REPUBLIQUE DU CAMEROUN

****

PAIX – TRAVAIL – PATRIE

****

MINISTERE DE L'ENSEIGNEMENT SUPERIEURE

****

UNIVERSITE DE BUEA

REPUBLIC OF CAMEROON

****

PEACE – WORK – FATHERLAND

****

MINISTRY OF HIGHER EDUCATION

****

UNIVERSITY OF BUEA

FACULTY OF ENGINEERING AND TECHNOLOGY
PO BOX 63
BUEA, SOUTH WEST REGION

DEPARTMENT OF COMPUTER ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

COURSE TITLE/CODE: INTERNET PROGRAMMING AND MOBILE PROGRAMMING, CEF 440
COURSE INSTRUCTOR: Dr. Eng. NKEMENI Valery

SYSTEM MODELLING AND DESIGN

Presented by:

BEH CHU NELSON	**FE22A170**
CHEMBOLI ROYWINFEEL	**FE22A180**
NFOR RINGDAH BRADFORD	**FE22A257**
SONE BILLE MILTON	**FE22A297**
TCHOUANI TODJIEU EMMANUEL	**FE22A313**

May 2025

ACADEMIC YEAR: 2024/2025

Contents

1. Introduction

1.1 Purpose

This document describes the software architecture and technical design of the Road Sign and Road State Mobile Notification App. It translates the requirements gathered during the earlier project phases into a structured design blueprint to guide the development and ensure consistency, usability, and scalability.

1.2 Scope

The system aims to enhance road safety by offering real-time notifications of road hazards, road sign education, and reporting functionalities. It integrates map, traffic, and weather data to provide contextual alerts to drivers. It supports multiple languages and offline usage.

1.3 Intended Audience

- Instructors
- Developers
- Road Safety Stakeholders (e.g., Ministry of Transport, Road Safety Officers)

2. System Overview / Architecture

2.1 High-Level Architecture

2.1.1 Overview

The Road Sign and Road State Mobile Notification Application is a mobile-based system designed to enhance road safety and user awareness through real-time alerts and educational content. The system is composed of three core components: the mobile frontend, the backend server, and external services (such as map and traffic APIs). These components collaborate to deliver a responsive, data-driven, and location-aware experience for users in urban and semi-urban environments in Cameroon.

2.1.2 System Components

a. Mobile Application (Frontend)

Developed using Flutter, the mobile application serves as the primary user interface. It provides features including:

- Live map display of road conditions (potholes, accidents, traffic congestion).
- Real-time notifications through pop-ups, voice alerts, or in-app messages.
- Road sign education modules with explanations and quizzes.
- User-generated reporting of road issues with category selection and GPS tagging.
- Offline support with cached data.
- Preference settings for alert type and language (English/French).

b. Backend Server (Logic and Storage)

The backend is responsible for managing application logic, data storage, and real-time communication. Technologies such as Node.js/Express or Firebase are used. Key responsibilities include:

- Handling user authentication and authorization.
- Receiving, storing, and distributing road condition reports.
- Sending push notifications using Firebase Cloud Messaging (FCM).
- Maintaining location-based filtering of alerts.
- Supporting synchronization and store-and-forward logic for offline reporting

c. External APIs

The system integrates third-party services to enhance functionality:

- Google Maps SDK: For rendering interactive maps and fetching geolocation data.
- Traffic APIs (future): To provide live traffic congestion updates.
- Weather APIs (optional): For detecting hazardous road conditions due to weather.

2.1.3 High-Level Architecture Diagram (Textual Representation)

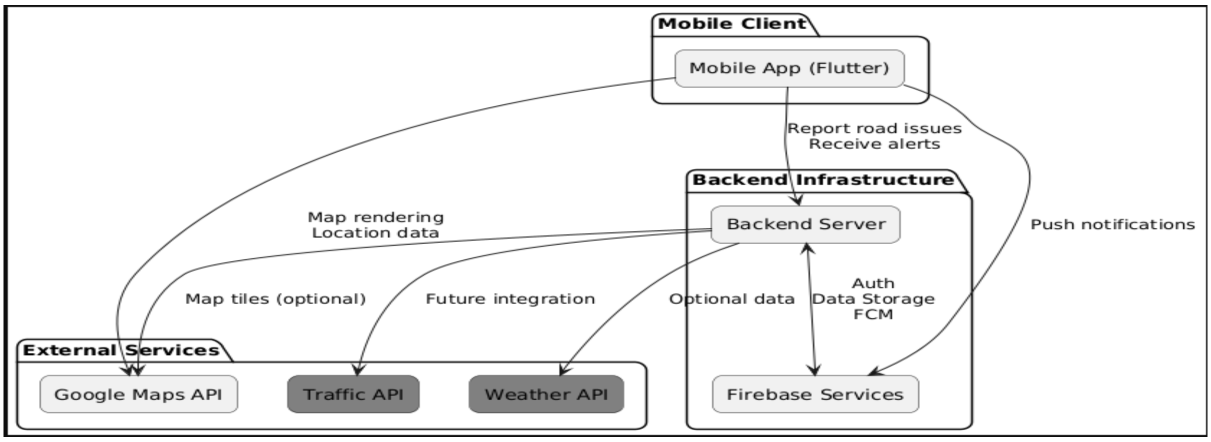


fig1: High Level Architecture

2.1.4 Data Flow Summary

1. User Interaction: The mobile app captures user actions such as submitting reports or viewing alerts based on location.
2. Backend Communication: Data is transmitted to the backend, which processes and stores it in the database.
3. Notification Delivery: Push notifications are dispatched using Firebase Cloud Messaging.
4. External Integration: Map data and potential future traffic/weather updates are retrieved from external APIs.
5. Offline Handling: When offline, data is cached locally and synchronized upon reconnection.

2.2 Components and Responsibilities

Component	**Function**
User Interface	**Shows alerts, road signs, and report form**
Report Handler	**Submits and stores user reports**
Alert Generator	**Creates alerts from APIs and user reports**
API Connector	**Connects to Google Maps and weather services**
Notification Engine	**Sends in-app, voice, or SMS alerts**
Sensor Monitoring Engine	**Detects road anomalies via motion and GPS sensors**

2.3 Component Interaction

This section explains how the major components interact within the system to support core functionalities such as reporting, notification, and data retrieval.

1. **User Interface → Report Handler :**
   - The user fills the form to report a road issues.
   - The UI send the data to the Report handler for processing.

2. **Report Handler → Backend Server:**
   - The report is submitted and stored in the backend
   - Data is tagged with location and type (e,g., potholes, accident).

3. **Backend Server → Alert Generator:**
   - Backend processes report and combines them with API data (e.g., traffic weather).
   - Alert Generator creates context-based alerts.

4. **Alert Generator → Notification Engine:**
   - Alerts are formatted for delivery (e.g., text, voice).
   - Notification Engine determines delivery method (push, voice, SMS).

5. **Notification Engine → User Interface:**
   - User receives the alert through the selected notification channel.

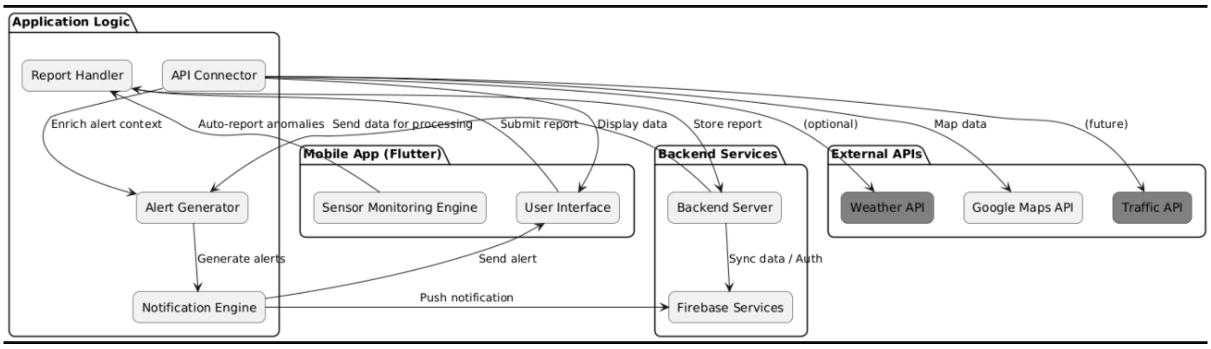
6. **API Connector → External APIs:**
   - Fetches map data, traffic updates, or weather conditions.
   - Supplies this data to both the Alert Generator and the UI (e.g., map overlays).

7. **Backend → Firebase Services:**
   - Authenticates users and manages data syncing.
   - Facilitates real-time updates and offline synchronization.

8. **Mobile Sensors → Sensor Monitoring Engine → Report Handler (Automated Detection)**
   - The app detects sudden motion patterns using the device's accelerometer and GPS (e.g., when a car hits a pothole).

- A potential road issue is flagged and temporarily logged with location data.
- If a second vehicle experiences the same anomaly at approximately the same location, the event is confirmed.
- The system forwards the confirmed report to the backend.
- An alert is generated and dispatched to nearby users.

This intelligent and automated data collection approach enhances the accuracy and timeliness of road condition alerts. By combining manual reports with sensor-driven detection, the system ensures proactive safety measures and reduces the reliance on manual input, thereby improving scalability and reliability.



3. Component Design

3.1 Module Descriptions

This section provides detailed descriptions of the key software modules, including their responsibilities, inputs, and outputs.

a. User Interface (UI)

- **Responsibilities:** Provide a clean, multilingual interface for user interaction.
- **Inputs**: User actions such as tapping buttons, filling forms, selecting preferences.
- **Outputs:** Visual feedback, alert pop-ups, educational content, map overlays.

b. Report Handler

- **Responsibilities:** Collect manual and automatic reports of road conditions.
- **Inputs:** Form data from the UI sensor data from the Sensor Monitoring Engine.

- **Outputs:** Structured road condition reports sent to the backend for processing.

c. Sensor Monitoring Engine

- **Responsibilities:** Monitor phone sensors (e.g., accelerometer, GPS) to detect anomalies.
- **Inputs:** Real-time sensor data from the mobile device.
- **Outputs:** Flagged locations of potential potholes or road issues forwarded to the Report Handler.

d. Alert Generator

- **Responsibilities:** Analyze input from user reports and external APIs to generate alerts.
- **Inputs**: Verified data from the backend and API Connector.
- **Outputs:** Prepared alert content with type, location, and severity.

e. Notification Engine

- **Responsibilities**: Deliver alerts to users using selected channels.
- **Inputs:** Alert content from the Alert Generator; user preferences.
- **Outputs**: In-app messages, voice alerts, or SMS notifications**.**

f. API Connector

- **Responsibilities**: Retrieve external contextual data such as map visuals, traffic conditions, and weather reports.
- **Inputs:** Requests from internal components (UI, Alert Generator).
- **Outputs**: API responses (e.g., map tiles, traffic data) sent back to internal components.

g. Backend Server

- **Responsibilities:** Handle core application logic, authentication, storage, and communication.
- **Inputs:** Incoming reports and data requests.
- **Outputs:** Stored data, notifications, and authenticated user sessions.

h. Firebase Services

- **Responsibilities:** Support real-time database operations, authentication, and push messaging.
- **Inputs:** Data sync and push requests.

- **Outputs:** Real-time updates to users, token management, and alert delivery.

i. Educational Module

- **Responsibilities:** Display informative content about road signs to enhance user understanding and road safety awareness.
- **Inputs:** User selection of road sign categories or topics.
- **Outputs:** Displayed explanations and examples within the app.

Each of these modules plays a crucial role in delivering a robust and intelligent road safety notification system by maintaining clear responsibilities, clean interfaces, and reliable communication pathways.

4. System Design Models

4.1. Context Diagram

The context diagram provides a high-level view of the Road Sign and Road State Mobile Notification Application and its interaction with external entities. It illustrates how the system exchanges data with users, third-party services, and backend platforms.

External Entities:

- **Driver/User:** Interacts with the mobile app to receive alerts, view road signs, and submit reports.
- **Mobile Sensors:** Provide raw data for detecting potholes and other anomalies.
- **Google Maps API:** Supplies mapping and location data.
- **Traffic and Weather APIs:** Provide additional context for road conditions (future integration).
- **Firebase Services:** Manage authentication, real-time database updates, and push notifications.

System Boundary:

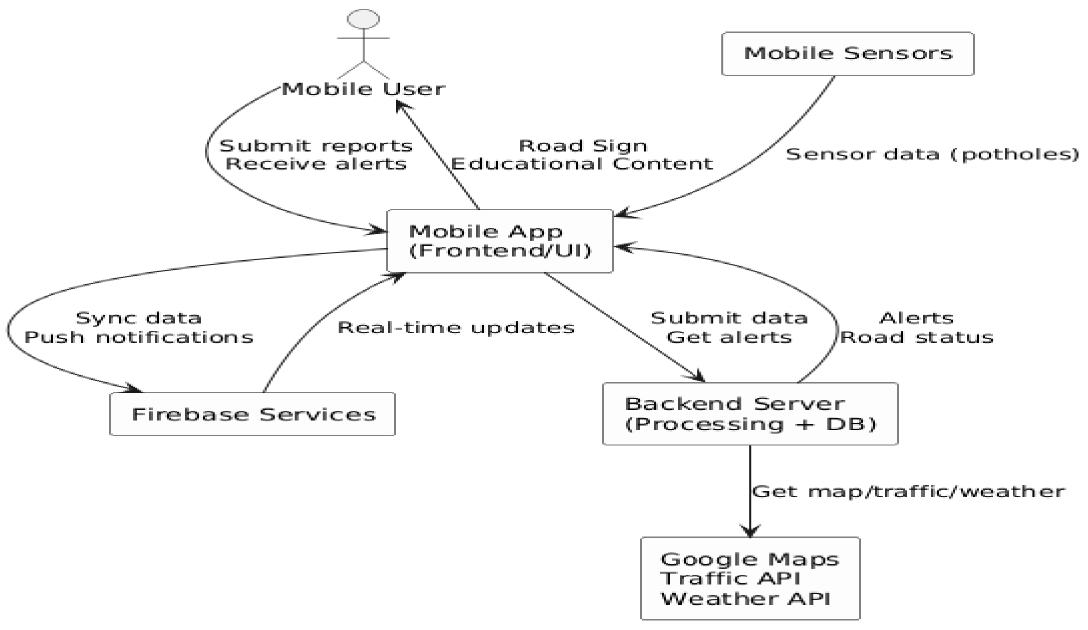
- Road Sign and Road State Mobile Notification Application

Data Flows:

- Users submit reports, view road sign educational content, and receive notifications.
- Sensors send data for road condition analysis.

- APIs send map, traffic, and weather data.
- Firebase handles backend data sync and user management.

Context Diagram :



This diagram highlights the central role of the mobile application and backend server in mediating communication between users, external APIs, and cloud infrastructure. It also emphasizes the hybrid data acquisition model (manual reporting and automatic sensor-based detection), while also showing the educational content pathway that provides users with road sign explanations via the mobile app.

4.2 Data Flow Diagram (Level 1)

The Level 1 Data Flow Diagram (DFD) decomposes the system into its major processes and shows the flow of data between these processes and external entities.

External Entities:

- User
- Mobile Sensors
- Google Maps API
- Traffic & Weather APIs
- Firebase Services

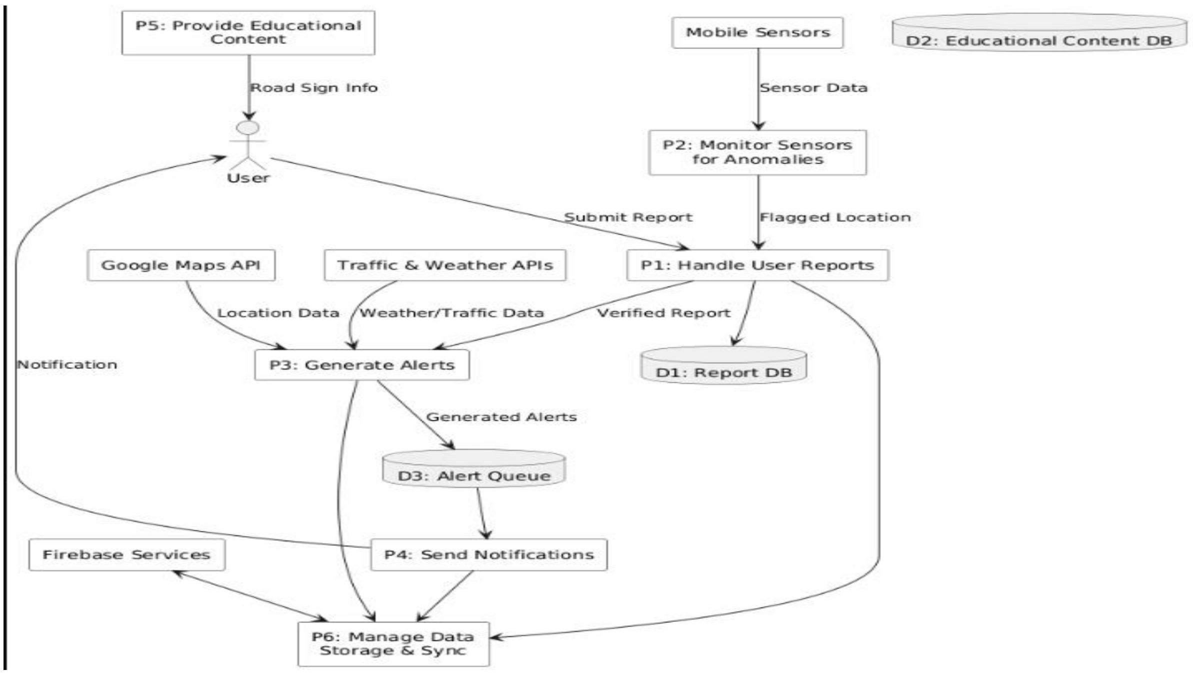
Internal Processes:

1. P1 - Handle User Reports
2. P2 - Monitor Sensors for Anomalies
3. P3 - Generate Alerts
4. P4 - Send Notifications
5. P5 - Provide Educational Contents
6. P6 - Manage Data Storage & Sync

Data Stores:

D1 - Report Database
D2 - Educational Content DB
D3 - Alert Queue

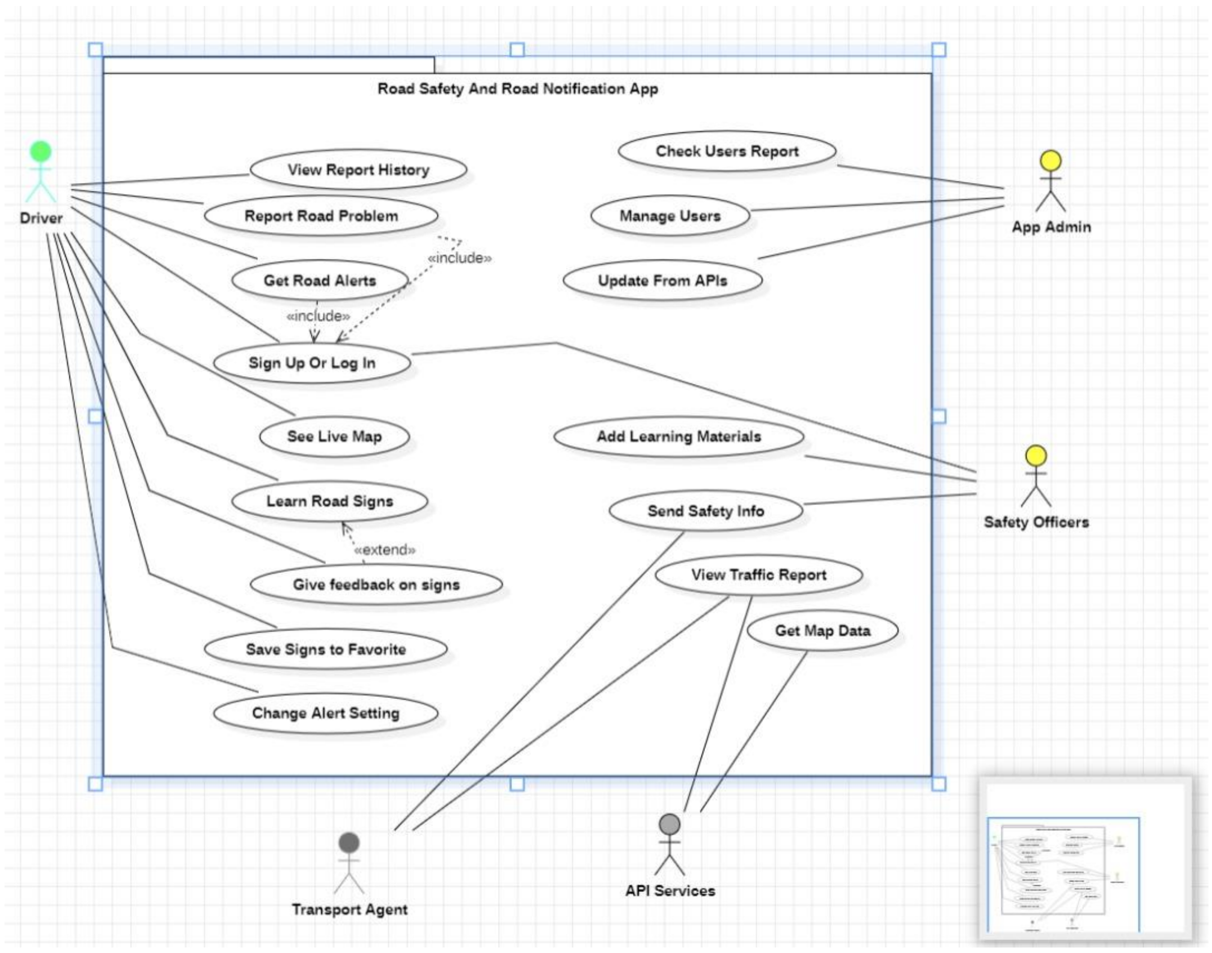
Dataflow Diagram (Level 1)



This Level 1 DFD illustrates how user inputs, sensor data, and external API responses are processed within the system to deliver timely alerts, educational content, and data management services.

4.3. Use Case Diagram:

The usecase diagram represents the interactions between users and the system's major functionalities, below show the diagram of this project:



4.4 Sequence Diagrams

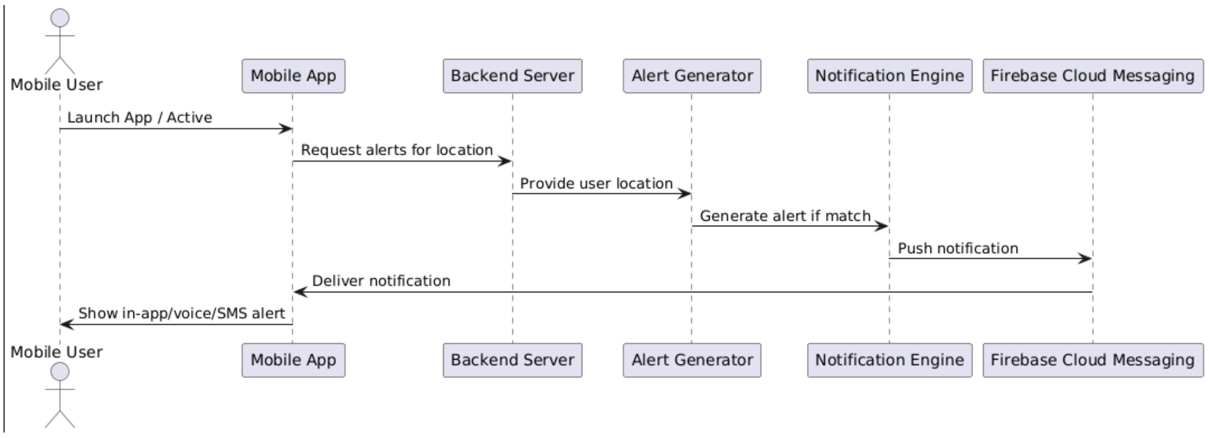
4.4.1. Receive Real-Time Alert

This sequence diagram illustrates how a user receives a real-time road alert after a report or detection is processed.

Actors & Components:

- Mobile User
- Mobile App
- Firebase Cloud Messaging (FCM)
- Backend Server
- Alert Generator
- Notification Engine

Sequence Diagram:



This sequence captures both the backend-driven alert logic and the Firebase-based notification delivery mechanism.

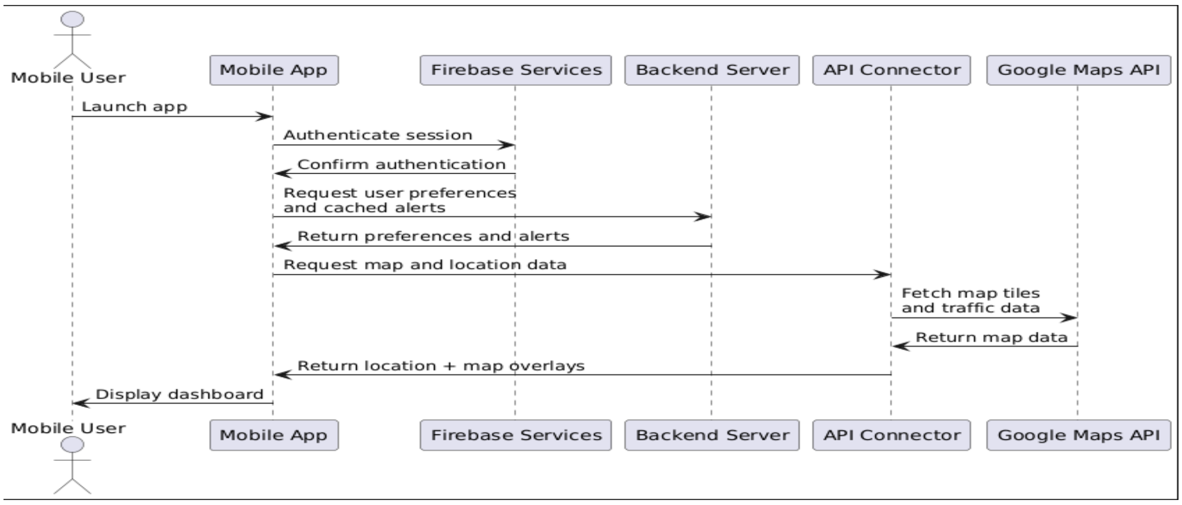
4.4.2. User Opens App

This sequence diagram describes what happens when a user launches the mobile application.

Actors & Components:

- Mobile Users
- Mobile App
- Firebase Services
- Backend Server
- API Connector

Sequence Diagram:



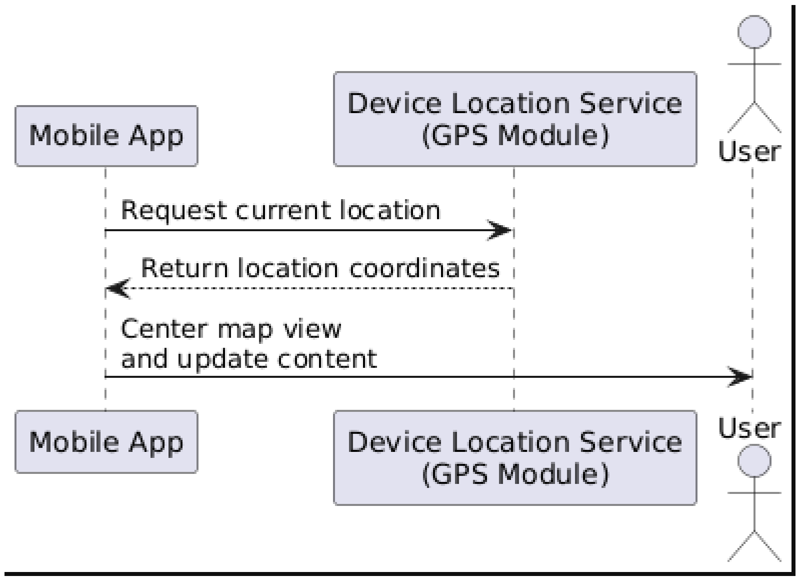
The flow shows how the system initializes and personalizes the user interface after app launch, combining user settings, real-time data, and external map services.

4.4.3. App Requests Location

Actors and Components:

- Mobile App
- Device Location Service (GPS)
- User

Sequence Diagram:


Mobile App
Device Location Service (GPS Module)
User
Request current location
Return location coordinates
Center map view and update content
Mobile App
Device Location Service (GPS Module)
User

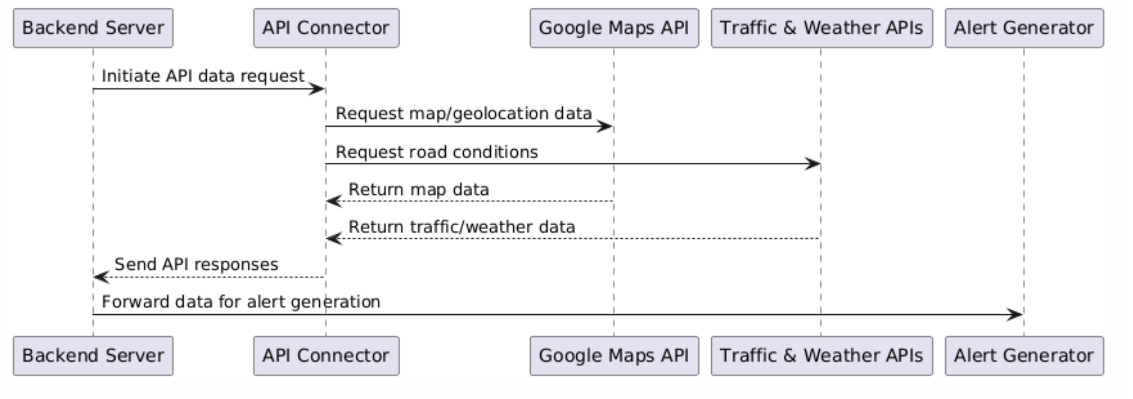
This sequence shows how the backend collects contextual road data to support intelligent alert generation and route planning.

4.4.4. Server fetches data from API

Actors and Components:

- Backend Server
- API Connector
- Google Map API
- Traffic & Weather APIs
- Alert Generator

Sequence Diagram:



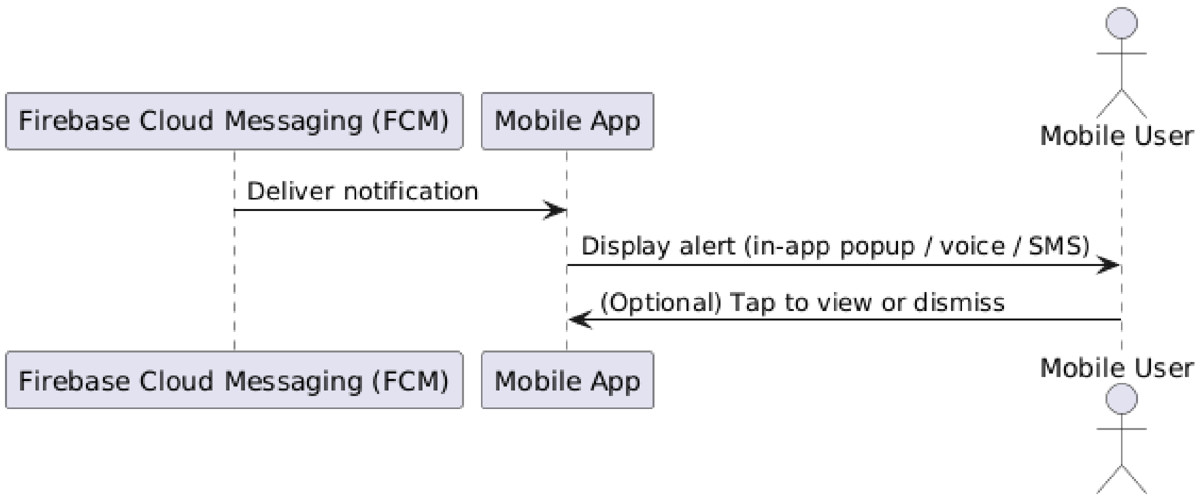
4.4.5. Alert is Displayed

This sequence shows how a generated and delivered alert is ultimately displayed to the user.

Actors & Components:

- Firebase Cloud Messaging (FCM)
- Mobile App
- Mobile User

Sequence Diagram:


Firebase Cloud Messaging (FCM)
Mobile App
Mobile User
Deliver notification
Display alert (in-app popup / voice / SMS)
(Optional) Tap to view or dismiss

4.5. Class Diagram

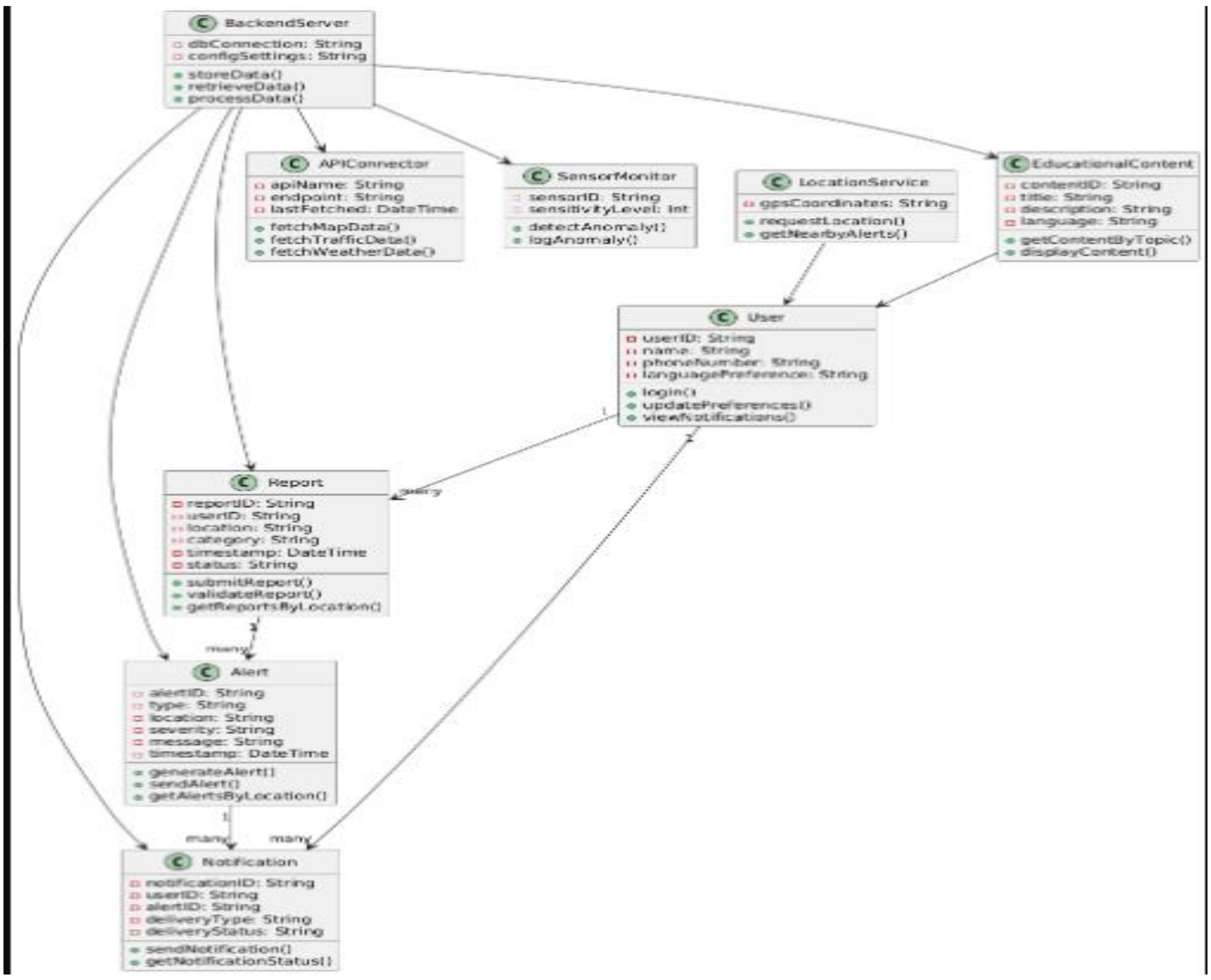
The class diagram represents the static structure of the system by showing the system's classes, their attributes and methods, and the relationships between them.

Key Classes:

1. **User**

- **Attributes:** userID, name, phoneNumber, languagePreference
- **Methods :** login(), updatePreferences(), viewNotifications()

2. **Report**
   - **Attributes:** reportID, userID, location, category, timestamp, status
   - **Methods:** submitReport(), validateReport(), getReportsByLocation()
3. **Alert**
   - **Attributes:** alertID, type, location, severity, message, timestamp
   - **Methods:** generateAlert(), sendAlert(), getAlertsByLocation()
4. **Notification**
   - **Attributes:** notificationID, userID, alertID, deliveryType, deliveryStatus
   - **Methods:** sendNotification(), getNotificationStatus()
5. **EducationalContent**
   - **Attributes:** contentID, title, description, language
   - **Methods:** getContentByTopic(), displayContent()
6. **LocationService**
   - **Attributes:** gpsCoordinates
   - **Methods:** requestLocation(), getNearbyAlerts()
7. **APIConnector**
   - **Attributes:** apiName, endpoint, lastFetched
   - **Methods:** fetchMapData(), fetchTrafficData(), fetchWeatherData()
8. **SensorMonitor**
   - **Attributes:** sensorID, sensitivityLevel
   - **Methods:** detectAnomaly(), logAnomaly()
9. **BackendServer**
   - **Attributes:** dbConnection, configSettings
   - **Methods:** storeData(), retrieveData(), processData()



4.6. Deployment Diagram

The deployment diagram illustrates the physical arrangement of system components and how they are deployed across different hardware nodes and environments**.**

Nodes:

- **Mobile Device (Android/iOS)**
  - Contains: Mobile App, Sensor Monitor, Location Service
- **Cloud Backend Server**
  - **Contains:** BackendServer, Alert Generator, Report Handler, API Connector
- **Firebase Cloud Platform**

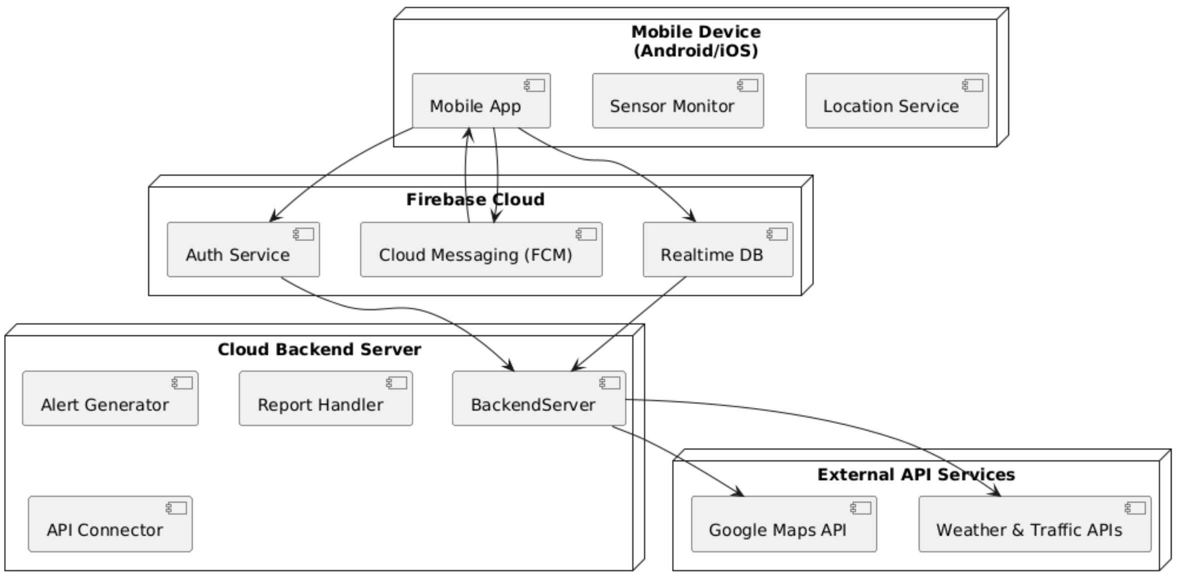
    - **Contains:** Firebase Authentication, Firebase Cloud Messaging, Realtime Database

- **External API Services**
    - Google Maps API
    - Weather and Traffic APIs

Deployment Relationships:

- The Mobile App communicates with the Backend Server and Firebase for authentication, data synchronization, and alert delivery.
- Backend Server interacts with External APIs to fetch map, traffic, and weather data.
- Firebase Cloud Messaging pushes notifications from Backend to Mobile App.

Deployment Diagram:



---

5. Security Design

- Authentication: OTP or email-based sign-in.
- Authorization: Role-based access (Driver, Officer, Admin).
- Data Encryption: All communication uses HTTPS.
- Privacy: Collects minimal user data; user consent enforced.

6. Design Constraints

- Must run on Android smartphones, including low-end devices.
- Offline mode supported for report viewing and submission.
- Supports English, French, and Pidgin languages.
- Optimized for low battery and data usage.

7. Assumptions and Dependencies

- External APIs (Maps, Weather) are always accessible.
- GPS and mobile internet are available on user devices.
- Users are willing to submit reports when the UI is simple.
- Sensor data and API data are accurate and timely.