PROJECT REPORT

Weway Fianl Project Report

Abstract

The document provides detailed senior design project report for WeWay system

University Name:

College:

Department:

List of Team Members including IDs and Majors:

Date of submission: Wednesday, 03 December 2025

Table of Contents

Table of Figures

C1) Software Requirements Specification (SRS)

C1.1. Introduction

C1.1.1 Purpose

The purpose of the document is to gather in one place all the functional and non-functional requirements which are going to affect and guide the development of the project. This document captures the finalized set of requirements and will be used for further refinement of the requirements and also for adding or removing more features in later updates. The document specifies the features that are going to be implemented, as well as the features that are not going to be implemented in the development. The document also shows the consensus of all the stakeholders involved regarding the requirements that should be included for development. Also, this document helps to specify requirements in more detail so that these can be easily mapped onto the architecture and design that would be built later on, also acts as input for system design, architecture, and developers' guidance and helps in testing. The intended audience consists of Project bidders, Project managers, Project Advisors, Team leaders, System designers, System Architects, Coders, Testers, Quality assurance teams, deployment engineers and other stakeholders.

C1.1.2 Document Conventions

This document follows the IEEE 830 / ISO/IEC/IEEE 29148 standards for Software Requirements Specifications. The conventions, notations, and formats used in this document are as follows:

1. **Text Conventions**:
    - *Italic text* is used for emphasis.
    - **Bold text** is used for section headers and key points.
    - **Monospace text** is used for code snippets and system outputs.
2. **Requirements Notation**:
    - Functional requirements are denoted by the prefix **FR** followed by a unique number (e.g., FR1, FR2).

- Non-functional requirements are denoted by the prefix **NFR** followed by a unique number (e.g., NFR1, NFR2).
- Use cases are referenced by the prefix **UC** followed by the use case number (e.g., UC1, UC2).

3. **Diagrams**:
   - Use case diagrams and other UML diagrams follow the standard UML notation.
   - Diagrams are included within the document with relevant descriptions and legends wherever appropriate.
   - Diagrams follow UML 2.x conventions.
   - Data modelling follows IDEF1X notation.
   - Interfaces are described using standard API and communication protocols (REST, JSON, HTTPS/TLS).
4. **Glossary**:
   - A glossary is provided at the end of the document to define terms and acronyms used throughout the SRS.

C1.1.3 Intended Audience and Reading Suggestions

This document is intended for a diverse group of stakeholders involved in the development, implementation, and use of the System. The key audiences and their respective reading suggestions are as follows:

1. **Project Managers**:
   - Focus on the sections outlining the overall project scope (Section 1.4), the functional and non-functional requirements (Sections 4 and 5), and the project schedule.
   - Review the risk analysis and mitigation strategies to understand potential project challenges.
2. **System Architects and Developers**:

- Pay special attention to the detailed functional requirements (Section 4) and system architecture.
- Study the use case diagrams and descriptions (Section 4.2) to gain a thorough understanding of system functionalities.
- Review the interface requirements for integration points with external systems and platforms.

3. **Quality Assurance (QA) and Testing Teams**:
    - Focus on the functional requirements (Section 4) to develop test cases and testing strategies.
    - Pay attention to the non-functional requirements (Section 5) to ensure performance, security, and usability standards are met.
    - Review the acceptance criteria and validation methods to prepare for system testing and validation.
4. **End Users and Clients**:
    - Read the overview of the system (Section 1.1) to understand the high-level functionality and benefits.
    - Review the user stories or use case descriptions (Section 4.2) to see how their needs and requirements are addressed.
    - Familiarize themselves with the user interface requirements (Section 5.1) to understand the expected user experience.
5. **Technical Support and Maintenance Teams**:
    - Focus on the sections detailing system architecture and interface requirements for technical insights.
    - Review the operational requirements to understand the system's deployment, monitoring, and maintenance needs.
6. **Academic Supervisors & Evaluators**:

- To assess the correctness, completeness, and compliance of system requirements with software engineering methodologies.

7. **Test Engineers & QA Teams**:
    - To derive test cases, acceptance criteria, and validation conditions.
8. **UI/UX Designers**:
    - To interpret user interface expectations, accessibility considerations, and interaction flows.

C1.1.4 Project Scope

The WeWay application is a next-generation mobile navigation platform designed to enhance driving safety, social collaboration, and AI-assisted decision-making. The application goes beyond traditional navigation systems by integrating contextual intelligence, landmark-based routing and community-driven updates.

C1.1.4.1. In-Scope Functionalities:

- Real-time map rendering with traffic layers
- AI-powered smart route recommendations
- Turn-by-turn navigation with landmark-based guidance
- Community event reporting (hazards, accidents, road closures)
- AI-based summarization of community reports
- POI recommendations (restaurants, gas stations, landmarks)
- User-friendly interface
- Authentication and profile management via Firebase
- Safe-driving mode with minimal UI
- Commenting on places and roads
- Saving places and organizing lists
- Admin moderation powered by AI filtering.

C1.1.4.1. Out Of Scope Functionalities:

- Family live location sharing & group invitation
- Emergency vehicle alerting

- Full offline navigation beyond cached routes
- Continuous real-time monitoring of emergency vehicles without official data sources
- Commercial integrations (ride-hailing, ticketing, advertising)
- Voice-activated conversational AI beyond navigation assistance
- Desktop/web version in this phase
- Hardware integrations (vehicle dashboards, IoT devices)

C1.1.5 References

The following documents and resources were referenced in the preparation of this Software Requirements Specification (SRS). These references provide additional context, standards, and guidelines that are integral to understanding and implementing the requirements outlined in this document.

External Standards & Resources

- IEEE 830 Software Requirements Specification Standard
- ISO/IEC/IEEE 29148 requirements engineering standard
- UML 2.x Specification for Use Case and Other UML Diagrams
- IDEF1X / Crow's Foot ERD Notation Standards
- UTF-8 Encoding Standard
- HTTPS/TLS 1.2+ security protocols
- Google Maps SDK & Google Places API Documentation
- Firebase Authentication, Firestore, Cloud Functions, Cloud Messaging

C1.1.6 Definitions, Acronyms, and Abbreviations

Term	Definition
SRS	Software Requirement Specifications
AI	Artificial Intelligence
POI	Point of Interest
GPS	Global Positioning System
ETA	Estimated Time of Arrival
API	Application Programming Interface
UI/UX	User Interface / User Experience
Firestore	NoSQL database service by Firebase
NLP	Natural Language Processing
FR	Functional Requirement
NFR	Non-Functional Requirement
UC	Use Case
Safe Driving Mode	Minimal interface mode triggered during vehicle movement
Geo-Anchor	Exact map location attached to user posts

C1.1.7 Overview of Document

Section / Subsection	Explanation (1–2 lines)
1. Introduction	Provides background, purpose, scope, conventions, and structure of the SRS document.
1.1 Purpose	States why the SRS exists and the goals it aims to achieve for all stakeholders.
1.2 Document Conventions	Describes the formatting, terminology, requirement identifiers, and modelling standards used.
1.3 Intended Audience and Reading Suggestions	Identifies all readers of this SRS and suggests which sections each should focus on.

1.4 Project Scope	Defines what the system will include and exclude, outlining the full boundary of the WeWay project.
1.4.1 In-Scope Functionalities	Defines what the system will include.
1.4.2 Out Of Scope Functionalities	Defines what the system will exclude.
1.5 References	Lists documents, standards, articles, and APIs referenced while preparing the SRS.
1.6 Definitions, Acronyms, and Abbreviations	Provides definitions of technical terms, acronyms, and abbreviations for clarity.
1.7 Overview of Document	Summarizes the structure of the SRS and briefly describes the purpose of each major section.
2. Overall Description	Gives high-level information about system context, environment, users, and constraints.
2.1 Product Perspective	Explains how WeWay fits into the broader ecosystem and its relationship with external systems.
2.1.1 System Context	Describes external actors and how the system interacts with APIs, databases, and services.
2.1.2 Interfaces	Provides a high-level summary of user, hardware, software, and communication interfaces.
2.1.3 Dependencies	Lists external APIs, services, and conditions the system relies on.
2.2 Product Functions	Summarizes the major capabilities of the system before detailed requirements are presented.
2.3 User Classes and Characteristics	Describes different user types (Guest, Registered User, Admin) and their access levels.
2.4 Operating Environment	Defines the hardware, platforms, OS, network, and environmental requirements.
2.5 Design and Implementation Constraints	State limitations such as required technologies, platform restrictions, and cost/time constraints.
2.6 Assumptions and Dependencies	List assumptions about user behaviour, environment, and dependencies on external systems.
3. External Interface Requirements	Specifies all external interfaces, including UI, APIs, hardware, and communication protocols.

3.1 User Interfaces	Describes the layout, interaction flow, visual design guidelines, and accessibility needs.
3.2 Hardware Interfaces	Identifies interactions with GPS, sensors, camera, microphone, and mobile hardware.
3.3 Software Interfaces	Defines interactions with SDKs, libraries, third-party services, and internal modules.
3.4 Communications Interfaces	Describes supported communication protocols and data exchange formats (REST, HTTPS, JSON).
3.5 API Interfaces	Provides details of internal and external APIs used for navigation, routes, events, and AI features.
4. System Features	Contains detailed use cases, functional descriptions, and all functional requirements.
4.1 Use Case Analysis	Introduces the use case modelling approach and how users interact with the system.
4.1.1 Use Case Diagrams	Presents UML diagrams showing actors and their interactions with system functions.
4.1.2 Use Case Descriptions	Provides informal narratives of system use cases, including flows and preconditions.
4.2 Functionality	Breaks system features into functional areas for clarity and implementation planning.
4.2.1 Functional Requirements	Lists all system behaviours in detail, each with a unique identifier (FR-x.x).
5. Nonfunctional Requirements	Covers performance, safety, security, usability, compliance, and system quality attributes.
5.1 Performance Requirements	Specifies expected system speed, response times, and scalability levels.
5.2 Safety Requirements	Describes safety-related behaviours like emergency alerts and safe-driving mode.
5.3 Security Requirements	States' login security, encryption needs, data privacy, and access control rules.
5.4 Software Quality Attributes	Covers reliability, maintainability, usability, portability, and system robustness.

5.5 Compliance Requirements	Identifies compliance with standards, regulations, and university guidelines.
5.6 Business Rules	Defines internal business logic governing user eligibility, posting rules, and system operation.
5.7 Regulatory Requirements	Includes legal, regional, and cultural requirements such as Saudi-specific language and privacy rules.
6. Other Requirements	Captures additional requirements such as database structure, backup, localization, and logging.
6.1 Database Requirements	Describes data entities, storage structures, Firestore/NoSQL rules, and indexing needs.
6.2 Logging, Monitoring, and Auditing Requirements	Details how the system logs user actions, admin moderation, errors, and events.
6.3 Localization / Internationalization Requirements	Specifies support for Arabic/English, RTL, and local formatting.
6.4 Disaster Recovery & Backup Requirements	Defines fallback mechanisms, data redundancy, and recovery processes.
6.5 Licensing Requirements	Lists licensing needs for Google APIs, AI services, Flutter packages, and assets.
6.6 Installation Requirements	Provides installation steps, app store packaging, and environment setup rules.
7. Appendix	Contains supporting details, auxiliary diagrams, glossary items, and traceability artefacts.
7.1 Supporting Information	Additional notes or reference information not included elsewhere.
7.2 Glossary	Defines all domain-specific terms used within the WeWay system.
7.3 Data Dictionary	Provides definitions, fields, and constraints of all major data elements.
7.4 Domain Models / Additional Diagrams	Includes ERDs, sequence diagrams, state charts, and architectural visuals.
7.5 Sample Inputs/Outputs	Shows example API payloads, event reports, and navigation outputs.
7.6 Traceability Aids	Contains requirement traceability matrices.

C1.2. Overall Description

C1.2.1 Product Perspective

The WeWay Social Map & Smart Navigation System is a **mobile-first, standalone cross-platform application** designed for iOS and Android using Flutter. It integrates with multiple cloud-based services and third-party APIs to deliver real-time navigation, social collaboration, emergency alerts, and AI-assisted decision support.

WeWay operates as a **next-generation navigation ecosystem**, extending beyond traditional map applications by embedding community-driven input, smart route recommendations. It functions as a **hybrid system**, part navigation engine, part social interaction layer, powered by Firebase for backend services and Google APIs for maps, places, and routing.

WeWay interacts with the following external systems:

Google Maps SDK for map visualization and route rendering

Google Places API for place search and landmark identification

Firebase Authentication for login and identity management

Firebase Firestore for storing events, comments, and user activity

Firebase Cloud Messaging for notifications

AI Platforms (OpenAI) for summarization, route intelligence, and content moderation

WeWay does not replace these systems; instead, it augments them with custom logic, front-end enhancements, and AI-driven social navigation capabilities.

C1.2.1.1 System Context

The system operates within a mobile environment where users interact through touch and voice interfaces.
The context includes:

Primary Actors

- **Guest User** can view the map, search, and start navigation, but cannot contribute or save content.

- **Registered users** can access all features, including posting, saving, and reporting.
- **Admin** oversees moderation, AI-flagged content, and platform safety.

External Dependencies

- Google APIs (Maps, Places, Directions)
- OpenAI services
- Firebase Authentication, Firestore, Storage, FCM
- Device GPS, accelerometer, and network services

Context Summary

At the highest level, the system:

1. Receives **input** from mobile users (navigation request, event report, comment, settings).
2. Uses external services to process data (routing, search, AI summarization).
3. Outputs **navigation instructions**, **alerts**, **recommendations**, and **community updates**.

C1.2.1.2 Interfaces

WeWay interacts with multiple interface types:

1. User Interfaces

- Touch-based mobile UI
- Map interaction (zoom, pan, tap, long-press)
- Voice interaction for hands-free reporting

2. Hardware Interfaces

- GPS for location
- Microphone for voice commands
- Camera for photo attachments
- Accelerometer & gyroscope for driving detection

3. Software Interfaces

- Firebase services for authentication, database, and storage
- Google Maps/Places APIs
- AI summarization and moderation engine

4. Communication Interfaces

- HTTPS/TLS 1.2+
- JSON-based REST services
- FCM notifications

C1.2.1.3 Dependencies

The system requires the following dependencies for proper functioning:

External Dependencies

- **Stable internet connection**
- **GPS accuracy** from device sensors
- **Google API availability**
- **Firebase uptime** (Auth, Firestore, Messaging)
- **AI platform availability**

Internal Dependencies

- Correct installation of Flutter packages
- Proper Firestore rules and indexes
- Accurate map tile rendering
- Device permissions (Location, Camera, Microphone)

User Dependencies

- Users enabling GPS and the internet
- Community participation for social updates
- Correct event reporting for reliable AI summaries

C1.2.1.4 Comparative Analysis: WeWay vs. Google Maps vs. Waze

High-Level Feature Category	WeWay	Google Maps	Waze
FR-1 Accounts & Access	✓ Full account system with social login, profile & recovery	✓ Full account system	✓ Basic account system
FR-2 Map, Search & Navigation	✓ Full map, search, routing	✓ Industry-leading map/search/routing	✓ Strong routing with traffic focus
FR-3 User Contributions (Places & Roads)	✓ Place & road comments, photos, threads	✓ Reviews on places only	✓ Hazard reports only (no threads or comments)
FR-4 AI Intelligence & Moderation	✓ AI summaries, NLP search, AI moderation	✕ No AI summaries or NLP	✕ No AI intelligence or moderation
FR-5 Safe-Driving Behavior	✓ Auto safe mode + passenger override	✕ No	✓ Minimal safe-driving UI
FR-6 Landmark-Based Guidance	✓ Landmark instructions & toggle	✕ No	✕ No
FR-7 Saving, Notifications & Admin	✓ Lists, saved places, alerts, admin console	✓ Saved places + limited alerts	✓ Basic alerts (traffic/police), no lists

C1.2.2 Product Functions

WeWay system provides a comprehensive set of functions designed to meet the needs of both end users and system administrators. These functions are derived from the extensive use cases identified for the system. This subsection provides a high-level summary of the major functionalities before detailed requirements appear in later sections.

1. Navigation & Routing

- Turn-by-turn navigation
- Smart route recommendation (fastest, safest, scenic)
- Landmark-based instructions
- Real-time rerouting

2. Map & Search

- Interactive maps with traffic
- Search for places, categories, and addresses
- Filter by distance, rating, “open now”

3. Community & Social Features

- Reporting road events (accident, hazard, traffic)
- Commenting on places and roads
- Likes, replies, abuse reports

4. AI Intelligence

- AI route scoring
- AI summarization of community reports
- AI-assisted moderation

5. Account & Personalization

- Login via Email, Google, Apple
- Profile editing
- Saving favourite places & creating lists

6. Admin Console

- Reviewing flagged posts
- Making moderation decisions
- Viewing user report history

C1.2.3 User Classes and Characteristics

The WeWay App is designed to cater to various user classes, each with distinct characteristics and needs. This section identifies and describes these user classes to ensure that the system meets the requirements of all potential users.

1. Guest User

- **Abilities:** Browse map, read comments, search places, start navigation
- **Restrictions:** Cannot post, react, report, or save places
- **Characteristics:** First-time users, casual users

2. Registered User

- **Abilities:** Full app access
- **Characteristics:** Daily commuters, drivers, residents, tourists
- **Motivations:** Navigation, social reporting, family travel

3. Admin

- **Abilities:** Moderate content, review reports, monitor abuse
- **Characteristics:** Internal platform staff
- **Tools:** Moderation console, AI-flagged content list

4. External Services (Non-human actors)

- Google APIs, AI platform, Firebase
- Provide critical navigation and data-processing functions

C1.2.4 Operating Environment

The WeWay App is designed to operate in a diverse technological environment, ensuring compatibility and accessibility for a wide range of users. The system must be adaptable to

various hardware and software configurations to deliver optimal performance and a seamless user experience.

WeWay must operate within:

Hardware Environment

- Smartphones (Android 8+, iOS 13+)
- GPS-enabled devices
- Devices with microphone and camera capabilities

Software Environment

- Flutter mobile application
- Firebase backend (Auth, Firestore, Storage, FCM)
- Google Maps SDK
- AI integration (Vertex AI or OpenAI)
- REST endpoints (FastAPI backend if needed)

Network Environment

- 4G/5G or Wi-Fi connection
- Stable latency for routing and data sync

Cultural Environment

- Language support (English)
- Landmark-based navigation

C1.2.5 Design and Implementation Constraints

The system is constrained by the following limitations:

1. Technological Constraints

- Must be built using **Flutter**
- Must use **Firebase** for cloud backend
- Must integrate Google Maps/Places APIs

- AI must rely on Vertex AI or OpenAI
- Data storage must comply with Firebase rules

2. Platform Constraints

- Mobile app only (Android/iOS)
- No desktop or web version in this phase

3. Cost & Budget Constraints

- Google API usage costs (future usage-based billing)
- Firebase storage and document read/write costs
- 100 SAR Figma course cost
- Two team members → limited time capacity

4. Security Constraints

- Mandatory HTTPS/TLS
- Authentication via Firebase only
- Location consent must be explicitly enabled

5. Environment Constraints

- Requires an active internet connection
- Requires permissions for GPS, camera, and microphone

C1.2.6 Assumptions and Dependencies

The development and deployment of the WeWay App are based on several key assumptions and dependencies. These factors must be considered to ensure the project's success.

Assumptions

- Users will grant location access for navigation.
- Internet connectivity will be available during active navigation.
- Google APIs will provide accurate and updated data.
- Community members will contribute reliable reports.
- Devices support push notifications.

Dependencies

- Firebase Authentication for login workflows
- Firestore for real-time data updates
- Google Maps SDK for navigation rendering
- AI model availability for summarization
- Device sensors for safe-driving mode detection

C1.3. External Interface Requirements

C1.3.1 User Interfaces

The user interface of WeWay must be intuitive, accessible, and optimized for mobile use cases involving driving, navigation, and quick interactions:

C1.3.1.1 General UI Requirements

- The UI **shall** follow a clean, minimalistic design to reduce user distraction while driving.
- The interface **shall** adapt to various screen sizes across Android and iOS devices.
- UI elements **shall** remain readable under bright sunlight and low-light environments.
- UI **shall** support responsive touch gestures: tap, swipe, pinch-to-zoom, long-press.
- All text **shall** be available in English.

C1.3.1.2 Screen Types

The primary app screens include:

1. Home / Map Screen

- Dynamic Google Map with zoom, pan, and traffic layers.
- Current location indicator.
- Search bar (top).
- “Report Event” button (floating).
- Nearby POIs.
- Re-centre location button.

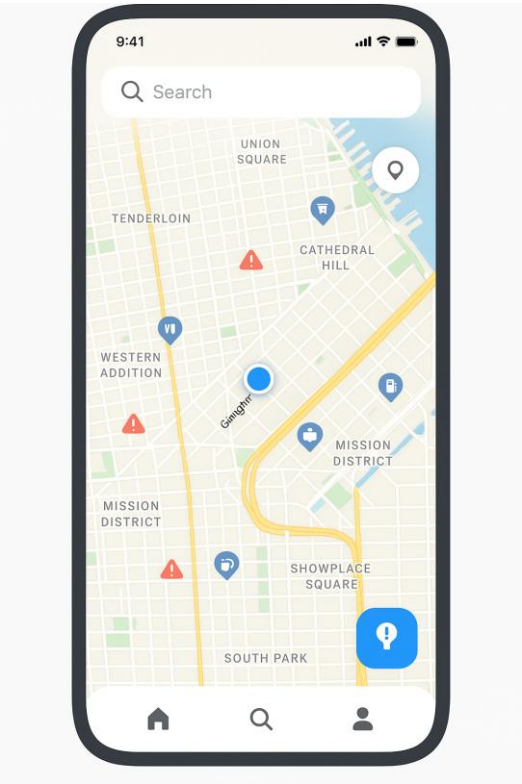

Figure 1: Home/Maps Screen

2. Navigation Screen

- Turn-by-turn instructions.
- Lane guidance (arrows).
- Landmark-enhanced instructions.
- ETA, distance, speed.
- Rerouting alerts.
- Safe-driving mode with simplified UI.

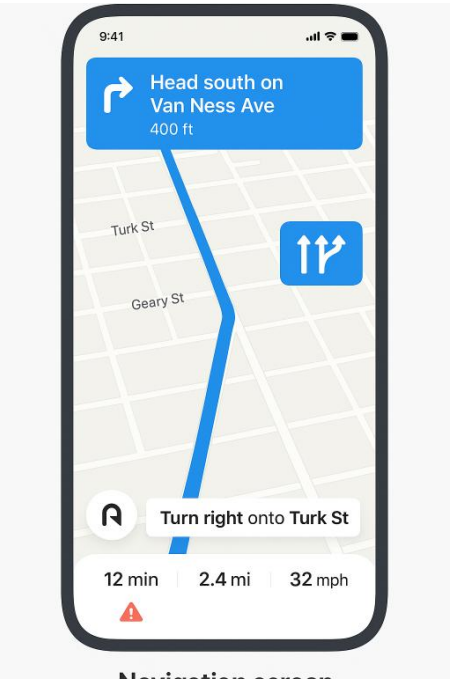

Figure 2: Navigation Screen

3. Search Screen

- Search bar with autocomplete suggestions.
- Filters (Open now, Distance, Rating).
- Search results list + map preview.

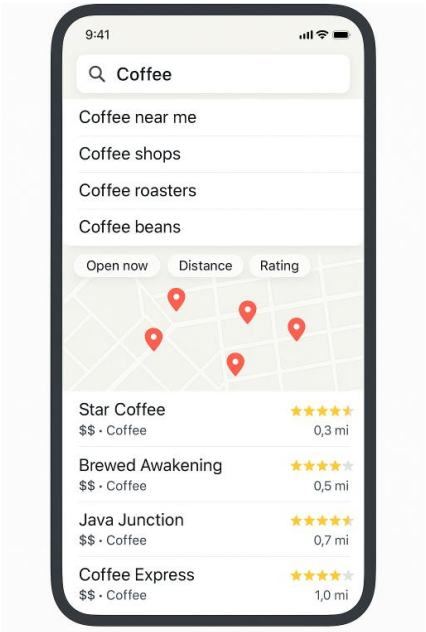

Figure 3: Search Screen

4. Place Detail Screen

- Name, rating, distance, opening hours.
- Photos.
- User comments and road threads.
- "Navigate," "Save," "Share," and "Report" options.

Figure 4: Place Details Screen

5. Event Reporting Screen

- Event categories (accident, road closure, traffic, hazard).
- Optional text and photo uploads.
- Location auto-filled.

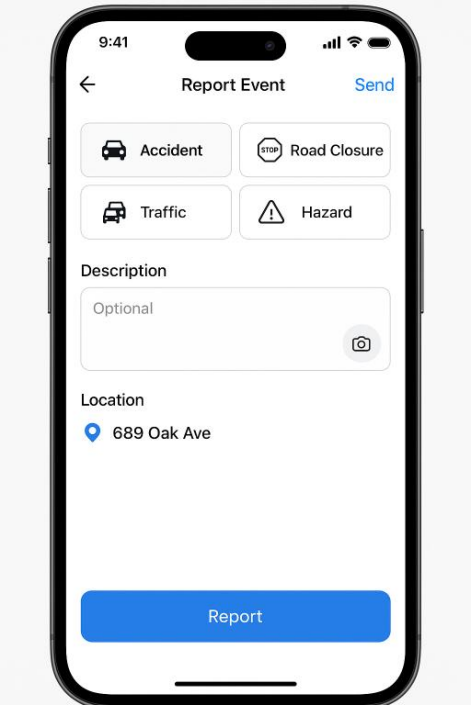

Figure 5: Report Event Screen

6. AI Insights Screen

- Route summaries.
- Community event summaries.
- Source links to original posts.

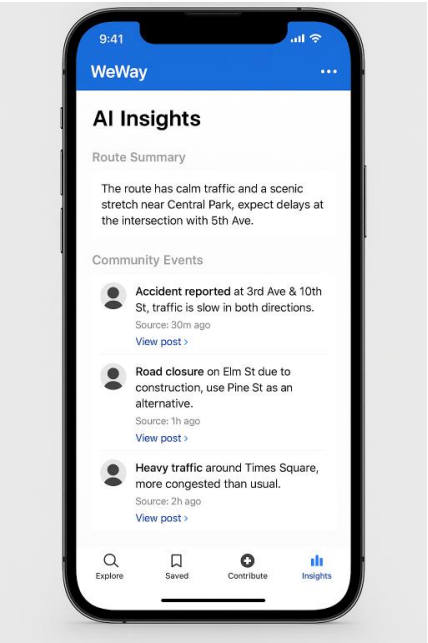

Figure 6: AI Insights Screen

7. Profile & Settings Screen

- Profile photo, name, bio.
- Account settings.
- Privacy controls.
- Saved lists.

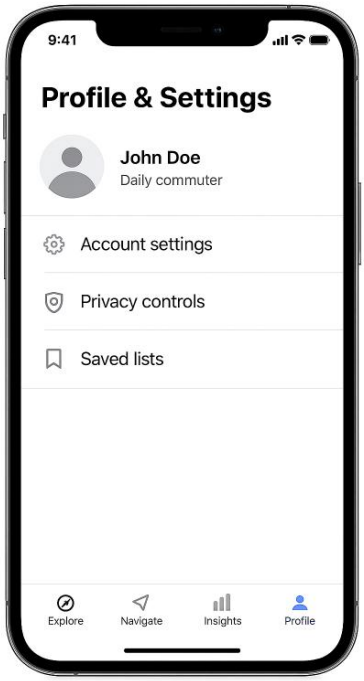

Figure 7: Profile and Setting Screen

8. Admin Moderation Screen (Admin Only)

- Flagged posts.
- Abuse reports.
- Decision actions (Approve, Remove).
- User violation logs.

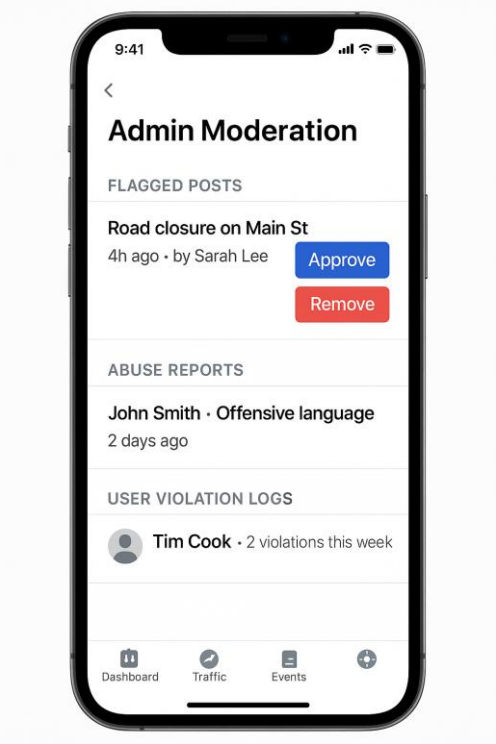

Figure 8: Admin Moderation Screen

9. Add Comments Flow Screen

- Displays selected place/road name, preview images, rating, distance, and status.
- Shows popular comments with user profiles, timestamps, likes, replies, and share options.
- Provides “Add Comment” entry point for posting a new contribution.
- Allows attaching photos, viewing location coordinates, and submitting the comment.
- Offers quick access to map preview, media thumbnails, and detailed place information.

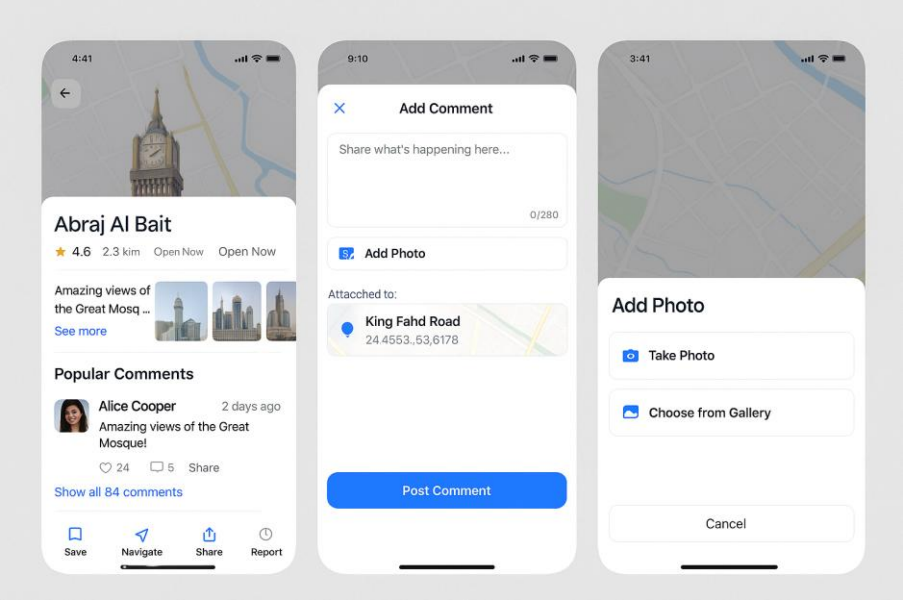

Figure 9: Add comments flow screen

10. Road Thread View Screen

- Shows road segment name and corresponding geographic location on the map.
- Displays a list of user comments related to that specific road segment.
- Includes avatars, usernames, timestamps, text comments, and attached photos.
- Provides an "Add Comment" button to contribute road-specific updates.
- Automatically links each comment to exact geo-coordinates for accuracy.

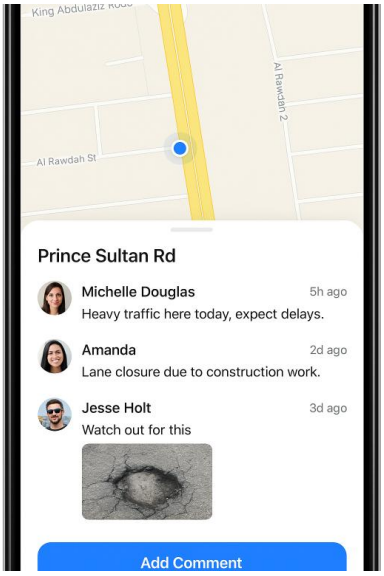

Figure 10: Road thread screen

10. Safe Driving Mode Panel Screen

- Activates a simplified interface optimized for minimal driver distraction.
- Displays quick-action safety tiles (Road Closure, Hazard, Police, Accident, Pothole, Quick Photo Report).
- Includes voice-input button for hands-free reporting.
- Features Safe Driving Mode toggle with clear visual indication.
- Provides accessible navigation tabs (Home, Search, Contribute, Profile) in reduced-distraction format.

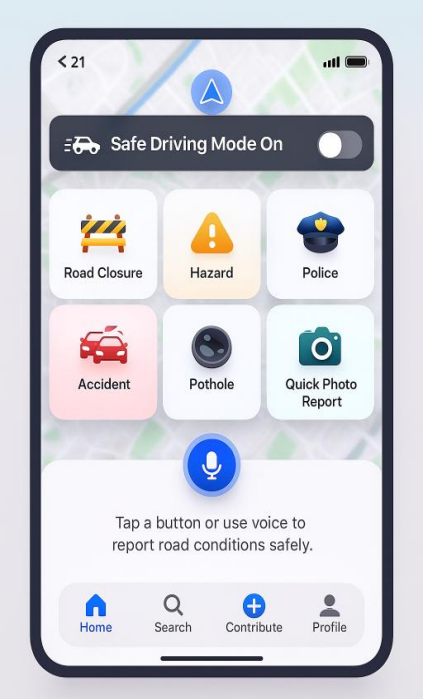

Figure 11: Safe driving mode panel screen

C1.3.1.3 Accessibility Requirements

- Font sizes must be adjustable.
- High-contrast mode must be supported.
- Buttons must be large enough for touch while driving.
- Screen reading (TalkBack / VoiceOver) must function on all essential elements.

C1.3.2 Hardware Interfaces

WeWay relies on specific device hardware to deliver navigation, reporting, and social features.

GPS Module

- Required for navigation, event reporting.
- The system **shall** request “Allow While Using App” permission at a minimum.

Mobile Network Hardware

- Required for accessing map tiles, route computation, AI summarization, and event synchronization.

Camera

- Used for attaching photos to place comments or event reports.
- Permission: `android.permission.CAMERA` / `NSCameraUsageDescription`.

Microphone

- Used for voice-based reporting in safe-driving mode.
- Permission: `RECORD_AUDIO`.

Accelerometer/Gyroscope

- Detects motion characteristics to enable Safe-Driving Mode.

C1.3.3 Software Interfaces

WeWay interacts with external software platforms and internal modules.

Firebase Authentication

- Provide login with Email/Password, Google Sign-In, and Apple Sign-In.
- Provides JWT tokens for secure sessions.

Firebase Firestore

- Stores reports, comments, user information, lists, group locations, and settings.
- Must follow NoSQL best practices and index frequently queried fields.

Firebase Cloud Storage

- Stores user-uploaded photos.
- Enforces size and file-type constraints.

Firebase Cloud Messaging (FCM)

- Push notifications for alerts, replies, and mentions.

Google Maps SDK

- Supplies map tiles, shapes, traffic layers, markers, and polylines.

Google Places API

- Provides POI search, categories, and landmark detection.

Directions API

- Computes turn-by-turn directions, ETA, and reroutes.

AI Engine (OpenAI)

- Summarizes community events.
- Provides natural language discovery.
- Moderates user content.

Internal FastAPI Backend (Optional)

If needed for advanced processing:

- AI caching
- Proxying AI calls
- Admin tools

C1.3.4 Communications Interfaces

The system uses multiple communication protocols:

HTTPS / TLS

- All communication must use encrypted HTTPS.
- API calls must support TLS 1.2 or higher.

REST API

- JSON-based request/response pattern.
- Idempotent GET endpoints.
- POST endpoints for reporting events and comments.

Notification Channels

- FCM channels for:
    - Nearby event warnings
    - Replies and mentions
    - Route updates

C1.3.5 API Interfaces

This section summarizes the key APIs used by WeWay.

Google Maps API

- **Functions:** Map rendering, polyline drawing, and traffic overlay.

Input:

- API key
- Latitude/longitude
- Map configuration

Output:

- Map tiles
- Marker positions
- Traffic overlays

Google Places API

- **Functions:** Search places, autocompletion, landmark detection.

Input:

- Query text
- Filters (Open now, rating, distance)
- Location bias

Output:

- Place IDs
- Coordinates
- Place details

Directions API

- **Functions:** Provides routing, ETA, and turn-by-turn geometry.

Input:

- Start + destination
- Route mode (driving)

Output:

- Steps
- Polyline
- ETA
- Alternative routes (fastest, safest, scenic)

AI Summarization API (OpenAI)

- **Functions:** Summarizes event clusters, filters noise, and detects patterns.

Input:

- Collection of event reports
- Road segment context

Output:

- Clean structured summary
- Confidence score
- Source reference links

Firebase Endpoints

(Not traditional APIs; uses SDK functions)

- Firestore read/write
- Authentication token lifecycle
- Media upload/download
- Messaging subscriptions

C1.4. System Features

C1.4.1 Use Case Analysis

C1.4.1.1. Use Case Diagrams

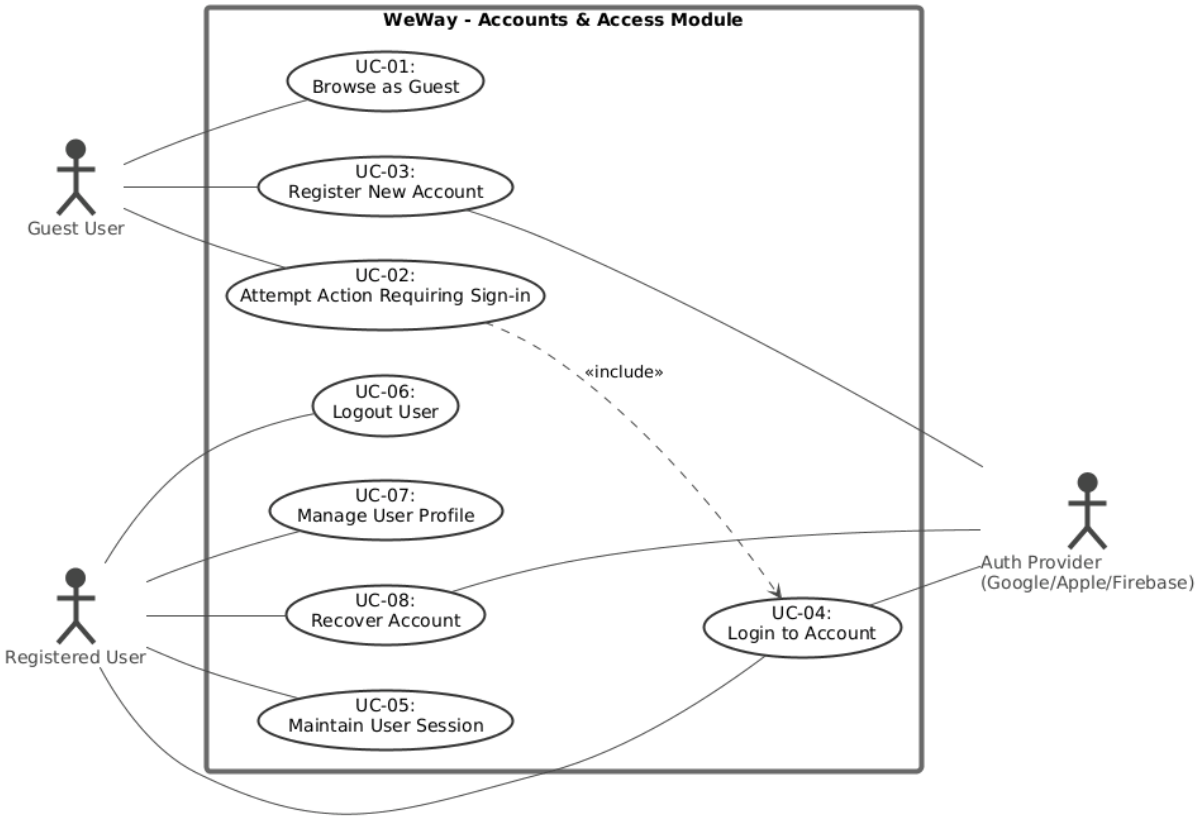

Figure 12: Use Case Diagram - Accounts & Access Module

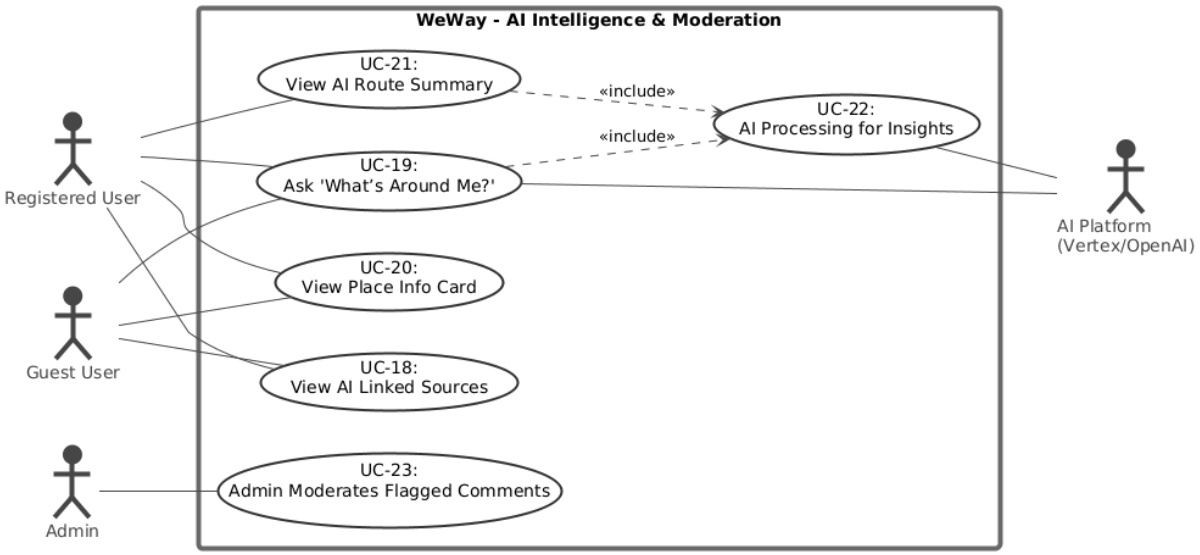

Figure 13: Use Case Diagram - AI Intelligence & Moderation Module

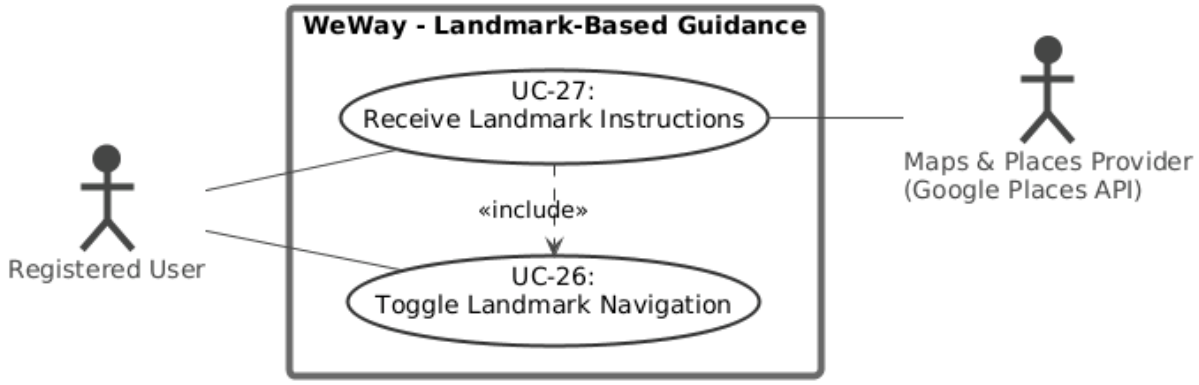

Figure 14: Use Case Diagram - Landmark-Based Guidance Module

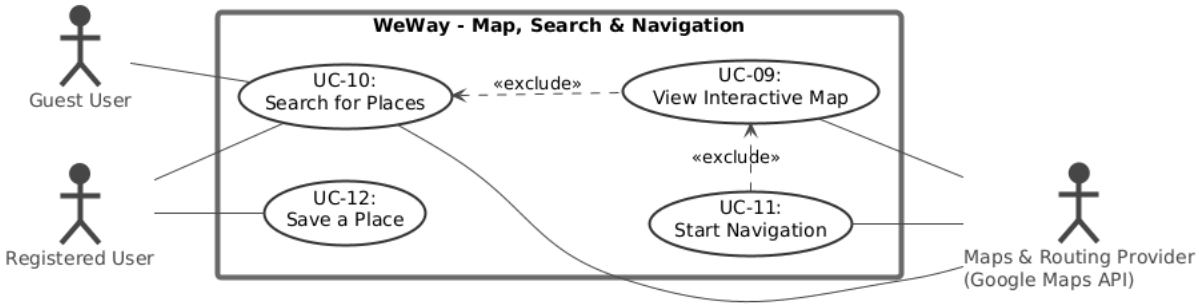

Figure 15: Use Case Diagram - Map, Search & Navigation Module

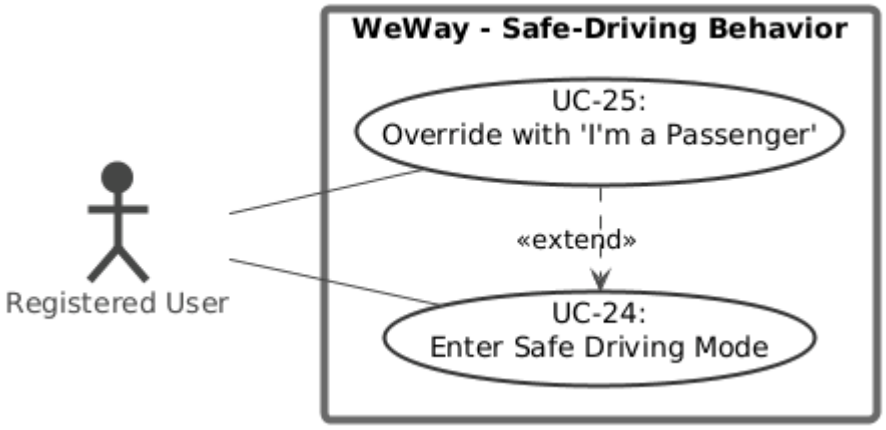

Figure 16: Use Case Diagram - Safe-Driving Behavior Module

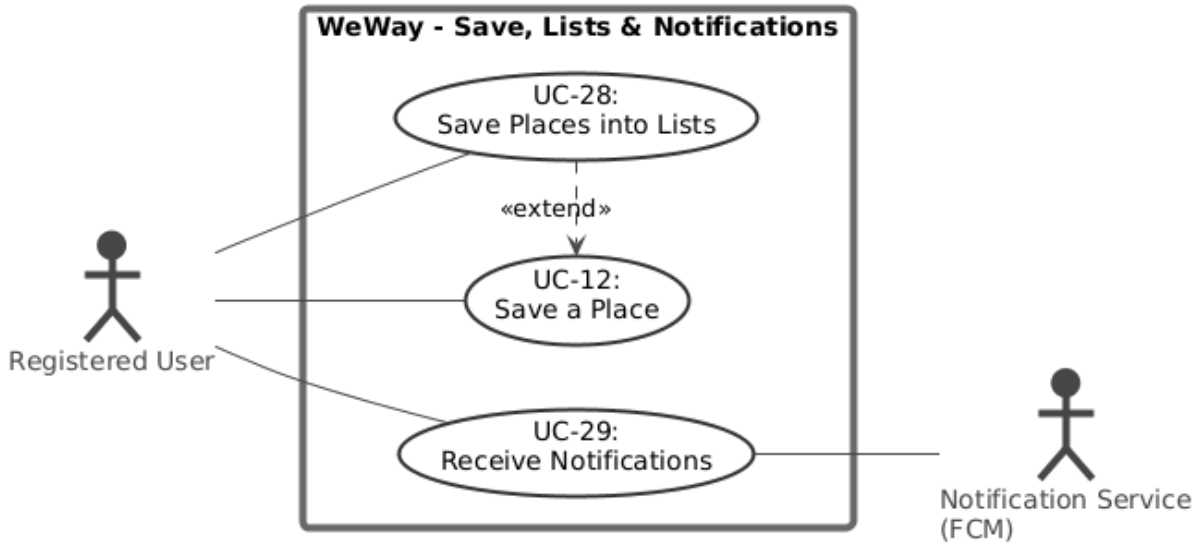

Figure 17: Use Case Diagram - Save, List & Notifications Module

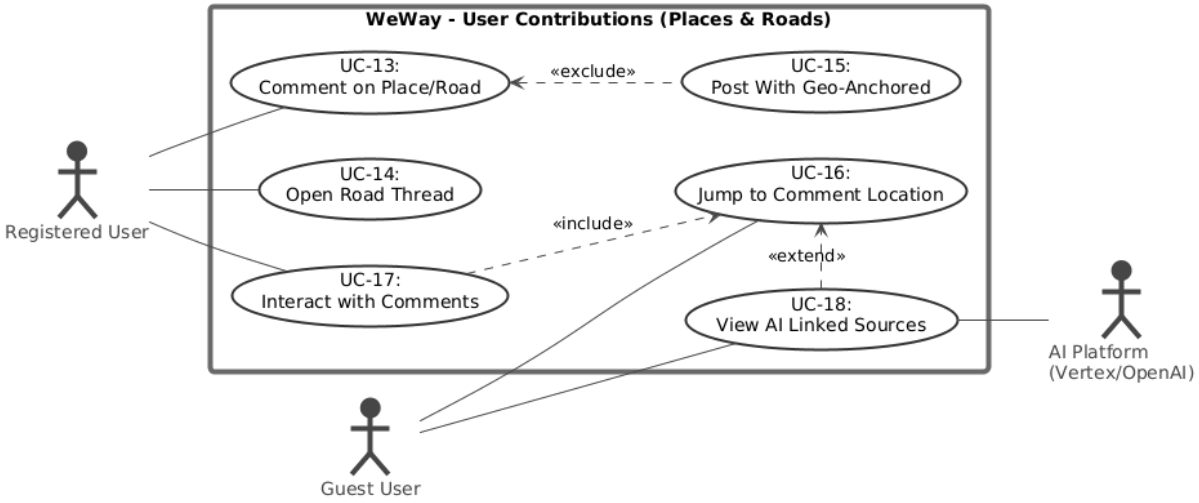

Figure 18: Use Case Diagram – User Contributions (Places & Roads) Module

C1.4.1.2 Use Cases Descriptions (Informal):

Use Case Title (ID)	Actor Names	Requirement Reference ID	Description
UC-01: Browse as Guest	Guest User	FR-1.1	This use case describes how a Guest user can browse the map, search places, view POI details, and start navigation without signing in.
UC-02: Attempt Action Requiring Sign-in	Guest User	FR-1.2	This use case describes how the system prompts a Guest to sign in when attempting to post a comment, react, or report an event.
UC-03: Register New Account	Guest User	FR-1.3	This use case describes how a user creates a new account using Email/Password, Google, or Apple Sign-In.
UC-04: Login to Account	Guest User / Registered User	FR-1.3	This use case describes how a user logs into the system using supported authentication methods.
UC-05: Maintain User Session	Registered User	FR-1.4	This use case describes how the system keeps a registered user signed in after closing and reopening the app.
UC-06: Logout User	Registered User	FR-1.4	This use case describes the process of a registered user logging out of their active session.
UC-07: Manage User Profile	Registered User	FR-1.5	This use case describes how a user updates profile details such as name, avatar, bio, and privacy settings.
UC-08: Recover Account	Registered User	FR-1.6	This use case describes how a user resets their password or

			relinks a social login method when access is lost.
UC-09: View Interactive Map	Guest + Registered User	FR-2.1	This use case describes how the user interacts with a smooth, zoomable map showing the current location and traffic.
UC-10: Search for Places	Guest + Registered User	FR-2.2	This use case describes how a user searches for places by name, address, or category and applies filters like rating or distance.
UC-11: Start Navigation	Guest + Registered User	FR-2.3	This use case describes how a user requests turn-by-turn directions with ETA and rerouting.
UC-12: Save a Place	Registered User	FR-2.4	This use case describes how a user saves a place to favorites or a custom list.
UC-13: Comment on Place or Road	Registered User	FR-3.1	This use case describes how a user posts text or a photo in a place or road comment thread.
UC-14: Open Road Thread	Registered User	FR-3.2	This use case describes how long-pressing a road segment opens a thread dedicated to that section of the road.
UC-15: Geo-Anchored Posting	Registered User	FR-3.3	This use case describes how a posted comment is attached to its exact map location.
UC-16: Jump to a Comment’s Location	Guest + Registered User	FR-3.4	This use case describes how selecting a comment moves the map to the exact location where it was posted.
UC-17: Interact with Comments (Like/Reply/Report)	Registered User	FR-3.5	This use case describes how users like, reply, report, or share comments within a discussion thread.

UC-18: View AI Linked Sources	Guest + Registered User	FR-3.6	This use case describes how users can view the original posts that contributed to an AI-generated summary.
UC-19: Ask “What’s Around Me?”	Guest + Registered User	FR-4.1	This use case describes how a user submits a natural-language query (e.g., “restaurants near me”) and receives contextual results.
UC-20: View Place Info Card	Guest + Registered User	FR-4.2	This use case describes how tapping a map marker displays the place info card with rating, distance, and quick actions.
UC-21: View AI Route Summary	Registered User	FR-4.3	This use case describes how users receive an AI-generated summary of community posts along their selected route.
UC-22: AI Processing for Insights	System / AI Platform	FR-4.4	This use case describes how the system sends route or event data to an AI platform and receives validated, safe output.
UC-23: Admin Moderates Flagged Comments	Admin	FR-4.5	This use case describes how the Admin reviews AI-flagged comments, then approves or removes them.
UC-24: Enter Safe Driving Mode Automatically	Registered User	FR-5.1	This use case describes how the app detects vehicle movement and switches UI to a simplified safe-driving mode.
UC-25: Override Safe Driving With “I’m a Passenger”	Registered User	FR-5.2	This use case describes how a passenger restores full app functionality while the car is moving.
UC-26: Toggle Landmark-Based Navigation	Registered User	FR-6.1	This use case describes how a user enables or disables landmark-based instruction mode.

UC-27: Receive Landmark Navigation Instructions	Registered User	FR-6.2	This use case describes how navigation instructions show a nearby landmark if one exists within 200 meters of a turn.
UC-28: Save Places into Lists	Registered User	FR-7.1	This use case describes how a user organizes saved places into lists such as “Favorites” or “Trip Plan”.
UC-29: Receive Notifications	Registered User	FR-7.2	This use case describes how users receive push notifications about replies, mentions, along-route alerts, or events.

C1.4.2 Functionality

C1.4.2.1 Functional Requirements

FR-1 Accounts & Access

Reference	**Feature Name**	**Actor**	**Functional Requirement**	**Use Case Reference**
FR-1.1	Guest Mode	Guest User	The user shall be able to browse the map, search places, view POI details, and initiate navigation without signing in. The user shall be able to interact with the live map, zoom, pan, and select points of interest, while being restricted from actions requiring authentication, such as posting, reacting, reporting, or saving places.	UC-01
FR-1.2	Sign-in Gate	Guest User	The user shall be able to trigger a sign-in prompt whenever they attempt an action requiring authentication, such as commenting, reacting, or reporting. The user shall be able to resume the originally	UC-02

			attempted action immediately after successfully signing in.	
FR-1.3	Sign-in & Registration	Guest User	The user shall be able to register or sign in using Email/Password, Google, or Apple credentials. The user shall be able to enter required fields, validate their identity, and gain access to a secure, authenticated profile through the supported login mechanism.	UC-03, UC-04
FR-1.4	Session Management	Registered User	The user shall be able to remain signed in across app restarts through secure token retention, and shall be able to manually log out at any time. Upon logout, the user shall be able to return to Guest Mode with all authentication data cleared.	UC-05, UC-06
FR-1.5	Profile Management	Registered User	The user shall be able to view and update their profile details, including name, avatar, and bio, as well as configure privacy preferences. The user shall be able to save updates, which must immediately reflect in their profile view.	UC-07
FR-1.6	Account Recovery	Registered User	The user shall be able to recover account access using password-reset emails or relinking options for Google/Apple sign-ins. The user shall be able to restore their account through a secure, validated recovery process.	UC-08

FR-2 Map, Search & Navigation

Reference	Feature Name	Actor	Functional Requirement	Use Case Reference
FR-2.1	Map Display	Guest + Registered User	The user shall be able to view an interactive map that supports zooming, panning, traffic layers, and live location tracking. The user shall be able to select any map point or POI to view details.	UC-09
FR-2.2	Search	Guest + Registered User	The user shall be able to search for places by entering text, selecting autocomplete suggestions, and applying filters such as rating, distance, or "open now." The user shall be able to view search results on the map or in a list.	UC-10
FR-2.3	Navigation	Guest + Registered User	The user shall be able to request turn-by-turn navigation, view ETA, distance, lane guidance, and receive automatic rerouting when deviating from the path. The user shall be able to follow visual and voice-guided instructions.	UC-11
FR-2.4	Save a Place	Registered User	The user shall be able to save any place displayed on the map or in search results to their favorites or a selected custom list. Saved places shall be accessible across sessions.	UC-12

FR-3 User Contributions (Places & Roads)

Reference	Feature Name	Actor	Functional Requirement	Use Case Reference
FR-3.1	Place & Road Comments	Registered User	The user shall be able to write and submit comments on places and road segments, attach photos, and contribute to community threads. Submitted comments shall immediately appear in the corresponding thread.	UC-13
FR-3.2	Road Threads	Registered User	The user shall be able to open the discussion thread for any road segment by long-pressing on the road. The user shall be able to browse and add comments within that specific segment's thread.	UC-14
FR-3.3	Geo-Anchored Posting	Registered User	The user shall be able to have each comment automatically tagged with its exact GPS coordinates. The user shall be able to later revisit these posts as map-based anchors representing the physical location of each contribution.	UC-15
FR-3.4	Jump to Post Location	Guest + Registered User	The user shall be able to tap any comment and instantly jump to the exact map location where the comment was originally posted. The user shall be able to view the geographic context of all contributions.	UC-16
FR-3.5	Comment Interactions	Registered User	The user shall be able to like, reply to, report, or share comments. The user shall be able to perform these interactions	UC-17

			smoothly, with all updates reflected immediately within the thread.	
FR-3.6	AI Summary Source Linking	Guest + Registered User	The user shall be able to open the original comment sources for any AI-generated summary item. The user shall be able to trace how the summary was produced by viewing linked posts.	UC-18

FR-4 AI Intelligence & Moderation

Reference	**Feature Name**	**Actor**	**Functional Requirement**	**Use Case Reference**
FR-4.1	Natural-Language Discovery	Guest + Registered User	The user shall be able to ask conversational queries such as “What’s around me?” or “Best restaurants nearby,” and receive AI-processed suggestions tailored to their current location and context.	UC-19
FR-4.2	Place Info Card	Guest + Registered User	The user shall be able to tap any place pin or search result to open a detailed place info card showing ratings, opening hours, distance, photos, and available actions like saving or navigating.	UC-20
FR-4.3	AI Route Summary	Registered User	The user shall be able to receive an AI-generated summary of meaningful comments and road-related updates along their navigation route. The user shall be able to review this summary before or during navigation.	UC-21

FR-4.4	AI Processing Engine	System / AI Platform	The system shall be able to process user queries, comments, and route data using AI models, delivering summarized insights and classifications. *(System-level, no direct user action; kept for traceability)*	UC-22
FR-4.5	AI-Assisted Moderation	Admin	The user (Admin) shall be able to review all AI-flagged comments and decide whether to approve or remove them. The admin user shall be able to access violation logs and moderation tools.	UC-23

FR-5 Safe-Driving Behavior

Reference	**Feature Name**	**Actor**	**Functional Requirement**	**Use Case Reference**
FR-5.1	Safe Driving Mode	Registered User	The user shall be able to automatically enter Safe Driving Mode when the system detects driving patterns. The user shall be able to interact with a simplified UI with large icons, minimal distractions, and one-tap or voice-based report actions.	UC-24
FR-5.2	Passenger Override	Registered User	The user shall be able to override Safe Driving Mode by selecting the “I am a passenger” toggle, which restores full app functionality and normal interaction options.	UC-25

FR-6 Landmark-Based Guidance

Reference	Feature Name	Actor	Functional Requirement	Use Case Reference
FR-6.1	Landmark Toggle	Registered User	The user shall be able to enable or disable landmark-based navigation through the settings interface. When enabled, the system shall enhance instructions with recognizable landmarks along the route.	UC-26
FR-6.2	Landmark 200-Meter Rule	Registered User	The user shall be able to receive landmark-based navigation instructions whenever a suitable landmark exists within 200 meters of a turn or exit. When none exist, the user shall receive standard street-based instructions.	UC-27

FR-7 Saving, Lists & Notifications

Reference	Feature Name	Actor	Functional Requirement	Use Case Reference
FR-7.1	Save & Lists	Registered User	The user shall be able to create custom lists, add saved places to them, rename lists, and organize places according to personal preferences. All list changes shall sync across sessions and devices.	UC-28
FR-7.2	Notifications	Registered User	The user shall be able to receive push notifications for replies, mentions, safety alerts, road events, and updates along their route. The user shall be able to manage notification preferences by category.	UC-29

C1.5. Nonfunctional Requirements

C1.5.1 Performance Requirements

- The system shall load the home map screen quickly under normal mobile data or Wi-Fi conditions.
- The user shall experience minimal delay when retrieving traffic overlays and GPS location.
- Search queries shall return autocomplete suggestions promptly with a smooth response.
- Full search results shall appear without noticeable lag under typical network conditions.
- Navigation routes shall generate quickly, and rerouting shall occur promptly when the user deviates.
- The user shall be able to submit comments, photos, and road reports with near-instant reflection in threads.
- AI-based features, such as summaries or "what's around me," shall return results within a reasonable time.
- The backend shall support stable performance during peak usage and heavy concurrency.

C1.5.2 Safety Requirements

- The system shall automatically activate Safe Driving Mode when driving movement is detected.
- The user shall be presented with a simplified UI containing large, low-distraction controls while driving.
- High-distraction interactions (scrolling, long text input, browsing) shall be limited during driving.
- The user shall be able to override Safe Driving Mode using an "I'm a passenger" option.
- Voice-based interactions shall be available when possible to reduce the need for manual input.

- Safety-related alerts (hazards, incidents, emergency warnings, landmark instructions) shall be prominent and non-intrusive to core navigation.

C1.5.3 Security Requirements

- The system shall encrypt all communication between the mobile app, Firebase services, Google APIs, and AI platforms using **HTTPS/TLS 1.2+**.
- The system shall authenticate users through secure, industry-standard methods including **Email/Password, Google Sign-In, and Apple Sign-In**.
- All passwords shall be stored only as **hashed and salted** values using Firebase Authentication's secure hashing mechanisms.
- The system shall enforce strong password rules (minimum **8–10 characters**, including uppercase letters, numbers, and optional symbols).
- Password fields in the UI shall always be masked to prevent shoulder-surfing, and never displayed or stored in plaintext.
- The system shall implement a secure **Forgot Password** feature that sends password-reset emails to verified users.
- Authentication tokens, session IDs, and sensitive data shall be stored exclusively in secure device storage (Secure Enclave / Android Keystore).
- The system shall enforce **role-based access control (RBAC)**, distinguishing Guest, Registered User, and Admin permissions.
- Administrative functions (moderation, removal actions, violation logs) shall be restricted to Admin accounts and protected with strict access controls.
- User-uploaded media (photos, attachments) shall be stored securely in Firebase Storage with **restricted read/write rules**.
- All API keys, Google Maps keys, and AI credentials shall be stored in a **secure secrets management service**, not inside the mobile app.

- The system shall sanitize all user inputs and use Firebase SDK operations to avoid injection-style attacks or malformed database queries.
- Rate-limiting mechanisms shall be applied to prevent brute-force login attempts, automated abuse, and excessive API usage.
- The system shall maintain tamper-proof audit logs for critical actions such as login attempts, posting, reporting, and admin moderation.
- The system shall automatically flag harmful or inappropriate user content using **AI-assisted moderation**, followed by human verification.

C1.5.4 Software Quality Attributes

The WeWay App must exhibit the following software quality attributes to ensure a high-quality user experience and maintainability:

Reliability

- The system shall operate consistently and predictably across all supported features.
- Automated testing shall cover the majority of core system logic to ensure reliable behavior.
- Failover strategies such as retries and fallbacks shall be implemented to prevent interruptions.

Availability:

- The system shall remain accessible with minimal downtime, except during scheduled maintenance.
- Modular architecture shall allow updates to individual services without affecting the whole system.
- Cloud infrastructure shall support continuous operation and quick recovery.

Maintainability:

- The system shall use modular, loosely coupled components that are easy to update and debug.

- Comprehensive technical documentation shall be provided for all modules and integrations.

- Logging and monitoring tools shall support quick issue identification and resolution.

Usability:

- The user interface shall be simple, intuitive, and consistent across all screens.

- Users shall be able to navigate key features with minimal learning effort.

- Accessibility practices such as clear text sizes, colour contrast, and simplified driving modes shall be applied.

- Built-in user feedback mechanisms shall be provided to help improve usability over time.

Portability:

- The backend shall support deployment on multiple operating systems.

- The mobile application shall function smoothly on both iOS and Android platforms.

- The system architecture shall allow migration across cloud environments if needed.

Scalability:

- The system shall support growth in the number of users, reports, and AI requests.

- Infrastructure shall scale horizontally and vertically as usage increases.

- Caching and batching mechanisms shall be used to optimize performance.

C1.5.5 Compliance Requirements

- The system shall comply with relevant privacy and data protection regulations.

- Users shall be clearly informed about data usage, background location access, and content storage practices.

- Cookie, tracking, and data-usage disclosures shall follow regulatory standards.

- Use of Google Maps APIs and AI services shall comply with licensing and terms of service.

- Moderation workflows shall follow fairness and transparency expectations.

- Device permissions (location, camera, storage, notifications) shall follow platform guidelines and require explicit user consent.

C1.5.6 Business Rules

- **BR-1:** All user accounts shall be unique and must be authenticated using email/password or approved social login providers (Google or Apple).
- **BR-2:** All user authentication credentials shall be securely stored and must follow Firebase Authentication's recommended security practices.
- **BR-3:** Guest users shall not be permitted to post comments, react to posts, report events, or save places; these features are restricted to Registered Users only.
- **BR-4:** All user-generated comments, photos, and reports must comply with community guidelines and shall be subject to moderation.
- **BR-5:** Any content flagged by AI or users must be reviewed by an Admin before removal or reinstatement.
- **BR-6:** Road and place comments must be associated with a valid location or road segment and shall always include a timestamp and user identifier.
- **BR-7:** Geo-anchored posts must be displayed at their correct coordinates and shall not be editable once submitted, except by Admin.
- **BR-8:** AI-generated route summaries and insights must only be created using recent, verified user contributions to ensure accuracy.
- **BR-9:** Users shall only receive notifications for categories they explicitly opted into, following platform-level permission rules.

- **BR-10:** Safe Driving Mode must automatically activate when driving conditions are detected, and shall restrict high-distraction actions unless overridden by the "I'm a passenger" declaration.
- **BR-11:** Landmark-based navigation shall only be used when the user has enabled the feature and when a suitable landmark exists within 200 meters.
- **BR-12:** Saved places must be stored under the user's authenticated profile and must sync across all logged-in devices.
- **BR-13:** AI natural-language discovery features must respond only to permitted query types and may not fulfil restricted or unsafe requests.
- **BR-14:** Admin users must maintain accurate moderation logs, including timestamp, action taken, and moderator identity.
- **BR-15:** All map, routing, and place data must comply with Google Maps Platform licensing and attribution rules.
- **BR-16:** Users must grant explicit permission for location access before navigation, live updates, or context-aware features can be activated.
- **BR-17:** Reported hazards or incidents must include required metadata (location, type, timestamp) before being accepted.
- **BR-18:** Duplicate or spam reports must be automatically filtered or consolidated by the system to prevent misinformation.
- **BR-19:** The system may not store any sensitive information (passwords, payment data, face images, etc.) outside approved secure storage services.
- **BR-20:** The platform must enforce account suspension or restriction for repeated violations of community or safety rules.

C1.5.6 Regulatory Requirements

- The system shall comply with the integration and usage rules of all external services, including Google Maps, Places, and Directions APIs.

- The system shall follow authentication and data-handling requirements defined by Firebase Authentication, Firestore, Cloud Storage, and related cloud services.

- The application shall comply with the operational and licensing policies of AI service providers, such as OpenAI.

- The system shall optimize resource usage, including processor, memory, and network bandwidth, to operate efficiently across a wide range of devices and conditions.

- The system shall avoid unnecessary processing and minimize data transfer overhead by using caching, batching, and optimized API usage patterns.

- The system shall maintain low latency for map rendering, AI outputs, and community content updates to ensure a responsive user experience.

- The architecture shall be designed to support the integration of future third-party services without requiring significant system redesign or violating platform restrictions.

C1.6. Other Requirements

C1.6.1 Database Requirements

- The system shall store all persistent data (users, comments, road threads, saved places, summaries, violation logs) using a cloud-based NoSQL database such as Firebase Firestore.
- The system shall maintain separate collections for *Users*, *Comments*, *RoadThreads*, *SavedLists*, *Notifications*, and *ModerationLogs*.
- Each database document shall contain unique IDs, timestamps, user references, and location metadata when applicable.
- Geo-anchored posts shall store precise latitude/longitude using Firestore GeoPoint for spatial queries.
- Road thread comments shall be indexed by road segment identifiers to enable fast retrieval.
- All write and read operations shall be secured using Firestore Security Rules defining user-specific access permissions.
- The database shall support real-time synchronization across all user devices.
- The system shall scale automatically for high-volume traffic using Firestore's horizontal sharding.

C1.6.2 Logging, Monitoring, and Auditing Requirements

- The system shall log key events, including authentication, navigation requests, comment submissions, AI requests, and system errors.
- Crash reports shall be captured using Firebase Crashlytics or equivalent monitoring tools.
- Performance logs (latency, API failures, slow responses) shall be recorded using platform monitoring tools.
- Administrative moderation actions shall be auditable, storing admin ID, timestamp, and decision details (approve/reject).
- AI moderation outputs (flagged items, confidence scores) shall be logged for later review.
- Logs shall be retained securely and made accessible only to authorized personnel.

- The system shall provide monitoring dashboards to track app health, feature usage, and error trends.

C1.6.3 Localization / Internationalization Requirements

- The system shall support operations initially offering English.
- Text strings shall be stored using internationalization frameworks (not hard-coded) to facilitate expansion.
- Dates, times, distance units, and locale formatting shall automatically adapt to user locale settings.

C1.6.4 Disaster Recovery & Backup Requirements

- The system shall automatically back up all critical data (comments, road reports, user profiles, lists, moderation logs) using Firestore's multi-region replication.
- Backups shall support point-in-time restoration in case of data corruption or accidental deletion.
- AI models, configuration files, and map metadata shall be versioned and stored in secure cloud storage.
- Disaster recovery procedures shall allow full restoration of services within acceptable RTO and RPO thresholds.
- System components shall fail gracefully, with fallbacks for temporary API or service outages.

C1.6.5 Licensing Requirements

- The system shall comply with all Google Maps API licensing requirements, including usage quotas and mandatory attributions.
- AI services (OpenAI) shall be used under appropriate licensing agreements and comply with service terms and safety policies.
- All SDKs, libraries, fonts, and third-party assets used in the mobile app shall be licensed appropriately (MIT, Apache, commercial, etc.).
- Any media assets, such as icons or illustrations shall either be created in-house or used with explicit licensing permissions.
- A dependency license inventory shall be maintained for audit and compliance purposes.

C1.6.6 Installation Requirements

- The mobile application shall be packaged and deployed for iOS and Android through the App Store and Google Play Store.
- Beta versions shall be distributed through TestFlight (iOS) and Internal Testing (Android).
- The installation package shall include correct signing keys, version numbers, and manifest configurations as required by platform policies.
- Users shall be able to install the application with standard platform permissions (location, camera, notifications, storage).
- The system shall support installation and operation across a wide range of devices, screen sizes, and OS versions.
- Installation shall require minimal user steps and shall not require external configuration.

C1.7. Appendix

C1.7.1 Supporting Information

This appendix provides foundational reference material that supports the understanding, maintenance, and further development of the WeWay Intelligent Navigation and Social Road Awareness System. It includes key terminology used throughout the system, a data dictionary generated strictly from the system's ER model, and the project's finalized conceptual ER diagram in Crow's Foot notation. This section ensures that all stakeholders, developers, testers, analysts, and administrators have a standardized representation of core system concepts and data structures.

C1.7.2 Glossary

This section provides definitions and explanations of key terms and acronyms used throughout the Software Requirements Specification (SRS) document.

Term	Definition
Abuse Report	A user-submitted report identifying inappropriate, harmful, or misleading content within a comment.
AI (Artificial Intelligence)	Systems used for summarization, natural-language discovery, and moderation of user-generated content.
AI-Assisted Moderation	Automated identification and flagging of potentially harmful comments for admin review.
AI Query	A request sent to the AI engine for tasks such as generating summaries, answering "What's Around Me?", or moderating content.
AI Summary	A generated text summarizing community events, route conditions, or comment patterns along a road or navigation path.
AI Summary Source	The set of original comments that contributed to an AI-generated summary, with traceable references.
API (Application Programming Interface)	A software interface allowing WeWay to communicate with external services like Google Maps, Places, Directions, and AI models.

Authentication	The process of verifying user identity via Email/Password, Google Sign-In, or Apple Sign-In using Firebase Authentication.
Avatar	A user's profile picture is displayed publicly.
Comment	A user-generated text or media message associated with a place or road segment.
Comment Thread	A chronological series of comments posted on a place or road segment.
Crow's Foot Notation	A standard entity-relationship diagram notation used to represent cardinality in the ER model.
Dashboard (Admin)	The interface where admins review flagged comments, reports, and moderation actions.
ETA (Estimated Time of Arrival)	The predicted time it will take to reach a destination during navigation.
Event Report	User-generated reports about accidents, hazards, traffic, or closures on a road.
FCM (Firebase Cloud Messaging)	The service used for delivering push notifications to WeWay users.
Firestore	The cloud-based NoSQL database is used to store users, comments, road threads, lists, summaries, and system activity.
Geo-Anchor	The precise latitude and longitude are attached to a comment or event posted on the map.
GPS (Global Positioning System)	Satellite-based location tracking is used for navigation and detecting safe-driving mode.
Guest User	A user not logged in who can browse maps, search places, and start navigation, but cannot post or save content.
Hazard	An obstacle or dangerous road condition reported by users (e.g., accident, debris, construction).
Interactive Map	The main interface shows traffic layers, POIs, road segments, and real-time navigation.
Landmark-Based Navigation	Navigation instructions are enhanced with recognizable landmarks within 200 meters of a turn.

Moderation Action	An admin decision to approve, reject, or remove flagged content after review.
Moderation Queue Item	Any comment flagged by AI or users awaiting admin review.
Navigation Session	A complete instance of a user starting, following, and completing a navigation route.
Natural-Language Discovery	An AI-powered feature allowing users to ask questions such as "What's around me?" in plain language.
NFR (Non-Functional Requirement)	A requirement describing performance, reliability, safety, security, usability, or system quality attributes.
Notification	A push message is delivered to the user for replies, mentions, warnings, route changes, or updates.
Place	A point of interest (restaurant, shop, location) identified via Google Places API or user interaction.
Place Info Card	A UI panel displaying details about a selected place, including ratings, hours, photos, and actions.
POI (Point of Interest)	A significant location such as a shop, restaurant, gas station, or landmark.
Reaction	User interaction with a comment, such as liking or replying.
Registered User	A logged-in user who can comment, save places, report events, and access all features.
Reporting (Road Event)	A user action to notify others about accidents, traffic, hazards, or road closures.
Road Segment	A discrete piece of a road on the map is used to anchor comments and threads.
Route Summary	An AI-generated overview of comments and conditions along the route selected for navigation.
Safe-Driving Mode	A simplified UI state automatically triggers when driving movement is detected to reduce distraction.
Saved List	A user-created list grouping saved places such as Favourites, Work, or Trip Plans.
Saved Place	A POI marked by a user and stored inside a saved list.
Search Filters	Criteria used during search, such as open-now, rating, distance, and categories.

SRS (Software Requirements Specification)	This official document details functional, non-functional, and technical requirements.
System Actor	Any external party interacting with WeWay, including users, admins, and third-party APIs.
TLS (Transport Layer Security)	Protocol enforcing encryption for secure communication between app, backend, and APIs.
UC (Use Case)	A narrative describing how a user interacts with a system feature to achieve a specific goal.
User	Any individual using the application, whether authenticated or not.
User Profile	Personal information displayed for a registered user, including name, avatar, and preferences.
Violation Log	The stored history of moderation actions and flagged content related to a user.
Waypoint	An intermediate navigation stop is added manually or through POI selection.
WeWay	The mobile application described in the SRS provides navigation, social updates, AI insights, and community reporting.

C1.7.3 Data Dictionary

Attribute/Entity	Type	Key	Description
Notification			
NotificationID	int	**PK**	Notification ID.
UserID	int	FK	Recipient user.
NotificationType	varchar		Type (reply, mention, hazard, etc.).
Title	varchar		Notification title.
Body	text		Notification description.
RelatedEntityType	varchar		Comment / Place / Route, etc.
RelatedEntityID	int		Entity reference.
IsRead	boolean		Read/unread flag.
SavedPlace			
SavedPlaceID	int	**PK**	Entry ID.
SavedListID	int	FK	Parent list.
PlaceID	int	FK	Saved place.
AddedAt	datetime		Timestamp added.
SavedList			
SavedListID	int	**PK**	List ID.
UserID	int	FK	Owner user.
ListName	varchar		Name of the list.
CreatedAt	datetime		Timestamp.
RouteSummary			
RouteSummaryID	int	**PK**	Summary ID.
NavigationSessionID	int	FK	Related navigation session.
AISummaryID	int	FK	AI summary content.
CreatedAt	datetime		Timestamp.
NavigationSession			
NavigationSessionID	int	**PK**	Unique navigation session.
UserID	int	FK	Linked user (nullable for guest).
OriginLat	decimal(9,6)		Origin latitude.
OriginLng	decimal(9,6)		Origin longitude.
DestLat	decimal(9,6)		Destination latitude.

DestLng	decimal(9,6)		Destination longitude.
StartedAt	datetime		Session start.
EndedAt	datetime		Session end.
SafeModeUsed	boolean		Indicates safe-driving mode usage.
ModerationAction			
ModerationActionID	int	**PK**	Action ID.
ModerationItemID	int	FK	Associated flagged item.
AdminID	int	FK	Admin performing the action.
ActionType	varchar		approve / remove.
ActionNote	text		Reason or notes.
ActionTime	datetime		Timestamp.
ModerationQueueItem			
ModerationItemID	int	**PK**	Queue record ID.
CommentID	int	FK	Comment flagged.
FlagSource	varchar		AI or User.
FlagReason	varchar		Reason text.
CreatedAt	datetime		Timestamp.
Status	varchar		pending / resolved.
AISummarySource			
AISummarySourceID	int	**PK**	Link ID.
AISummaryID	int	FK	Related summary.
CommentID	int	FK	Source comment.
Weight	decimal(4,2)		Contribution weight.
AISummary			
AISummaryID	int	**PK**	AI-generated summary ID.
AIQueryLogID	int	FK	Originating query.
SummaryText	text		Generated output text.
SummaryType	varchar		area / route / thread.
LanguageCode	varchar		Summary language.
CreatedAt	datetime		Timestamp.
AIQueueLog			
AIQueryLogID	int	**PK**	AI query entry.

UserID	int	FK	User who initiated (nullable for guest).
QueryText	text		Raw natural language query.
QueryType	varchar		discovery, summary, moderation.
CreatedAt	datetime		Timestamp.
AbuseReport			
AbuseReportID	int	**PK**	Report record ID.
CommentID	int	FK	Comment being reported.
UserID	int	FK	User who reported.
Reason	varchar		Report description.
CreatedAt	datetime		Timestamp.
Status	varchar		open, reviewed, dismissed.
Reaction			
ReactionID	int	**PK**	Reaction record ID.
CommentID	int	FK	Comment being reacted to.
UserID	int	FK	User performing reaction.
ReactionType	varchar		like / etc.
CreatedAt	datetime		Timestamp.
Comment			
CommentID	int	**PK**	Unique comment identifier.
UserID	int	FK	User who posted the comment.
PlaceID	int	FK	Place associated (nullable).
RoadSegmentID	int	FK	Road segment associated (nullable).
ParentCommentID	int	FK	For replies; nullable.
Content	text		Textual content.
MediaUrl	varchar		Optional media attachment.
CommentType	varchar		place / road / event.
Latitude	decimal(9,6)		Geo-anchor.
Longitude	decimal(9,6)		Geo-anchor.
CreatedAt	datetime		Posted timestamp.
IsDeleted	boolean		Logical deletion flag.
RoadSegment			
RoadSegmentID	int	**PK**	Unique road segment ID.

ExternalRoadRef	varchar		External reference (e.g., Maps API).
Name	varchar		Road or highway name.
RoadType	varchar		Type of road (highway, street).
GeoHash	varchar		Spatial hash for indexing.
Place			
PlaceID	int	**PK**	Unique place record ID.
ExternalPlaceRef	varchar		Google Place ID reference.
Name	varchar		Place name.
Address	varchar		Full address.
Latitude	decimal(9,6)		Geographical coordinate.
Longitude	decimal(9,6)		Geographical coordinate.
AvgRating	decimal(3,2)		Aggregated rating.
IsLandmarkEligible	boolean		Marks if usable for landmark navigation.
CreatedAt	datetime		Record creation timestamp.
Admin			
AdminID	int	**PK**	Unique administrator record ID.
UserID	int	FK	Associated user account.
AdminLevel	varchar		Role hierarchy level.
CreatedAt	datetime		When admin privileges were granted.
UserAuthMethod			
UserAuthMethodID	int	**PK**	Unique ID for authentication method entry.
UserID	int	FK	Reference to associated User.
ProviderType	varchar		Type of login provider: Email, Google, Apple.
ProviderUserId	varchar		Provider-specific identifier.
LinkedAt	datetime		Timestamp when linked to the account.
User			
UserID	int	**PK**	Unique identifier for each user.
Email	varchar		User's login email.

DisplayName	varchar		User’s visible display name.
Role	varchar		User role (Registered, Admin).
LanguageCode	varchar		Preferred language (e.g., en, ar).
AvatarUrl	varchar		Profile picture URL.
CreatedAt	datetime		Timestamp of account creation.
Status	varchar		Account status (active, suspended).

C1.7.4 Additional Diagrams

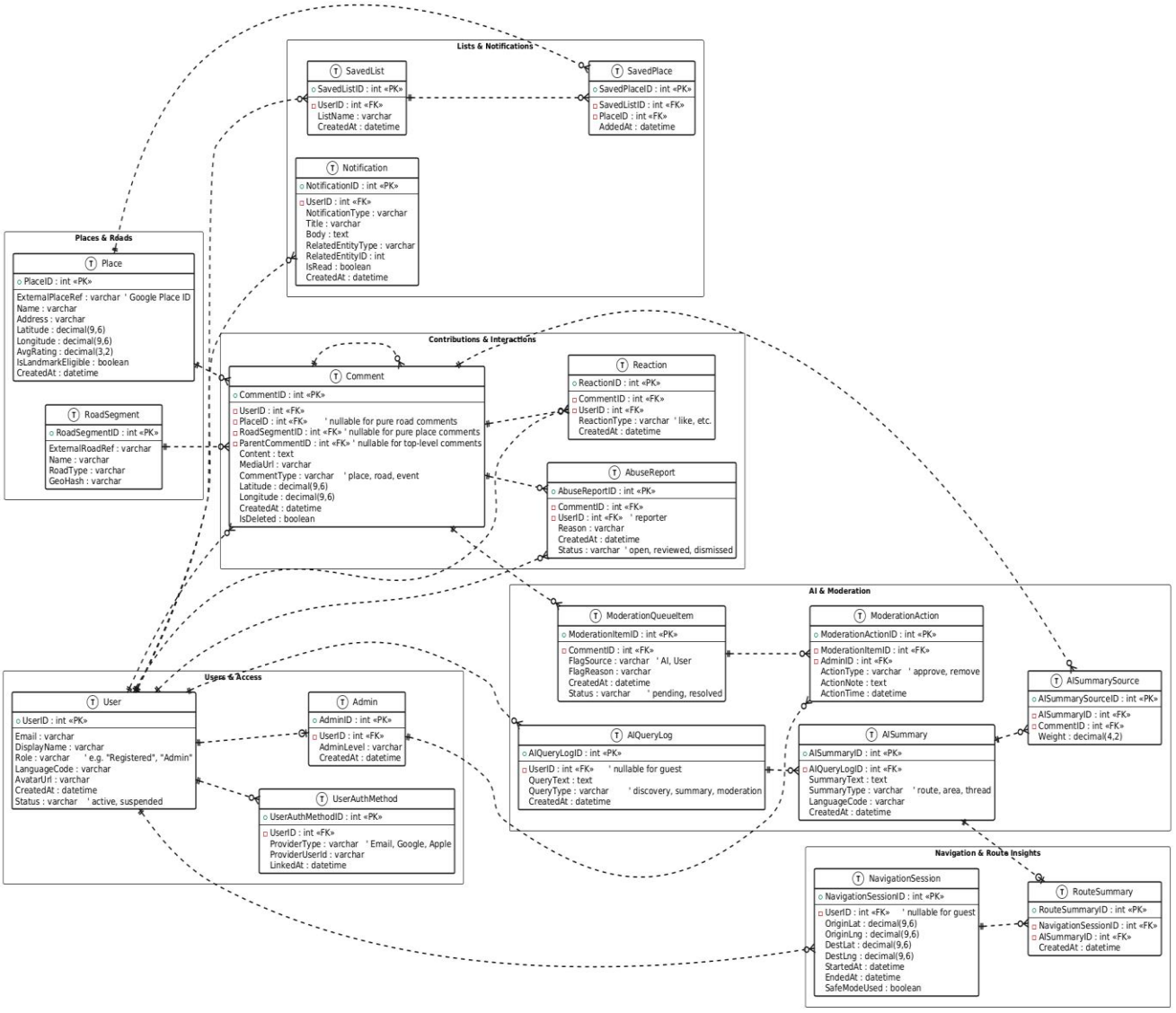

Figure 19: Er-Diagram Crowfeet Notation

C1.7.5 Sample Inputs/Outputs

Input	Description	Output	Result
User submits a comment.	Text + geolocation	Comment object	Comment displayed on the map and thread
User reports abuse	Comment ID + reason	Queue item	Added to moderation queue
AI summary request	Query text	Summary text	Displayed to the user and linked to sources
Save place	PlaceID + ListID	SavedPlace entry	Updated saved list
Start navigation	Origin + destination	NavigationSession	Turn-by-turn navigation UI

C1.7.6 Traceability Aids

- All Functional Requirements (**FR-1.1 → FR-7.2**) map to Use Cases (**UC-01 → UC-29**).
- All Use Cases map to at least one entity in the ERD.
- Data Dictionary items correspond directly to storage and processing needs defined in functional requirements.
- Moderation, AI, and navigation workflows are fully trace to both UC diagrams and ERD relationships.

C1.7.6.1 Traceability Matrix

FR ID	Feature Name	Actor	Mapped Use Case(s)	Mapped Entity/Entities (From ERD)	Summary
FR-1.1	Guest Mode	Guest User	UC-01	User, Place, RoadSegment, NavigationSession	Guests can browse the map, search, view POI details, and navigate.
FR-1.2	Sign-in Gate	Guest User	UC-02	User	The guest is prompted to sign in before restricted actions.
FR-1.3	Sign-in & Registration	Guest User	UC-03, UC-04	User, UserAuthMethod	User logs in/registers with Email/Google/Apple.
FR-1.4	Session Management	Registered User	UC-05, UC-06	User, UserAuthMethod	User session persists; logout ends the session.
FR-1.5	Profile Management	Registered User	UC-07	User	User updates profile details and preferences.
FR-1.6	Account Recovery	Registered User	UC-08	User, UserAuthMethod	User resets password or relinks login provider.
FR-2.1	Map Display	Guest + Registered User	UC-09	User, Place, RoadSegment, NavigationSession	User views the interactive map and POIs.
FR-2.2	Search	Guest + Registered User	UC-10	User, Place	User searches for places and applies filters.

FR-2.3	Navigation	Guest + Registered User	UC-11	User, NavigationSession, RouteSummary	User starts navigation, receives ETA & rerouting.
FR-2.4	Save a Place	Registered User	UC-12	User, SavedList, SavedPlace, Place	User saves places to lists.
FR-3.1	Place & Road Comments	Registered User	UC-13	Comment, User, Place, RoadSegment	User posts comments with or without photos.
FR-3.2	Road Threads	Registered User	UC-14	RoadSegment, Comment, User	User opens road thread on road long-press.
FR-3.3	Geo-Anchored Posting	Registered User	UC-15	Comment, RoadSegment, Place, User	Comments store exact geo-coordinates.
FR-3.4	Jump to Post Location	Guest + Registered User	UC-16	Comment, Place, RoadSegment	Selecting a comment zooms the map to its origin location.
FR-3.5	Comment Interactions	Registered User	UC-17	Comment, Reaction, AbuseReport, User	User likes, replies, reports or shares comments.
FR-3.6	AI Summary Source Linking	Guest + Registered User	UC-18	AISummary, AISummarySource, Comment	Users open exact posts contributing to the AI summary.
FR-4.1	Natural-Language Discovery	Guest + Registered User	UC-19	AIQueryLog, AISummary, User, Place	User asks “What’s around me?” and gets AI results.
FR-4.2	Place Info Card	Guest + Registered User	UC-20	Place, Comment, Reaction, SavedPlace	User views detailed place information.
FR-4.3	AI Route Summary	Registered User	UC-21	AISummary, AISummarySource,	User gets an AI summary along the selected route.

				Comment, NavigationSession	
FR-4.4	AI Processing Engine	System / AI Platform	UC-22	AIQueryLog, AISummary, AISummarySource	AI processes input and generates structured insight.
FR-4.5	AI-Assisted Moderation	Admin	UC-23	ModerationQueueItem, ModerationAction, Comment, Admin	Admin reviews AI-flagged comments.
FR-5.1	Safe Driving Mode	Registered User	UC-24	NavigationSession, User	The system enters simplified mode when movement is detected.
FR-5.2	Passenger Override	Registered User	UC-25	NavigationSession, User	User overrides Safe Driving Mode.
FR-6.1	Landmark Toggle	Registered User	UC-26	User, Place	User enables/disables landmark-based navigation.
FR-6.2	Landmark 200m Rule	Registered User	UC-27	Place, NavigationSession	The system uses landmarks within 200m in instructions.
FR-7.1	Save & Lists	Registered User	UC-28	SavedList, SavedPlace, User, Place	User manages lists & organises saved places.
FR-7.2	Notifications	Registered User	UC-29	Notification, User	User receives push notifications for events & replies.

C2) Software Project Management Plan (SPMP)

The WeWay project will be delivered using an iterative, milestone-based approach over the semester. The plan is organized week-by-week and aligns with standard SDLC phases requirements, design, implementation, prototyping, testing, and deployment.

C2.1. Software Engineering Process Used

The WeWay project was developed using a hybrid Agile–Prototyping Model, chosen for its flexibility and rapid iteration cycle. This model combines:

Agile Components

- Short development iterations (1–2 weeks)
- Continuous feedback from supervisor and team
- Incremental delivery of working features
- Regular testing and refinement after each sprint

Prototyping Components

- Early low-fidelity UI prototypes
- Iterative refinements of map UI, comments flow, and AI interaction screens
- Multiple functional prototypes (Prototype 1, 2, 3) across the semester

Why This Process Was Used

The WeWay system includes:

- Real-time map rendering
- Firebase integration
- AI-powered workflows
- GPS and sensor-based features

These components require experimentation, interface refinement, and continuous evaluation. Thus, a hybrid Agile + Prototyping approach ensured the team could adapt, refine, and improve the design throughout development.

C2.2 Project Overview

Purpose

WeWay is a mobile application designed to enhance real-world navigation using community-driven insights, location-based comments, AI route summaries, and safe-driving interaction modes.

Objectives

- Provide real-time place and road information using crowdsourced comments
- Enable fast, landmark-based navigation supported by AI summaries
- Improve road safety using Safe-Driving Mode and passenger override
- Empower admins to moderate harmful content via AI + manual review
- Allow users to save places and build personal interest lists
- Provide multilingual AI summaries & localized navigation guidance

Project Scope

The project includes:

- A full Flutter mobile application (Android/iOS)
- Firebase backend (Auth, Firestore, Storage, FCM)
- Google Maps, Places, and Directions APIs
- OpenAI API for summarization and content analysis
- Admin moderation console
- Core modules: navigation, comments, road threads, saved lists, AI queries, safe driving

The system does **not** include:

- Payment gateways
- Desktop-based navigation
- High-precision offline maps

C2.3 Project Schedule

Milestone	Description	Week
Project Initiation	Problem identification, concept design	Week 1
Requirements Phase	Detailed SRS, UML, user stories	Week 2–3
Architecture & Planning	ERD, class diagrams, high-level architecture	Week 4
Prototype 1	Map UI, navigation basics	Week 6
Prototype Testing	UI/Navigation test	Week 7
Prototype 2	Comments, lists, road threads	Week 8–9
Prototype 3	AI summaries, AI moderation, safe-driving	Week 10
System Testing	Integration, performance, security	Week 11
Supervisor Demo	First demo to supervisor	Week 12
Final Improvements	Optimization, bug fixes	Week 13
Final System Delivery	Full working mobile app	Week 14
Documentation	Report + PPT + submission	Week 15
Project Initiation	Problem identification, concept design	Week 1
Requirements Phase	Detailed SRS, UML, user stories	Week 2–3
Architecture & Planning	ERD, class diagrams, high-level architecture	Week 4

C2.4 Sub-Team Responsibilities and Team Structure

The team work in this project consisted of:

The WeWay project will be completed by a two-member team, with both individuals contributing equally while maintaining distinct leadership roles and areas of expertise.

1. Fager Alyati Team Leader

Student ID: *202200750*

Expertise:

- Mobile application development using Flutter
- UI/UX design and prototyping
- Systems modelling (UML, ERD, architectural diagrams)
- Cloud technologies: Firebase Authentication, Firestore, Storage, FCM
- Requirements engineering & technical documentation

Responsibilities:

- Lead system architecture and conceptual design
- Develop core front-end modules: map interface, navigation flow, safe-driving mode
- Integrate Google Maps, Places, and Directions APIs
- Implement authentication flows, Firestore CRUD operations, and app-wide state management
- Coordinate weekly progress, task breakdown, and version control
- Ensure documentation quality, consistency, and compliance with academic standards

2. Rehab Sldossary Team Leader #2

Student ID: *201900816*

Expertise:

- Backend workflows & cloud database design (Firestore, Storage, FCM)
- AI integration: OpenAI API for summaries, moderation, NLP queries
- Testing, debugging, and performance optimization
- Mobile UI logic, forms, validation, and responsive layout design
- API integration and asynchronous data handling

Responsibilities:

- Implement backend-dependent modules: comments, road threads, reaction system, saved lists
- Develop AI-driven features (route summaries, contextual analysis, moderation engine)
- Manage testing cycles (unit testing, integration testing, performance testing)
- Optimize database rules, security constraints, and API request flows
- Support prototype development and help prepare project deliverables
- Contribute to report writing, diagrams, and refinement of high-level architecture

C2.5 Budget and Resources

Software Resources

- Flutter & Dart SDK – free
- Firebase Authentication & Firestore – free tier
- Firebase Storage – limited free tier
- Google Maps Platform – free credit + usage quotas
- OpenAI API – minimal usage for summaries/moderation
- GitHub/GitLab – version control
- Project Management Tool – Trello/Notion free tier

Hardware Resources

- Android smartphone
- iPhone (for iOS testing)
- Laptop for development
- GPS-enabled device sensors

Human Resources

- 2 student developers (both team leaders with shared roles)

Estimated Budget

Most resources fall under free or educational tier.
Only AI usage and Maps API may incur small usage-based costs.

C2.7 Risk Management

Identified Risks & Mitigation Strategies

Risk	Impact	Mitigation Strategy
Unstable GPS or inaccurate readings	Medium	Use fused location provider & smoothing filters
AI moderation false positives	Medium	Add manual admin review fallback
Map API quota limits	High	Cache routes & summaries, avoid repeated API calls
Performance issues on low-end devices	High	Optimize map rendering, lazy load comments
App crashes during startup	Medium	Implement error boundaries and crash-safe state restoration
Firestore security vulnerabilities	High	Apply Firebase Security Rules, authentication checks
Schedule delays	Medium	Weekly sprint reviews, flexible backlog
Device compatibility issues	Medium	Test on multiple physical and virtual devices
Unexpected API errors or downtime	Medium	Retry logic, graceful fallback UI

C3) Software Design Description (SDD)

C3.1 Introduction

C3.1.1 Design Scope

The Software Design Description (SDD) describes the **complete technical design** of the WeWay mobile application, including architecture, behavior, interactions, data structures, UI layouts, and all software components required to implement the system. The SDD translates the SRS into a workable blueprint that guides development, testing, and integration.

C3.1.2 Design Objectives

- Provide a clear, scalable architecture suitable for real-time map rendering and cloud-based interactions.
- Ensure modularity through well-defined classes, services, and data entities.
- Enable secure, efficient, and maintainable integration with Firebase, Google Maps, and OpenAI.
- Support a clean separation between presentation logic (Flutter UI), application logic, and cloud services.
- Provide models to support future extensions (e.g., richer AI, heatmaps, content ranking).

C3.1.3 References

- WeWay SRS Document
- UML Diagrams (Use Case, Class, Activity, Sequence)
- ER Diagram and Database Model
- Flutter Documentation
- Firebase Architecture Guides
- Google Maps API & OpenAI API references
- IEEE 42010 Architecture Description Standard

C3.2 Design Viewpoints and Views

We adopt IEEE 42010 architecture viewpoints to structure the design.

C3.2.1 Context View (System Boundary and Actors)

Purpose

Shows how WeWay interacts with external actors and services.

System Boundary Includes:

- Flutter Mobile Client
- Firebase Backend (Auth, Firestore, FCM, Storage)
- AI Engine (OpenAI)
- Google Maps Platform

Primary Actors

- **User** – posts comments, navigates, saves places
- **Admin** – moderates flagged content
- **Device Sensors** – provides GPS and motion data
- **External APIs** – Google Maps, OpenAI

Referenced Diagram:

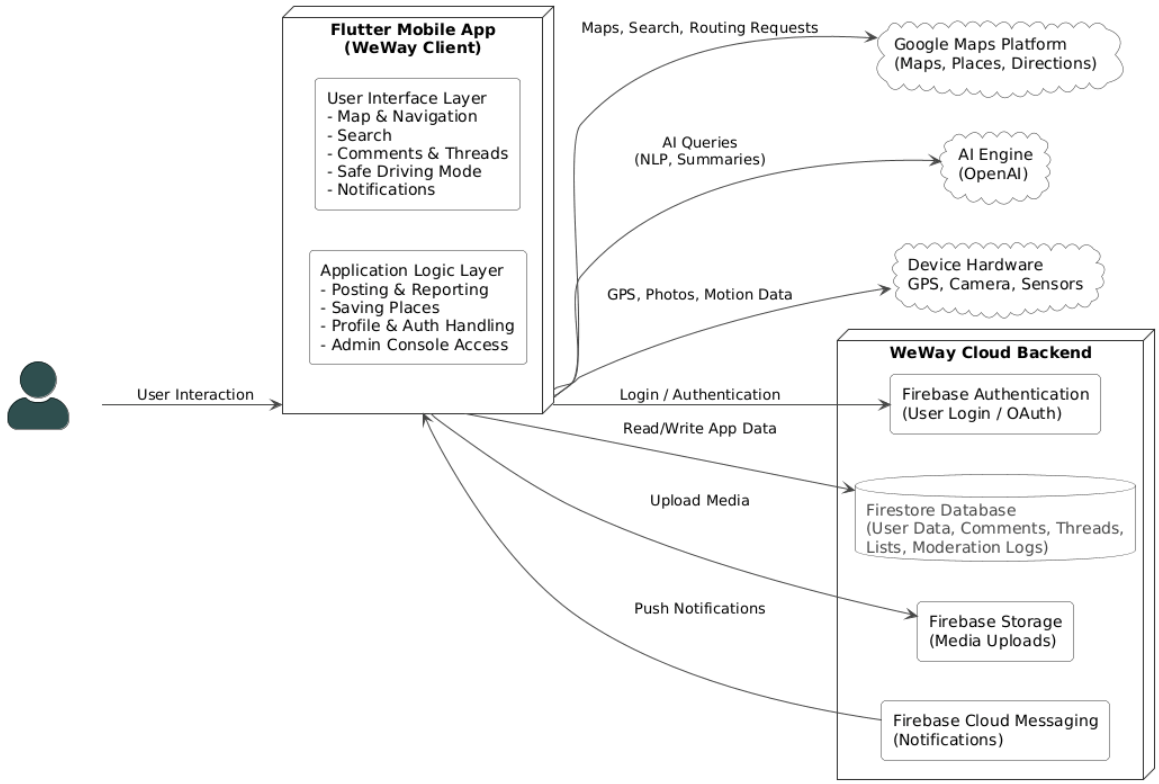

Figure 20: Block Diagram

C3.2.2 Logical View (Modules, Classes, Packages)

Purpose

Describes the structural organization of the software.

Main Logical Components

1. **Presentation Layer (Flutter UI)**
    - Map View
    - Place Details
    - Comments Thread
    - Road Thread View
    - Saved Lists UI
    - AI Summary View
    - Safe-Driving Mode View
2. **Application Logic Layer**

- Auth Controller
- Navigation Controller
- Comment Controller
- AI Integration Service
- SafeDrivingManager
- ListService
- ModerationService

3. **Data Access Layer**
   - Firestore Repository
   - Storage Repository
   - API Request Layer
4. **Backend Services (Firebase)**
   - Authentication
   - Firestore Collections
   - Storage Buckets
   - FCM Notifications
5. **External Services**
   - Google Maps/Places/Directions APIs
   - OpenAI Generative APIs

Referenced Diagram:

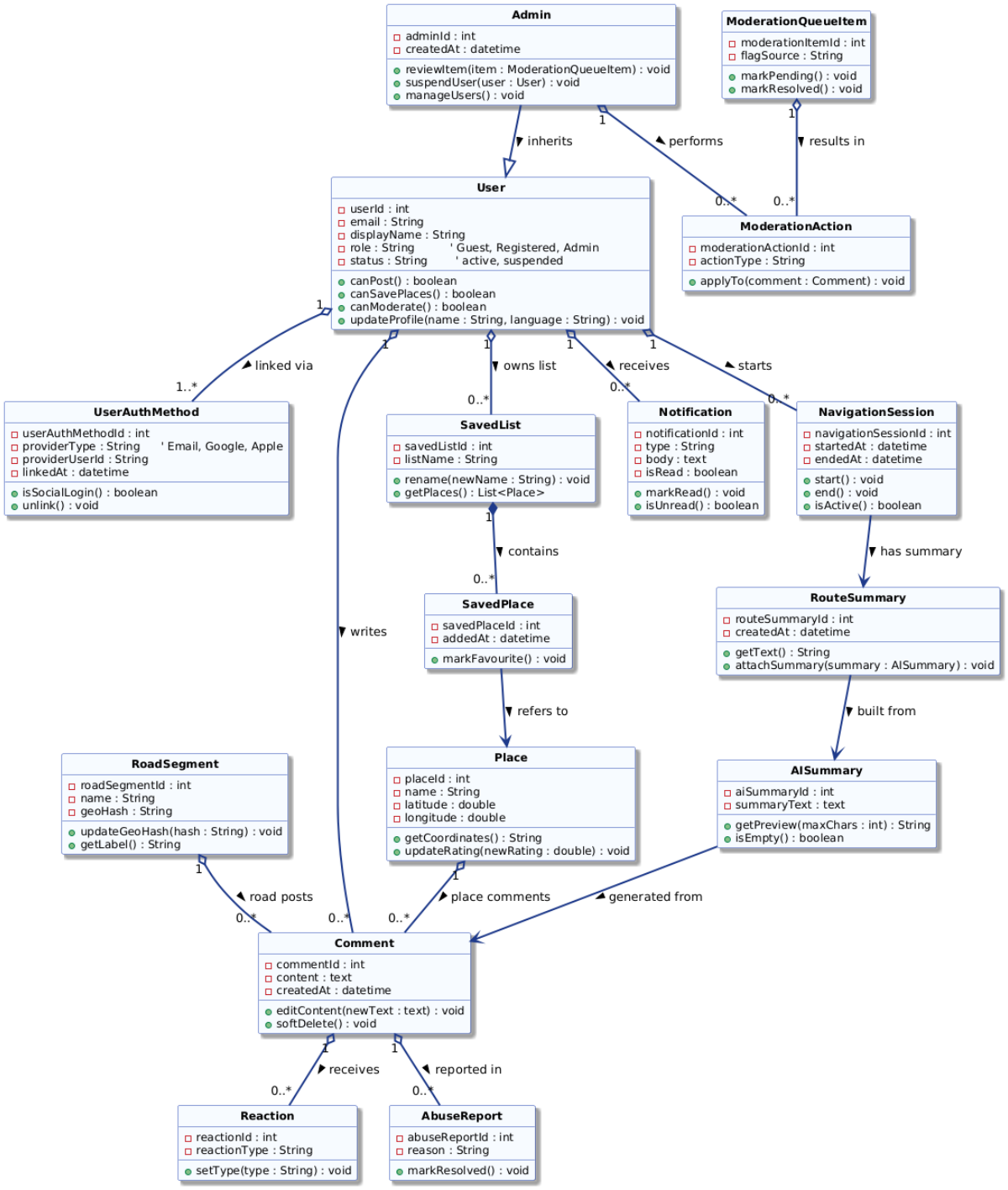

Figure 21: UML Class Diagram

C3.2.3 Interface View (APIs, Data Formats, Integration Points)

External API Interfaces

- **Google Maps SDK**
  - getRoute(lat1, lon1, lat2, lon2)
  - searchPlaces(query)
  - getNearbyPlaces(location)
- **OpenAI API**
  - generateRouteSummary(routeData)
  - moderateComment(text)
  - whatIsAroundMe(query)

Firebase Interfaces

- **Firestore Collections:**
  - /users
  - /comments
  - /roadThreads
  - /places
  - /savedLists
  - /moderationQueue
- **Authentication Providers:**
  - Email/Password
  - Google Sign-in
  - Apple Sign-in

Data Formats

- JSON for API communication
- Base64/Multipart for media upload
- GeoPoint data format for map coordinates
- AI summaries as plain text

C3.2.4 Interaction View (Sequence Diagrams)

This viewpoint describes the dynamic behavior between system components.

Referenced Diagrams

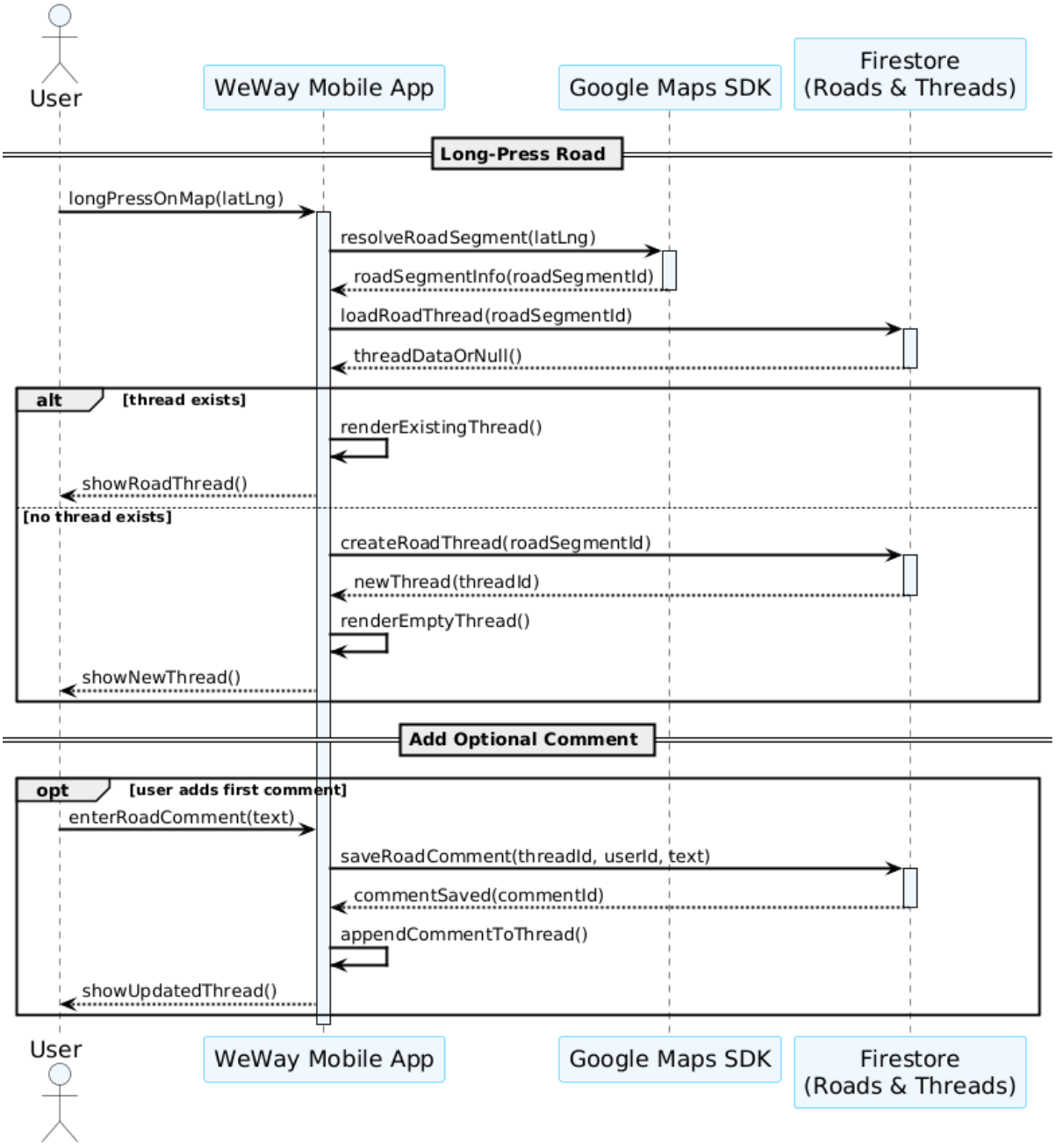

Figure 22: Road Thread View sequence

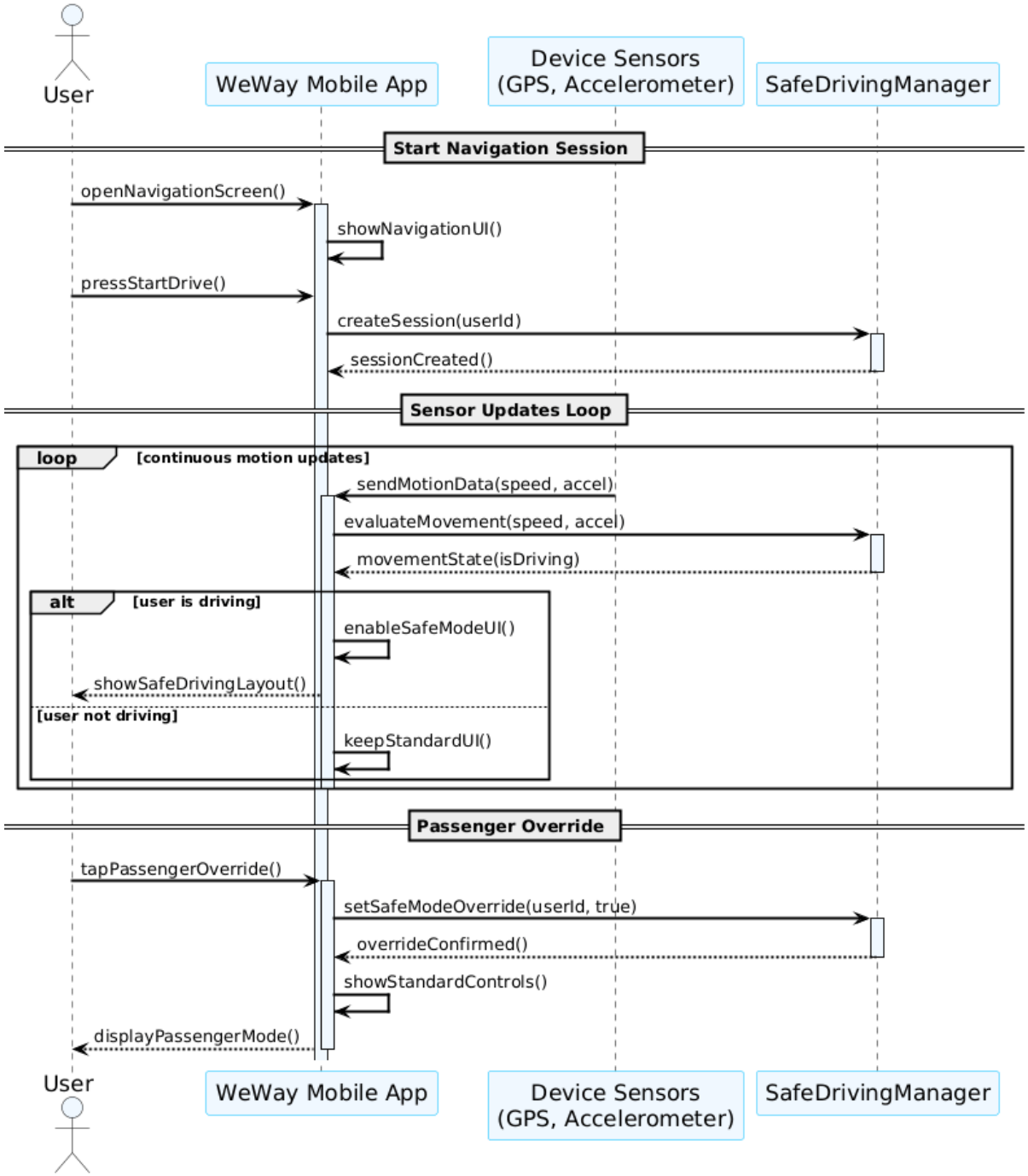

Figure 23: Activate safe driving mode sequence

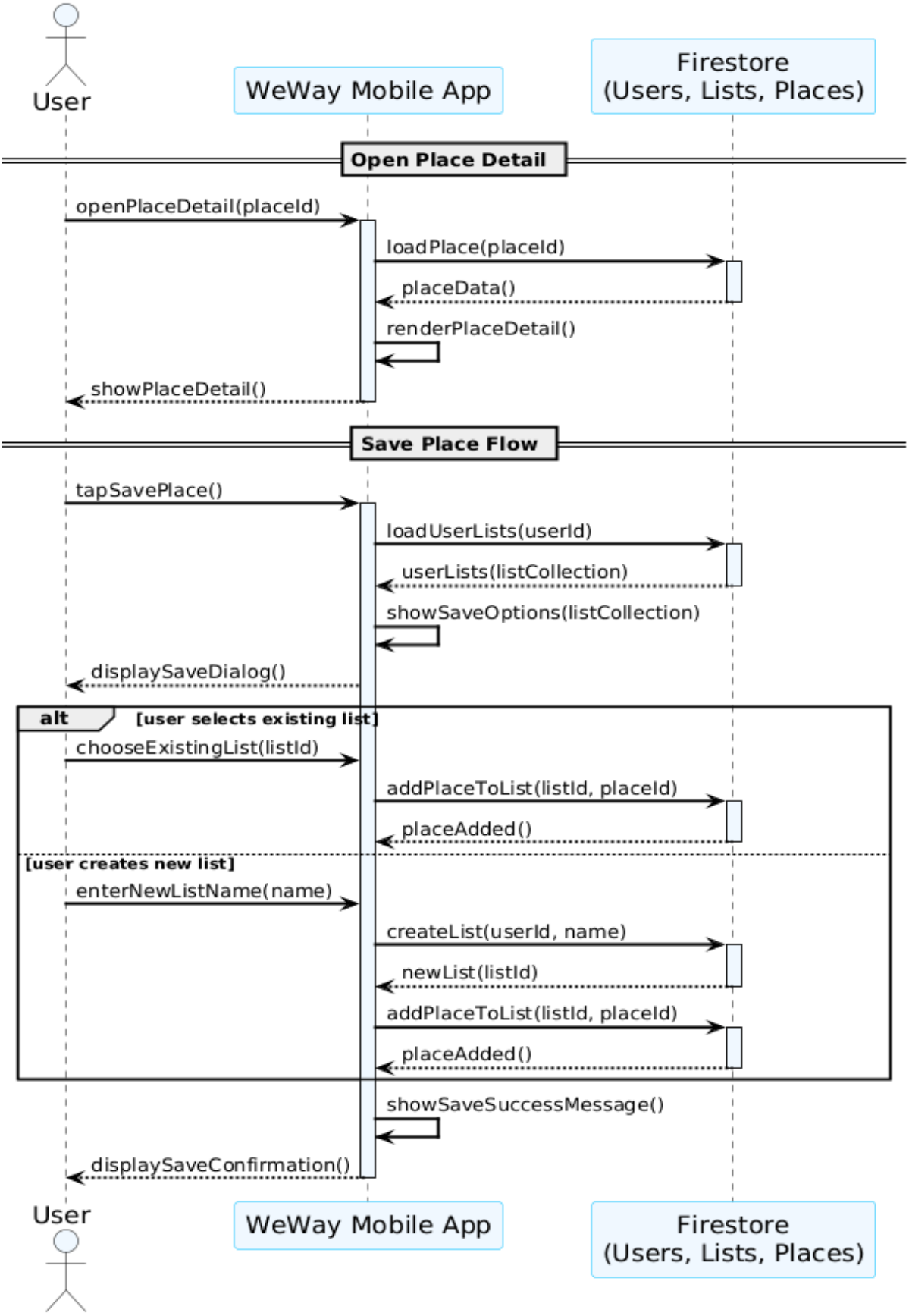

Figure 24: Save a place sequence

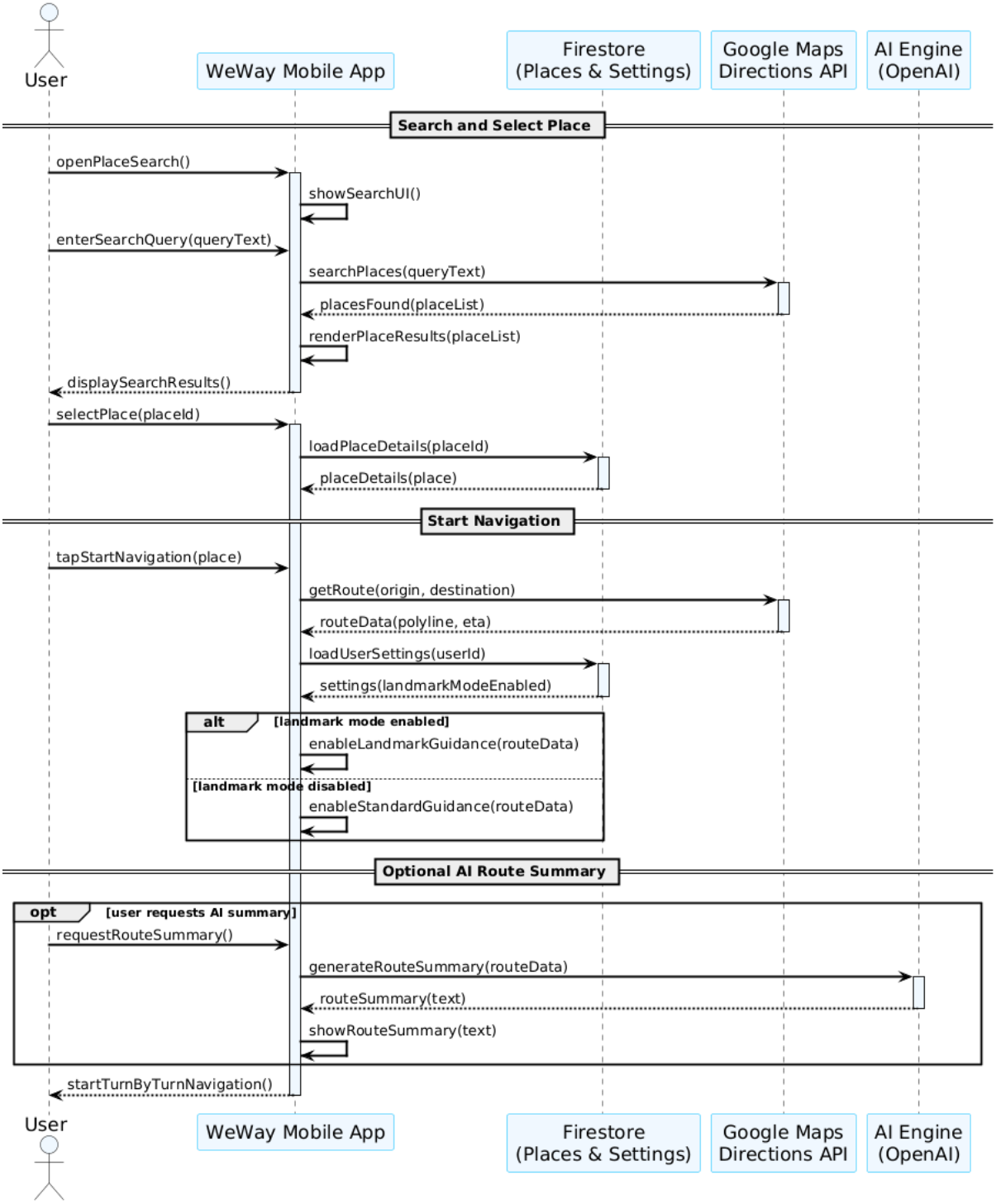

Figure 25: Start navigation sequence

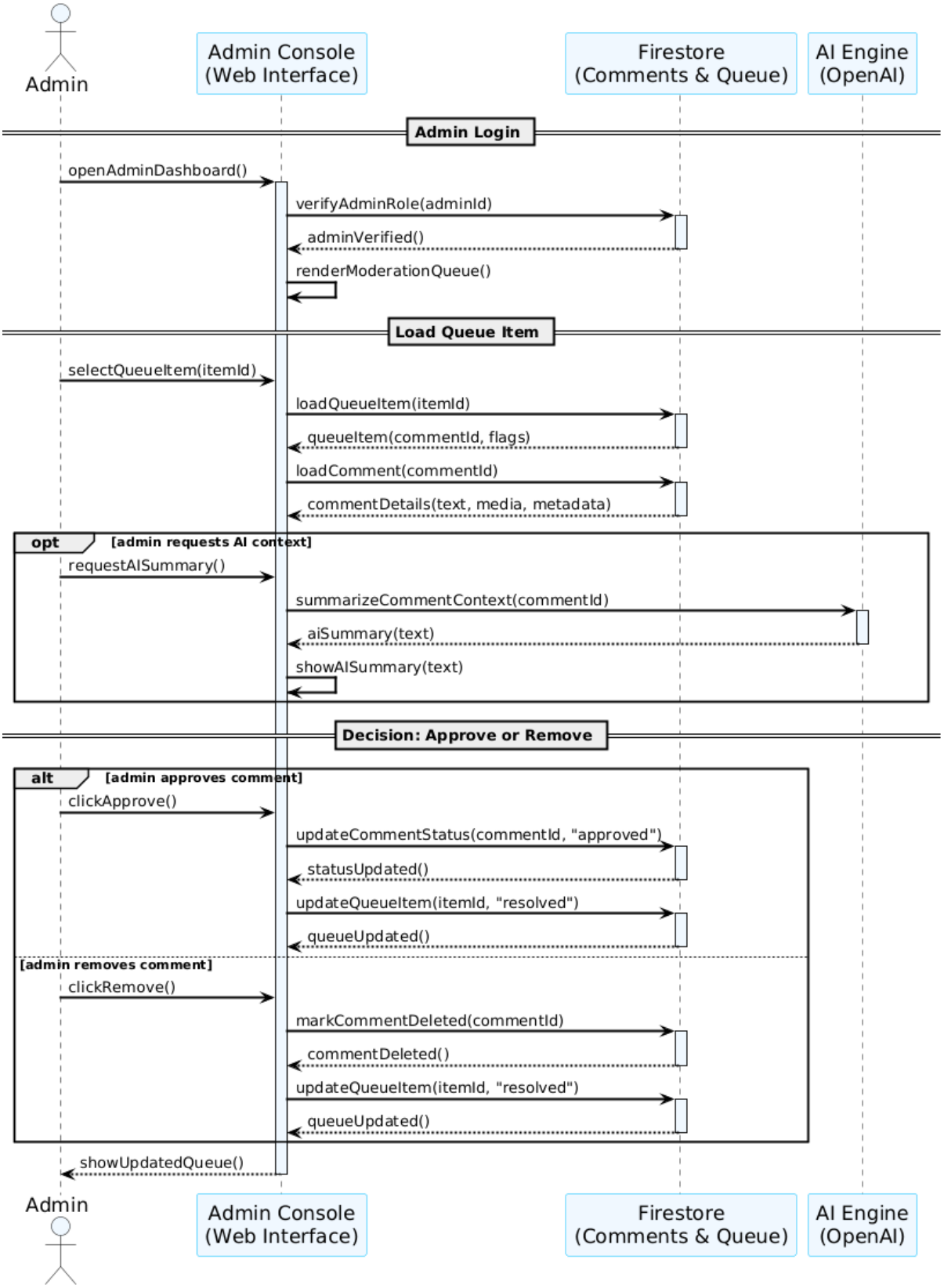

Figure 26: Admin Moderation review sequence

These diagrams describe message flows, activation bars, function calls, asynchronous operations, and return values across multiple lifelines.

C3.2.5 State View (State Machines)

This viewpoint describes the dynamic behavior between system states.

Referenced Diagrams

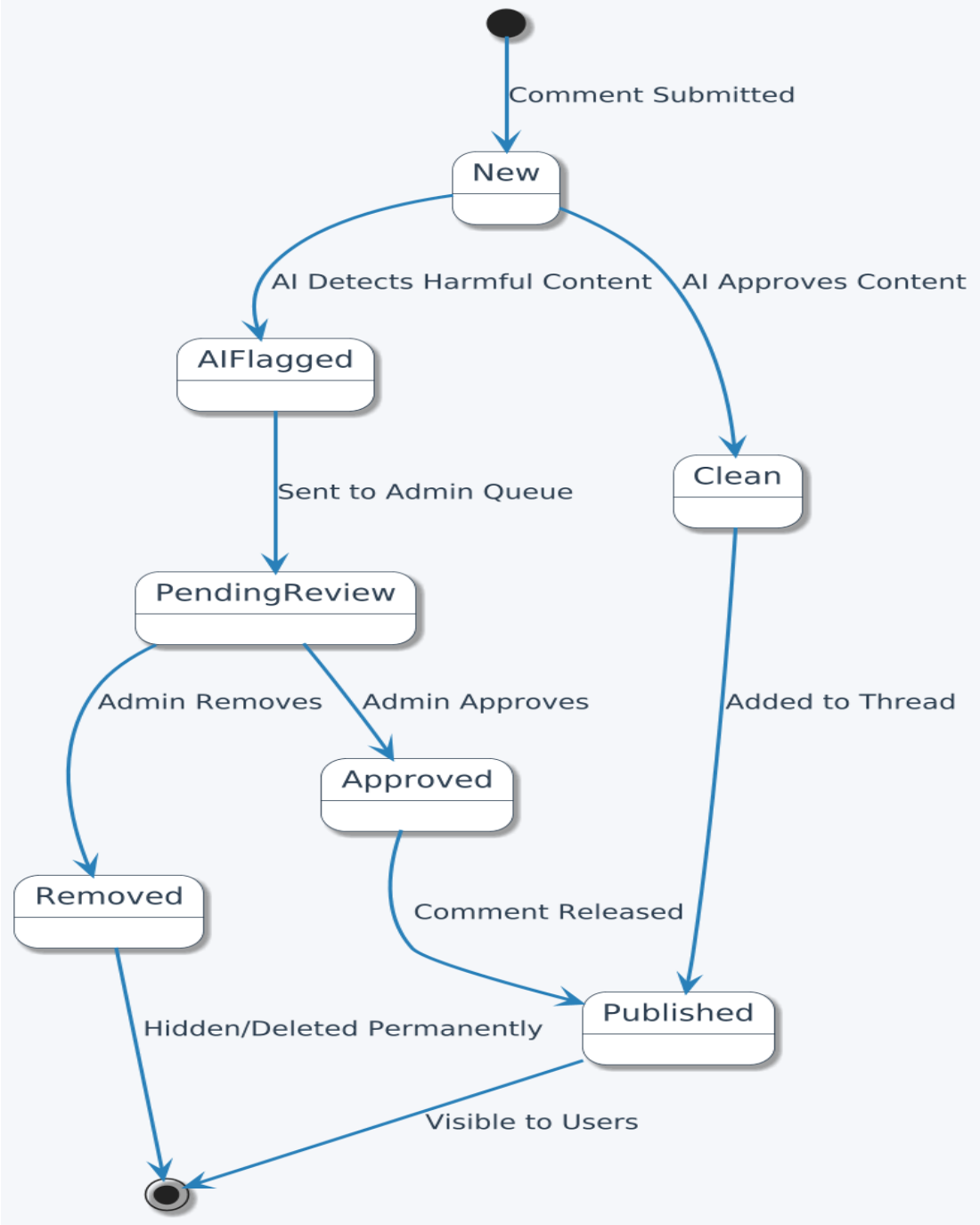

Figure 27: Comment moderation state machine diagram

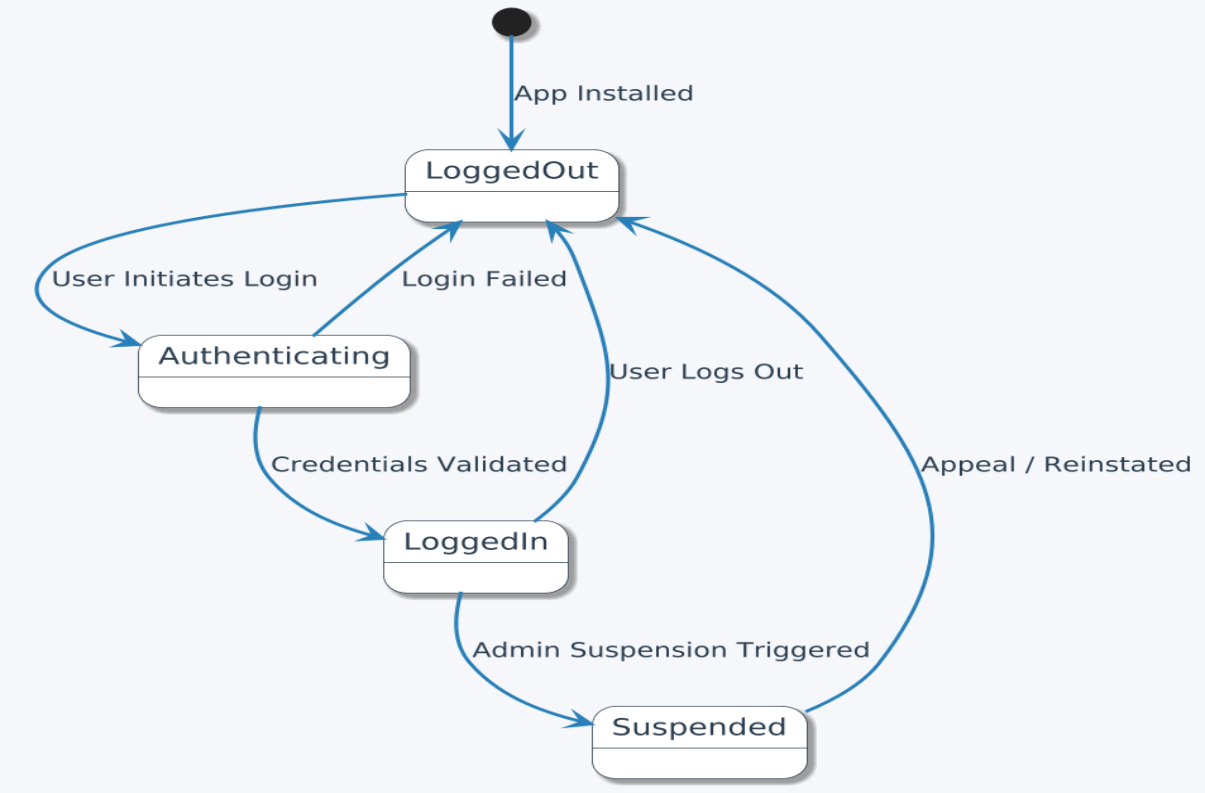

Figure 28: User session state machine diagram

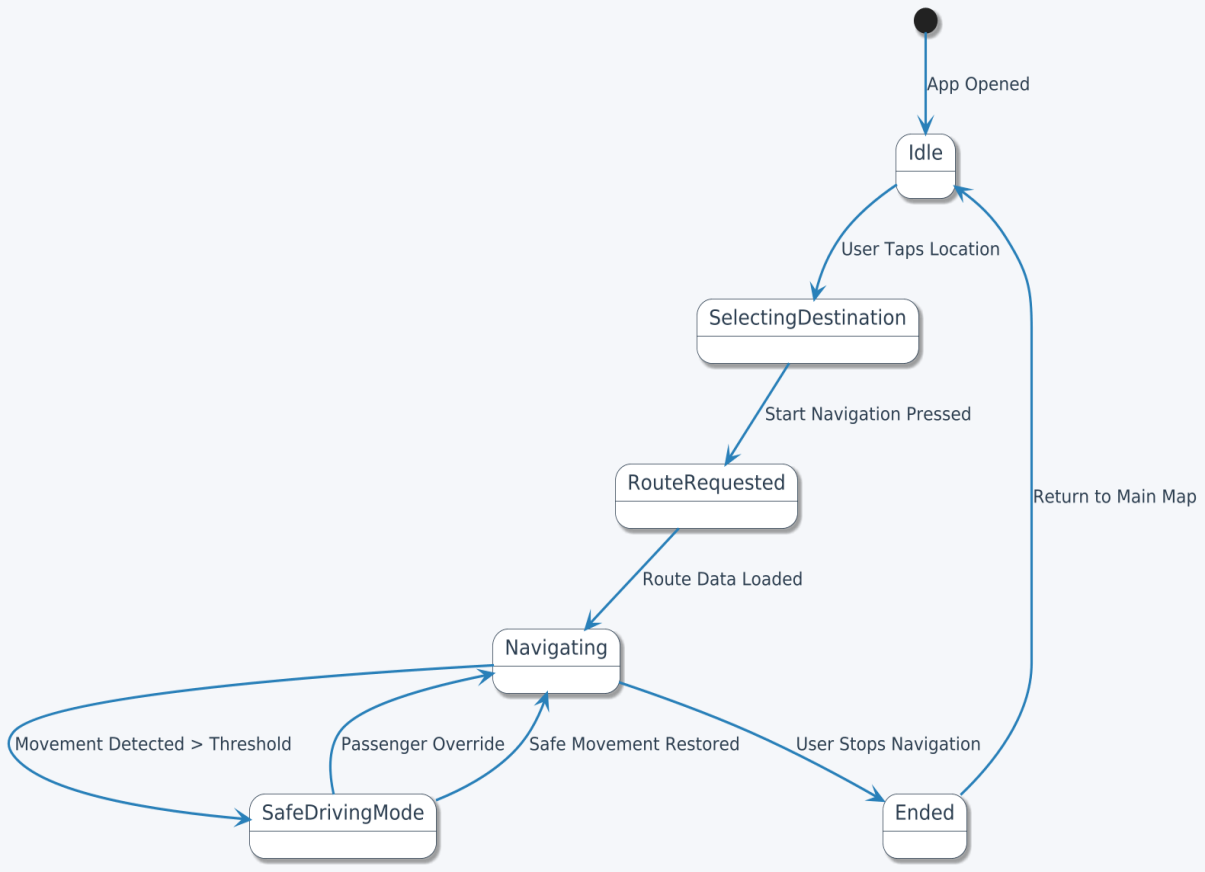

Figure 29: Navigation session state machine diagram

C3.2.6 Deployment View (Hardware/Software Mapping)

Client-Side (Mobile Device)

- Flutter Runtime
- Mobile Sensors (GPS, accelerometer)
- Secure Storage for tokens

Backend / Cloud

- **Firebase Authentication** (user login)
- **Firestore** (data storage)
- **Firebase Storage** (media uploads)
- **Firebase Cloud Messaging** (notifications)
- **Google Cloud Load Balancer** (implicitly used by APIs)

External Systems

- Google Maps Platform
- OpenAI Cloud APIs

Deployment reflects a serverless, scalable mobile-cloud architecture.

C3.2.7 Database Design

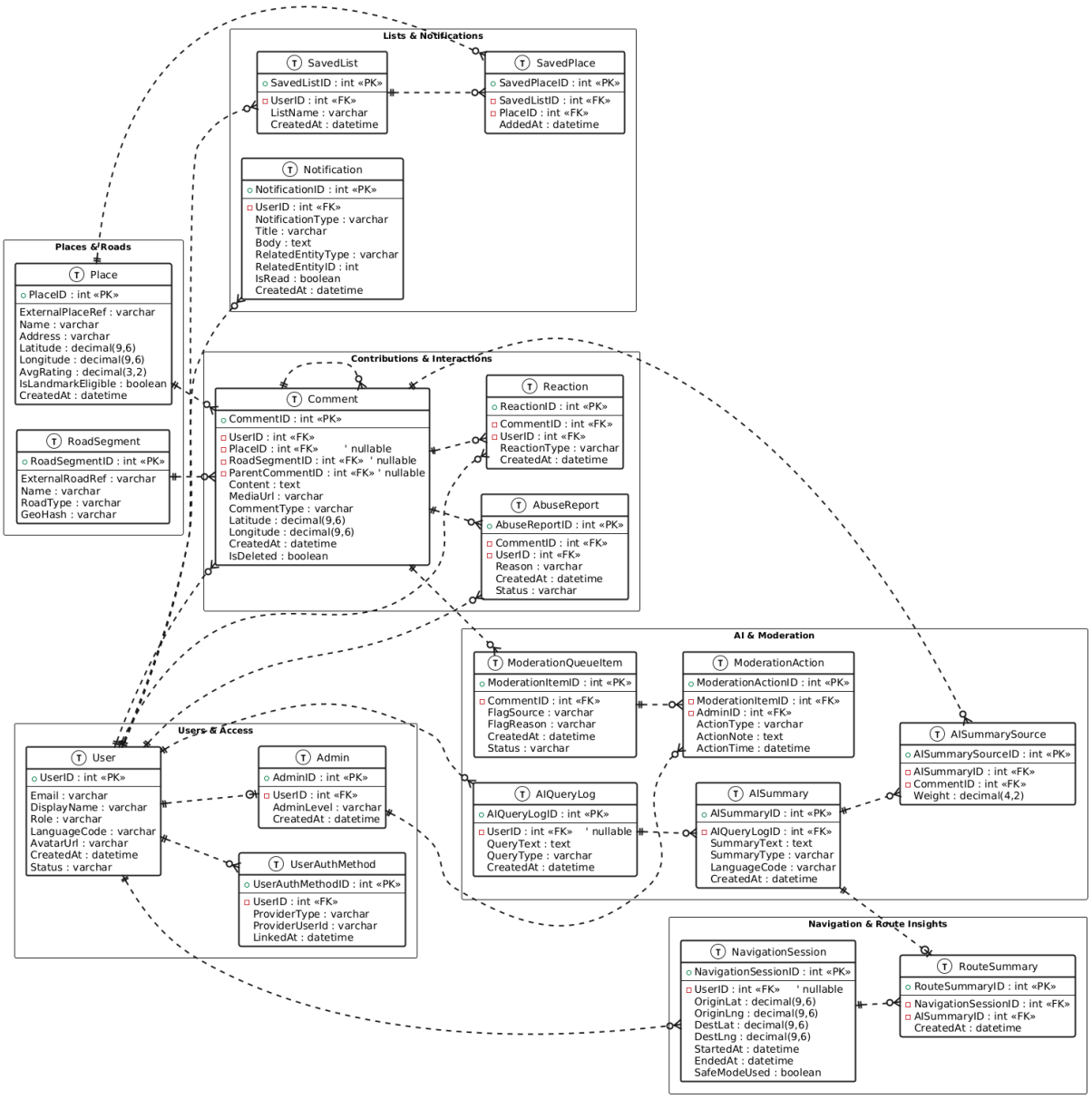

Figure 30: ER Diagram

C3.2.8 UI Design

Below are some high-level user interface designs:

Home / Map Screen

- Dynamic Google Map with zoom, pan, and traffic layers.
- Current location indicator.
- Search bar (top).
- “Report Event” button (floating).
- Nearby POIs.
- Re-center location button.

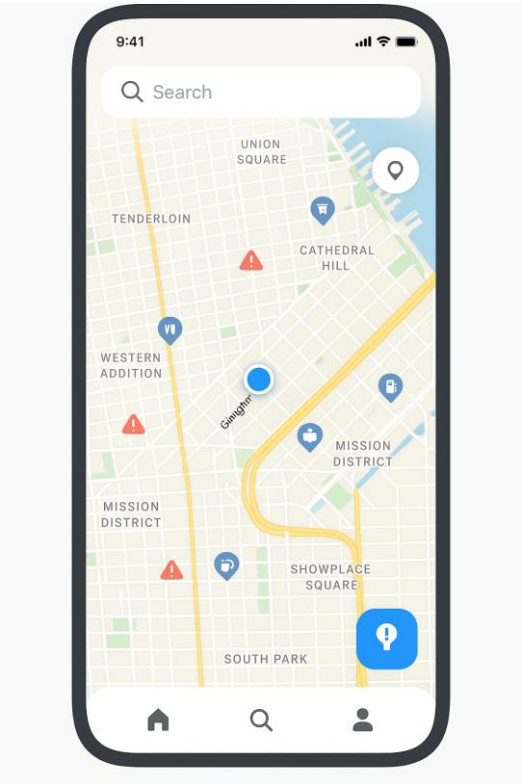

Figure 31: Home/Maps Screen

Navigation Screen

- Turn-by-turn instructions.
- Lane guidance (arrows).
- Landmark-enhanced instructions.
- ETA, distance, speed.
- Rerouting alerts.

- Safe-driving mode with simplified UI.

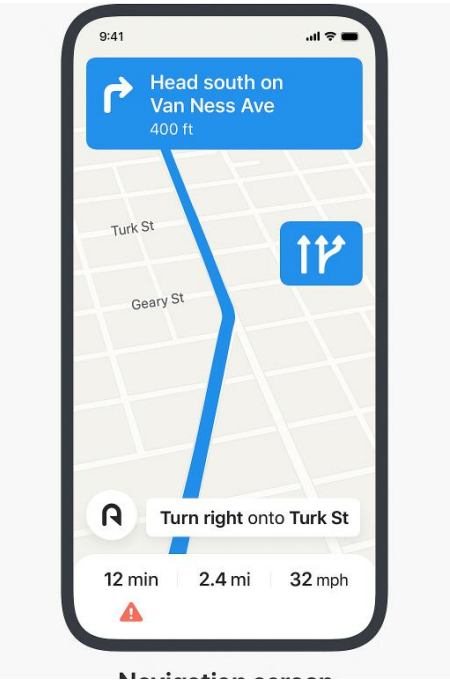

Figure 32: Navigation Screen

Search Screen

- Search bar with autocomplete suggestions.
- Filters (Open now, Distance, Rating).
- Search results list + map preview.

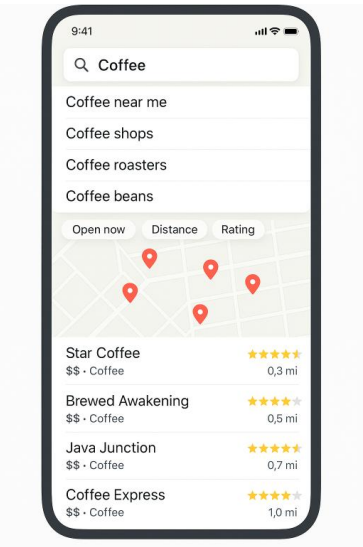

Figure 33: Search Screen

Place Detail Screen

- Name, rating, distance, opening hours.
- Photos.
- User comments and road threads.
- "Navigate," "Save," "Share," and "Report" options.

Figure 34: Place Details Screen

Event Reporting Screen

- Event categories (accident, road closure, traffic, hazard).
- Optional text and photo uploads.
- Location auto-filled.

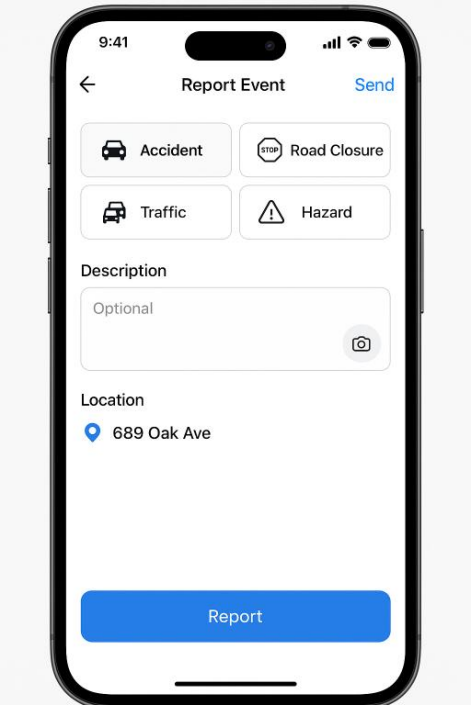

Figure 35: Report Event Screen

AI Insights Screen

- Route summaries.
- Community event summaries.
- Source links to original posts.

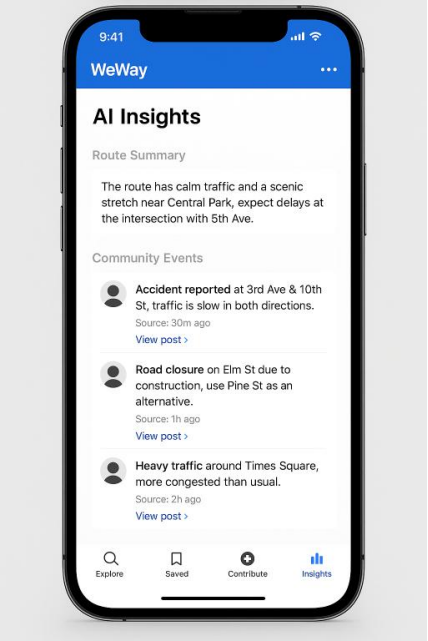

Figure 36: AI Insights Screen

Profile & Settings Screen

- Profile photo, name, bio.
- Account settings.
- Privacy controls.
- Saved lists.

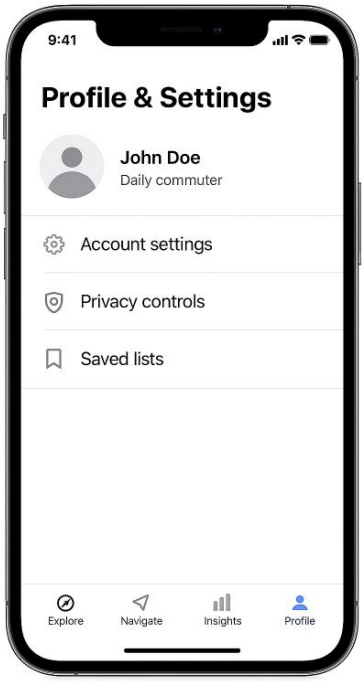

Figure 37: Profile and Setting Screen

Admin Moderation Screen (Admin Only)

- Flagged posts.
- Abuse reports.
- Decision actions (Approve, Remove).
- User violation logs.

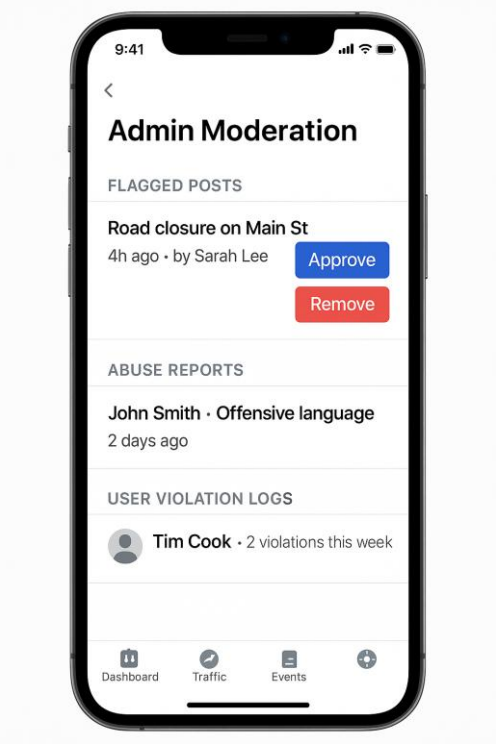

Figure 38: Admin Moderation Screen

Add Comments Flow Screen

- Displays selected place/road name, preview images, rating, distance, and status.
- Shows popular comments with user profiles, timestamps, likes, replies, and share options.
- Provides “Add Comment” entry point for posting a new contribution.
- Allows attaching photos, viewing location coordinates, and submitting the comment.
- Offers quick access to map preview, media thumbnails, and detailed place information.

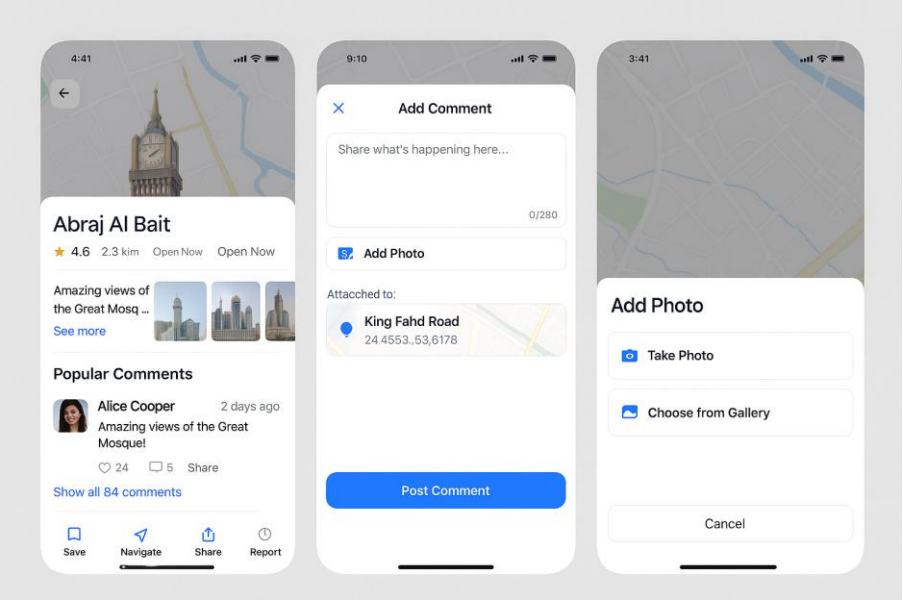

Figure 39: Add comments flow screen

Road Thread View Screen

- Shows road segment name and corresponding geographic location on the map.
- Displays a list of user comments related to that specific road segment.
- Includes avatars, usernames, timestamps, text comments, and attached photos.
- Provides an "Add Comment" button to contribute road-specific updates.
- Automatically links each comment to exact geo-coordinates for accuracy.

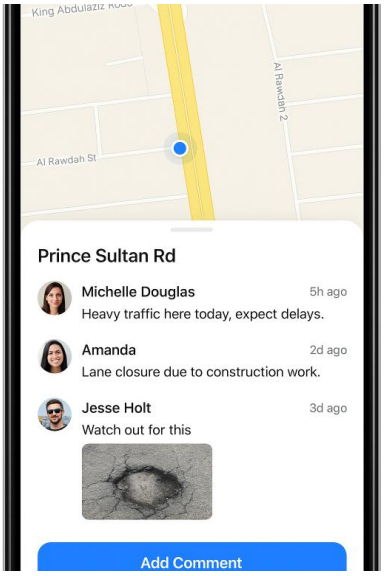

Figure 40: Road thread screen

Safe Driving Mode Panel Screen

- Activates a simplified interface optimized for minimal driver distraction.
- Displays quick-action safety tiles (Road Closure, Hazard, Police, Accident, Pothole, Quick Photo Report).
- Includes voice-input button for hands-free reporting.
- Features Safe Driving Mode toggle with clear visual indication.
- Provides accessible navigation tabs (Home, Search, Contribute, Profile) in reduced-distraction format.

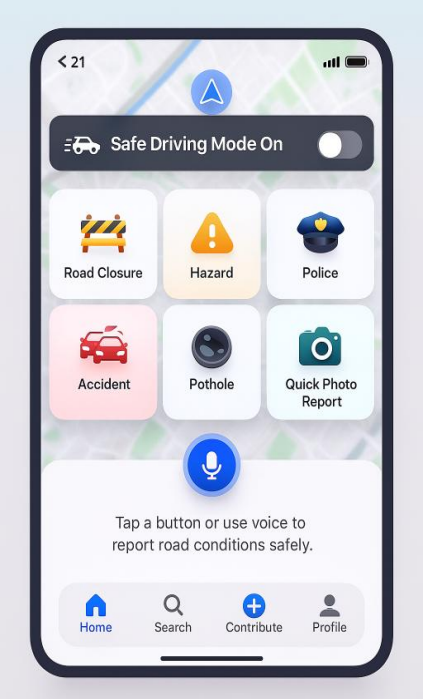

Figure 41: Safe driving mode panel screen:

C3.3. Design Rationale

Why Flutter?

- Cross-platform support (Android + iOS)
- High UI performance for map rendering
- Strong plugin ecosystem (Google Maps, Firebase)

Why Firebase Backend?

- Real-time syncing
- Simple NoSQL structure
- Scales automatically
- Secure with built-in authentication

Why OpenAI Integration?

- High-quality text summarization
- Effective moderation filtering
- Provides unique differentiating features (AI route awareness)

Why Modular Architecture?

- Separates UI, logic, data layers
- Enables independent development
- Easier maintenance & testing

Why Prototyping + Agile?

- Rapid iteration needed for UI-heavy mobile apps
- Frequent adjustments based on map UI feedback
- AI features require experimentation and refinement

C3.4. Design Constraints

Technological Constraints

- Limited device processing power (mobile phones)

- GPS errors and inconsistent sensor accuracy
- API quotas (Google Maps, OpenAI, Firebase)

Architecture Constraints

- Serverless architecture limits complex server-side logic
- Must use structured Firestore rules for security
- Offline use is limited since map APIs require connectivity

Performance Constraints

- Map rendering must remain < 200ms frame time
- Real-time comment loading must avoid blocking UI
- AI calls must return within reasonable latency

Security Constraints

- Authentication must follow industry standards (OAuth, secure storage)
- All communication must use HTTPS/TLS
- No plaintext sensitive data allowed

C3.5 Design Languages

Modeling Languages Used

- **UML 2.5**
    - Use Case Diagram
    - Class Diagram
    - Activity Diagram
    - State Diagram (recommended)
    - Sequence Diagrams
    - State Machine Diagrams
- **Informal Notation**
    - Block Diagram

Database Modeling

- ER Diagram (Chen/Crow's Foot combined)
- Firestore NoSQL data model derived from ERD

UI Wireframing

- Figma-style UI mockups
- High-fidelity screens provided for main flows
- Light-mode design theme matching project branding

C4) Software Implementation

C4.1 Initial Implementation

The initial implementation of the system is based on the UML class diagram shown in figure 21.

```
/* ======================

   CORE USER DOMAIN

   ====================== */


public class User {

    private int userId;

    private String email;

    private String displayName;

    /** "Guest", "Registered", "Admin" */

    private String role;

    /** "active", "suspended" */

    private String status;

    // Associations (from UML, 1..* UserAuthMethod, 0..* Comment, 0..* SavedList, 0..* Notification, 0..* NavigationSession)

    private final List<UserAuthMethod> authMethods = new ArrayList<>();

    private final List<Comment> comments = new ArrayList<>();

    private final List<SavedList> savedLists = new ArrayList<>();

    private final List<Notification> notifications = new ArrayList<>();

    private final List<NavigationSession> navigationSessions = new ArrayList<>();

    public User() {

    }

    public User(int userId, String email, String displayName, String role, String status) {

        this.userId = userId;

        this.email = email;

        this.displayName = displayName;

        this.role = role;
```

```
    this.status = status;
}
// --- Methods from UML ---
public boolean canPost() {
    return "active".equalsIgnoreCase(status);
}
public boolean canSavePlaces() {
    return "active".equalsIgnoreCase(status)
            && !"Guest".equalsIgnoreCase(role);
}
public boolean canModerate() {
    return "Admin".equalsIgnoreCase(role)
            && "active".equalsIgnoreCase(status);
}
public void updateProfile(String name, String language) {
    this.displayName = name;
    // languageCode wasn't in this version of the diagram, so we just
    // ignore `language` or you can extend User with a language field if needed.
}
// --- Convenience association helpers (implementation detail) ---
public void addAuthMethod(UserAuthMethod method) {
    authMethods.add(method);
}
public List<UserAuthMethod> getAuthMethods() {
    return authMethods;
}
public void addComment(Comment comment) {
    comments.add(comment);
}
```

```
public List<Comment> getComments() {
    return comments;
}
public void addSavedList(SavedList list) {
    savedLists.add(list);
}
public List<SavedList> getSavedLists() {
    return savedLists;
}
public void addNotification(Notification notification) {
    notifications.add(notification);
}
public List<Notification> getNotifications() {
    return notifications;
}
public void addNavigationSession(NavigationSession session) {
    navigationSessions.add(session);
}
public List<NavigationSession> getNavigationSessions() {
    return navigationSessions;
}
// Getters/Setters for attributes (not in UML, but practical)
public int getUserId() {
    return userId;
}
public void setUserId(int userId) {
    this.userId = userId;
}
```

```
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getDisplayName() {
        return displayName;
    }
    public void setDisplayName(String displayName) {
        this.displayName = displayName;
    }
    public String getRole() {
        return role;
    }
    public void setRole(String role) {
        this.role = role;
    }
    public String getStatus() {
        return status;
    }
    public void setStatus(String status) {
        this.status = status;
    }
}
```

```
/* UserAuthMethod */
class UserAuthMethod {
    private int userAuthMethodId;
    /** "Email", "Google", "Apple" */
    private String providerType;
    private String providerUserId;
    private LocalDateTime linkedAt;
    // back-reference optional
    private User user;
    public UserAuthMethod() {
    }
    public UserAuthMethod(int userAuthMethodId, String providerType, String providerUserId,
                          LocalDateTime linkedAt) {
        this.userAuthMethodId = userAuthMethodId;
        this.providerType = providerType;
        this.providerUserId = providerUserId;
        this.linkedAt = linkedAt;
    }
    public boolean isSocialLogin() {
        return !"Email".equalsIgnoreCase(providerType);
    }
    public void unlink() {
        this.providerUserId = null;
    }
    // Getters/Setters
    public int getUserAuthMethodId() {
        return userAuthMethodId;
    }
```

```
    public void setUserAuthMethodId(int userAuthMethodId) {
        this.userAuthMethodId = userAuthMethodId;
    }
    public String getProviderType() {
        return providerType;
    }
    public void setProviderType(String providerType) {
        this.providerType = providerType;
    }
    public String getProviderUserId() {
        return providerUserId;
    }
    public void setProviderUserId(String providerUserId) {
        this.providerUserId = providerUserId;
    }
    public LocalDateTime getLinkedAt() {
        return linkedAt;
    }
    public void setLinkedAt(LocalDateTime linkedAt) {
        this.linkedAt = linkedAt;
    }
    public User getUser() {
        return user;
    }
    public void setUser(User user) {
        this.user = user;
    }
}
```

```
/* Admin extends User */
class Admin extends User {
    private int adminId;
    private LocalDateTime createdAt;
    public Admin() {
    }
    public Admin(int adminId, LocalDateTime createdAt) {
        this.adminId = adminId;
        this.createdAt = createdAt;
    }
    public void reviewItem(ModerationQueueItem item) {
        // domain logic stub
    }
    public void suspendUser(User user) {
        user.setStatus("suspended");
    }
    public void manageUsers() {
        // domain logic stub
    }
    public int getAdminId() {
        return adminId;
    }
    public void setAdminId(int adminId) {
        this.adminId = adminId;
    }
    public LocalDateTime getCreatedAt() {
        return createdAt;
    }
```

```
    public void setCreatedAt(LocalDateTime createdAt) {
        this.createdAt = createdAt;
    }
}
/* ========================
   PLACES & ROADS
   ======================== */
class Place {
    private int placeId;
    private String name;
    private double latitude;
    private double longitude;
    // Implementation detail (not in UML but needed by updateRating)
    private Double rating;
    // Comments association
    private final List<Comment> comments = new ArrayList<>();
    public Place() {
    }
    public Place(int placeId, String name, double latitude, double longitude) {
        this.placeId = placeId;
        this.name = name;
        this.latitude = latitude;
        this.longitude = longitude;
    }
    public String getCoordinates() {
        return latitude + "," + longitude;
    }
    public void updateRating(double newRating) {
        this.rating = newRating;
```

```
}
public void addComment(Comment comment) {
    comments.add(comment);
}
public List<Comment> getComments() {
    return comments;
}
// Getters/Setters
public int getPlaceId() {
    return placeId;
}
public void setPlaceId(int placeId) {
    this.placeId = placeId;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public double getLatitude() {
    return latitude;
}
public void setLatitude(double latitude) {
    this.latitude = latitude;
}
public double getLongitude() {
    return longitude;
}
```

```
    public void setLongitude(double longitude) {
        this.longitude = longitude;
    }
    public Double getRating() {
        return rating;
    }
}
class RoadSegment {
    private int roadSegmentId;
    private String name;
    private String geoHash;
    private final List<Comment> comments = new ArrayList<>();-
    public RoadSegment() {
    }
    public RoadSegment(int roadSegmentId, String name, String geoHash) {
        this.roadSegmentId = roadSegmentId;
        this.name = name;
        this.geoHash = geoHash;
    }
    public void updateGeoHash(String hash) {
        this.geoHash = hash;
    }
    public String getLabel() {
        return name + " (" + geoHash + ")";
    }
    public void addComment(Comment comment) {
        comments.add(comment);
    }
```

```
    public List<Comment> getComments() {
        return comments;
    }
    // Getters/Setters
    public int getRoadSegmentId() {
        return roadSegmentId;
    }
    public void setRoadSegmentId(int roadSegmentId) {
        this.roadSegmentId = roadSegmentId;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getGeoHash() {
        return geoHash;
    }
    public void setGeoHash(String geoHash) {
        this.geoHash = geoHash;
    }
}
```

```
/* ========================
   COMMENTS & INTERACTIONS
   ======================== */
class Comment {
    private int commentId;
    private String content;
    private LocalDateTime createdAt;
    private boolean deleted;
    private final List<Reaction> reactions = new ArrayList<>();
    private final List<AbuseReport> abuseReports = new ArrayList<>();
    public Comment() {
    }
    public Comment(int commentId, String content, LocalDateTime createdAt) {
        this.commentId = commentId;
        this.content = content;
        this.createdAt = createdAt;
        this.deleted = false;
    }
    public void editContent(String newText) {
        this.content = newText;
    }
    public void softDelete() {
        this.deleted = true;
    }
    public void addReaction(Reaction reaction) {
        reactions.add(reaction);
    }
    public void addAbuseReport(AbuseReport report) {
        abuseReports.add(report);
```

```
    }

    // Getters/Setters
    public int getCommentId() {
        return commentId;
    }
    public void setCommentId(int commentId) {
        this.commentId = commentId;
    }
    public String getContent() {
        return content;
    }
    public LocalDateTime getCreatedAt() {
        return createdAt;
    }
    public boolean isDeleted() {
        return deleted;
    }
    public List<Reaction> getReactions() {
        return reactions;
    }
    public List<AbuseReport> getAbuseReports() {
        return abuseReports;
    }
}
```

```
class Reaction {
    private int reactionId;
    private String reactionType;
    public Reaction() {
    }
    public Reaction(int reactionId, String reactionType) {
        this.reactionId = reactionId;
        this.reactionType = reactionType;
    }
    public void setType(String type) {
        this.reactionType = type;
    }
    // Getters/Setters
    public int getReactionId() {
        return reactionId;
    }
    public void setReactionId(int reactionId) {
        this.reactionId = reactionId;
    }
    public String getReactionType() {
        return reactionType;
    }
}
class AbuseReport {
    private int abuseReportId;
    private String reason;
    private boolean resolved;
    public AbuseReport() {
    }
```

```
    public AbuseReport(int abuseReportId, String reason) {
        this.abuseReportId = abuseReportId;
        this.reason = reason;
        this.resolved = false;
    }
    public void markResolved() {
        this.resolved = true;
    }
    // Getters/Setters
    public int getAbuseReportId() {
        return abuseReportId;
    }
    public void setAbuseReportId(int abuseReportId) {
        this.abuseReportId = abuseReportId;
    }
    public String getReason() {
        return reason;
    }
    public boolean isResolved() {
        return resolved;
    }
}
```

```
/* ========================
   LISTS & SAVED PLACES
   ======================== */
class SavedList {
    private int savedListId;
    private String listName;
    private final List<SavedPlace> savedPlaces = new ArrayList<>();
    public SavedList() {
    }
    public SavedList(int savedListId, String listName) {
        this.savedListId = savedListId;
        this.listName = listName;
    }
    public void rename(String newName) {
        this.listName = newName;
    }
    public List<Place> getPlaces() {
        List<Place> result = new ArrayList<>();
        for (SavedPlace sp : savedPlaces) {
            if (sp.getPlace() != null) {
                result.add(sp.getPlace());
            }
        }
        return result;
    }
    public void addSavedPlace(SavedPlace savedPlace) {
        savedPlaces.add(savedPlace);
    }
```

```
    public List<SavedPlace> getSavedPlaces() {
        return savedPlaces;
    }
    // Getters/Setters
    public int getSavedListId() {
        return savedListId;
    }
    public void setSavedListId(int savedListId) {
        this.savedListId = savedListId;
    }
    public String getListName() {
        return listName;
    }
}
class SavedPlace {
    private int savedPlaceId;
    private LocalDateTime addedAt;
    private boolean favourite;
    private Place place;
    public SavedPlace() {
    }
    public SavedPlace(int savedPlaceId, LocalDateTime addedAt, Place place) {
        this.savedPlaceId = savedPlaceId;
        this.addedAt = addedAt;
        this.place = place;
        this.favourite = false;
    }
    public void markFavourite() {
        this.favourite = true;
```

```
    }

    // Getters/Setters
    public int getSavedPlaceId() {
        return savedPlaceId;
    }
    public void setSavedPlaceId(int savedPlaceId) {
        this.savedPlaceId = savedPlaceId;
    }
    public LocalDateTime getAddedAt() {
        return addedAt;
    }
    public void setAddedAt(LocalDateTime addedAt) {
        this.addedAt = addedAt;
    }
    public boolean isFavourite() {
        return favourite;
    }
    public Place getPlace() {
        return place;
    }
    public void setPlace(Place place) {
        this.place = place;
    }
}
```

```
/* ========================
   NOTIFICATIONS
   ======================== */
class Notification {
    private int notificationId;
    private String type;
    private String body;
    private boolean isRead;
    public Notification() {
    }
    public Notification(int notificationId, String type, String body) {
        this.notificationId = notificationId;
        this.type = type;
        this.body = body;
        this.isRead = false;
    }
    public void markRead() {
        this.isRead = true;
    }
    public boolean isUnread() {
        return !isRead;
    }
    // Getters/Setters
    public int getNotificationId() {
        return notificationId;
    }
    public void setNotificationId(int notificationId) {
        this.notificationId = notificationId;
    }
```

```
    public String getType() {
        return type;
    }
    public String getBody() {
        return body;
    }
    public boolean isRead() {
        return isRead;
    }
}
/* ======================
   AI & MODERATION
   ====================== */
class ModerationQueueItem {
    private int moderationItemId;
    private String flagSource; // "AI", "User"
    private String status;     // pending/resolved
    private final List<ModerationAction> actions = new ArrayList<>();
    public ModerationQueueItem() {
        this.status = "pending";
    }
    public ModerationQueueItem(int moderationItemId, String flagSource) {
        this.moderationItemId = moderationItemId;
        this.flagSource = flagSource;
        this.status = "pending";
    }
    public void markPending() {
        this.status = "pending";
    }
```

```
    public void markResolved() {
        this.status = "resolved";
    }
    public void addAction(ModerationAction action) {
        actions.add(action);
    }
    public List<ModerationAction> getActions() {
        return actions;
    }
    // Getters/Setters
    public int getModerationItemId() {
        return moderationItemId;
    }
    public void setModerationItemId(int moderationItemId) {
        this.moderationItemId = moderationItemId;
    }
    public String getFlagSource() {
        return flagSource;
    }
    public void setFlagSource(String flagSource) {
        this.flagSource = flagSource;
    }
    public String getStatus() {
        return status;
    }
}
```

```
class ModerationAction {
    private int moderationActionId;
    private String actionType; // "approve", "remove"
    private Comment targetComment;
    public ModerationAction() {
    }
    public ModerationAction(int moderationActionId, String actionType, Comment targetComment) {
        this.moderationActionId = moderationActionId;
        this.actionType = actionType;
        this.targetComment = targetComment;
    }
    public void applyTo(Comment comment) {
        this.targetComment = comment;
        if ("remove".equalsIgnoreCase(actionType) && comment != null) {
            comment.softDelete();
        }
        // "approve" can be a no-op or clear moderation flags in a richer model
    }
    // Getters/Setters
    public int getModerationActionId() {
        return moderationActionId;
    }
    public void setModerationActionId(int moderationActionId) {
        this.moderationActionId = moderationActionId;
    }
    public String getActionType() {
        return actionType;
    }
    public Comment getTargetComment() {
```

```
        return targetComment;
    }
}
class AISummary {
    private int aiSummaryId;
    private String summaryText;
    private Comment sourceComment;
    public AISummary() {
    }
    public AISummary(int aiSummaryId, String summaryText) {
        this.aiSummaryId = aiSummaryId;
        this.summaryText = summaryText;
    }
    public String getPreview(int maxChars) {
        if (summaryText == null) {
            return "";
        }
        if (summaryText.length() <= maxChars) {
            return summaryText;
        }
        return summaryText.substring(0, maxChars) + "...";
    }
    public boolean isEmpty() {
        return summaryText == null || summaryText.isBlank();
    }
    public void setSourceComment(Comment comment) {
        this.sourceComment = comment;
    }
    public Comment getSourceComment() {
```

```
        return sourceComment;
    }
    // Getters/Setters
    public int getAiSummaryId() {
        return aiSummaryId;
    }
    public void setAiSummaryId(int aiSummaryId) {
        this.aiSummaryId = aiSummaryId;
    }
    public String getSummaryText() {
        return summaryText;
    }
    public void setSummaryText(String summaryText) {
        this.summaryText = summaryText;
    }
}
/* ========================
   NAVIGATION & SESSIONS
   ======================== */
class NavigationSession {
    private int navigationSessionId;
    private LocalDateTime startedAt;
    private LocalDateTime endedAt;
    private RouteSummary routeSummary;
    public NavigationSession() {
    }
    public NavigationSession(int navigationSessionId) {
        this.navigationSessionId = navigationSessionId;
    }
```

```
public void start() {
    this.startedAt = LocalDateTime.now();
    this.endedAt = null;
}
public void end() {
    this.endedAt = LocalDateTime.now();
}
public boolean isActive() {
    return startedAt != null && endedAt == null;
}
public void setRouteSummary(RouteSummary routeSummary) {
    this.routeSummary = routeSummary;
}
public RouteSummary getRouteSummary() {
    return routeSummary;
}
// Getters/Setters
public int getNavigationSessionId() {
    return navigationSessionId;
}

public void setNavigationSessionId(int navigationSessionId) {
    this.navigationSessionId = navigationSessionId;
}
public LocalDateTime getStartedAt() {
    return startedAt;
}
public LocalDateTime getEndedAt() {
    return endedAt;
```

```
    }
}
class RouteSummary {
    private int routeSummaryId;
    private LocalDateTime createdAt;
    private AISummary aiSummary;
    public RouteSummary() {
    }
    public RouteSummary(int routeSummaryId) {
        this.routeSummaryId = routeSummaryId;
        this.createdAt = LocalDateTime.now();
    }

    public String getText() {
        return aiSummary != null ? aiSummary.getSummaryText() : "";
    }
    public void attachSummary(AISummary summary) {
        this.aiSummary = summary;
    }
    // Getters/Setters
    public int getRouteSummaryId() {
        return routeSummaryId;
    }
    public void setRouteSummaryId(int routeSummaryId) {
        this.routeSummaryId = routeSummaryId;
    }
    public LocalDateTime getCreatedAt() {
        return createdAt;
    }
```

```
public AISummary getAiSummary() {

  return aiSummary;  }}
```

C7) Technical and non-technical Constraints

C7.A Technical Constraints

Type	Description	Examples (Specific to WeWay)	Possible Solutions
Hardware Limitations	Mobile devices have varying CPU/GPU performance, affecting map rendering.	Slow rendering of Google Maps on low-tier Android devices.	Apply pixel caching; reduce map overlay complexity.
Hardware Limitations	Inconsistent GPS accuracy impacts road-segment detection.	GPS drift in dense urban areas or tunnels.	Use fused location provider; apply Kalman filters.
Hardware Limitations	Accelerometer readings differ across devices, affecting Safe Driving Mode triggers.	Some devices report motion late or with noise.	Sensor smoothing; threshold tuning based on device capabilities.
Software / Platform Constraints	The app must be built using Flutter only.	No native Android/iOS code for navigation customisation.	Use platform channels only when absolutely needed.
Software / Platform Constraints	Firestore NoSQL limits nested structures and query complexity.	Cannot easily join lists of comments, places, and reactions in one query.	Use subcollections and composite indexes; denormalise when needed.

Software / Platform Constraints	Google Maps SDK imposes quota and usage restrictions.	High usage of auto-complete search or route recalculation.	Cache responses; throttle requests.
Integration Constraints	Dependency on Google Maps, Places, Directions.	API downtime or rate limit blocks navigation.	Use fallback cached routes; show “offline navigation unavailable” UI.
Integration Constraints	Reliance on OpenAI for route summaries and moderation.	AI latency or content filter failures.	Implement server-side timeout fallback; show manual summary placeholder.
Integration Constraints	Strong coupling with Firebase Authentication.	Outages affect login flow.	Implement local “guest mode” as fallback.
Security Constraints	All authentication must follow OAuth2.	Google/Apple login flows must not expose tokens.	Store tokens in Secure Enclave/Keystore only.
Security Constraints	Firebase Security Rules must prevent unauthorised writes.	Users attempting to edit someone else’s comments.	Role-based access rules; server-side validation.
Security Constraints	All communication must use TLS 1.2+.	Mobile networks dropping back to insecure HTTP via proxies.	Enforce HTTPS-only endpoints and certificate pinning.
Performance Constraints	Autocomplete search must be near real-time.	Query delays during high latency.	Debounce search input; prefetch nearby places.
Performance Constraints	AI summary must return quickly.	AI delay >3 seconds disrupts UX.	Show progress loader; handle async updates.

Performance Constraints	Firestore read/write must remain efficient.	Large comment threads may slow down reads.	Pagination; lazy-loading comments.
Reliability / Availability Constraints	App must remain functional under weak network.	Blurry map tiles; delayed thread loading.	Offline caching; retry queues.
Reliability / Availability Constraints	API outages must not break core flows.	Google Maps temporary downtime.	Graceful error UI + fallback cached routes.
Data Constraints	Firestore doc size must remain under limits.	Large comment threads exceed storage limits.	Move replies into subcollections.
Data Constraints	Unicode support for Arabic/English comment content.	Various fonts rendering differently on devices.	Use UTF-8 and standard Flutter fonts.
Architectural / Methodology Constraints	System must follow layered architecture (UI → Logic → Data).	Direct access to Firestore from UI discouraged.	Enforce repository pattern.
Architectural / Methodology Constraints	Must follow UML modelling and SDD format.	Use case/class/activity/sequence diagrams required.	Maintain consistent modelling standards.

C7.B Non-Technical Constraints

Type	Description	Examples (Specific to WeWay)	Possible Solutions
Time Constraints	Fixed 15-week development timeframe.	Only limited iterations possible for navigation + AI features.	Prioritize critical flows first; timebox AI tasks.
Time Constraints	Two full prototypes + final demo required within semester.	Limited buffer time for redesign.	Adopt weekly sprint goals.
Budget Constraints	Must operate under free-tier cloud resources.	Google Maps API quota; OpenAI token usage.	Cache results; minimise frequent API calls.
Budget Constraints	Cannot purchase premium navigation SDKs.	No paid offline-map SDK support.	Use Google free-tier capabilities only.
Team Skill Constraints	Team must learn advanced cloud integrations during project.	Moderate experience with OpenAI, Firebase rules, and geospatial APIs.	Divide learning tasks; internal knowledge sharing.
Team Skill Constraints	Only two developers available.	Complex features like moderation pipeline require careful planning.	Limit feature scope; reuse built-in tools.
Legal / Ethical Constraints	User-generated content must be moderated.	Harmful comments, defamation.	Use AI filtering + admin review.
Legal / Ethical Constraints	GPS data must be handled responsibly.	User location cannot be misused.	Store only necessary metadata; restrict visibility.
Cultural / Localization Constraints	Must support both Arabic and English.	Bidirectional UI; reversed layouts.	Use Flutter localization + RTL support.

Cultural / Localization Constraints	AI summaries must respect cultural norms.	Sensitive phrasing in summaries.	Add AI safety prompts; human review for flagged content.
Operational Constraints	Application will be used outdoors.	Network fluctuations; device battery consumption.	Provide offline map fallback UI; optimise sensor usage.
Operational Constraints	Must run in regions with strict GPS behaviour differences.	Urban canyon effect.	Fuse accelerometer + GPS data.
Environmental Constraints	Physical surroundings impact GPS.	Indoors/tunnels causing location drift.	Show “low accuracy mode” notice.
Environmental Constraints	Weather conditions may limit usability.	User might lose GPS signal during rain/storms.	Device sensor fallback; cache last route.
Type	**Description**	**Examples (Specific to WeWay)**	**Possible Solutions**
Time Constraints	Fixed 15-week development timeframe.	Only limited iterations possible for navigation + AI features.	Prioritize critical flows first; timebox AI tasks.

C8) Standards

C8.1 Standards

The WeWay project adheres to several internationally recognized engineering and software standards to ensure correctness, reliability, interoperability, and quality throughout the system lifecycle. These standards guide our modelling practices, communication protocols, database conventions, security mechanisms, and development methodologies.

1. Software Engineering & Requirements Standards

- IEEE 830 / ISO/IEC/IEEE 29148 – Standards for Software Requirements Specifications followed in the creation of the WeWay SRS.
- ISO/IEC 25010 – Software product quality model guiding maintainability, usability, reliability, and performance expectations.

2. Modelling & Design Standards

- UML 2.5 Standard – Used for use case diagrams, activity diagrams, sequence diagrams, and system modelling.
- IDEF1X / Crow's Foot Notation – Used for Entity Relationship Diagrams (ERDs) and database modelling.

3. Communication & Network Standards

- TCP/IP Model – Underlies all network communication.
- TLS 1.2+ (Transport Layer Security) – Ensures secure, encrypted communication between app, APIs, and backend systems.
- HTTPS (RFC 2818) – Mandatory protocol for client–server communication.

4. Data Representation & Encoding Standards

- UTF-8 Unicode Standard – Ensures proper representation of international characters across the app.
- JSON (RFC 8259) – Used as the standard lightweight data format for REST communications with APIs.

5. API & Integration Standards

- Google Maps / Places API Specifications – Followed for map rendering, geocoding, routing, and POI data.
- OpenAI / AI Platform Safety Standards – Applied during content moderation and summarization tasks.

6. Security & Authentication Standards

- OAuth 2.0 – Used for secure social login with Google and Apple.
- NIST Password Guidelines (SP 800-63B) – Basis for password rules and secure authentication policy.
- Firebase Security Rules Standard – Applied to enforce role-based access, data validation, and secure read/write controls.

7. Mobile Development & Packaging Standards

- Google Android Developer Guidelines – Followed for permissions, UI behaviour, and performance optimization.
- Apple Human Interface Guidelines (HIG) – For UI consistency on iOS devices.

8. Media & File Standards

- JPEG / PNG Standards (ISO/IEC 10918 & PNG RFC 2083) – Used for uploaded images, compression, and rendering.
- MIME Type Standards (IETF) – For consistent handling of media files between the app and backend.

C9) References & Appendices

C9.1 References

1. **Google Maps Platform Documentation**
   Comprehensive reference for Maps SDK, Places API, Routes API, Geocoding, and Navigation behaviours used in WeWay.
   https://developers.google.com/maps
2. **Firebase Documentation (Authentication, Firestore, Storage, Cloud Messaging)**
   Official backend documentation covering authentication, secure storage, NoSQL schema design, cloud messaging, and backend operations.
   https://firebase.google.com/docs
3. **OpenAI API Reference**
   Full API specifications for text summarization, moderation, NLP, embeddings, and content safety models used in WeWay's AI features.
   https://platform.openai.com/docs
4. **Flutter & Dart Documentation**
   Programming language and framework used to build the WeWay mobile client; includes UI widgets, state management, performance, and platform integration guidelines.
   https://docs.flutter.dev
5. **ISO/IEC/IEEE 29148:2018 – Systems and Software Requirements Engineering**
   International standard governing how software requirements are defined, validated, and managed throughout the lifecycle.
   https://www.iso.org/standard/72057.html
6. **UML 2.5 Specification (OMG)**
   Official modeling standard used to create WeWay's Use Case, Class, Activity, Sequence, and State Machine diagrams.
   https://www.omg.org/spec/UML
7. **Android Human Interface Guidelines (HIG)**
   UI/UX guidelines for mobile interactions, gestures, navigation patterns, and accessibility for Android devices.
   https://developer.android.com/design

8. **Apple Human Interface Guidelines (iOS HIG)**
   Guidelines for designing intuitive iOS user experiences, gestures, safe-area behaviours, and platform-specific UI structure.
   https://developer.apple.com/design/human-interface-guidelines/
9. **Academic Research on Navigation UX (Waze, Google Maps, Landmark Navigation)**
   Usability studies on cognitive load, map interpretation, driving distraction, and landmark-based navigation behaviour.
   Examples:
   a. https://dl.acm.org/doi/10.1145/2858036.2858213
   b. https://www.sciencedirect.com/science/article/pii/S0968090X15300729
10. **Software Engineering, 10th Edition – Ian Sommerville (Pearson)**
    Widely used academic reference covering requirements, architecture, modelling, lifecycle processes, and project management best practices.
    https://www.pearson.com/en-us/subject-catalog/p/software-engineering/P200000004354/9780133943030

C9.2 Appendices

C9.2.1 Glossary

Term	Definition
Sprint	Iterative development cycle in Agile.
Prototype	Initial working version demonstrating UI or logic.
SRS (Software Requirements Specification)	Document describing system requirements.
SDD (Software Design Description)	Document detailing architectural and design decisions.
SPMP (Software Project Management Plan)	Document detailing project planning and execution strategy.
Backlog	Ordered list of features, tasks, or enhancements.
Milestone	Major project checkpoint or deliverable.
Versioning	Tracking changes in software over time.
Traceability Matrix	Table mapping requirements to design elements and test cases.
WeWay	A mobile navigation and social interaction platform combining maps, comments, AI summaries, and road threads.
Architecture Layer	Logical separation of UI, application logic, and backend/cloud services.
API	Rules enabling communication between WeWay and external services like Google Maps or OpenAI.
Backend	Cloud-side services (Firebase, Google APIs, OpenAI) responsible for authentication, data storage, and AI processing.
Frontend (Client App)	The Flutter mobile app installed on user devices.
NoSQL Database	Non-relational database (Firestore) storing documents and collections.
Firestore	Cloud-hosted NoSQL database used for users, comments, lists, and navigation sessions.
Firebase Storage	Cloud storage for user-uploaded media (photos, attachments).

FCM (Firebase Cloud Messaging)	Service used to send push notifications to user devices.
OAuth 2.0	Secure authentication standard used for Google and Apple Sign-In.
Secret Management	Secure storage of API keys and tokens outside the app binary.
Google Maps SDK	Mapping engine used for map rendering, gestures, and user interactions.
Google Places API	Service returning place details, ratings, photos, and metadata.
Directions API	API used to compute routes and provide turn-by-turn directions.
Geocoding	Conversion between textual addresses and latitude/longitude coordinates.
Geohash	Encoded spatial identifier used for fast location grouping.
GPS Drift	Natural inaccuracy of GPS coordinates due to environmental interference.
Landmark Navigation	Navigation approach using landmarks instead of road names.
Polyline	Encoded path representing a route on the map.
POI (Point of Interest)	Relevant real-world location such as cafés, schools, or landmarks.
Safe-Zone Radius	Distance threshold used when displaying nearby places or threads.
Comment Thread	A chronological list of comments and replies for places or roads.
Road Thread	A discussion mapped to a specific road segment, triggered via long-press.
Reaction	A user interaction such as a like or upvote on a comment.
Saved List	A user-defined collection of places (e.g., Favorites).
Saved Place	An individual place stored inside a Saved List.
Report / Abuse Report	User-submitted flag indicating inappropriate or harmful content.
Reply	A direct response to an existing comment.
Content Flagging	The process of marking content as inappropriate for moderation review.

Guest User	Unauthenticated user with limited app privileges.
Registered User	Authenticated user able to post, save places, and interact fully.
Admin	User with elevated privileges to review flagged content or moderate discussions.
OpenAI Model	AI model used for summarization, NLP queries, and moderation analysis.
AI Summary	Auto-generated overview of an area, route, or comment thread.
AI Moderation	System that detects harmful or inappropriate content before display.
Moderation Queue	List of flagged comments awaiting admin review.
Moderation Action	Final decision by an admin (approve, remove, warn).
Content Safety	Ensuring AI-generated text avoids harmful or unethical output.
AI Query Log	Record of user-issued AI queries for transparency and debugging.
Summary Source Weighting	Relative contribution of comments used in AI summarization.
Safe Driving Mode	UI mode that limits user interaction when motion exceeds speed thresholds.
Motion Detection	Use of accelerometer and gyroscope to detect driving behaviour.
Passenger Override	Manual user input confirming they are not the driver.
Movement Threshold	Acceleration or speed value that triggers Safe Driving Mode.
Sensor Fusion	Combining multiple sensors to improve movement detection accuracy.
TLS 1.2+	Encryption standard required for all network communication.
Token Storage	Secure persistence of authentication and session tokens.
RBAC (Role-Based Access Control)	Permission system controlling user abilities by role.
Input Validation	Process of ensuring user input is safe and well-formed.
Authentication Provider	External identity service (Google/Apple Sign-In).

Security Rule (Firebase)	Access-control instructions protecting Firestore & Storage.
Hashing	One-way encryption of passwords or sensitive tokens.
Rate Limiting	Preventing excessive API calls to avoid abuse or security risks.
Session ID	Identifier that maintains user login state.
Primary Key (PK)	Unique identifier for a database record.
Foreign Key (FK)	Reference linking two related records.
Collection	Top-level grouping in Firestore analogous to a SQL table.
Document	A JSON-like object stored inside a Firestore collection.
Indexing	Technique for speeding up search and query operations.
Denormalization	Duplicating data in NoSQL for faster reads.
Composite Index	Firestore index supporting complex multi-field queries.
Subcollection	Nested collection inside a document for related data (e.g., comment replies).
Pagination	Loading data in smaller segments to improve performance.