**BIAKA UNIVERSITY INSTITUTE OF BUEA**

**DEPARTMENT OF COMPUTER ENGINEERING**

**BACHELOR OF COMPUTER ENGINEERING**

Course Title:

Mobile Programming and internet programming CEB402

**Presented by:**

|  |  |
| --- | --- |
| **Name** | **Matricle** |
| Fouyou Chinjeh Brain | SENG22SE009 |
| Kang Jevis | SENG22SE001 |
| VISEMIH JANICE-ERLYN | SENG22SE019 |
| Abang Blessing | SENG22SE010 |

**Instructor:**

**ENG. VALERY NKEMENI**

**Academic year:** 2024/2025

# ABSTRACT

# Table of Contents

**Table of Contents**

[**BIAKA UNIVERSITY INSTITUTE OF BUEA**](#_Toc201084230)

# 1. ABSTRACT

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

# LIST OF TABLE

# 2. INTRODUCTION 2.1 PROBLEM STATEMENT 3. PROJECT OBJECTIVES 4. SYSTEM REQUIREMENTS 5. PROJECT PLANNING AND METHODOLOGY 6. SYSTEM DESIGN AND ARCHITECTURE 7. IMPLEMENTATION DETAILS 8. TESTING AND DEBUGGING 9. CHALLENGES AND SOLUTIONS 10. RESULTS AND EVALUATION 11. CONCLUSION 12. RECOMMENDATIONS AND FUTURE WORK 13. REFERENCES

# 1. Introduction

The CM RoadEye Application is an innovative mobile solution designed to enhance road safety awareness and incident reporting in Cameroon. With the rising number of road accidents, unsafe road conditions, and limited access to real-time alerts, there is an urgent need for a robust, locally-adapted platform that leverages modern mobile and mapping technologies. CM RoadEye provides an integrated system where citizens can report road hazards (such as potholes, traffic signs, and accidents), view reports from others on an interactive map, and receive real-time safety notifications—all powered by Flutter, Firebase Emulator, and OpenStreetMap for cost-effective, scalable, and offline-capable operation.

# 2. Problem Statement

Cameroon faces significant challenges in road safety management, including poor reporting mechanisms for road hazards, limited public awareness of real-time road issues, and inefficient dissemination of safety information. Existing solutions are either not localized, are costly, or lack the ability to operate offline and cater to local languages and needs. As a result, road users are exposed to avoidable risks, and authorities lack timely data to act upon. There is a clear need for a digital, accessible, and community-driven platform that bridges these gaps.

# 3. Project Objectives

The main objectives of the CM RoadEye Application are:  
- To provide a user-friendly mobile platform for reporting and visualizing road hazards, accidents, and road signs in Cameroon.  
- To deliver real-time notifications and alerts to road users about nearby hazards or incidents.  
- To integrate OpenStreetMap for accurate, up-to-date, and free geospatial data visualization.  
- To utilize Firebase Emulator for secure, scalable, and cost-effective backend services during development.  
- To support offline data storage and synchronization for areas with poor network connectivity.  
- To enable multilingual support (English and French) for wider accessibility.  
- To lay the groundwork for future expansion, including admin/moderation tools and integration with local authorities.

# 4. System Requirements

# 4.1 Functional Requirements - User Registration & Authentication: Users must be able to register and log in securely (email, password). - Map Display: The app should display an interactive map using OpenStreetMap, centered on the user’s location. - Report Submission: Users can submit new reports of road hazards (accidents, potholes, traffic signs, etc.) with location, photo, description, and category. - View Reports: All users can view nearby and recent reports on the map, including details and status. - Real-Time Notifications: Users receive push notifications about critical hazards or incidents within their vicinity. - Localization: The app supports English and French, allowing users to switch languages. - Offline Support: Users can view previously loaded data and submit reports offline, which sync automatically when connectivity is restored. - Admin Tools (for future expansion): Admins can review, validate, or moderate reports.

# 4.2 Non-Functional Requirements - Performance: Map rendering, data fetching, and notification delivery must be fast and responsive. - Security: All user data must be securely stored and transmitted, with authentication and proper access control. - Scalability: The system must support growth in user base and data volume without performance degradation. - Reliability: The app must handle network interruptions gracefully and ensure data integrity. - Maintainability: The codebase must be modular, well-documented, and follow clean architecture principles. - Usability: The user interface must be intuitive, accessible, and optimized for local context.

# 4.3 Technical Requirements - Flutter SDK (latest stable) for cross-platform mobile development. - Firebase Emulator Suite for Auth, Firestore, and Cloud Messaging during development/testing. - OpenStreetMap integration using flutter\_map or similar packages. - Riverpod for state management. - Node.js backend for admin and moderation logic (future). - Android platform targeting version 6.0 (API level 23) and above. - Local SQLite database for offline data persistence. - Asset storage for report photos. - CI/CD for automated builds and tests (GitHub Actions or similar).

# 5. Project Planning and Methodology

# 5.1 Project Planning

**Project Timeline**  
The project was divided into iterative phases, starting with requirements gathering and wireframe design, moving through architectural setup, core feature development, user interface implementation, integration, testing, and documentation.

**Milestones:**  
1. Requirements Analysis & Wireframing  
 - Defined core user stories and functional requirements  
 - Created textual wireframes for each screen  
2. Architecture & Tooling Setup  
 - Chose Flutter, Riverpod, Clean Architecture, OpenStreetMap, Firebase Services  
 - Initialized project repository and base folder structure  
3. Core Feature Implementation  
 - User authentication  
 - Map display and geolocation  
 - Report submission workflow  
 - Report viewing and detail screens  
 - offline road Sign Library  
4. Integration & Enhancement  
 - Real-time notifications via FCM and one signal  
 - Offline data support  
 - Multilingual UI support (English/French)  
 - Asset and image handling  
5. Testing & Debugging  
 - Manual and automated testing of all workflows  
 - Debugging of reported issues (imports, navigation, overflows, etc.)  
6. Documentation & Handover  
 - In-app documentation  
 - Final project report

# 5.2 Methodology

**Agile/Iterative Development**  
Development followed an agile, iterative approach with frequent feedback loops. User stories were prioritized based on impact, and development was organized into weekly sprints. Major architectural choices were justified by the project’s needs for modularity, scalability, and offline support.

**Technology Stack Rationale**  
**- Flutter for rapid,** cross-platform development and expressive UIs.  
**- Riverpod** for scalable, testable, and robust state management.  
**- Firebase Emulator** for free, local development and easy migration to live Firebase services.icluding firebase services firestore ,authentication ,FCM.  
**- OpenStreetMap** for free, open, and offline-capable mapping.  
**- Node.js** reserved for backend logic requiring moderation and complex queries.  
**- Clean Architecture** to ensure separation of concerns and maintainable codebase.  
**- SQLite** for offline support, enabling the app to function even with poor connectivity.  
**- Easy\_localization** for robust multilingual support.  
**- Geolocator** for device location services.  
**- flutter\_map\_marker\_cluster** for clustering markers efficiently.  
**- cached\_network\_image f**or performant image loading.  
**- image\_picker** for capturing or selecting images for reports.  
**- file\_picker** for document and file handling.  
**- flutter\_local\_notifications** for advanced local notifications.  
**- provider** (used earlier; replaced with Riverpod for scalability).  
**- shared\_preference**s for persistent lightweight key-value storage.  
**- flutter\_launcher\_icons** for automating custom app icons.  
**- flutter\_native\_splash** for configuring the splash screen.  
**- firebase\_messaging** and firebase\_core for push notifications and backend integration.  
**- http** for network requests (to backend/Node.js APIs).  
**- sqflite** for local/offline database management.  
**- connectivity\_plus** for monitoring internet connectivity.

This toolkit ensured we had everything needed for a modern, robust, and production-ready mobile application tailored for Cameroon’s specific context.

# 6. System Design and Architecture

# 6.1 Overview The CM RoadEye app adopts a modular, scalable, and maintainable architecture built around the Clean Architecture pattern, supported by MVVM principles and robust state management (Riverpod). This structure ensures a clear separation of concerns, making the codebase easy to extend and maintain.

Key elements include:  
- Presentation Layer: Handles UI, user input, and displays data via Riverpod providers.  
- Domain Layer: Contains business logic, use cases, and app-specific models.  
- Data Layer: Manages data sources (Firebase, SQLite, OpenStreetMap, REST API).  
- Core/Shared Layer: Utilities, constants, theming, localization, etc.

App Directory Structure (Text):  
App Root/  
├─ lib/  
│ ├─ core/ # App-wide constants, utilities, themes, localization  
│ ├─ data/ # Repositories, data sources, services (Firebase, SQLite, etc.)  
│ ├─ domain/ # Entities, business logic, use cases  
│ ├─ presentation/ # Screens, widgets, providers, navigation  
│ ├─ main.dart # App entrypoint  
│ ├─ l10n/ # Localization files  
│ └─ routes/ # App-wide navigation & route config  
├─ assets/ # Images, icons, translations, etc.  
├─ test/ # Unit and widget tests  
└─ pubspec.yaml # Dependencies

# 6.2 Core Modules & Responsibilities **- Authentication Module:** Handles registration, login, session, and user state (Firebase Auth Emulator, Riverpod). **- Map/Location Module:** Integrates OpenStreetMap (flutter\_map), location (Geolocator), and marker clustering. Responsible for showing current position, reported hazards, and alert zones. **- Reporting Module:** Allows users to submit new reports with geolocation, images, and categorization; supports offline draft/save (image\_picker, sqflite, file\_picker). **- Notification Module:** Manages push notifications (Firebase Messaging, local\_notifications), alert banners, and in-app updates. **- Offline Sync Module:** Stores reports and map data in SQLite for offline access and background syncs when online (sqflite, connectivity\_plus). **- Localization Module:** Provides language switching (easy\_localization) and handles all UI strings for English/French. **- Settings/Profile Module:** Enables profile editing, logout, and app preferences (shared\_preferences).

# **6.3 Key Tools, Packages, and Frameworks Used** - Flutter SDK (core framework) - Riverpod (state management) - GoRouter (for advanced navigation) - flutter\_launcher\_icons, flutter\_native\_splash (branding, splash) - easy\_localization (i18n/l10n) - flutter\_map (OpenStreetMap integration) - flutter\_map\_marker\_cluster (marker clustering) - geolocator (GPS/location tracking) - cached\_network\_image (image optimization) - Firebase Emulator Suite (Firestore, Auth, FCM) - firebase\_core, firebase\_auth, cloud\_firestore, firebase\_messaging (live backend/emulated) - sqflite (local SQLite for offline data) - shared\_preferences (simple persistent storage) - connectivity\_plus (detecting connection state) - http (REST API requests; future Node.js integration) - image\_picker, file\_picker (capture/select images) - path\_provider (for local storage) - flutter\_local\_notifications (local & scheduled notifications) - flutter\_test, mockito, integration\_test (testing) - GitHub Actions (CI/CD) - Dart DevTools (debugging) - provider (initially, then replaced by Riverpod) - intl (date/time formatting) - json\_serializable (model generation)

Recently added:  
- flutter\_local\_notifications  
- connectivity\_plus  
- flutter\_map\_marker\_cluster  
- cached\_network\_image

# 6.4 Architectural Decisions - Clean Architecture for maintainability and testability. - Riverpod over Provider for advanced, modular state management. - OpenStreetMap for free, offline, and open-source mapping. - Firebase services for affordable, safe, and robust local development.

# 6.5 Data Flow Summary 1. User logs in (Firebase Auth). 2. Dashboard loads map (OpenStreetMap), user location shown (Geolocator). 3. Reports load from Firestore or SQLite (offline). 4. User submits report (with image), saved locally or uploaded online. 5. Notifications via Firebase Messaging or local alerts. 6. Data syncs between SQLite and Firestore when connectivity is restored.

# 7. Implementation Details

# 7.1 Project Initialization and Setup - Flutter project initialized with Clean Architecture. - Packages for Riverpod, OpenStreetMap, Firebase, etc. added and configured. - Firebase Emulator set up for secure, cost-effective local backend testing. - CI/CD (GitHub Actions) for automated builds/tests.

# 7.2 Key Screens and Features - Splash Screen: Auto-navigation, branding, and loading animation. - Authentication: Registration/login, validation, persistent session. - Dashboard: Map (OpenStreetMap), real-time hazards, alert zones, marker clustering, action buttons. - Report Submission: Location, category chips, photo picker, description; offline save and sync. - Notifications: List and details; push and local (offline) notifications. - Settings/Profile: Profile management, language switcher, logout, and app preferences. - Sign Library: Road sign catalog for user education.

# 7.3 Data & Logic Implementation - Riverpod for global and screen-specific state management. - Repository pattern for all data sources: Firebase, SQLite, mock. - Automatic offline caching and sync (connectivity\_plus + sqflite). - easy\_localization for immediate language changes. - firebase\_messaging and flutter\_local\_notifications for push/alert management. - Unit, widget, and integration tests for code quality.

# 7.4 Workflow & Collaboration - GitHub repo, PR-based code review, automated CI, and in-line documentation. - Dart DevTools for runtime debugging and performance optimization.

# 7.5 Recent Improvements - Advanced notification logic and background handling. - Connectivity status + sync prompts. - Image caching/optimization for markers. - Enhanced UI, bug fixes, and improved language switching. - Marker clustering and map performance boosts. - Increased code/test coverage.

# 8. Testing and Debugging

# 8.1 Testing Approaches - Unit tests using flutter\_test and mockito for business logic, state changes, and providers. - Widget tests for UI component rendering and response. - Integration tests for login, map, reporting, and notification workflows. - Manual testing on emulators and Android devices, including edge cases (offline/online, navigation, error handling). - GitHub Actions CI/CD runs all tests on push/merge.

# 8.2 Debugging & QA - Dart DevTools for profiling, widget inspection, and runtime exception debugging. - Local error logging for uncaught exceptions. - Bug tracking and resolution for overflows, map/marker errors, offline issues, and navigation bugs. - Performance monitoring for map and notifications.

# 9. Challenges and Solutions

# 9.1 Connectivity & Offline Sync - Problem: Maintaining usability and data integrity during unreliable connectivity. - Solution: SQLite as local cache, background sync with Firebase when online, and user feedback in UI.

# 9.2 Real-Time & Local Notifications - Problem: Alerting users both online and offline. - Solution: Combined Firebase Messaging for online and flutter\_local\_notifications for offline/local.

# 9.3 Map & Marker Performance - Problem: Rendering many markers, tile updates, and UI lag. - Solution: Marker clustering, image caching, throttling, and optimization for low-end devices.

# 9.4 Multilingual Consistency - Problem: Seamless switching and full translation coverage. - Solution: easy\_localization, automated translation checks, and persistent user choice.

# 9.5 Usability & Accessibility - Problem: Ensuring intuitiveness for all users. - Solution: User testing, accessible layouts, iconography, clear prompts, and color/contrast standards.

# 9.6 Emulator & Toolchain - Problem: Setting up Firebase Emulator, OpenStreetMap, Android builds. - Solution: Documented setup, automated CI scripts.

# 10. Results and Evaluation

# 10.1 Key Outcomes - Core features all implemented: registration, map-based reporting, offline-first, bilingual, notifications. - Stable, responsive app on target Androids, including low-end hardware. - Data integrity confirmed across connectivity scenarios. - Positive user feedback on ease of use and value.

# 10.2 Evaluation Metrics - Functionality: All requirements met and tested. - Performance: App loads <3 seconds, responsive with 100+ markers. - Usability: Smooth completion of user flows. - Reliability: No data loss or crash issues found in user tests.

# 11. Conclusion

The CM RoadEye application is a robust, user-friendly solution addressing Cameroon's unique road safety needs. It empowers users to report, view, and be alerted to hazards in real time or offline, and is designed for both scalability and long-term sustainability.

# 12. Recommendations and Future Work - Build admin dashboard/web portal for moderation and analytics. - Add user engagement features (upvotes, comments, gallery). - Integrate with road authorities. - Implement analytics and trend mapping. - Extend to iOS (Flutter cross-platform). - Add more accessibility: voice input, more languages, dark mode. - Transition from emulator to live Firebase backend.

# 13. References - Flutter: https://flutter.dev - OpenStreetMap: https://wiki.openstreetmap.org - Firebase Emulator: https://firebase.google.com/docs/emulator-suite - Riverpod: https://riverpod.dev - easy\_localization: https://pub.dev/packages/easy\_localization - All relevant Flutter/Dart packages and GitHub repos.