**THE UNIVERSITY OF BUEA** **REPUBLIC OF CAMEROON**

P.O Box 63, PEACE-WORK-FATHERLAND

Buea, Southwest Region

Cameroon

Tel: (237) 674354327

Fax: (237) 3332 22 72

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**DEPARTMENT OF COMPUTER ENGINEERING**

**CEF 440: Internet programming and mobile programming**

**Design and Implementation of a Road Sign and Road State Mobile Notification Application**

Group 15

**TASK 3: REQUIREMENT ANALYSIS**

[GitHub - Asumu22/group-15](https://github.com/Asumu22/group-15)

**Presented by:**

|  |  |  |  |
| --- | --- | --- | --- |
| No | Name | Matriculation No | Specialty |
| 01 | KONGNYUY RAYMOND AFONI | FE21A219 | NE |
| 02 | BESONG ELIAS ASUMU | FE21A149 | SE |
| 03 | KANKO KEMEDJEU DUPLEX | FE21A210 | NE |
| 04 | KENEDY MALLEY ITUKA | FE21A212 | NE |
| 05 | AZEFACK JUNIOR | FE21A146 | NE |

**Dr. NKEMENI Valery**

**Course Supervisors 2023 - 2024 Academic Year**

Abstract

This document details the system modeling and design for a mobile application aimed at enhancing driver safety through real-time road sign recognition and road state notifications. The application leverages various backend services, such as traffic management systems, GPS, and weather updates, to provide drivers with timely and relevant information. The system modeling includes context diagrams, use case diagrams, sequence diagrams, class diagrams, and deployment diagrams to represent the system's structure and interactions. This comprehensive design framework ensures that the application is robust, user-friendly, and capable of delivering critical information efficiently to aid in safe and informed driving decisions.

Table of content

[Introduction 3](#_Toc167663518)

[1. Context diagram 4](#_Toc167663519)

[Entities: 4](#_Toc167663520)

[Visual Representation: 5](#_Toc167663521)

[Explanation of the Diagram: 5](#_Toc167663522)

[2. Use case diagram 6](#_Toc167663523)

[Actors 6](#_Toc167663524)

[Relationships 6](#_Toc167663525)

[Visual Representation: 7](#_Toc167663526)

[Use Cases 8](#_Toc167663527)

[3. Sequence diagram 9](#_Toc167663528)

[Participants: 9](#_Toc167663529)

[Message Flow: 9](#_Toc167663530)

[Visual Representation 11](#_Toc167663531)

[4. Class diagram 12](#_Toc167663532)

[Visual Representation 12](#_Toc167663533)

[Classes and their Attributes and Methods WITH EXPLANATION: 1](#_Toc167663534)

[Relationships: 4](#_Toc167663535)

[5. Deployment diagram 5](#_Toc167663536)

[Visual Representation 5](#_Toc167663537)

[Explanation 5](#_Toc167663538)

[Relationships and Communications 6](#_Toc167663539)

[CONCLUSION 7](#_Toc167663540)

**SYSTEM MODELING AND DESIGN for Road Sign and Road State Mobile Notification Application.**

# **Introduction**

In today's fast-paced world, ensuring road safety is paramount, particularly with the increasing number of vehicles on the road. One effective way to enhance driver safety is through real-time road sign recognition and road state information. This project focuses on the system modeling and design of a mobile application that provides drivers with crucial information about road signs, traffic conditions, and weather updates, aiding them in making safer driving decisions.

The design process follows a structured approach, including various diagrams to model the system comprehensively:

1. **Context Diagram:** Establishes the system foundation, detaling data flow and interactions between the user, the application, and backend services.
2. **Use Case Diagram:** Based on the context , it details the functionalities and interactions between the user and the application.
3. **Sequence Diagram:** Breaks down specific use cases into sequences of actions and messages between components.
4. **Class Diagram:** Defines the system's internal building blocks, their attributes, and methods.
5. **Deployment Diagram:** Illustrates the physical layout of the system components and their interactions. (can be created later).

These models provide a clear and concise blueprint for developing a reliable, efficient, and user-friendly application that integrates seamlessly with various external services to deliver real-time information to drivers.

# 1. Context diagram

It establishes the system's foundation and Illustrates data flow (location data, traffic updates, etc.) between these entities. The context diagram illustrates the system's environment, highlighting how the mobile application (the central system) interacts with the driver (user) and various backend services, which are supported by external entities. The links represent the flow of data and interactions between these entities.

## Entities:

1. Driver (User)
2. Application
3. Backend Services

* Traffic Management Center (TMC
* Sign Recognition System
* Emergency Alert System

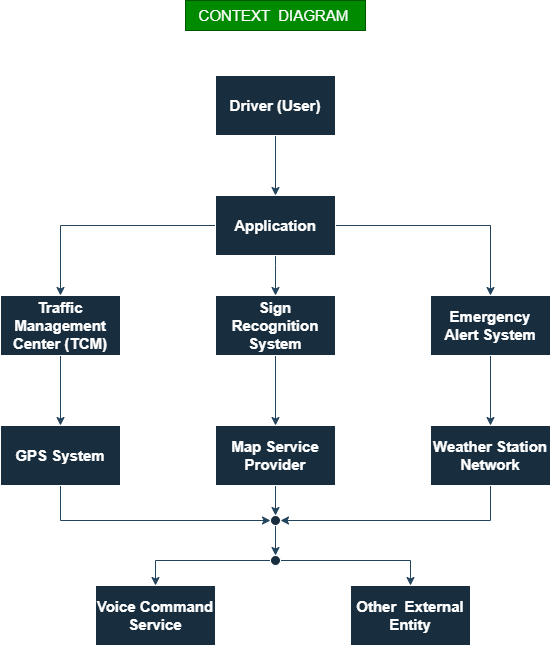
1. External Entities

* GPS System
* Map Service Provider
* Weather Station Network
* Voice Command Service

### NOTE

The arrows in the below context diagrm indicate the flow of data between the entities, showing how the application acts as a central hub, integrating data from backend services (supported by external entities) to provide comprehensive navigation and information services to the driver.

## Visual Representation:



## Explanation of the Diagram:

1. Driver (User): The end user interacts with the application on their mobile device.
2. Application: The main system that processes user inputs, sends requests to backend services, and displays information to the user.
3. Traffic Management Center (TMC): Offers real-time traffic updates to inform the driver about traffic conditions, integrated via the backend.
4. Sign Recognition System: Detects road signs and informs the application via the backend.
5. Emergency Alert System: Sends emergency alerts to the application via the backend, which are then displayed to the user.
6. GPS System: Provides real-time location data to the application through the backend.
7. Map Service Provider: Supplies map data for navigation and route planning via the backend.
8. Weather Station Network: Provides weather updates to help the application adjust routes based on weather conditions, integrated via the backend.
9. Voice Command Service: Allows the user to interact with the application using voice commands, supported via the backend.

# 2. Use case diagram

Details functionalities based on the context. Focuses on user actions (navigation, report hazards, etc.). Shows interactions between users and the app functionalities.This use case diagram effectively illustrates the interactions between the actors and the system, highlighting the primary functionalities and their dependencies.

## Actors

**1. Driver (User):** - The primary user of the system who interacts with various features of the application.

**2. GPS System:** - Provides location data to the application for navigation and route planning.

**3. Map Service Provider:** - Supplies map data and routing information.

**4. Weather Station Network:** - Provides real-time weather updates to the application.

The diagram shows the relationships between the actors and use cases, including "include" and "extend" relationships where applicable.

## Relationships

### Primary User Actions:

- The driver interacts with all the use cases directly. Their relationship is a normal **association**

### Included and exten Use Cases:

**Includes**: Indicates a use case that is always called as part of another use case.

- Navigation via Google Maps includes Weather Updates and Voice Commands.

- Profile Management Includes Voice Commands:

**Extends**: Indicates a use case that is sometimes called as part of another use case, usually based on a condition.

-Emergency alerts maybe sometimes called as costumized alerts

### External Interactions:

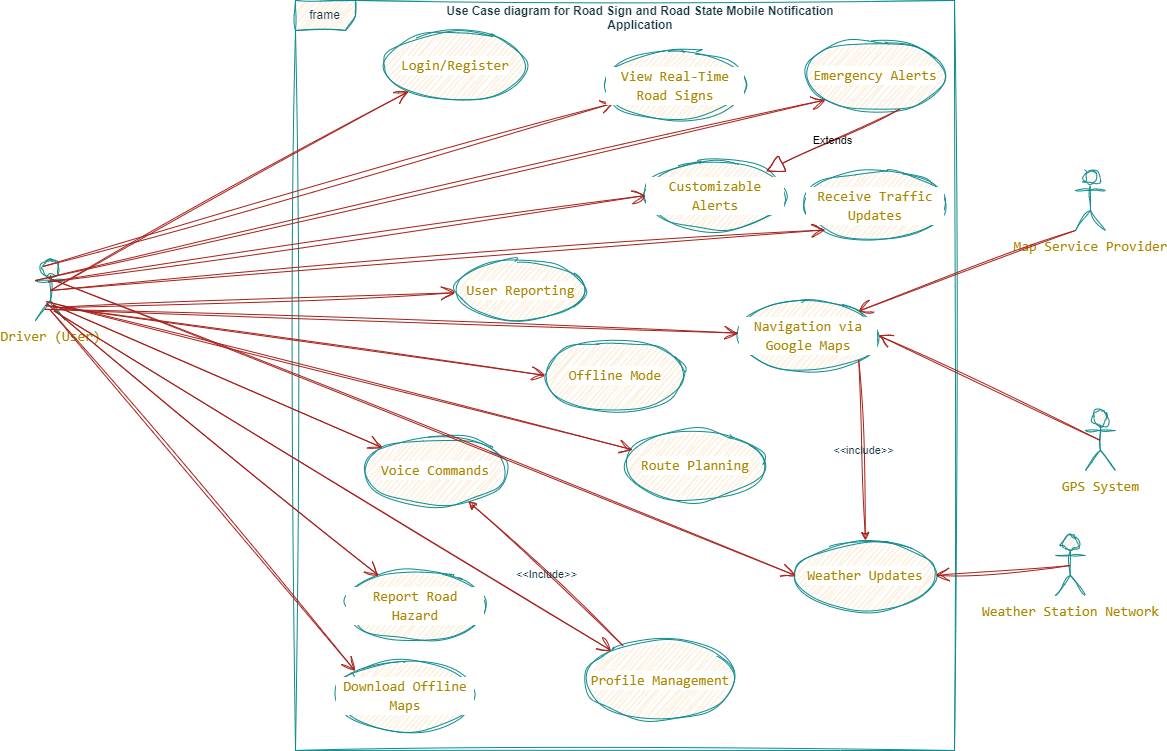
The GPS System, Map Service Provider, and Weather Station Network provide data to the application:

- GPS System interacts with Navigation via Google Maps.

- Map Service Provider interacts with Navigation via Google Maps.

- Weather Station Network interacts with Weather Updates.

## Visual Representation:



## Use Cases

**1. Login/Register:** - Allows the user to log in or register to the application.

**2. View Real-Time Road Signs**: - Displays real-time road signs to the user.

**3. Receive Traffic Updates:** - Shows current traffic conditions.

**4. Navigation via Google Maps:** - Provides navigation using Google Maps.

**5. User Reporting:** - Enables users to report road hazards.

**6. Route Planning:** - Plans routes considering current road conditions.

**7. Weather Updates:** - Displays current weather conditions and forecasts.

**8. Profile Management:** - Manages user profiles and preferences.

**9. Voice Commands:** - Allows interaction with the app via voice commands.

**10. Offline Mode:** - Provides access to offline maps and road sign recognition.

**11. Emergency Alerts:** - Sends alerts for emergencies.

**12. Customizable Alerts:** - Allows customization of the types of alerts received.

**13. Download Offline Maps:** - Enables users to download maps for offline use.

**14. Report Road Hazard:** - Allows users to report hazards on the road.

# 3. Sequence diagram

Dives into specific actions within use cases. **Zooms** in on **a specific use case** (e.g., navigation). llustrates the sequence of messages exchanged between app components to complete a usecase.

## Participants:

1. Driver: Represents the user interacting with the application.
2. Application: The mobile application itself.
3. ComputerVision: Module responsible for analyzing video frames for road signs and traffic lights.
4. MapService: Provides map data.
5. WeatherStation: Supplies real-time weather data.
6. GPS: Monitors the user's location.
7. AppBackend: Application server responsible for processing reports and providing route information.

## Message Flow:

1. The Driver selects the language preference in the application.

2. The Driver signs up or logs into the application.

3. The Driver customizes preferences related to the application.

4. The Driver initiates navigation by entering a destination.

5. The Application checks for internet connectivity.

* If online:
* Downloads maps for offline use.
* Requests map data, monitors the user's location, requests weather data, and requests traffic data.
* If offline, uses pre-downloaded maps.

6. Real-time Sign Recognition:

* Application captures video frames, analyzes them for road signs.
* If signs are detected, their type and meaning are identified, and information is displayed to the Driver with text notifications and audio alerts if necessary.

7. Traffic Light Detection:

* Application captures video frames, analyzes them for traffic lights.
* If traffic lights are detected, their color and meaning are identified, and information is displayed to the Driver with text notifications and audio pop-up notifications for red lights.

8. User Reporting:

* The Driver reports hazards on the road.
* The Application sends report data to the server for processing and updates road state information.

9. Navigation with Route Planning:

* Application provides turn-by-turn instructions to the Driver with text and audio notifications.
* Requests map data for the route, calculates the route based on road conditions from the AppBackend, displays the route on the map, and provides real-time traffic and road sign information.

10. User Profile Management:

* The Driver manages their profile, views or edits preferences.
* Application displays profile information to the Driver with text notifications.

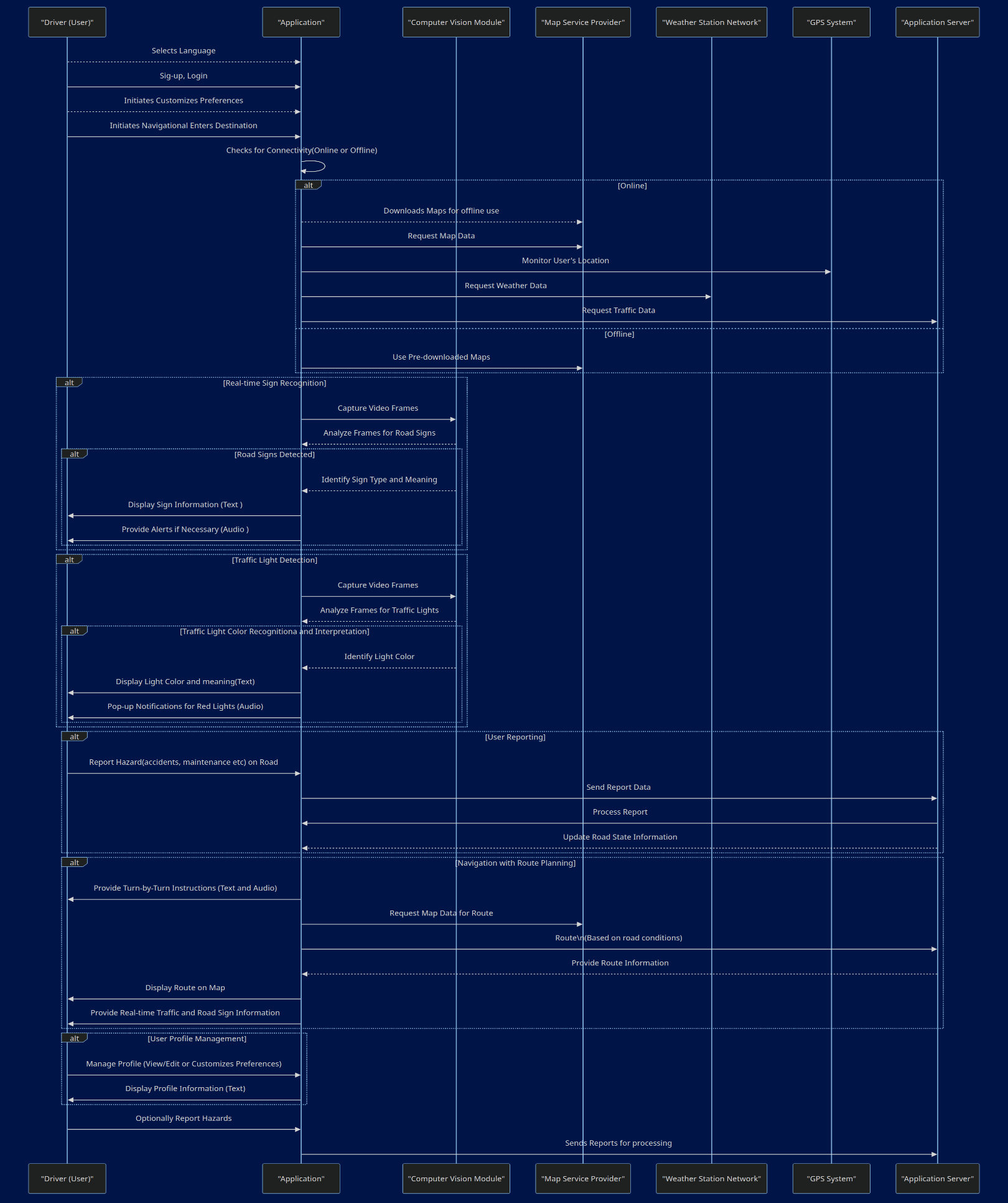
11. Optionally, the Driver can report hazards to the application.

12. The Application sends reports to the server for processing.

### NOTE

The synchronous messages (`->`) indicate communication where the sender waits for a response from the receiver before proceeding, ensuring a direct and immediate interaction. Asynchronous messages (`-->>`) indicate communication where the sender does not wait for an immediate response and can continue with other activities without blocking.

## Visual Representation



# 4. Class diagram

Defines the system building blocks internal components (classes) with their properties (data) and functionalities (methods). Can be developed alongside the Use Case Diagram as they inform each other.

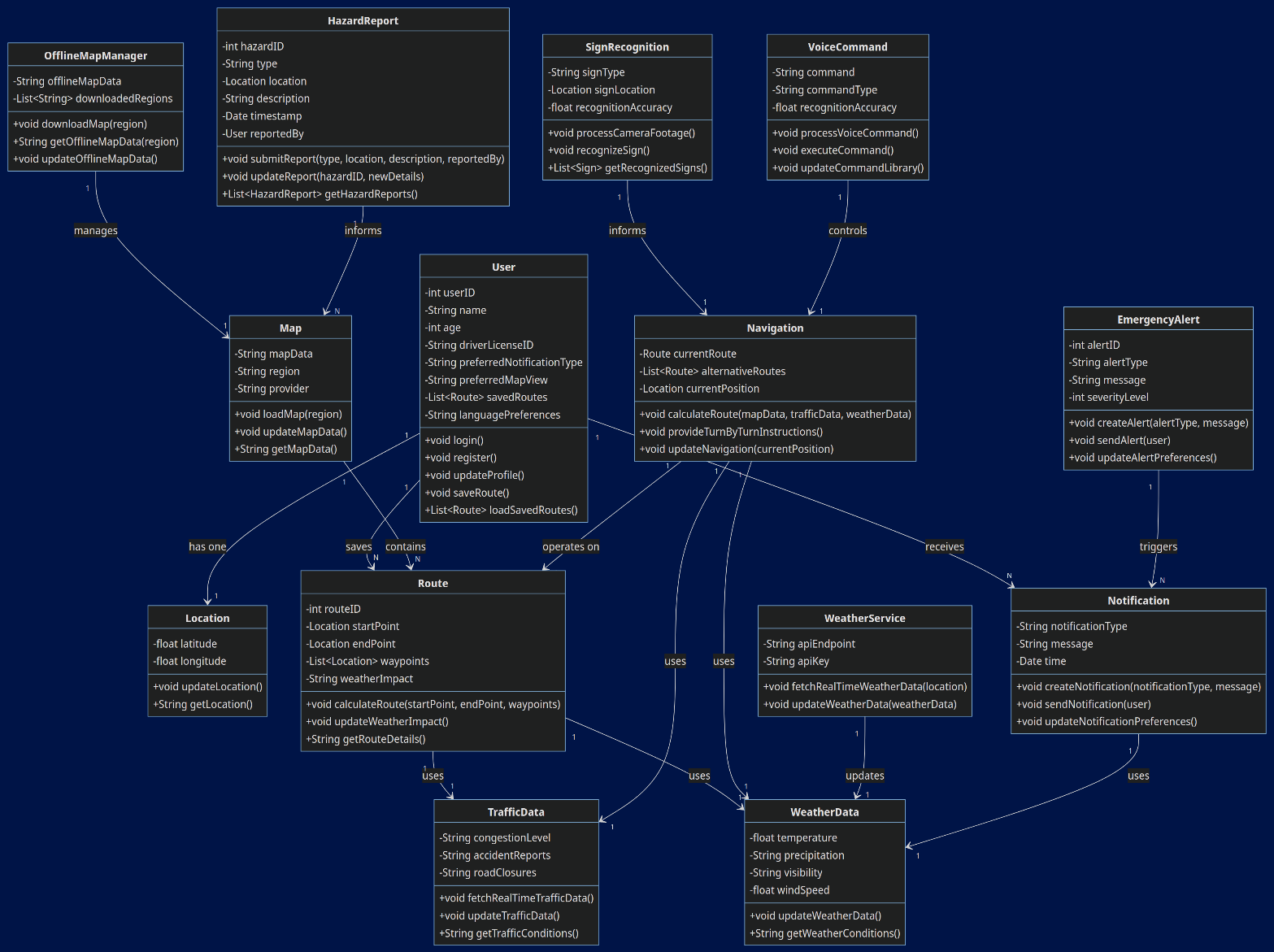
**Here is a simple list of the classes:**

1. User 2. Location 3. Map 4. Route 5. TrafficData 6. WeatherData

7. WeatherService 8. SignRecognition 9. Navigation 10. Notification

11. OfflineMapManager 12. HazardReport 13. VoiceCommand 14. EmergencyAlert

## Visual Representation



## Classes and their Attributes and Methods WITH EXPLANATION:

### 1. User:

**Attributes**: `userID`, `name`, `age`, `driverLicenseID`, `preferredNotificationType`, `preferredMapView`, `savedRoutes`, `languagePreferences` (all private).

**Methods**: `login()`, `register()`, `updateProfile()`, `saveRoute()`, `loadSavedRoutes()`.

**Explanation**: User attributes are private to ensure the confidentiality and integrity of user information. Methods to interact with user data are public.

### 2. Location:

**Attributes**: `latitude`, `longitude` (both private).

**Methods**: `updateLocation()`, `getLocation()`.

**Explanation**: Location details are private to prevent unauthorized access. Methods provide controlled access to update and retrieve location data.

### 3. Map:

**Attributes**: `mapData`, `region`, `provider` (all private).

**Methods**: `loadMap(region)`, `updateMapData()`, `getMapData()`.

**Explanation**: Map details are private to protect the integrity of map data. Methods allow controlled loading and updating of map data.

### 4.Route:

**Attributes**: `routeID`, `startPoint`, `endPoint`, `waypoints`, `weatherImpact` (all private).

**Methods**:`calculateRoute(startPoint, endPoint, waypoints)`, `updateWeatherImpact()`, `getRouteDetails()`.

**Explanation**: Route details are private to ensure accurate and secure route information. Public methods allow route calculation and retrieval.

### 5. TrafficData:

**Attributes**: `congestionLevel`, `accidentReports`, `roadClosures` (all private).

**Methods**: `fetchRealTimeTrafficData()`, `updateTrafficData()`, `getTrafficConditions()`.

**Explanation**: Traffic data attributes are private to maintain data integrity and control over updates. Methods allow fetching and updating traffic conditions.

### 6. WeatherData:

**Attributes**: `temperature`, `precipitation`, `visibility`, `windSpeed` (all private).

**Methods**: `updateWeatherData()`, `getWeatherConditions()`.

**Explanation**: Weather data attributes are private to ensure the accuracy and security of weather information. Methods allow updating and retrieving weather conditions.

### 7. WeatherService:

**Attributes**: `apiEndpoint`, `apiKey` (both private).

**Methods**: `fetchRealTimeWeatherData(location)`, `updateWeatherData(weatherData)`.

**Explanation**: API details are private to secure the API access and prevent misuse. Methods enable fetching and updating weather data.

### 8. SignRecognition:

**Attributes**: `signType`, `signLocation`, `recognitionAccuracy` (all private).

**Methods**: `processCameraFootage()`, `recognizeSign()`, `getRecognizedSigns()`.

**Explanation**: Recognition data is private to ensure accuracy and prevent tampering. Methods provide the functionality for processing and recognizing signs.

### 9. Navigation:

**Attributes**: `currentRoute`, `alternativeRoutes`, `currentPosition` (all private).

**Methods**: `calculateRoute(mapData, trafficData, weatherData)`, `provideTurnByTurnInstructions()`, `updateNavigation(currentPosition)`.

**Explanation**: Navigation attributes are private to maintain the integrity of the navigation process. Methods enable route calculation and updates.

### 10. Notification:

**Attributes**: `notificationType`, `message`, `time` (all private).

**Methods**:`createNotification(notificationType, message)`, `sendNotification(user)`, `updateNotificationPreferences()`.

**Explanation**: Notification attributes are private to ensure the correct delivery and privacy of notifications. Methods allow creating and sending notifications.

### 11. OfflineMapManager:

**Attributes**: `offlineMapData`, `downloadedRegions` (both private).

**Methods**: `downloadMap(region)`, `getOfflineMapData(region)`, `updateOfflineMapData()`.

**Explanation**: Offline map data is private to ensure it is accurately maintained. Methods allow downloading and updating offline maps.

### 12. HazardReport:

**Attributes**: `hazardID`, `type`, `location`, `description`, `timestamp`, `reportedBy` (all private).

**Methods**: `submitReport(type, location, description, reportedBy)`, `updateReport(hazardID, newDetails)`, `getHazardReports()`.

**Explanation**: Hazard report attributes are private to ensure the accuracy and privacy of reports. Methods allow submitting and updating reports.

#### 13. VoiceCommand:

**Attributes**: `command`, `commandType`, `recognitionAccuracy` (all private).

**Methods**: `processVoiceCommand()`, `executeCommand()`, `updateCommandLibrary()`.

**Explanation**: Voice command data is private to ensure command accuracy and prevent misuse. Methods provide voice command processing and execution.

### 14. EmergencyAlert:

**Attributes**: `alertID`, `alertType`, `message`, `severityLevel` (private).

**Methods**: `createAlert(alertType, message)`, `sendAlert(user)`, `updateAlertPreferences()`.

**Explanation**: Alert attributes are private to ensure accuracy and timely delivery. Methods allow creating and sending emergency alerts.

## Relationships:

- User **has** one Location.

- User can **receive** multiple Notifications.

- User can **save** multiple Routes.

- Map **contains** multiple Routes.

- Route **uses** TrafficData and WeatherData.

- Navigation **operates** on a Route and **uses** TrafficData and WeatherData.

- SignRecognition **informs** Navigation.

- OfflineMapManager **manages** Map.

- HazardReport **informs** multiple Maps.

- VoiceCommand **controls** Navigation.

- EmergencyAlert **triggers** multiple Notifications.

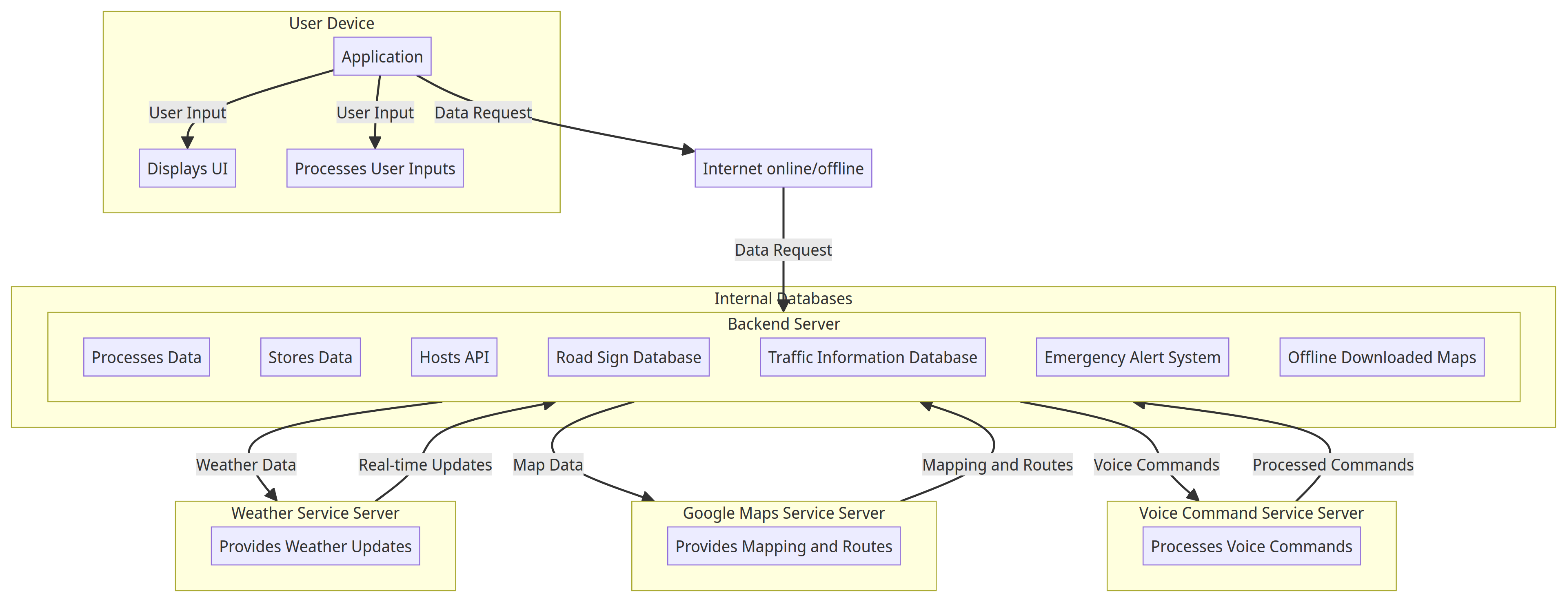
- WeatherService **updates** WeatherData.

- Notification **uses** WeatherData.

# 5. Deployment diagram

Shows the physical layout (can be created later). It models the physical aspects of an information system, illustrating how software components, hardware, and other devices interact and are arranged in the system. For Our System the deployment diagram is crucial for planning and maintaining a robust, scalable, and efficient system architecture. It ensures that all components are correctly integrated and function harmoniously to provide the desired services to users.

## Visual Representation



## Explanation

### 1. User Device (Client)

* Runs the application
* Displays the user interface (UI): This component is responsible for presenting information and options to the user.
* Processes user inputs: This component captures and processes the inputs from the user.

### 2. Internet (online/offline)

Acts as the communication medium between the User Device and the Backend Server and external entities.

### 3. Backend Server

**Processes and stores data**

* Processes Data: Handles the processing of data received from the user device.
* Stores Data: Manages the storage of data.
* Hosts API: Provides an API to communicate with the user device and other servers.
* Internal Databases: Contains:
* Road Sign Database: Stores road sign data.
* Traffic Information Database: Stores traffic light and sign information.
* Emergency Alert System: Sends emergency alerts.
* Offline Downloaded Maps: Stores maps for offline use.

### 4. Weather Service Server

Provides weather updates: Supplies real-time weather data to the backend server.

### 5. Google Maps Service Server

Provides mapping and routes: Supplies map data and route information to the backend server.

### 6. Voice Command Service Server

Processes voice commands: Receives and processes voice commands from the backend server and returns the results.

## Relationships and Communications

1. **User Device -> Internet**: The client application on the user device sends data requests through the internet.
2. Internet -> Backend Server: The internet relays data requests from the user device to the backend server.
3. Backend Server -> Road Sign Database (internal): The backend server uses the road sign database for storing and retrieving road sign data.
4. Backend Server -> Traffic Information Database (internal): The backend server uses the traffic information database for storing and retrieving traffic data.
5. Backend Server -> Emergency Alert System (internal): The backend server uses the emergency alert system to send alerts.
6. Backend Server -> Offline Downloaded Maps (internal): The backend server uses offline downloaded maps for navigation without an internet connection.
7. Backend Server -> Weather Service Server: The backend server retrieves real-time weather updates from the weather service server.
8. Backend Server -> Google Maps Service Server: The backend server retrieves map data and route information from Google Maps.
9. Backend Server -> Voice Command Service Server: The backend server sends voice commands to the voice command service server for processing.

# CONCLUSION

The system modeling and design for the road sign and road state mobile notification application demonstrate a comprehensive approach to enhancing driver safety. By integrating real-time data from various services and presenting it through an intuitive mobile interface, the application ensures that drivers receive timely and relevant information. The detailed diagrams—context, use case, sequence, class, and deployment—serve as a robust framework guiding the development process.

This project lays the groundwork for a practical and efficient application that not only aids in safe driving but also sets a standard for future innovations in road safety technology. The next phase involves implementing the design through UI development and integrating the modeled components to create a fully functional application that meets the needs of modern drivers.