THE UNIVERSITY OF BUEA
P.O.BOX 63,
Buea, South West Region Cameroon

REPUBLIC OF CAMEROON

CEF 440

Internet Programming and Mobile Programming

Department: Computer Engineering

Faculty Of Engineering and Technology

TASK 4

Road Sign and Road State Mobile Notification Application

SYSTEM MODELING AND DESIGN

Submitted to:

Dr. Nkemeni Valery

By:

Group 14

2023/2024

Contents

1.Introduction

System modeling and design is a crucial process in software development. It involves creating a blueprint for the system, outlining its functionalities, structure, and interactions with the external world.

1.1. System Modeling:

System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.

System modeling may represent a system using graphical notation, e.g., the Unified Modeling Language (UML).

System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers. It also provides the developer and the customer with the means to assess quality once the software is built.

The models built for the system include:

- Use Case Diagram: This provides a high-level overview of the system's functionality and how users interact with it.
- Context Diagram: This depicts the system's boundaries and its interaction with external entities (like traffic data providers).
- Class Diagram: This details the system's internal structure, showing the classes, their attributes, operations, and relationships.
- Sequence Diagram: This can be included to illustrate a specific interaction scenario between objects in the system, focusing on the message flow.
- Deployment Diagram: This depicts the physical deployment of the system components across different hardware or software environments.

1.2. System Design:

Translating Models into Implementation: Based on the system models, this phase focuses on how the system will be built. It involves:

- Choosing Technologies: Selecting programming languages, frameworks, and tools suitable for developing the system.
- Designing System Architecture: Defining the overall structure of the system, including components, their interactions, and data communication.

1.3. Need for System Modeling and Design:

- Improved Communication: Models provide a common language for stakeholders (developers, clients) to understand the system.
- Early Detection of Issues: Identifying potential problems early in the development cycle saves time and resources.
- Clearer Road Map: A well-defined system design guides development efforts and ensures everyone is on the same page.

2.Modelling the system

2.1. Use Case Diagram

The Use Case Diagram provides a visual representation of how our users interact with the system. It serves as a blueprint for understanding the functional requirements of the system from the user's perspective, aiding in the communication between stakeholders and guiding the development process.

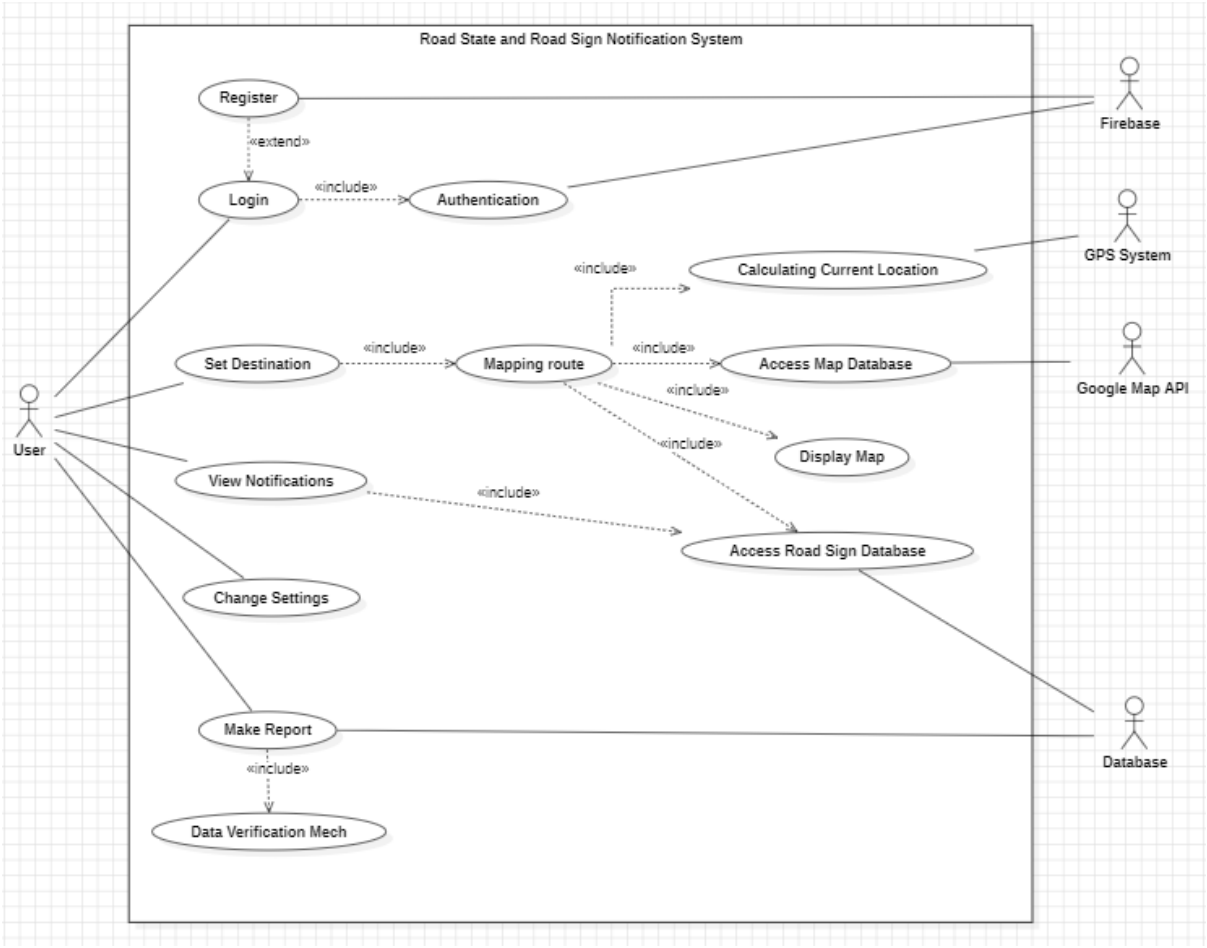
Login:

- Login into the system may require the user to register if not already a user of the system.
- Login into the system automatically triggers the authentication case which checks to see if the user is a valid user of the system or not.

Set Destination: When the user inputs his set destination, the system starts mapping out a route.

- The GPS location of the user is determined, and the route is mapped out.
- This mapped out route is then overlayed with the necessary road states and road sign fetched from the database.

- The end result is then displayed to the user.



View Notification:

- The appropriate road states and road states and road signs are fetched from the database and displayed to the user.

Change Settings: The user can change the system settings to their desired preferences.

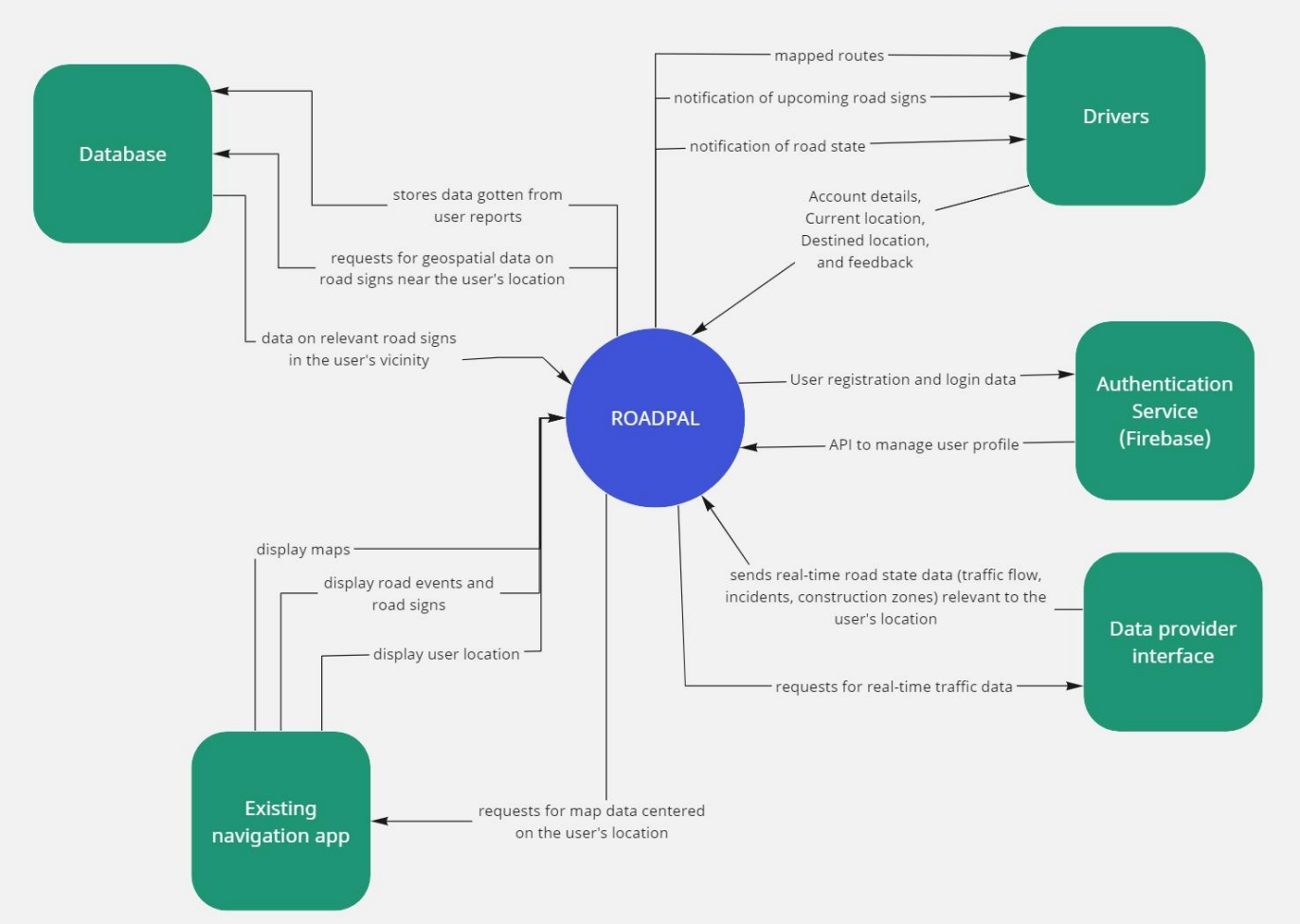
Make Report:

- When the user submits a report, the report is verified by the data verification mechanism.
- If the report is a valid one it is then stored in the database else, it is discarded.

2.2. Context Diagram

This context diagram illustrates the mobile application's interaction with external entities:

- **User:** Interacts with the mobile application to receive road state and road sign notifications.
- **Traffic Data Provider:** An external system that provides real-time traffic flow, incident, and construction zone data. This is a crowd-sourced traffic data interface.
- **Map Service:** An external system (e.g., Google Maps) that provides base maps and functionalities to display the user's location and road events.
- **Road Sign Database:** External database containing geospatial information on road signs.
- **User Authentication service:** Manages the registration, login, and user profiles.



User <=> Mobile App:

- **Input:** User interacts with the application (e.g., sets preferences, provide account details, current location, final destination, feedback).
- **Output:** Mobile App provides road state information (visual or audio alerts), upcoming road sign notifications, and map visualizations.

Mobile App <=> Data Provider Interface:

- **Input:** Mobile App sends requests for real-time traffic data (including user location).
- **Output:** Traffic Data Provider sends real-time road state data (traffic flow, incidents, construction zones) relevant to the user's location.

Mobile App <=> Map Service:

- **Input:** Mobile App sends requests for map data centered on the user's location.
- **Output:** Map Service sends map tiles, functionalities for displaying user location, road events, and potentially road signs.

Mobile App <=> Road Sign Database:

- **Input:** Mobile App sends requests for geospatial data on road signs near the user's location (if not using computer vision) and stores data provided by user reports.
- **Output:** Road Sign Database sends data on relevant road signs in the user's vicinity.

Mobile App <=> User authentication service:

- **Input:** Mobile App sends requests for API to manage user’s profile and validate registration and login.
- **Output:** User registration and login data

2.3. Class Diagram

Class diagrams are used to visually represent the structure and relationships of classes in a system.

The class diagram below explains the various relationships that exist between entities in our system:

User: This represents the drivers of our system.

- A user can register (first timers) or login (already have an account) into the system.
- A user can create one or multiple routes after inputting his/her current and final locations.
- A user can receive one or multiple notifications from the system.
- The user can set notification preferences.
- A user can give zero or more feedback to system on the accuracy of the given information.

Route: This the path the user will follow to arrive at his/her destination

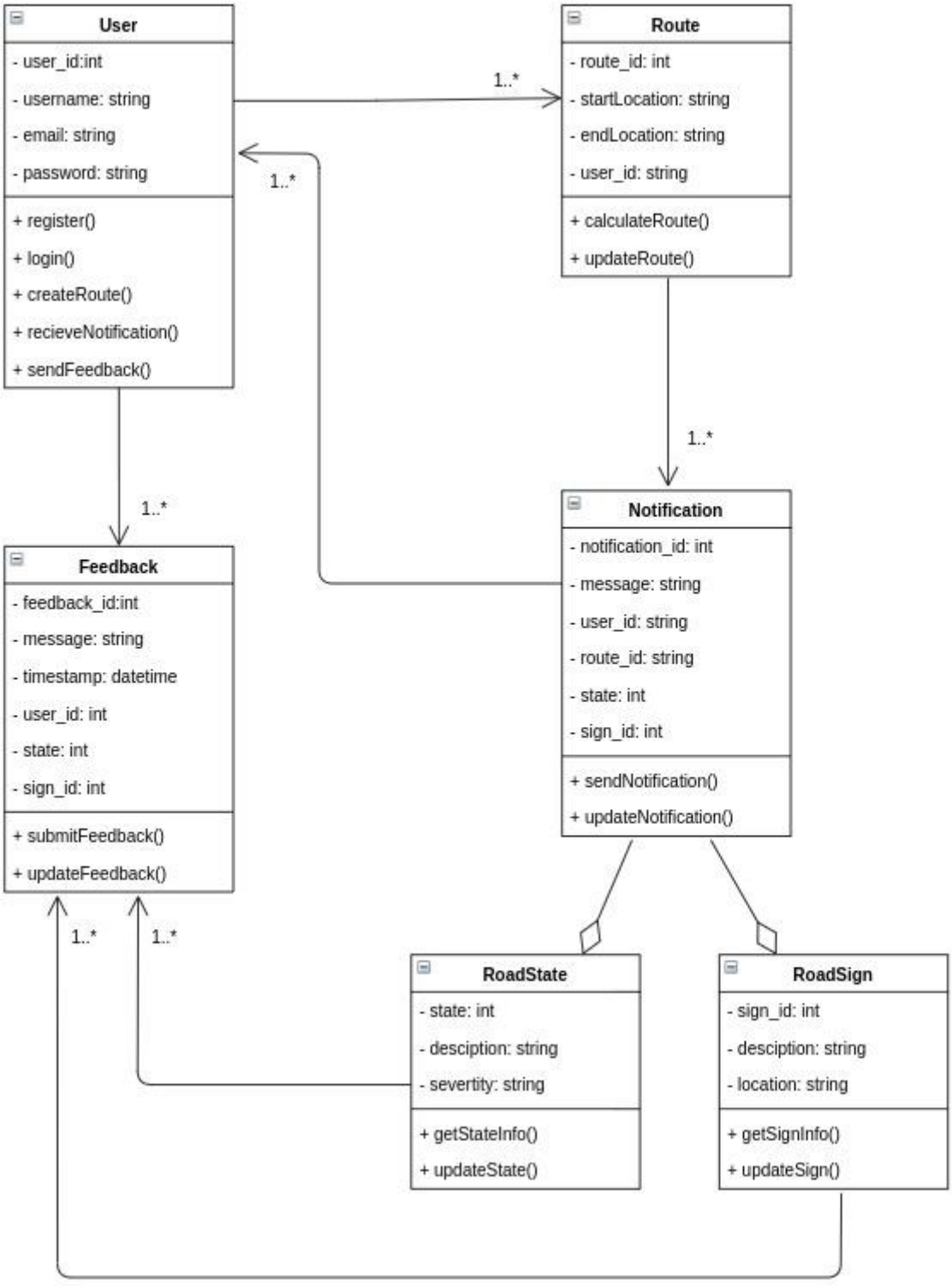
- The system can calculate the users route using his/her current location and final destination.
- The system can display multiple routes.
- The system can update routes depending on feedback from users.

Notification: These are messages that tell the user about the state of the road and approaching road signs.

- The system can send zero or more notifications.
- The system can update notifications.

Feedback: Messages given to the system concerning the accuracy of reported data

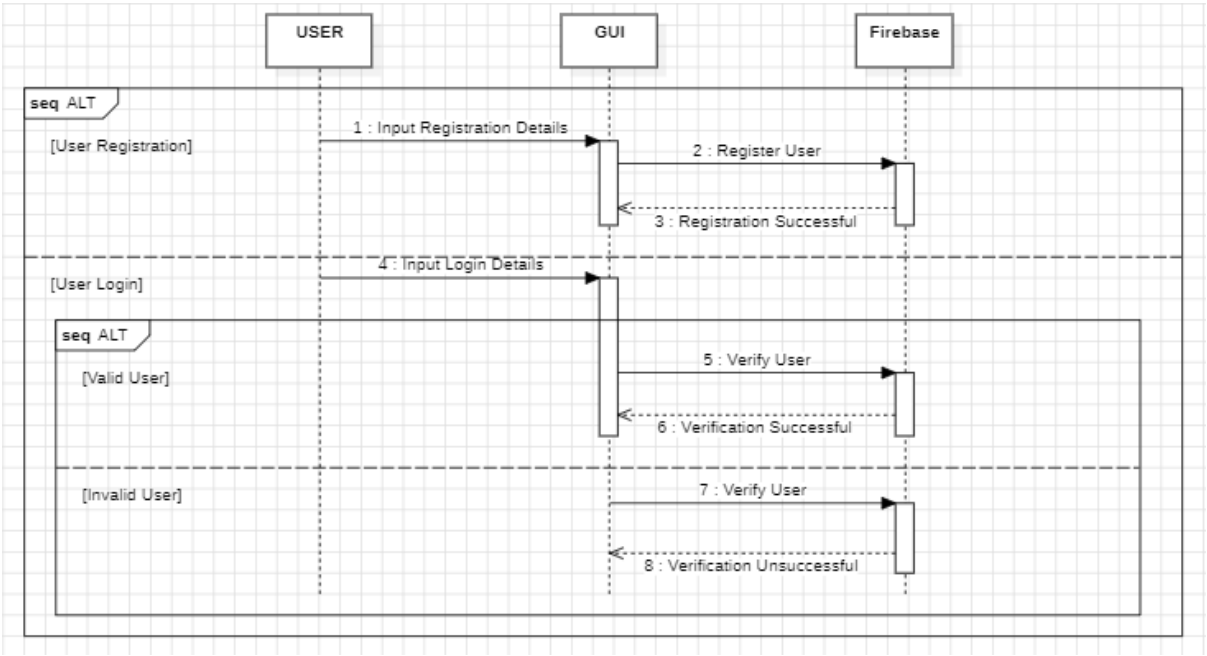
- The system can receive zero or multiple feedback from users.


User
- user_id:int
- username: string
- email: string
- password: string
+ register()
+ login()
+ createRoute()
+ recieveNotification()
+ sendFeedback()
Route
- route_id: int
- startLocation: string
- endLocation: string
- user_id: string
+ calculateRoute()
+ updateRoute()
1..*
1..*
1..*
1..*
Feedback
- feedback_id:int
- message: string
- timestamp: datetime
- user_id: int
- state: int
- sign_id: int
+ submitFeedback()
+ updateFeedback()
Notification
- notification_id: int
- message: string
- user_id: string
- route_id: string
- state: int
- sign_id: int
+ sendNotification()
+ updateNotification()
1..*
1..*
RoadState
- state: int
- desciption: string
- severtity: string
+ getStateInfo()
+ updateState()
RoadSign
- sign_id: int
- desciption: string
- location: string
+ getSignInfo()
+ updateSign()

2.4. Sequence Diagram

A sequence diagram is a Unified Modeling Language (UML) diagram that illustrates the sequence of messages between objects in an interaction.

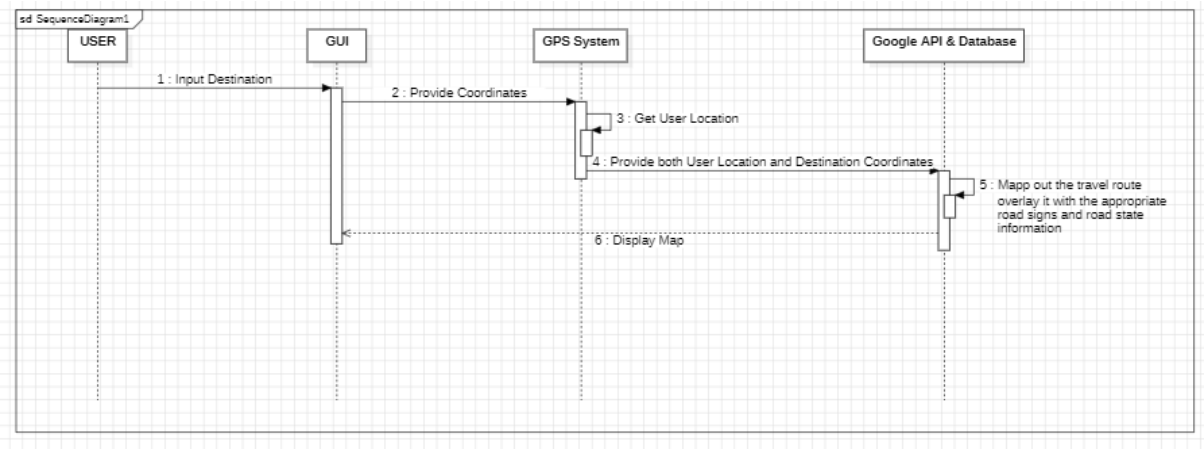
2.4.1. Login and Registration



Here the user has two options to either login or register.

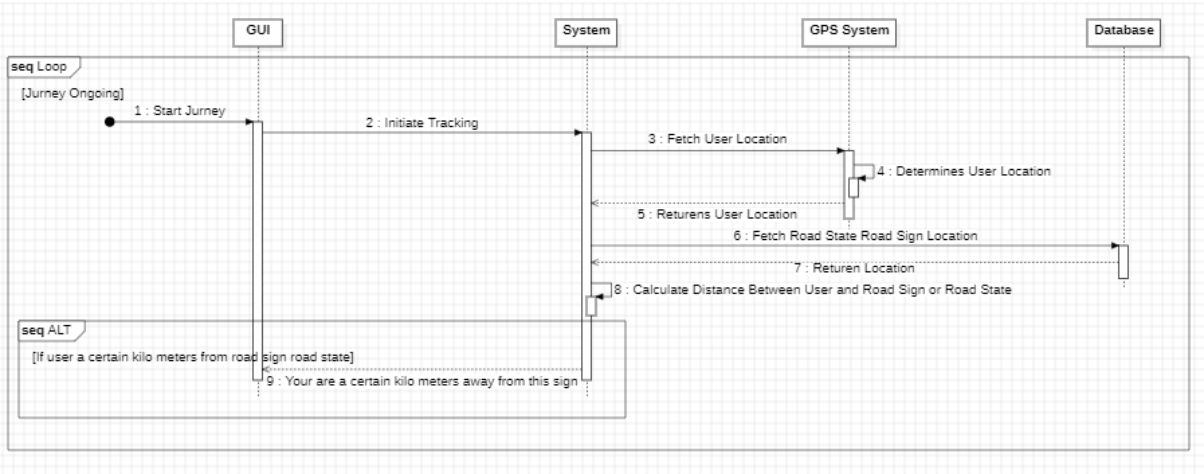
- During the registration process, the user fills in their details and it is stored in the database.
- During the login process, the user credentials are checked to see if the user is a valid user of the system. If he or she is a user of the system, they are granted access else, they are denied access, and an error message is printed on the screen.

2.4.2. Make Route



- The user provides his/her destination.
- The GPS system locates the user's location.
- These two coordinates are sent to the google API and a route is mapped out.
- The map is then overlayed with the appropriate road state and road sign information and displayed to the user.

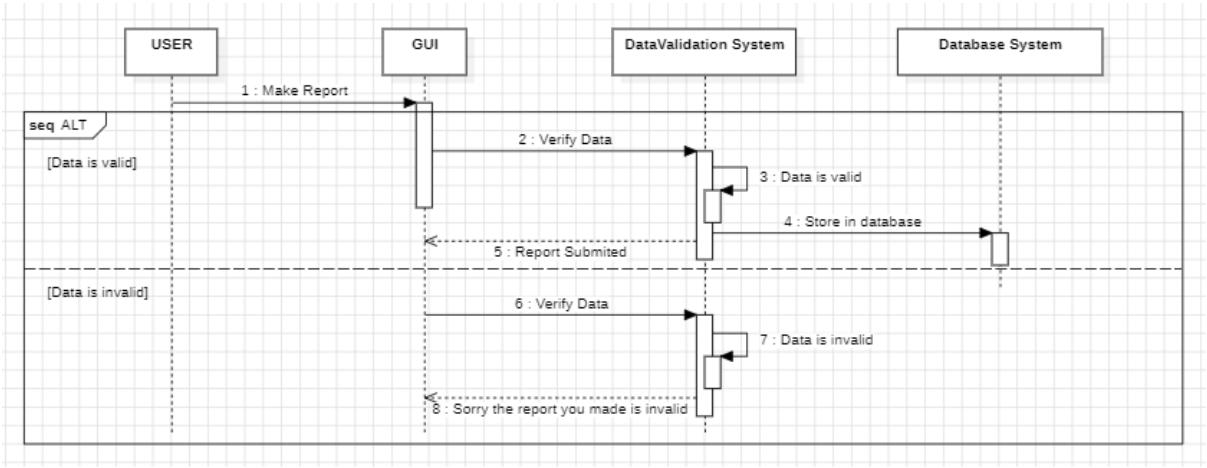
2.4.3. Road State and Road Signs Notification



- When the user starts the journey, the system initiates tracking.
- The system sends a fetch message to the devices GPS location system requesting for the user position.
- The system sends a fetch message to the data base requestion the road states and road sings as well as their location.

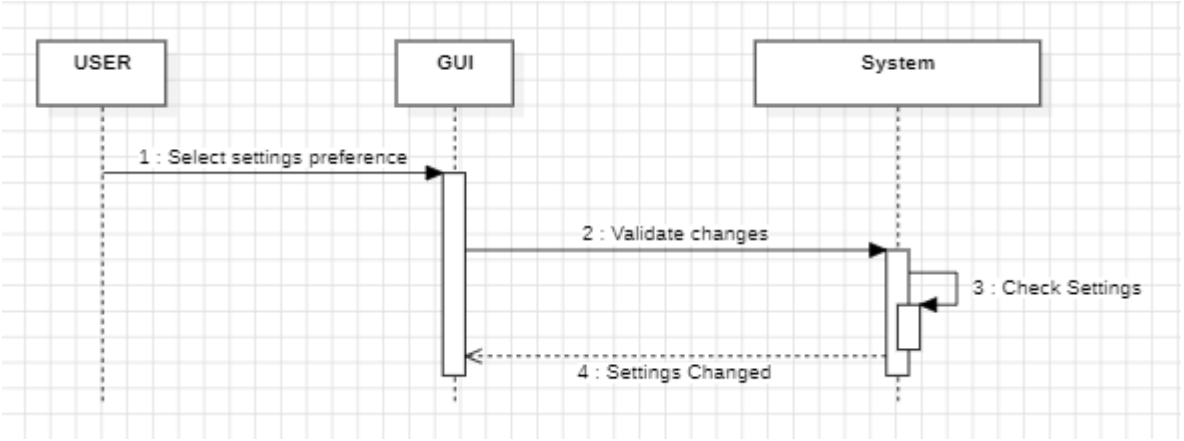
- The system then calculates the distance between the user and the road state or road sign.
- If the distance is equal to a specified value (distance for which the user is notified), the system returns a notification to the user.
- This process continues until the journey comes to an end.

2.4.4. Making a Report



- The user provides the system with the report.
- The Data validation system checks to see if the data is valid. If the data is valid, it is stored in the database and a report submitted message is displayed to the user else, the data is discarded, and a message is printed to the user saying invalid data provided.

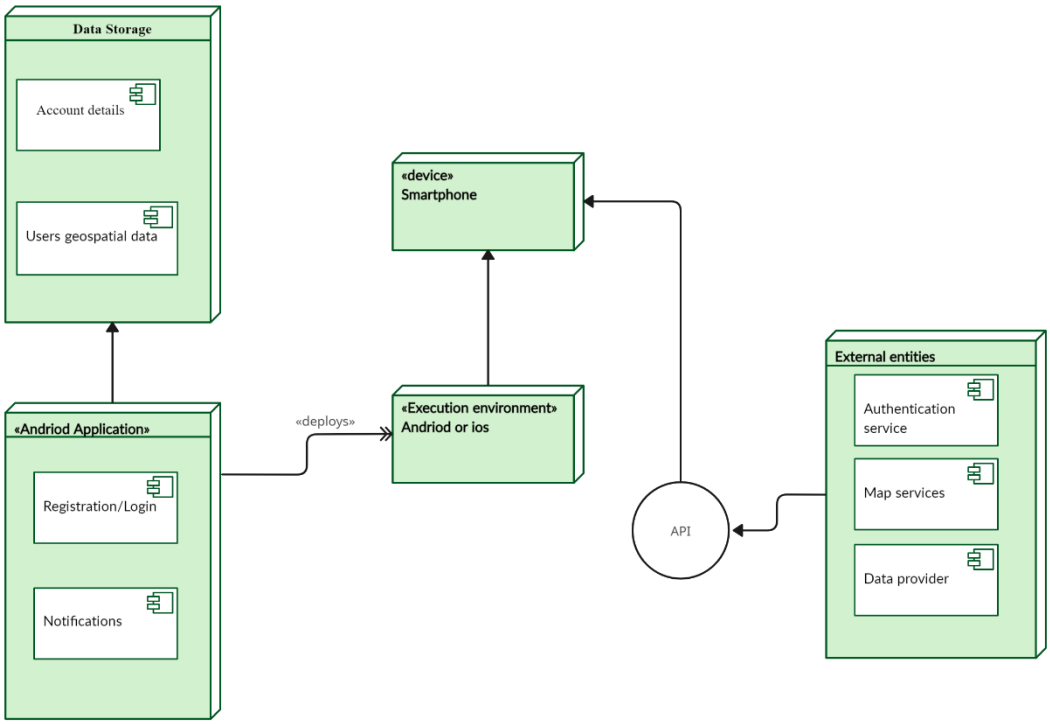
2.4.5. Changing Settings



- User changes settings.
- The system checks and validates the changes and a message is displayed to the user.

2.5. Deployment Diagram

A deployment diagram visually depicts the physical distribution of software components across different hardware or software environments. It shows how the various parts of a system are deployed onto real-world resources like servers, databases, and user devices.



This deployment diagram depicts the physical distribution of the Road State and Road Sign Mobile Notification App:

- **Device:** Represents the user with their device (laptop/phone) running the mobile application.
- **Mobile App:** The core application installed on the user's device.
- **External entities:** These are external services that the mobile app interacts with:
    - **Traffic Data Provider:** An external system (server) providing real-time traffic data through an API (e.g., government agency, crowd-sourced platform).
    - **Map Service:** An external system (server) offering map data and functionalities (e.g., Google Maps API).

    - **Authentication service:** Manages user profile.

- **Road Sign Database:** An external system (server) containing geospatial data on road signs and user account details.
- **Cloud (API):** Represents cloud servers hosting the APIs for the external systems.

Connections:

- **Device <-> Mobile App:** User interacts with the application on their device.
- **Device -> Traffic Data API:** Mobile app sends requests for real-time traffic data based on user location.
- **Device -> Map Service API:** Mobile app sends requests for map data (optional).
- **Mobile App -> Road Sign Database API:** Mobile app sends requests for road sign data.
- **Cloud -> (Data Flow) -> External Systems:** Data flows from the cloud servers to the device. The device leverages the API services.

3.Design of the system

3.1. Selecting appropriate Technologies

- **Frameworks:**
    - **Cross-Platform Development:** Frameworks like React Native or Flutter allow development using a single codebase for both iOS and Android. This can save time and resources but might have some limitations compared to native development.
    - Cross-platform frameworks like React Native or Flutter offer their own UI component libraries, reducing the need for platform-specific UI code.
    - Due to limited time and budget, a cross-platform framework will be used as this will allow for faster development across both platforms.

- **Location Services:**
    - **Technology:** The application will leverage the device's built-in GPS capabilities to determine the user's location.

- **Road State Data Acquisition:**
    - **Technology:** APIs provided by traffic management agencies or crowd-sourced traffic data platforms can be integrated to access real-time road state information.

      Example APIs:

    - Google Maps Platform offers traffic layer information.

    - Waze provides real-time traffic data through developer programs.

- **Road Sign Recognition:**

    - **Pre-Loaded Databases:** Alternatively, the app can utilize a pre-loaded database containing geospatial data on road signs, reducing reliance on processing power.

- **Notification System:**
    - **Technology:** The app can leverage the device's built-in notification framework to deliver visual and audio alerts to users.
- **Other Technologies:**
    - **Mapping Services:** Integration with existing mapping APIs (e.g., Google Maps) simplifies displaying the user's location, road events, and potentially road signs.
    - Utilizing existing APIs and leveraging device functionalities keeps development efficient.
    - **Data Storage:** A cloud-based database (e.g., MongoDB) can be used to store user preferences and potentially some cached traffic data.

3.2. Designing System Architecture

Every successful app has a good architecture that forms its structure and features. There are several architectural patterns that are used in mobile app development. It is essential to pick the most suitable architecture for our app to ensure its stability, performance, and quality.

The Model-View-Presenter (MVP) architectural pattern can be effectively applied to design our Road State and Road Sign Notification Mobile App.

Model:

- Encapsulates the data layer and business logic of the application.
- Responsibilities include:
    - Accessing real-time road state data through external APIs.
    - Processing and interpreting the acquired data.
    - Storing user preferences (potentially using the Data Access Layer).
    - Providing methods to retrieve relevant road state information and upcoming road signs based on user location and preferences.

View:

- Represents the user interface (UI) elements of the application.
- Responsibilities include:
    - Displaying the map view with user location and road events (traffic flow, incidents, construction zones).
    - Showing upcoming road sign notifications.
    - Presenting user settings for notification preferences and sign categories.
    - Sending user interactions (location updates, preference changes) to the Presenter.

- Updating the UI based on data received from the Presenter (new road events, sign notifications).

Presenter:

- Acts as the intermediary between the View and the Model.
- Responsibilities include:
  - Receiving user interaction events from the View.
  - Fetching data from the Model based on user location and preferences.
  - Processing the retrieved data and preparing it for presentation.
  - Calling appropriate methods on the View to update the UI with new road state information, upcoming signs, and notifications.

Benefits of using MVP:

- **Separation of Concerns:** Clear separation between UI, business logic, and data access promotes better code organization and maintainability.
- **Testability:** Each component (Model, View, Presenter) can be independently tested, simplifying the development process.
- **Loose Coupling:** The View and Model don't directly communicate, making the system more flexible and easier to modify.
- **Flexibility:** The Presenter can handle complex data processing and UI updates without affecting the View or Model.

Implementation:

1. The View (UI) interacts with the user and sends events (location updates, preference changes) to the Presenter.
2. The Presenter receives user events and retrieves data from the Model (road state information, upcoming signs based on location and preferences).

3. The Model performs data access (API calls, on-device recognition) and business logic (data processing).

4. The Presenter processes the retrieved data and prepares it for presentation (formatting, filtering).

5. The Presenter instructs the View to update the UI with the processed data (new road events, upcoming signs, notification triggers).

6. The View displays the updated information and generates notifications based on the Presenter's instructions.

4.Conclusion

This report has outlined the system modelling and design for a mobile application that enhances driver awareness of road conditions and signage. The application leverages real-time data and user location to provide timely notifications about traffic flow, incidents, construction zones, and relevant road signs.

The report presented a use case diagram depicting user interactions, a context diagram illustrating external system interactions, a class diagram detailing the application's internal structure, a sequence diagram (optional) for specific interaction scenarios, and a deployment diagram showcasing the physical distribution of system components. The MVP architectural pattern was proposed as a suitable approach for development, promoting separation of concerns and improved maintainability.

5.References

- https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/
- https://www.geeksforgeeks.org/what-is-system-design-learn-system-design/
- https://venngage.com/blog/context-diagram/
- https://medium.com/@brainbeanapps/designing-the-architecture-of-your-mobile-product-4-patterns-to-choose-among-d47b7d3c2e06