrics Data to Data Aggregation.
- **Feedback Database:** Stores Store Feedback from Feedback Collection and Store Issue from Issue Reporting. It provides Feedback Data to Data Aggregation.
- **Location Data:** Stores Store Location data from Feedback Database and Issue Reporting (implied, as both involve location). It provides Location Data to Data Aggregation and Analytics and Visualization.

- **Notification Log:** Stores Log Notifications from Notification Management.
- **Operator Activity Log:** Stores Log Operator Activity from Dashboard Interaction.

C. Key Data Flows

Table 3: Key data flows in the system

Data Flow	Source → Destination	Description
Login Request	Mobile Network Subscribers → User Authentication	User initiates login with credentials (e.g., mobile number).
Login Response	User Authentication → Mobile Network Subscribers	System confirms successful authentication.
User Credentials	User Authentication → Authentication Providers	Sends credentials for verification (e.g., OTP).
Authentication Response	Authentication Providers → User Authentication	Returns verification result (success/failure).
TOTP Verification	Authentication Providers → User Authentication	Validates Time-based One-Time Password.
Store User Credentials	User Authentication → User Profiles	Saves authenticated user data (e.g., mobile number, preferences).
Network Metrics	Mobile Device Hardware → Background Monitoring	Collects real-time signal strength, speed, etc.

Network Data Request	Background Monitoring → Mobile Device Hardware	Requests updated network metrics from the device.
Store Network Metrics	Background Monitoring → Network Metrics Log	Logs automated network performance data.
Prompt Survey	Feedback Collection → Mobile Network Subscribers	Triggers periodic survey prompts (every 2 days).
Submits Feedback	Mobile Network Subscribers → Feedback Collection	User provides survey responses or manual feedback.
Store Feedback	Feedback Collection → Feedback Database	Saves survey data with timestamps.
Report Issue	Mobile Network Subscribers → Issue Reporting	User submits structured issue reports (e.g., "no signal").
Store Issue	Issue Reporting → Feedback Database	Logs issue details (type, timestamp).
Store Location	Issue Reporting → Location Data	Attaches GPS coordinates to issues/feedback.
User Data	User Profiles → Data Aggregation	Provides user metadata for contextual analysis.

Network Metrics Data	Network Metrics Log → Data Aggregation	Supplies automated metrics for aggregation.
Feedback Data	Feedback Database → Data Aggregation	Combines survey/issue reports for analysis.
Location Data	Location Data → Data Aggregation	Adds geospatial context to aggregated data.
Aggregated Data	Data Aggregation → Analytics and Visualization	Processes raw data into trends/heatmaps.
Notification Triggers	Analytics and Visualization → Notification Mgm't	Alerts for reminders/resolutions based on rules.
Log Notifications	Notification Management → Notification Log	Records sent notifications (timestamp, recipient).
Send Notifications	Notification Management → Mobile Network Subscribers	Pushes alerts (e.g., "Issue resolved").
Analytics Data	Analytics and Visualization → Dashboard Interaction	Supplies visualized insights (graphs, heatmaps) to operators.
Login	Mobile Network Operators → Dashboard Interaction	Operator authenticates to access the dashboard.

View Dashboard	Dashboard Interaction → Mobile Network Operators	Displays analytics interface with filters.
Report Request	Dashboard Interaction → Report Generation	Operator initiates report generation (e.g., CSV/PDF).
Generated Reports	Report Generation → Mobile Network Operators	Delivers formatted reports for download.
Export Reports	Report Generation → Mobile Network Operators	Allows saving reports locally.
Log Operator Activity	Dashboard Interaction → Operator Activity Log	Tracks admin actions (ex: report exports) for auditing.

4.3 USE CASE DIAGRAM

4.3.1 Definition

A use case diagram is a type of behavioral diagram in the Unified Modeling Language (UML) that visually represents the interactions between external actors and the system under consideration. It models the functional requirements of a system by identifying the users (actors) and the various tasks (use cases) they perform in collaboration with the system

4.3.2 Importance

The importance of a use case diagram lies in its ability to:

- Clearly define system boundaries and scope.
- Identify and document system requirements from the user's perspective.
- Facilitate communication between stakeholders (developers, clients, testers).
- Serve as a foundation for further system analysis, design, and validation.

4.3.3 Purpose

The purpose of this use case diagram is to model the functional interactions between key actors and the system within the context of a mobile application developed for collecting Quality of Experience (QoE) data from mobile network subscribers. This diagram captures the high-level functional requirements and defines how different stakeholders interact with the application to fulfill both user-facing and administrative responsibilities.

4.3.4 System Boundary

The system boundary defines the scope of the application and distinguishes what is internal (automated system functions and user interface logic) from what is external (human users and hardware devices).

The system is labeled as: **QoE Data Collection Mobile Application**

All use cases are encapsulated within this boundary, representing features offered by the system. Actors such as the subscriber, network operator, and administrator interact with the system from outside its boundary.

Actors

Primary Actors

- **Network subscriber –** Interacts with the app to provide QoE feedback.
- **Network operator –** Accesses and manages data collected via the app.
- **Mobile device –** Automatically monitors & collects network metrics in the background.

Secondary Actor

- **System Administrator –** Validates credentials for network operators to ensure secure access.

Use Cases

Table 4: Use cases of the system and their description

Use case	Description
Provide Phone Number	Subscriber is assigned a UserID using their mobile number.
Provide Permissions	Subscriber grants access to phone features such as GPS, data usage, etc.
Report Issues Manually	Subscriber logs complaints about poor network experience.
Answer Periodic Questions	Subscriber answers scheduled feedback questions on network performance.
Submit Feedback	Subscriber voluntarily shares additional feedback.
View Heatmap	Subscriber and operator view a map visualizing network performance by location.
Access Dashboard	Subscriber views performance metrics and personal data history.
Monitor Network Conditions	The mobile device observes and forwards network behavior (e.g., signal loss, connectivity) to the system in real time.
Provides Network Metrics	The mobile device collects structured, performance-related data and sends it to the system.
Receive Notifications	Subscriber receives alerts and periodic prompts from the system.

Operator Login	Network operator logs into the platform.
Authenticate Operators	Administrator validates operator credentials.
View Reports	Operator can see reports generated by the system based on collected QoE data.
Export Reports	Operator downloads or shares generated reports.
View Feedback Analytics	Operator analyzes data via dashboards and visualizations.
Update Resolved Issues	Operator marks issues as resolved in the system.

Relationships and Interactions

Includes:

- Operator Login → includes → Authenticate Operators

Extends:

- Submit Feedback → may extend → Report Issues Manually
- Submit Feedback → may extend → Answer Periodic Questions
- Export Reports → may extend → View Reports
- View Reports → may extend → View Feedback Analytics

Justification of Actor Placement

Table 5: Justification of actor placement

Actor	Justification
Network Subscriber	Main user of the application; responsible for providing QoE data.
Network Operator	Uses collected data for analysis and reporting.
System Administrator	Only intervenes to authenticate operators.
Mobile Device	Non-human actor; essential for passive background monitoring.

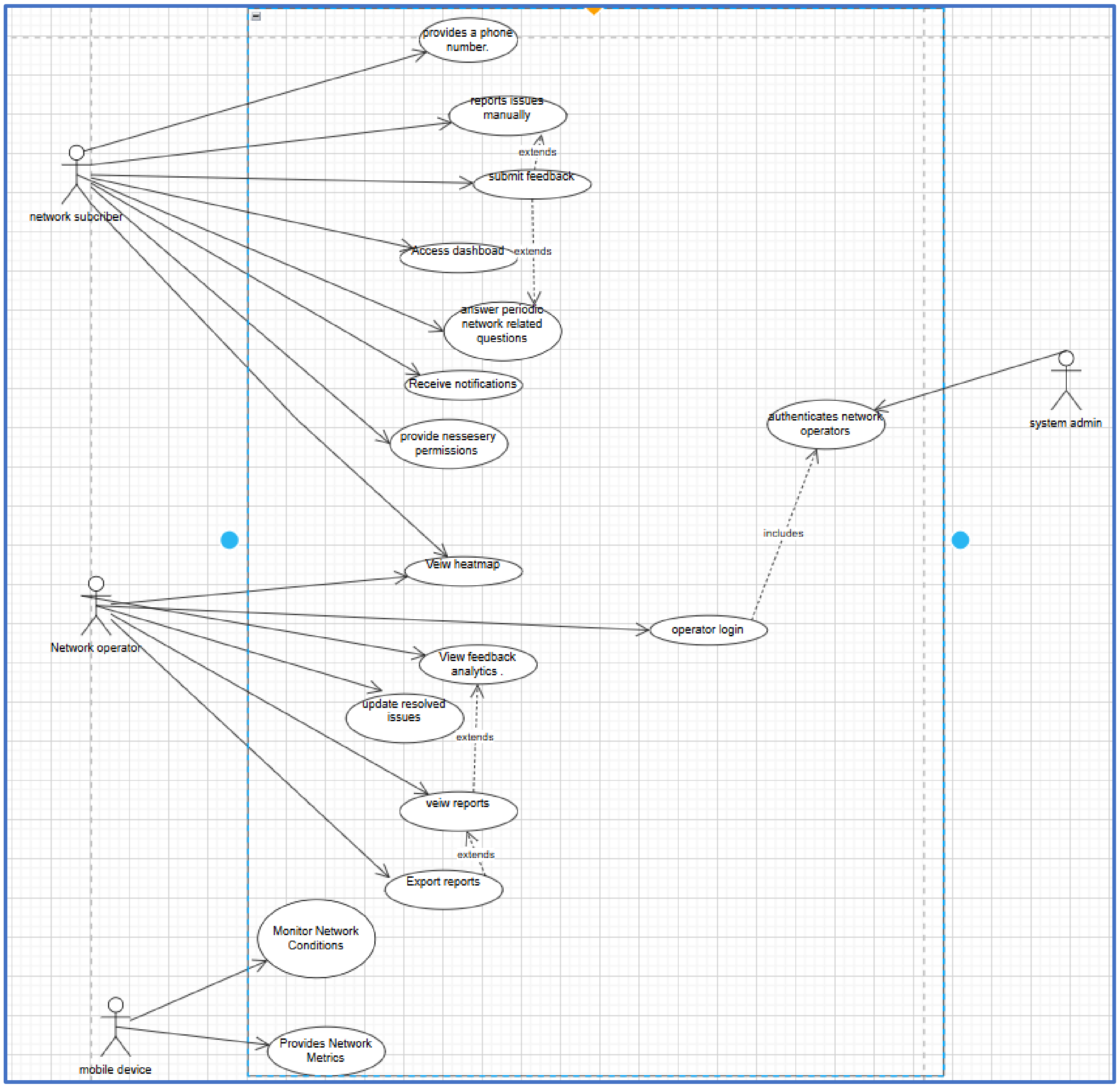

Figure 3: Use case diagram of the system

This use case diagram follows UML modeling standards as outlined in ONF TR-514 and Visual Paradigm guidelines. It captures the functional behavior of the mobile application by showing interactions between users and system functions.

4.4 SEQUENCE DIAGRAM

4.4.1 Purpose of the sequence diagram

The sequence diagram is used to model the dynamic behavior of a system by illustrating how objects interact with each other to perform a specific use case or scenarios. Its primary purpose includes:

- Showing interaction flow
- Capturing system behavior
- Supporting design and development
- Validating requirements
- Facilitating communications

4.4.2 Design considerations

We need to focus on overarching principles that ensure consistency, align with the project's requirements, and address its technical and user-centric constraints.

General design considerations were made for the sequence diagrams and they are outlined as follows:

- Alignment with functional and non-functional requirements
- Performance optimization and resource efficiency
- Security and privacy compliance
- Usability for low-literacy users
- Scalability for concurrent users

4.4.3 Sequence diagrams visualizations

a) Subscriber registration/Login sequence diagram

In the design of this sequence diagram, the following key architectural decisions were made:

Key architectural decisions:

1. **Progressive permissioning**
    - Location/camera permissions requested before registration
    - Prevents mid-flow permission dialogs that increase abandonment
2. **Defensive input validation**

    - Cameroon-specific phone number validation (+237 prefix check)
    - Early rejection of malformed numbers saves SMS costs
3. **Authentication resilience**
    - 5-minute TOTP validity window balances security and usability
    - Resend capability handles SMS delivery failures
4. **Session management**
    - JWT token storage enables passwordless re-authentication
    - Token invalidation not shown (would require logout flow)

Technical constraints addressed:

- TLS 1.3 for all external communications
- Firebase Auth rate limiting considerations
- Mobile carrier SMS reliability assumptions

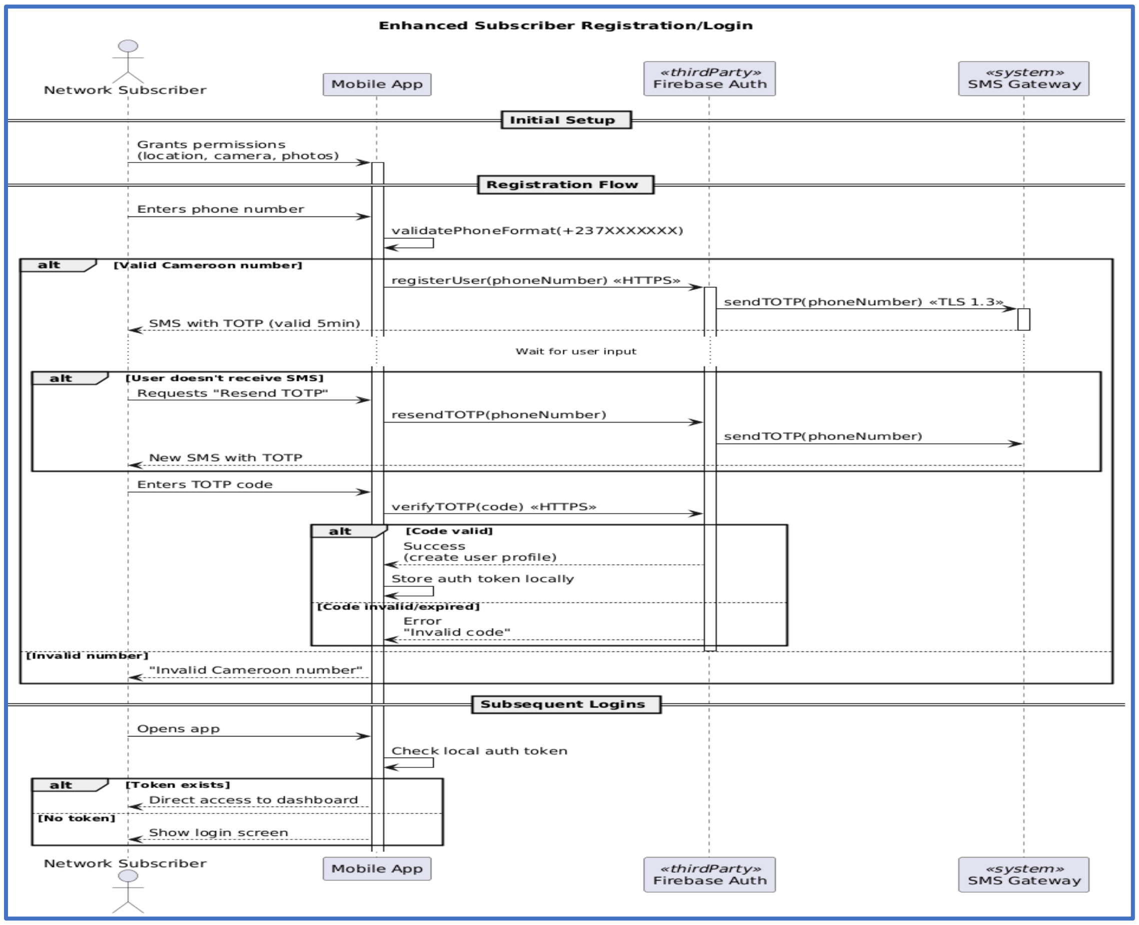

Figure 4: Sequence diagram for the registration and login of network subscribers

b) Issue reporting and feedback sequence diagram

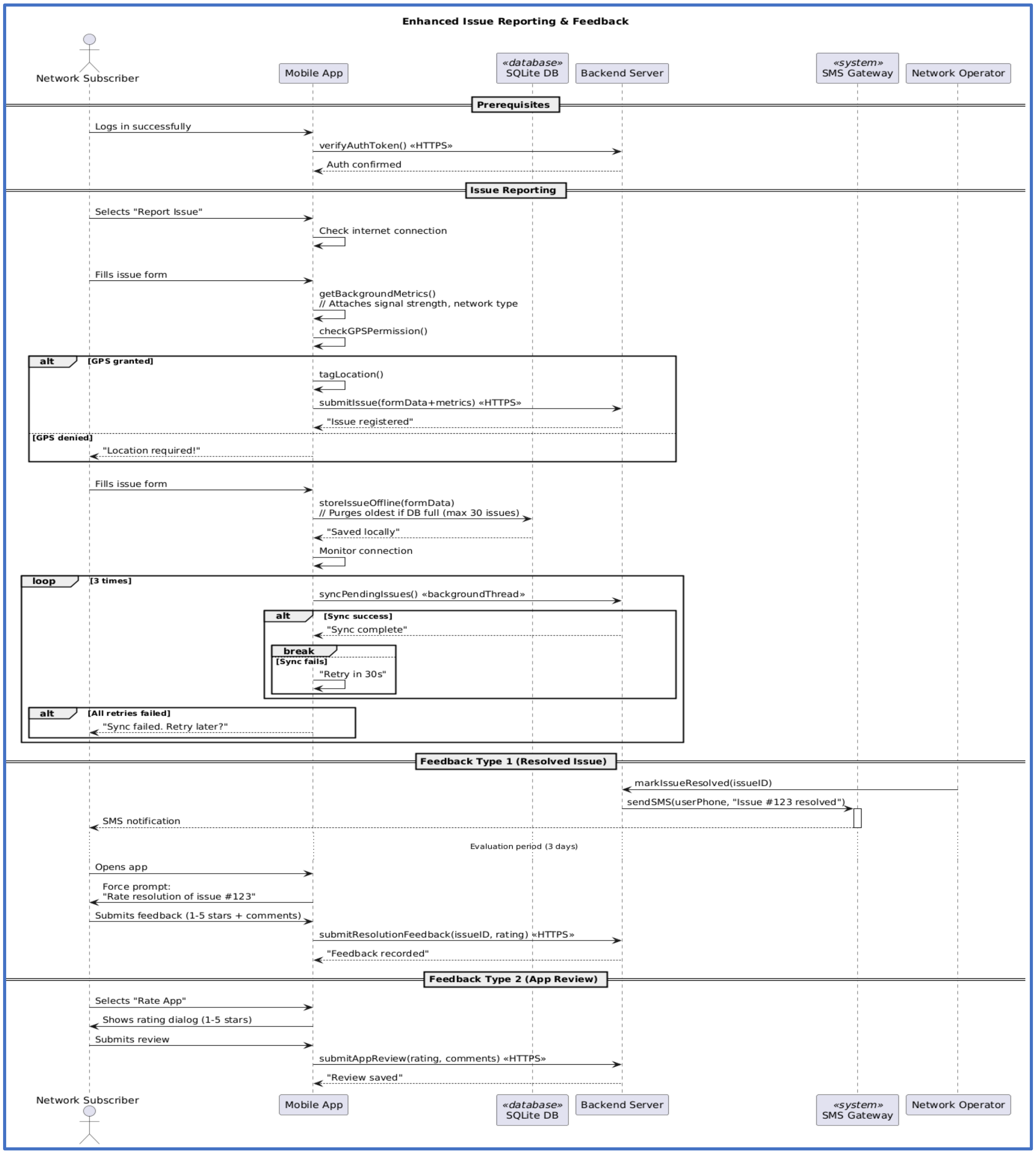
Enhanced Issue Reporting & Feedback
Network Subscriber
Mobile App
«database» SQLite DB
Backend Server
«system» SMS Gateway
Network Operator
Prerequisites
Logs in successfully
verifyAuthToken() «HTTPS»
Auth confirmed
Issue Reporting
Selects "Report Issue"
Check internet connection
Fills issue form
getBackgroundMetrics() // Attaches signal strength, network type
checkGPSPermission()
alt [GPS granted]
tagLocation()
submitIssue(formData+metrics) «HTTPS»
"Issue registered"
[GPS denied]
"Location required!"
Fills issue form
storeIssueOffline(formData) // Purges oldest if DB full (max 30 issues)
"Saved locally"
Monitor connection
loop [3 times]
syncPendingIssues() «backgroundThread»
alt [Sync success]
"Sync complete"
break [Sync fails]
"Retry in 30s"
alt [All retries failed]
"Sync failed. Retry later?"
Feedback Type 1 (Resolved Issue)
markIssueResolved(issueID)
sendSMS(userPhone, "Issue #123 resolved")
SMS notification
Evaluation period (3 days)
Opens app
Force prompt: "Rate resolution of issue #123"
Submits feedback (1-5 stars + comments)
submitResolutionFeedback(issueID, rating) «HTTPS»
"Feedback recorded"
Feedback Type 2 (App Review)
Selects "Rate App"
Shows rating dialog (1-5 stars)
Submits review
submitAppReview(rating, comments) «HTTPS»
"Review saved"

Figure 5: Sequence diagram for issue reporting and feedback survey

In the design of the sequence diagram for the issue reporting and feedback survey, the following critical design choices were taken into considerations:

1. **Offline resilience**
    - SQLite storage with FIFO purge (30-issue limit)
    - Exponential backoff in sync retries (3 attempts)
2. **Automated diagnostics**
    - Background metrics attached to manual reports:
        - Signal strength (dBm)
        - Network type (4G/5G/WiFi)
        - Timestamped logs
3. **Feedback timing**
    - 3-day evaluation period for resolved issues
    - Force prompt ensures 100% feedback capture
4. **Resource optimization**
    - Background thread isolation for sync
    - GPS lazy-loading only when reporting

Trade-offs acknowledged:

- No battery optimization tags shown
- SMS single point of failure
- No data compression for sync

c) Background monitoring of network metrics sequence diagram

The design of this sequence diagram incorporates several general and specific considerations to meet the project's requirements and constraints:

1. **Periodic Monitoring for Efficiency**: Metrics are collected at regular intervals (e.g., every few minutes) to balance data freshness with resource usage. This prevents continuous polling, which could drain the device's battery. Less frequent monitoring might miss transient network issues, but it ensures the app meets the performance constraint of consuming ≤3% battery per hour.
2. **Background Execution**: The process runs in a background thread (Monitor Module) to avoid blocking the app's user interface, ensuring a seamless user

experience. This aligns with the project's use of background services for metric collection.

3. **Minimal Data Collection**: Only essential metrics (signal strength, network speed, latency) are collected to reduce data transmission and processing overhead, addressing the constraint of minimal data usage.

4. **Modularity and Decoupling**: The Monitor Module is a distinct component within the Mobile App, interacting with the Backend Server via well-defined interfaces. This adheres to the design principle of loose coupling, making the system easier to maintain and extend.

5. **Data Storage and Scalability**: Metrics are stored in the Network Metrics Log, a dedicated data store, ensuring scalability for large volumes of data and supporting the system's requirement for 25+ concurrent users.

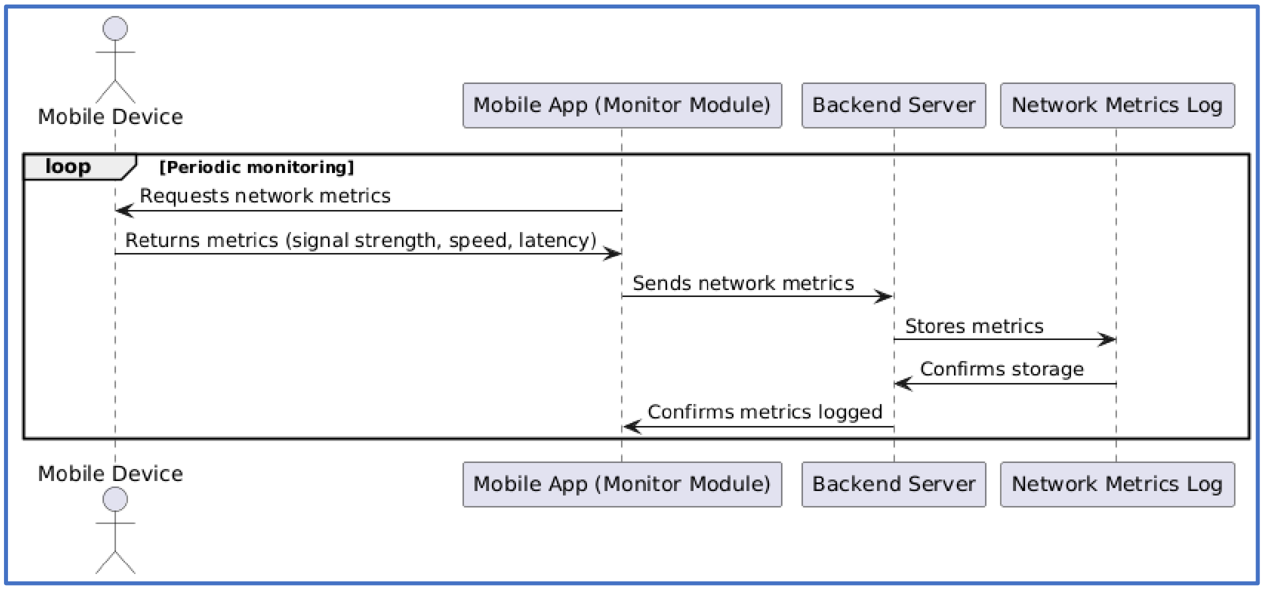

Figure 6: Network metrics background monitoring sequence diagram

d) View and download network analytic data from dashboard sequence diagram

1. **Authentication and Security**: The operator's interaction assumes prior authentication (e.g., via Firebase Auth, as in "Subscriber Registration/Login"), ensuring secure access to the dashboard. A secure communication channel is established between the Operator Dashboard and Backend Server.

2. **Data Aggregation for Performance**: Analytics data is pre-aggregated by the Backend Server to reduce load time on the Operator Dashboard, supporting scalability for concurrent users.

   Trade-off: Pre-aggregation may introduce slight delays in real-time updates but ensures efficient rendering of trends and heatmaps.

3. **Export Flexibility**: Reports can be generated in CSV or PDF formats to meet diverse operator needs, as specified in the use case diagram.

   Design Choice: The Report Generation Module handles format generation, decoupling it from analytics processing.

4. **Scalability and Efficiency**: Uses GraphQL (implied from Section 4.6.4) for efficient data fetching between the Dashboard and Backend Server, supporting the system's scalability requirement.
   Trade-off: GraphQL requires additional setup but reduces data overhead compared to REST.
5. **User Feedback and Usability**: The Operator Dashboard provides immediate feedback by displaying analytics and delivering reports, enhancing operator trust and efficiency.
   Design Choice: Clear message flows (e.g., "Displays analytics," "Delivers report") ensure a straightforward interaction.

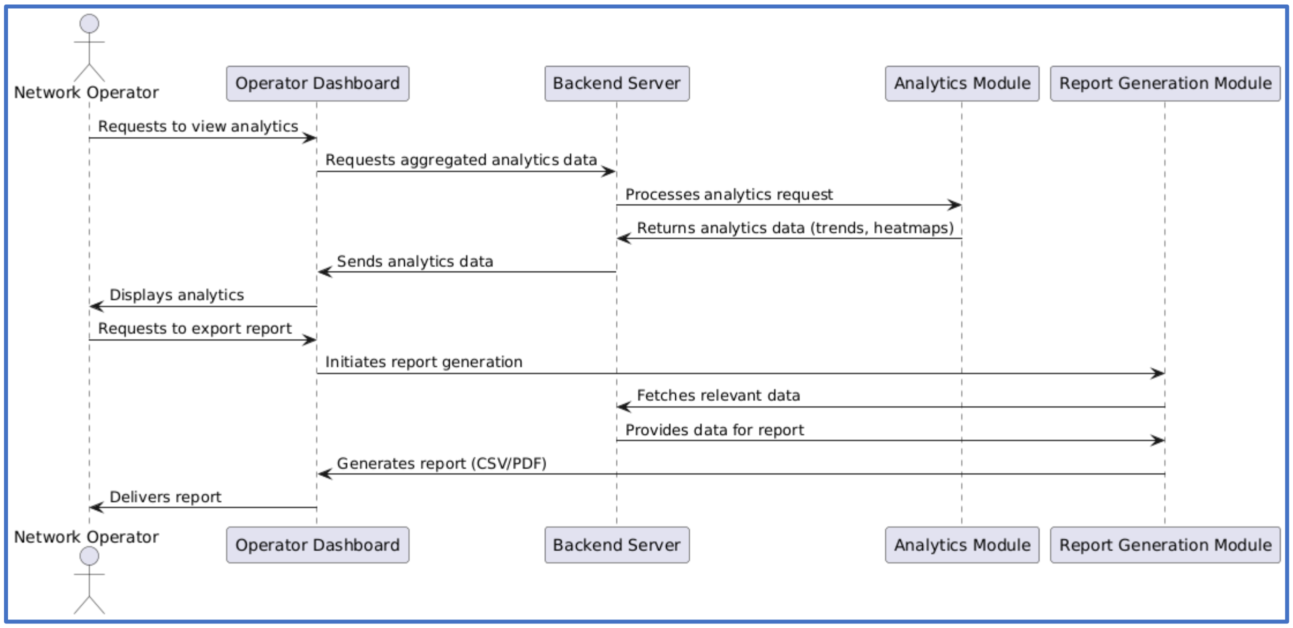
Network Operator
Operator Dashboard
Backend Server
Analytics Module
Report Generation Module
Requests to view analytics
Requests aggregated analytics data
Processes analytics request
Returns analytics data (trends, heatmaps)
Sends analytics data
Displays analytics
Requests to export report
Initiates report generation
Fetches relevant data
Provides data for report
Generates report (CSV/PDF)
Delivers report

Figure 7: View & download of network analytics from dashboard sequence diagram

4.5 CLASS DIAGRAM

4.5.1 Purpose of the class diagram

The Class Diagram defines the static structure of the QoE mobile application, modeling its core entities, attributes, operations, and relationships. It ensures a modular, extensible design that aligns with system's requirements, supporting the Model-View-Controller (MVC) architectural pattern for clear separation of concerns.

4.5.2 Design considerations

The design and modeling of the class diagram of the system involved following key considerations to accurately represent the system's structure, relationships and behaviors. Below are the key considerations that were taken into account when generating the class diagram of the system.

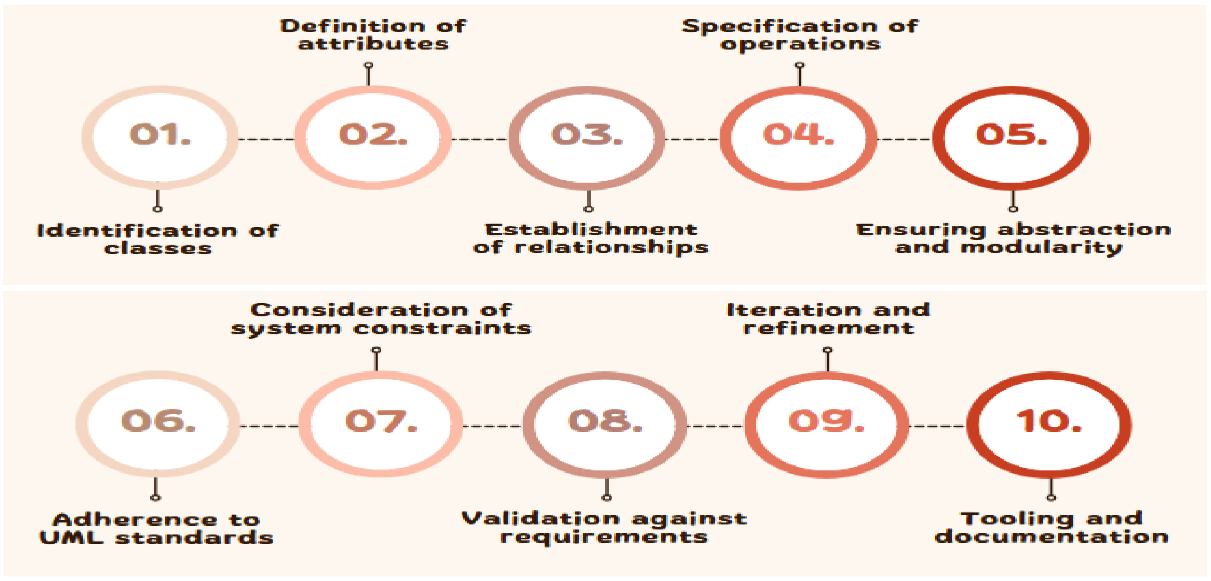
01.
Identification of classes
02.
Definition of attributes
03.
Establishment of relationships
04.
Specification of operations
05.
Ensuring abstraction and modularity
06.
Adherence to UML standards
07.
Consideration of system constraints
08.
Validation against requirements
09.
Iteration and refinement
10.
Tooling and documentation

Figure 8: Steps involved in the design of the system's class diagram

1. Identification of classes

Core entities were identified based on system requirements:

1. **User**: Represents mobile network subscribers reporting issues.
2. **NetworkOperator**: Represents administrators accessing analytics.
3. **IssueReport**: Captures user-reported network issues.

4. **FeedbackSurvey**: Collects user satisfaction data.
5. **NetworkMetric**: Logs technical network data.
6. **Notification**: Manages alerts to users.
7. **OperatorDashboard**: Visualizes analytics for operators.
8. **GPSLocation**: Tags reports with location data.
9. **Authentication**: Manages user and operator sessions.

2. Definition of attributes

Attributes describe each class's state, encapsulated as private properties:

Table 6: Definition of attributes the classes

Classes	Attributes
User	userID, userName, location, operator, preferredLanguage
NetworkOperator	operatorID, operatorName (MTN, Orange, CAMTEL)
IssueReport	reportID, userID, issueType, description, timestamp, status, activityPerform
FeedbackSurvey	surveyID, userID, questions, rating, timestamp
NetwrokMetric	metricID, userID, signalStrength, networkSpeed, latency, networkType, timestamp
Notification	notificationID, type, message, timestamp
OperatorDashboard	dashboardId, visualizationType, filter
GPSLocation	locationID , latitude, longitude
Authentication	sessionID, role, lastLogin

3. Establishment of relationships

Relationships define how classes interact, using UML notations:

Association (navigable, with multiplicity):

- User submits IssueReport (1 to *).
- User completes FeedbackSurvey (1 to *).
- IssueReport is tagged with GPSLocation (1 to 1).
- IssueReport is associated with NetworkOperator (1 to 1).

- NetworkMetric is logged for User (* to 1).
- Notification is sent to User (* to 1).
- NetworkOperator accesses OperatorDashboard (1 to *).
- Authentication manages sessions for User and NetworkOperator (1 to 1).
- OperatorDashboard displays data from IssueReport, FeedbackSurvey, and NetworkMetric (1 to *).

Aggregation: OperatorDashboard aggregates IssueReport, FeedbackSurvey, and NetworkMetric (shared data, independent existence).

Composition: IssueReport contains GPSLocation (cannot exist independently).

Inheritance: Notification may have subclasses ResolvedIssueNotification and SurveyNotification.

4. Specification of operations

Table 7: Definition of the operations of the classes

Classes	Attributes
User	- getUserDetails ( ): Retrieves user details. - submitIssueReport ( ): Allows the user to submit a new issue report. - completeFeedbackSurvey ( ): Allows the user to complete a feedback survey. - viewNotification ( ): Allows users to view notifications.
NetworkOperator	- registerOperator ( ): Authenticates the operator. - getOperatorDetails ( ): Retrieves operator details. - accessDashboard ( ): Grants access to the operator dashboard. - respondToIssueReport ( ): Allows the operator to respond to issue reports. - sendNotification ( ): Sends notifications to users.
IssueReport	- createReport ( ): Creates a new issue report. - updateStatus ( ): Updates the status of an issue report (e.g., pending, resolved). - addActivityPerform ( ): Adds a log of performed activities - getReportDetails ( ): Retrieves issue report details.
FeedbackSurvey	- createSurvey(): Creates a new feedback survey. - submitResponse(): Submits a user's response to the survey. - getSurveyDetails(): Retrieves survey details. - getSurveyResponse(): Retrieves a user's response.
NetwrokMetric	- logMetric(): Logs a new network metric. - getMetricsByUser(userID): Retrieves network metrics for a specific user.
Notification	- createNotification(): Creates a new notification. - sendNotification(): Sends the notification to a user.

Classes	Attributes
OperatorDashboard	- generateDashboard(): Generates the dashboard with selected visualizations. - applyFilter(): Applies filters to the data displayed on the dashboard. - getDashboardData(): Retrieves the data for the dashboard.
GPSLocation	- getLocation(): Retrieves the GPS location. - setLocation(): Sets the GPS location.
Authentication	- login() - logout() - authenticate(): Authenticates a user or operator. - createSession(): Creates a new session upon successful authentication. - getSessionDetails(): Retrieves session details. - updateLastLogin(): Updates the last login timestamp.

5. Ensuring abstraction and modularity

- **Single Responsibility**:
  - User manages user actions (reporting, feedback).
  - NetworkMetric handles technical data logging.
  - OperatorDashboard focuses on analytics visualization.
- **Decoupling**:
  - Authentication centralizes access control, reducing dependencies.
  - GPSLocation is composed within IssueReport, preventing direct access by other classes.
- **MVC Alignment**: Classes are organized to fit the MVC pattern:
  - **Model**: User, IssueReport, NetworkMetric, FeedbackSurvey, GPSLocation, Authentication manage data and logic.
  - **View**: OperatorDashboard (visualizations), UI components for User (Flutter widgets).
  - **Controller**: Mediates interactions (e.g., submitIssueReport updates Model, refreshes View).

6. Adherence to UML standard

The diagram uses standard UML notations: rectangles for classes with compartments for attributes and operations, symbols for relationships: lines for associations, filled diamonds for composition, empty diamonds for aggregation, visibility notations: public and private, multiplicity and navigability arrows.

7. Consideration of system constraints

Constraints are embedded in the design of the class diagram. For example: Scalability is addressed by decoupling data collection (NetworkMetric) from visualization (OperatorDashboard).

8. Validation against requirements

This process ensures that the designed class diagram for the system covers all the FRs and NFRs of the system. In order to proper verify that, a traceability matrix was generated. All the FRs were fully addressed by the class diagram. Classes such as User, IssueReport and OperatorDashboard directly support core functionalities. The NFRs were embedded in the class diagram. The class diagram adheres to the MVC patterns.

9. Iteration and refinement

The diagram was developed iteratively with an initial design which focuses on core classes. Later on, the class diagram can be refined by adding other classes for extensibility based on evolving requirements. Team member feedback also helped in the designing process in order to ensure that the class diagram align with the use cases.

10. Tooling and documentation

draw.io was used to create the UML class diagram, enabling precise modeling and collaboration. The diagram includes clear notation in order to ease its interpretation.

4.5.3 Class diagram visualization

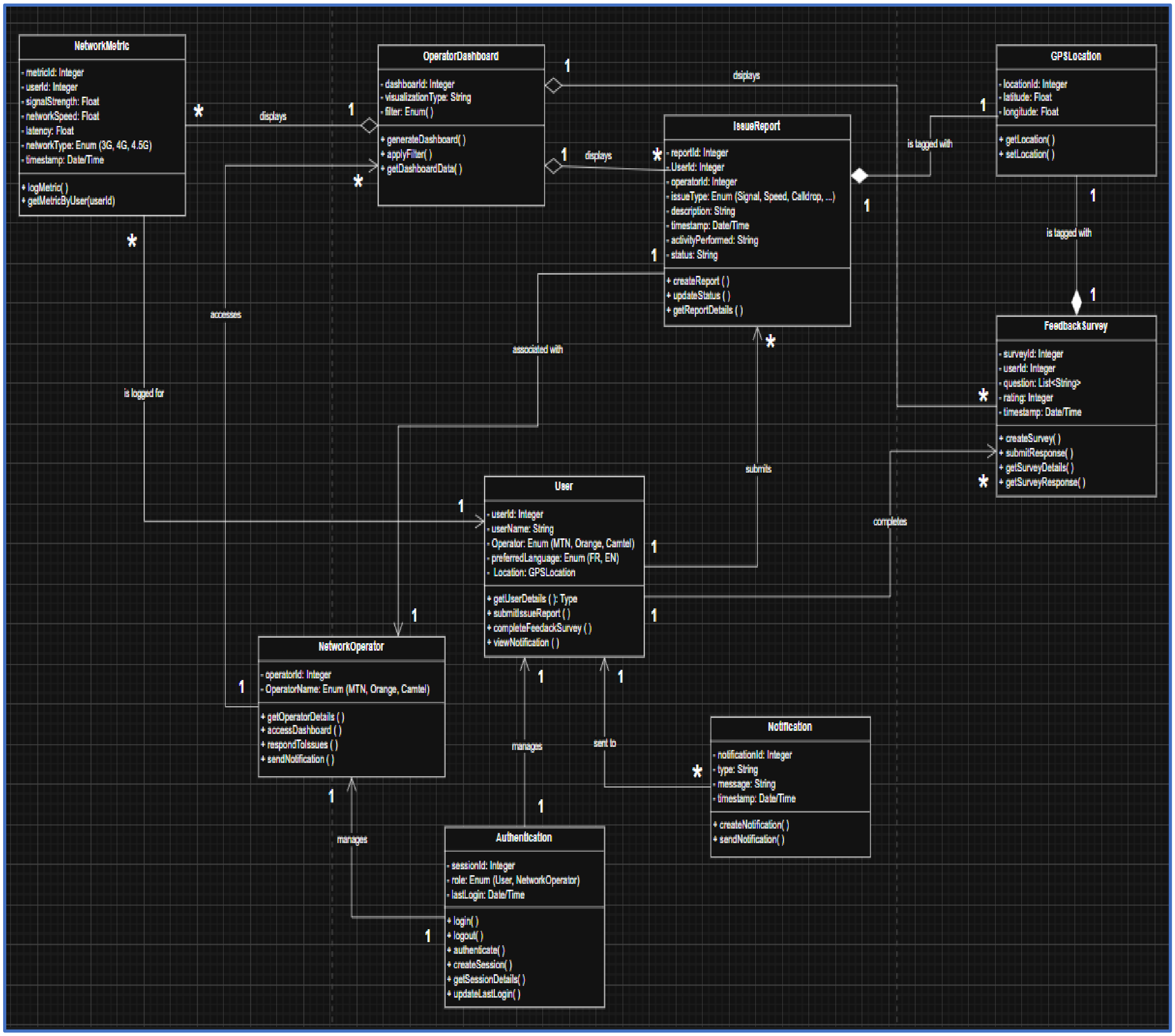

Figure 9: System class diagram visualization

4.6 DEPLOYMENT DIAGRAM

4.6.1 Purpose of the diagram

The deployment diagram visually represents:

- **Communication protocols** between components (e.g., HTTPS, REST).
- **Dependencies** (e.g., background services relying on Android APIs).
- **Failover mechanisms** (e.g., backup Firestore database).
- **Physical hardware nodes** (mobile devices, cloud servers) and **software components** that execute on them.

It serves as a **blueprint** for developers and operators to:

- Plan infrastructure setup.
- Validate system scalability and reliability.
- Troubleshoot deployment issues.

4.6.2 Importance of the deployment diagram in the project

For the QoE Mobile App, this diagram:

1. Clarifies System Architecture: Shows how Flutter apps interact with Firebase services and operator dashboards.

2. Guides Development: Ensures background services (e.g., `TelephonyManager`) are properly integrated.

3. Supports DevOps: Maps cloud resources (Firestore, Cloud Functions) for deployment scripting.

4. **Documents Interfaces**: Defines contracts (e.g., `«interface» OTP API`) for team alignment.

4.6.3 Key design considerations

Based on the system requirements, we prioritized:

Table 8: Key design considerations

Requirement	Design Choice	Diagram Implementation
Background data collection	Android-native services	Monitor Module with «Android API» dependency.
4-day data aggregation	Cloud Functions triggers	Cloud Functions → Firestore path labeled «Pub/Sub».
Battery efficiency	Limited background tasks	Implied via WorkManager constraints (not shown explicitly).
Offline support	SQLite local storage	Storage Module and sync arrows to Firestore.
Operator analytics	Flutter Web Dashboard	Dedicated Analytics Module and Heatmap Module.
Failover resilience	Backup database	Firestore → Backup Firestore with «failover».

4.6.4 Deployment diagram description

A. Nodes & Components

1. **Mobile Device (**<<mobile>>**)**
   - **Flutter App**: Frontend for users.
     - *Subcomponents*:
       - UI Module: Handles surveys/issue reports.
       - Monitor Module: Background service for network metrics.
       - Storage Module: SQLite offline storage.
   - **Interfaces**:
     - Geolocator API: Captures GPS data.
     - Google Maps Interface: Renders heatmaps.
2. **Firebase Cloud (**<<cloud>>**)**
   - **Firebase Auth (**<<interface>>**)**: OTP-based login.
   - **Firestore (**<<database>>**)**: Stores feedback/metrics.
   - **Cloud Functions (**<<service>>**)**: Aggregates data every 4 days.
   - **Backup Firestore**: Failover database.
3. **Operator Dashboard (**<<webServer>>**)**

- **Heatmap Module**: Visualizes poor-signal zones.
- **Analytics Module**: Generates trends.
- **Export Module**: Creates CSV/PDF reports.

B. Critical Communication Paths

Table 9: Critical communication paths of the deployment diagram

Path	Protocol	Purpose
Flutter App ↔ Firebase Auth	HTTPS	User authentication (OTP).
Flutter App ↔ Firestore	REST	Syncs feedback/metrics.
Monitor Module ↔ Device Sensors	Android API	Collects signal strength/speed.
Dashboard ↔ Firestore	GraphQL	Fetches aggregated data for analytics.

C. Key Dependencies

- Monitor Module depends on Android OS for background execution.
- Heatmap Module depends on Google Maps API for rendering.
- Firestore depends on Cloud Functions for periodic aggregation.

D. Visual Highlights

- **Stereotypes**: Labels like <<mobile>>, <<database>> clarify roles.
- **Failover Mechanism**: Backup Firestore ensures data resilience.
- **Modular Design**: Components are nested for readability.

4.6.5 Deployment diagram visualization

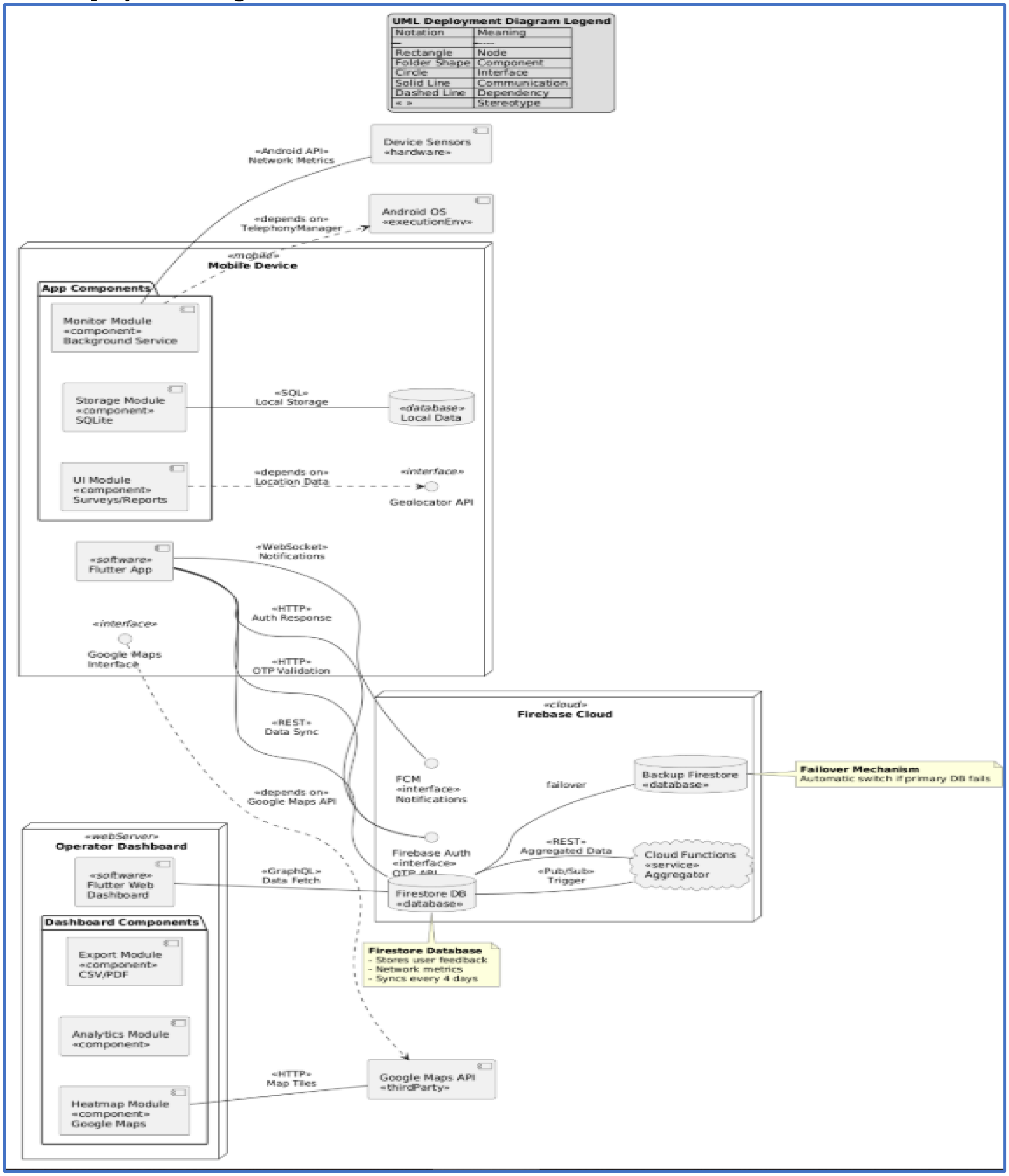
UML Deployment Diagram Legend
Notation | Meaning
Rectangle | Node
Folder Shape | Component
Circle | Interface
Solid Line | Communication
Dashed Line | Dependency
« » | Stereotype
Device Sensors «hardware»
«Android API» Network Metrics
Android OS «executionEnv»
«depends on» TelephonyManager
«mobile» Mobile Device
App Components
Monitor Module «component» Background Service
Storage Module «component» SQLite
«SQL» Local Storage
«database» Local Data
UI Module «component» Surveys/Reports
«depends on» Location Data
«interface» Geolocator API
«software» Flutter App
«WebSocket» Notifications
«HTTP» Auth Response
«interface» Google Maps Interface
«HTTP» OTP Validation
«REST» Data Sync
«depends on» Google Maps API
«cloud» Firebase Cloud
FCM «interface» Notifications
failover
Backup Firestore «database»
Failover Mechanism
Automatic switch if primary DB fails
Firebase Auth «interface» OTP API
«REST» Aggregated Data
Cloud Functions «service» Aggregator
«Pub/Sub» Trigger
Firestore DB «database»
Firestore Database
- Stores user feedback
- Network metrics
- Syncs every 4 days
«webServer» Operator Dashboard
«software» Flutter Web Dashboard
«GraphQL» Data Fetch
Dashboard Components
Export Module «component» CSV/PDF
Analytics Module «component»
Heatmap Module «component» Google Maps
«HTTP» Map Tiles
Google Maps API «thirdParty»

Figure 10: Deployment diagram of the system

5.0 CONCLUSION

The system modeling and design phase of the QoE mobile app project has successfully translated functional and non-functional requirements into a structured blueprint using UML diagrams. The Context and Data Flow Diagrams clarified system boundaries and data interactions, while the Use Case Diagram captured user-system functionalities. Sequence Diagrams detailed critical workflows, such as authentication and issue reporting, and the Class Diagram defined the system's static structure. Finally, the Deployment Diagram outlined the physical and software components, ensuring scalability and reliability.

This design addresses Cameroon's mobile network challenges by bridging the gap between subscribers and operators, fostering transparency and improved service quality. The modular, user-centric approach, combined with technologies like Flutter and Firebase, ensures efficient performance and adaptability for future enhancements. Moving forward, the next phase will focus on implementation, testing, and validation to bring this solution to life, ultimately enhancing the mobile experience for users across Cameroon.