Republic of Cameroon
******
Peace - Work - Fatherland
*****
Minister of Higher Education
*****
University of Buea

Republique du Cameroun
******
Paix - Travail - Patrie
*****
Ministere de l'enseignement Superieur
*****
Universite de Buea

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

MOBILE APPLICATION DEVELOPMENT

CEF 440

TASK 4 REPORT:

ROAD SIGN AND ROAD STATE NOTIFICATION MOBILE APP

SYSTEM MODELLING AND DESIGN

Presented by:

GROUP 8

N°	NAMES	MATRICULES
1	FOWEDLUNG ATSAFAC AGAFINA	FE21A196
2	KAMDEM KAMGAING GILLES CHRISTIAN	FE21A209
3	NEGUE KWAHAM MAEL GRACE	FE21A252
4	NGUEPI GNETEDEM PATERSON	FE21A264
5	NOUPOUWO DONGMO STEPHANE MERCI	FE21A283

Course Instructor: Dr Valery Nkemeni

April 2024

TABLE OF CONTENTS

I- INTRODUCTION

System design and modelling are critical phases in the development of robust, efficient, and scalable systems. The modeling process involves defining the core entities, their attributes, and the relationships among them. System design is the process of defining the architecture, components, interfaces, and data for a system to satisfy specified requirements.

This document seeks to detail the design rationale, highlight the interactions between different components, and provide a clear roadmap for translating the conceptual model into a functional system. By presenting an organized and detailed model, this document serves as a critical reference for developers, stakeholders, and project managers involved in the system's development and deployment.

II- SYSTEM MODELLING

System modeling is a process used in systems engineering to create abstract representations of complex systems. These models were used to understand, analyze, and communicate the structure, behavior, and interactions of our system's components. For this system, UML (Unified Modelling Language) diagrams were used, which include;

II.1 CONTEXT DIAGRAM

A context diagram is a high-level view of a system. It's a basic sketch meant to define an entity based on its scope, boundaries, and relation to external components. The diagram below typically includes the main system, represented as a single process, and all the entities that interact with it. Arrows are used to depict the flow of data between these entities and the system, ensuring clarity on what information is exchanged and how the system interfaces with the outside world.

II.1.1 Identification of Entities

The different entities involved in this system are segmented into internal and external components as follows;

A. Internal Components:

1. **Road Sign and Road State Notification System:**
    - ***Core of the system*:** It manages the collection, processing, and dissemination of information related to road signs and road conditions.
2. **Drivers:**
    - ***Users of the system*:** They receive notifications about road signs and road conditions. They also send requests for route information and location updates.
3. **Road Sign Database:**
    - ***Repository*:** It stores data about road signs, including their positions and descriptions. The system queries this database to obtain road sign information.
4. **Data Providers:**
    - ***Sources of Information*:** These are users that supply updates on road signs and conditions. They send updates to the system, which validates and integrates this data.

B. External Components:

1. **Google API:**

   - ***Map Service*:** It provides map related services to the system such as route information, traffic, speed monitoring.

2. **Weather API:**

   - ***Weather Information:*** It provides weather related information which might affect road conditions.

II.1.2 The Context Diagram

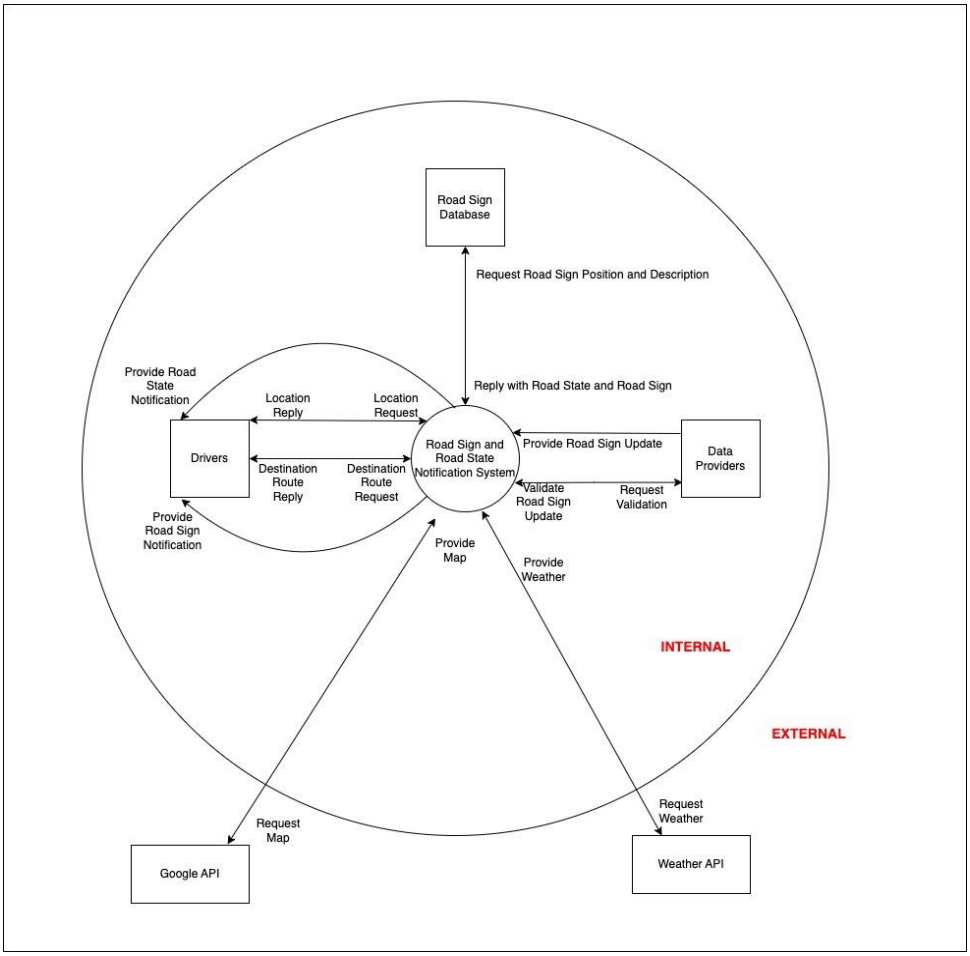

Figure 1 Context diagram

II.1.3 Interaction of the Entities

1. **Drivers:**
    - **Request Route and Location:**
        - ***Destination Route Request*:** Drivers request route information.
        - ***Location Request*:** Drivers request location-based updates.
    - **Receive Notifications:**
        - ***Receive Road State Notification*:** System sends notifications about road conditions to drivers such as floods, traffic.
        - ***Receive Road Sign Notification*:** System sends notifications about road signs to drivers.
    - **Reply:**
        - ***Location Reply*:** The system replies with the driver's current location.
        - ***Destination Route Reply:*** The system responds to route requests.
2. **Road Sign Database:**
    - ***Request Information*:** The system requests data from the database.
    - ***Reply*:** The database responds with the requested information.
3. **Data Providers:**
    - ***Send Updates*:** Data providers send updates on road signs.
    - ***Validation*:** The system requests validation from data providers.
4. **Google API:**
    - ***Map Requests*:** The system requests maps to provide drivers with route information.
    - ***Provide Map*:** Google API responds with the requested map data.
5. **Weather API:**
    - ***Weather Requests*:** The system requests weather data.
    - ***Provide Weather*:** Weather API responds with the requested weather information.

II.2 USE CASE DIAGRAM

A use case diagram is a visual representation of how users interact with the system to achieve a specific goal. It's like a blue print showing the functionalities of the system from user's perspectives. It also helps us understand the system behavior and clearly represents some functionalities that the user will be able to perform.

II.2.1 Identification of Actors

1. **User:** Represents both new and returning users who interact with the system to receive notifications and view information.
2. **Admin:** Manages and monitors the system for updates.
3. **External Systems:**
   - ***Road Sign Database*:** Stores data related to road signs and road state.
   - ***Google Map API*:** Provides map services.
   - ***Data Provider*:** Supplies and verifies updates on road conditions.

II.2.2 Use Cases

1. **Login:** The system verifies user credentials during login. If verification fails, the system displays a login error.
2. **Sign Up:** Allows new users to create an account in the system.
3. **Request for Destination Route:** The user provides his destination and request for a route. This use case also leads to the occurrence of the following use cases (includes):
   - ***Provide map with route*:** The google map API provides the user with a map and the route to his destination.
   - ***Provide current location*:** The google map API provides the user's current location within the map.
4. **View Map:** Allows users to view the provided maps. The map contains information about;
   - ***Traffics:*** The map can include traffic information.
   - ***Hazards:*** The map can include hazard information.
5. **Set Notification Preference:** This use case permits users to set their preferences.
6. **Display Road Sign:** The system displays the relevant road sign information. The road signs are displayed from the road sign database based on the user's preference.

7. **Display Road State:** The system displays the relevant road state information. The road states data are gotten from the data providers or weather provider, and displayed based on the user's preference. The road state data can either be;
    - ***Traffics*:** Traffic information is provided by google API.
    - ***Hazards:*** Hazards information are provided by data providers or weather API.
8. **Verify Trustfulness of Hazards:** Data providers validates the reliability of hazard notifications reported by other data providers.
9. **Monitor System for Updates:** The administrators monitor the system for updates to ensure it is functioning correctly and receiving accurate data.

II.2.2 The Use Case Diagram

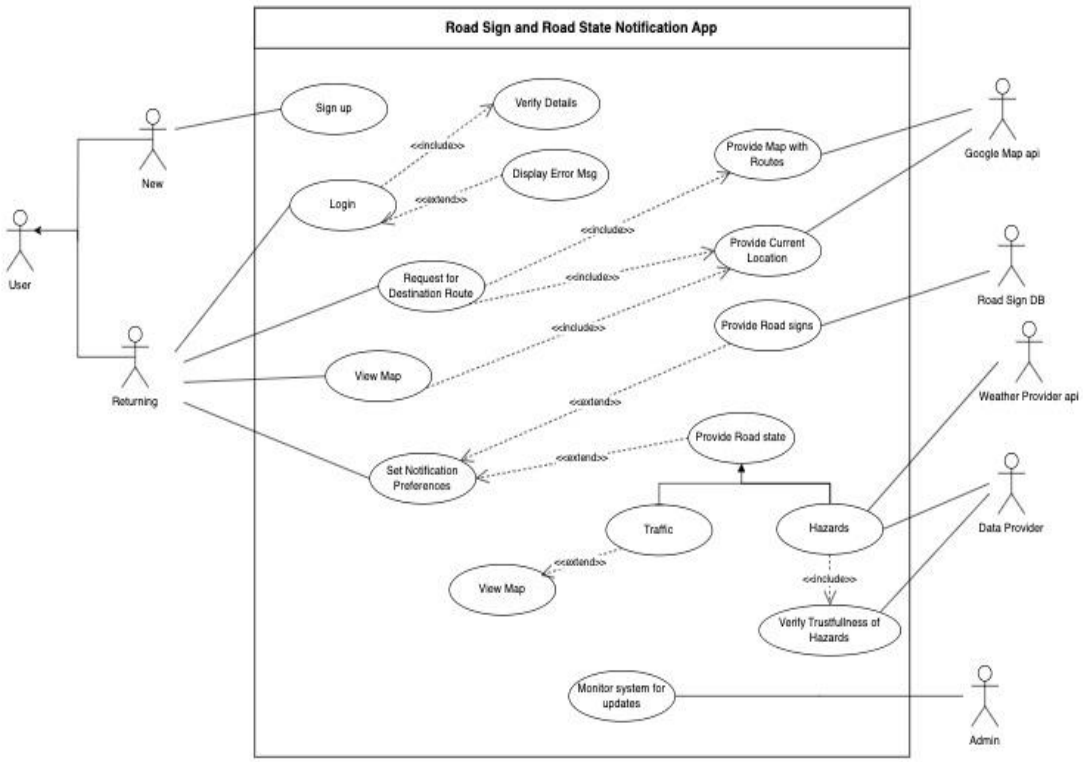

Figure 2 Use case diagram

II.3 SEQUENCE DIAGRAM

A sequence is a UML diagram that provides a visual representation of how objects in systems interact with each other over time. It shows the message exchange between objects, highlighting the sequence of interaction. This diagram helps us understand the system behavior and visualize interactions.

II.3.1 The Sequence Diagram

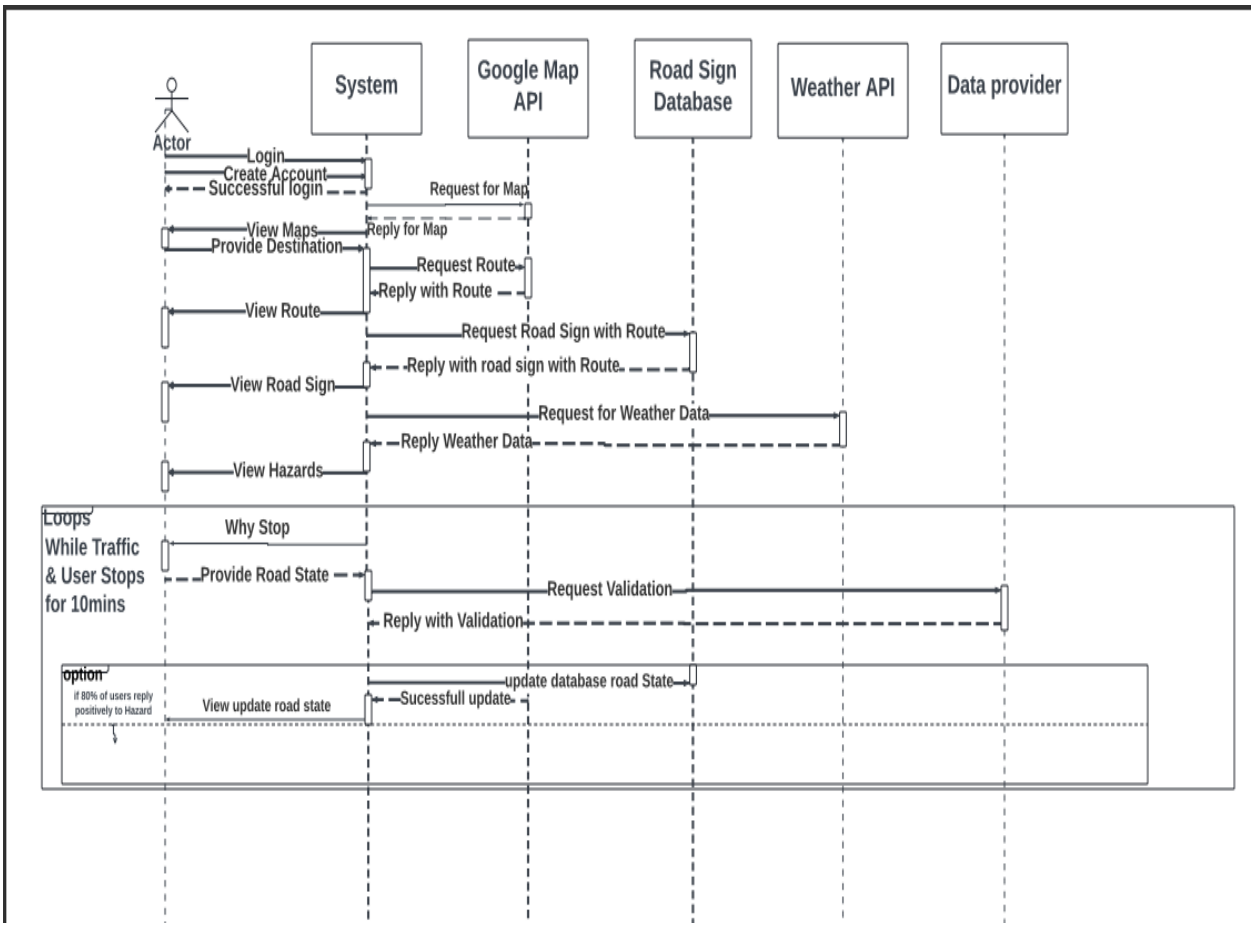

Figure 3 Sequence Diagram

II.3.3 Sequence of Interactions

1. **Login:** The user is authenticated into the system, then the system verifies the login details and grants access. If he does not have an account, he creates one, and he is automatically given access to his account.

2. **View Maps:** After the user is login, the system automatically requests map data from the Google Map API. The Google Map API replies with the requested map data, and the map data is shown to the user.

3. **Provide Destination:** The user provides a destination for route planning. This leads to the system requesting for route information from the Google Map API. The Google Map API replies with the route information and the route information is shown to the user.

4. **View Road Sign:** After the route is provided, the system requests for road signs information from the road sign database about the specific route. The road signs database replies with the road sign data for that specific route, and the information is notified to the user when needed.

5. **View Hazards:** The system requests weather data from the Weather API about a specific route. The Weather API replies with the weather data for that specific route, and the information is notified to the user when needed.

6. **While Traffic & User Stops for 10 mins (Loop):** The system continuously monitors the user's status and traffic conditions. When the system detects the user has stopped for 10 minutes, and there is traffic, it asks the user (data provider) about the road state. The data provider then replies with the road condition. The system requests for validation of road state information from other data providers in a specified perimeter. The other data providers reply with validation data.

    - **Option:** If greater than 80% of data provides reply positively to the hazard notification:

        - The system sends an update request to the database.
        - The database confirms the update.
        - Users in the specified perimeter are notified about the new road state update.

II.4 CLASS DIAGRAM

A class diagram is a blueprint that visually represents the building blocks of a system. It focuses on the static structure, showing classes and their attributes, operations and the relationship between classes.

II.4.1 Class Identification

- ***Map***: Responsible for displaying the map and containing elements like roads, buildings, water bodies, and landscapes.
- ***User*:** Handles user-specific functionalities like logging in, signing up, requesting routes, choosing notification preferences, and providing and validating road states.
- ***Itinerary***: Helps track user's movement from his current location to his destination.
- ***Road Sign and Road State***: Handle notifications and hold information about specific locations marked by latitude and longitude.

II.4.2 Relationships

- ***Map and User*:** There is a one-to-many (1...*) relationship between map and user, indicating that each map can have multiple users, but each user is associated with only one map.
- ***Map and Itinerary*:** There is a one-to-one (1...*) relationship between map and itinerary, indicating that a map can have multiple itineraries, but each itinerary is associated with a single map.
- ***Itinerary with Road Sign and Road State*:** Every road signs and road states is part of an itinerary but there can be an itinerary with neither road signs nor road states.

II.4.2 The Class Diagram

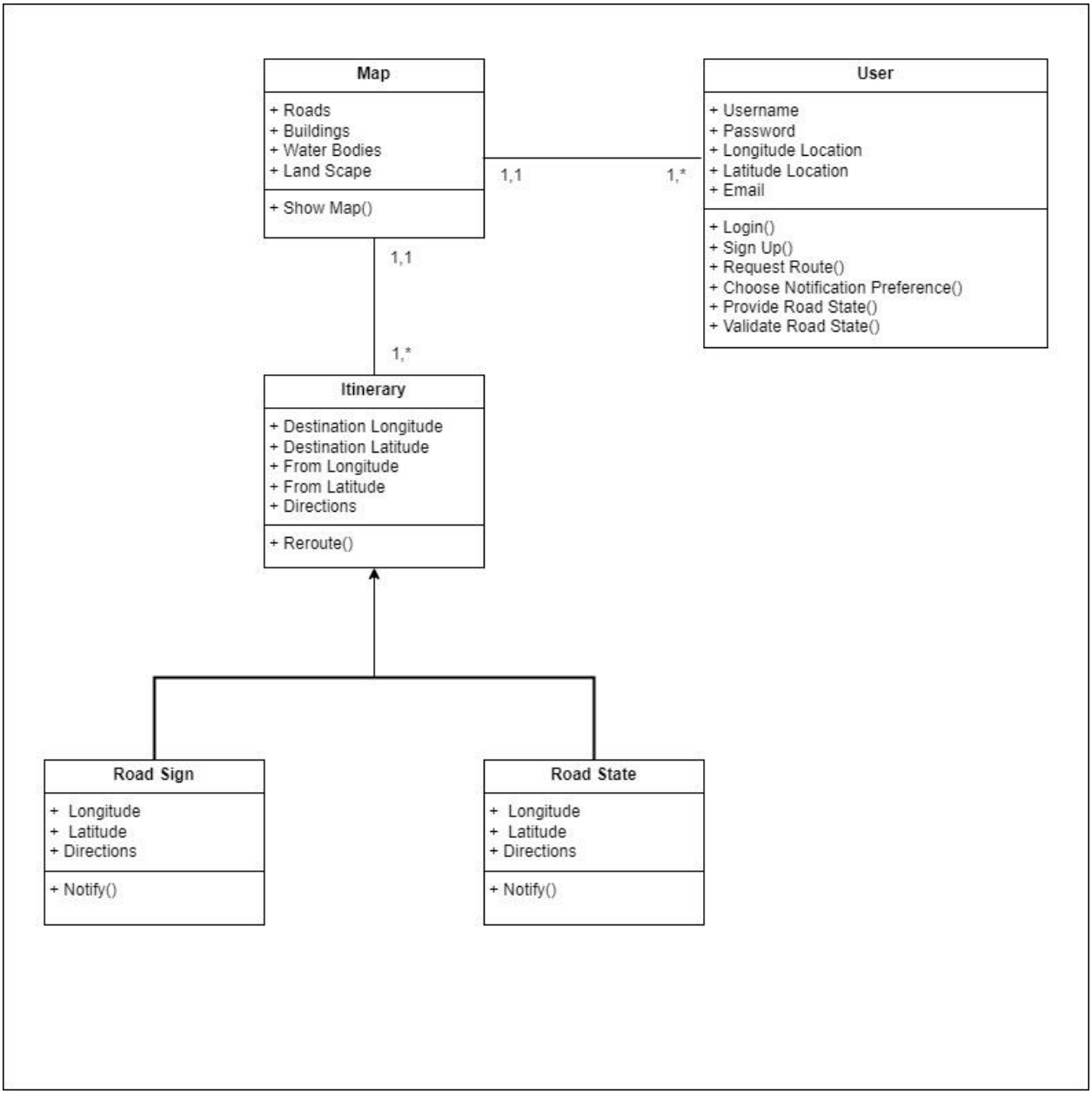

Figure 4 Class Diagram

II.5 DEPLOYMENT DIAGRAM

A deployment diagram is a way to visualise how the software and its components will be deployed physically on a system. It shows the hardware conponents (nodes) and software components(artifacts) that run on them, along with the connections between them. This helps to understand how the different parts work together.

II.5.1 Component Identification

1. Mobile Device:

- *User Interface*: The front end where users interact with the app.
- *App.js:* The client-side script that handles user requests and interactions, sending them to the application server via HTTP.

2. Application Server: The application server processes request from the mobile device and interacts with external servers (Google server and Weather provider server) via TCP/IP.

- *Backend Logic:* The core server-side functionality implemented in **server.js** and configured via **app.config**.
- *User Account DB*: A database for storing user account information.
- *System DB:* A database for storing system-related data such as road sign, road state information.

3. Google Server: The application server communicates with the google server to retrieve map data through the google map API.

- *Google Map API*: The interface provided by Google to access its mapping services.
- *Google DB:* The database used by Google to store and retrieve map-related data.

4. Weather Provider Server: The application server communicates with the weather provider server to retrieve weather data through the weather API.

- *Weather API*: The interface provided by the weather service to access weather data.
- *Weather DB*: The database used by the weather provider to store and retrieve weather information.

II.5.2 Interactions

- The Mobile Device sends user requests via HTTP to the Application Server.
- The Application Server handles these requests using its backend logic. It may need to query the User Account DB or System DB for certain operations.
- For map-related requests, the Application Server communicates with the Google Server via TCP/IP, utilizing the Google Map API to fetch the necessary data from the Google DB.
- For weather-related requests, the Application Server communicates with the Weather Provider Server via TCP/IP, utilizing the Weather API to fetch the necessary data from the Weather DB.
- Responses from the external servers are processed by the Application Server and sent back to the Mobile Device via HTTP.

II.5.3 The Deployment Diagram

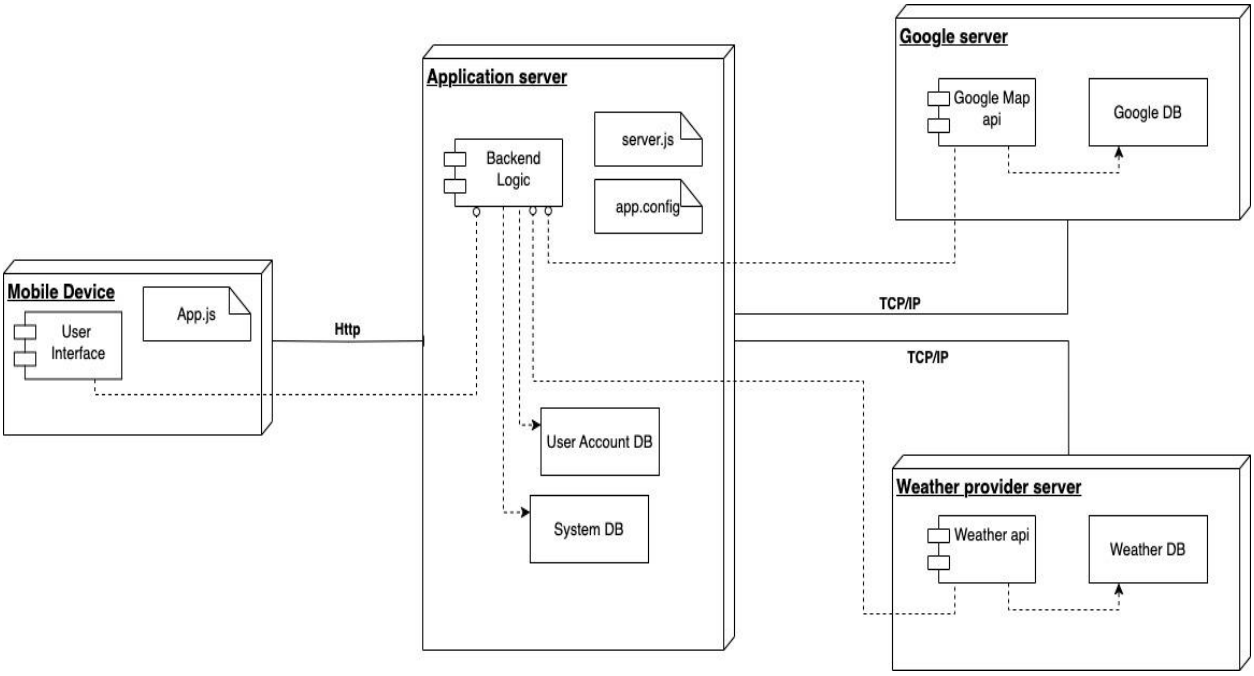

Figure 5 Deployment Diagram

II.6 ACTIVITY DIAGRAM

An activity diagram is a visual representation of a workflow or process. It shows the sequential steps involves, including decision points and potential alternative paths. Activity diagrams are used to model the dynamic aspects of a system, such as how it behaves in response to events.

II.6.1 User Activities

1. **Login:** The user starts by logging into the application.

2. **View Map:** After logging in, the user views the map.

3. **Set Destination for Route:** The user sets a destination for their route.

4. **Route Viewing or Motion:** The user either views the route they have set or remains in motion.

5. **View Routes:** If the user decides to view their routes, they proceed to view routes.

6. **Set Notification Preferences:** The user decides whether they want to set notification preferences. If they do, they move to the next step.

7. **Set Notifications Preferences:** The user sets their notification preferences.

8. **Provide Road Signs / Provide Road State:** The user can choose to either provide road signs or provide the road state. This step is broken down further:

   - *Provide Road Signs*: The user provides information about road signs.
   - *Select Road State*: The user selects the road state they want to be notified about.
   - *Provide Road State*: The user provides information about the state of the road.
   - *Select Road Signs*: The user selects road signs they want to be notified about.

9. **Select Road State:** The user makes the final selection of the road state.

10. **End:** The process ends.

Additional Details:

Loops:

- There is a loop for **while user in motion**, indicating continuous checks or updates while the user is on the move.
- There is also a loop if the user wants to set notification preferences again after viewing routes.

II.6.1 The Activity Diagram

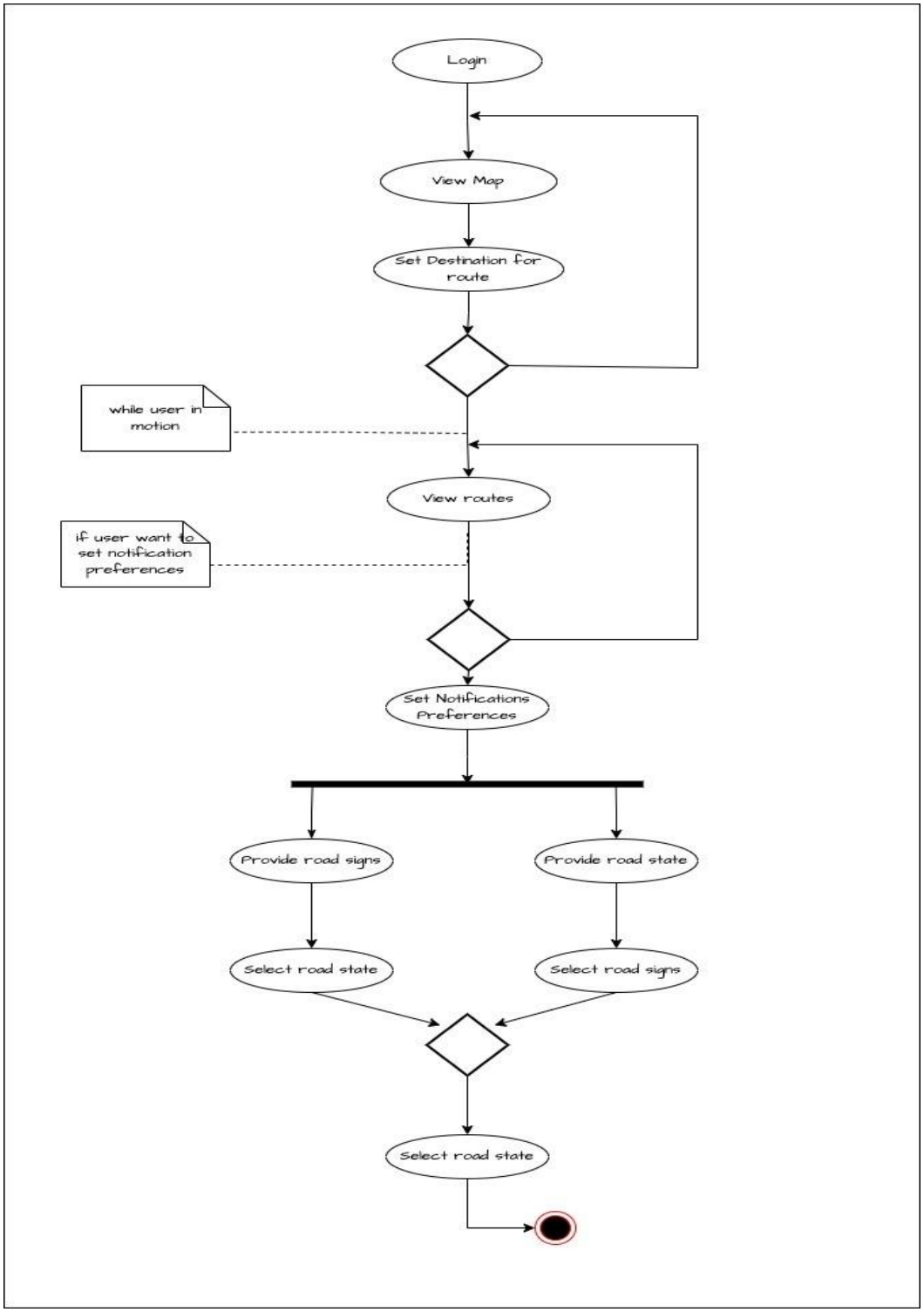

Figure 6 Activity Diagram

III- SYSTEM DESIGN

System design is the process of defining the architecture, components, interfaces, and data for a system to satisfy specified requirements. It is a blueprint for building a system that outlines how the system will function, its components, and the interaction between those components. System design focuses on creating a high-level structure that provides an effective and efficient solution to the problems identified during the system analysis phase.

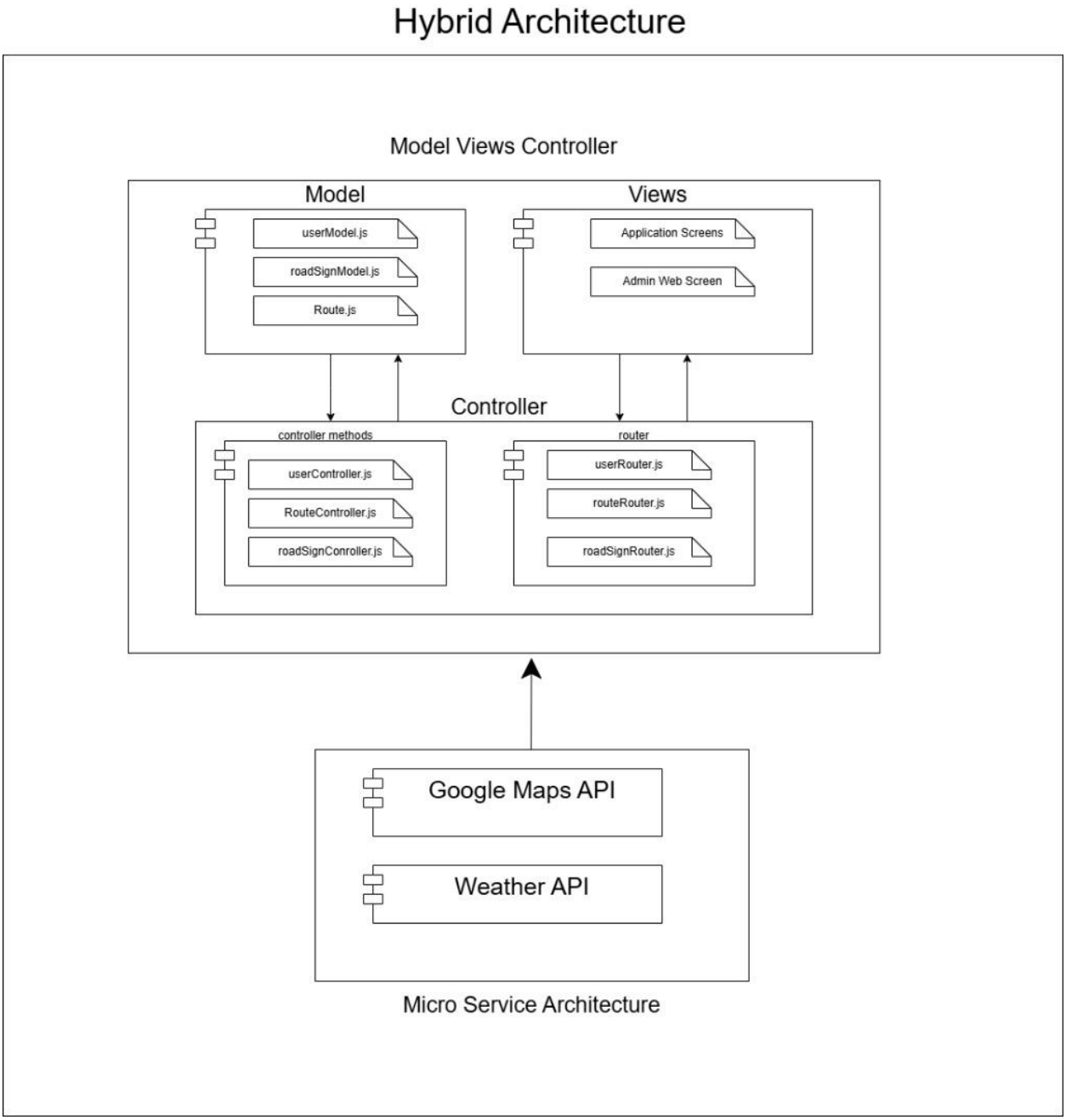

Figure 7 Hybrid Architecture

III.1 Explanation of the Hybrid Architecture Diagram

The figure above represents a hybrid architecture combining the Model-View-Controller (MVC) pattern and Microservice Architecture.

A. Model-View-Controller (MVC)

1. **Model**:

- *userModel.js:* This represents the user data and business logic associated with users.
- *roadSignModel.js*: This represents the road sign data and business logic associated with road signs.
- *Route.js:* This represents the route data and business logic associated with routes.

2. **View**:

- *Application Screens:* This is the user interface for end-users.
- *Admin Web Screen:* This is the user interface for administrative tasks.

3. **Controller:**

Controller Methods:

- *userController.js:* It manages the user-related operations and interactions between the user model and views.
- *RouteController.js:* It manages the route-related operations and interactions between the route model and views.
- *roadSignController.js*: It manages the road sign-related operations and interactions between the road sign model and views.

Router:

- *userRouter.js*: This defines the routes/endpoints for user-related API calls.
- *routeRouter.js:* This defines the routes/endpoints for route-related API calls.
- *roadSignRouter.js*: This defines the routes/endpoints for road sign-related API calls.

B. Microservice Architecture

- *Google Maps API:* A microservice responsible for providing map-related functionalities.
- *Weather API*: A microservice responsible for providing weather-related data.

C. Interaction Flow

1. Model Layer: It contains the core data and business logic of the application. Each model file represents a specific part of the application's data structure.

2. View Layer: It represents the user interface elements that the end-users interact with. It includes both application screens and admin web screens.

3. Controller Layer:

- Acts as an intermediary between models and views. It handles user input, processes it using the appropriate models, and then updates the views accordingly.
- The routers within the controller layer define the API endpoints that the application will respond to.

D. Microservice Interaction

- The controllers can interact with external microservices like Google Maps API and Weather API to fetch relevant data (e.g., map data, weather information).
- These external APIs provide specialized functionalities that are integrated into the main application through the controller logic.

CONCLUSION

System modeling and design are critical processes in the development of complex systems, ensuring that all aspects of the system are well-understood, properly planned, and effectively implemented. By utilizing structured approaches and standardized tools like UML (Unified Modelling Language), system designers can create clear, comprehensive representations of both the system's architecture and its behavior.

REFERENCES

[1] https://venngage.com/blog/context-diagram/ (Visited on the 05/25/2024)

[2] https://www.geeksforgeeks.org/deployment-diagram-unified-modeling-languageuml/

(Visited on the 05/27/2024)

[3] https://www.lucidchart.com/pages/uml-deployment-diagram (Visited on the 05/27/2024)