#### 

**UNIVERSITY OF BUEA REPUBLIC OF CAMEROON**

Buea, South West Region PEACE-WORK-FATHERLAND

Cameroon

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**DEPARTMENT OF COMPUTER ENGINEERING**

**Task 4: System Modelling and Design**

**Course Master: Dr Valery Nkemeni**

**CEF440:Internet Programming and Mobile Programming**

**GROUP 22**

|  |  |
| --- | --- |
| **NAMES** | **MATRICULE** |
| NGOUH KAMBI MARCBRYAN | FE22A264 |
| AMBASSA LISE-ASTRID | FE22A146 |
| EPIE-NGOLE MCCAULEY | FE22A445 |
| EPANDA RICHARD JUNIOR | FE22A206 |
| NKIANGWA NKEGOA KAREL | FE22A277 |

Table of Contents

**Introduction……………………………………………………………………………3**

1. **Context Diagram………………...………………………………………………….3**
2. **Dataflow Diagram………………………………………………………………….4**
3. **Use case Diagram………………..…………………………………………………6**
4. **Sequence Diagram………………………………………………………………….8**
5. **Class Diagram…….……………………………………………………………….13**
6. **Deployment Diagram...……………………………………………………………17**

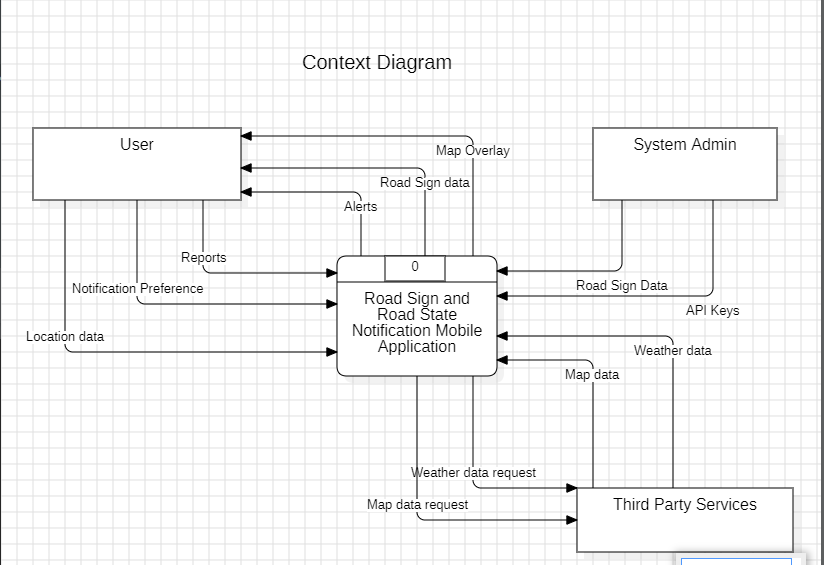
**Introduction**

This report outlines the system modelling phase of the Road Sign and Road State Notifications Mobile App, designed to improve driver safety through real-time alerts and road sign education. The app integrates crowdsourced reports, third-party APIs (e.g., Google Maps, OpenWeatherMap), and Firebase for backend operations.

1. **Context Diagram:**

A **context diagram**,focuses on the high-level interactions between a system and its external entities. It provides an overview of the system's boundaries and its relationships with users, admins, and third-party services.

This diagram effectively outlines the interactions and data exchanges among users, admins, and external services, illustrating the application’s functionality and scope.



**Entities**:

* **User**: Interacts with the application, specifying preferences, receiving alerts, and viewing reports.
* **System Admin**: Manages the application, likely responsible for overseeing data and system functions.
* **Third Party Services**: Provides external data, such as weather information and road sign data.

**System**:

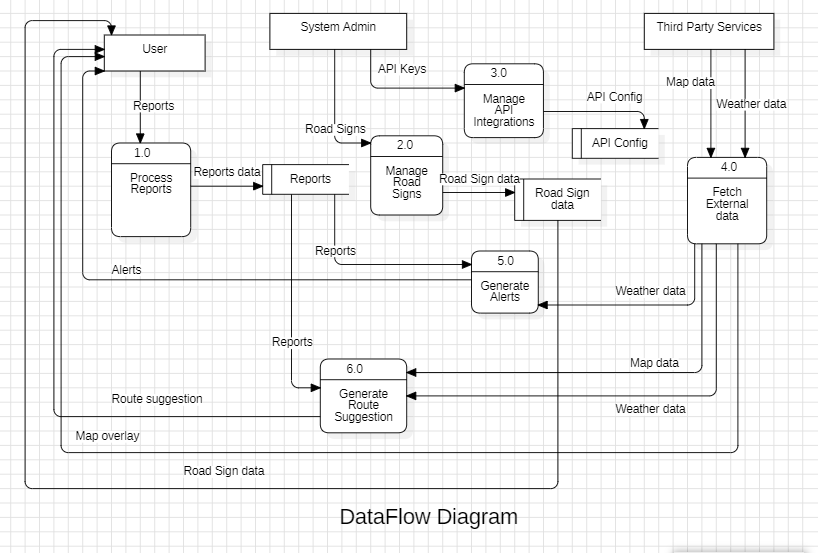
* **Road Sign and Road State Notification Mobile Application**: The central system that connects users, admins, and third-party services.

**Data Flows**:

* **User to Application**: Users send location data and notification preferences to the application.
* **Application to User**: The application sends alerts and reports back to the user.
* **Admin to Application**: The system admin likely updates or manages API keys and oversees application data.
* **Application to Third Party Services**: The application requests weather and map data from external services.
* **Third Party Services to Application**: These services provide the requested data back to the application.

1. **Dataflow Diagram:**

**A data flow diagram (DFD)**,delves deeper into the internal workings of the system. It illustrates how data moves within the system, showing processes, data stores, and the flow of information between different components, unlike the context diagram which only visually represents the relationships between a system and its external entities.



### Explanation of the Data Flow Diagram

**Entities**:

* **User**: Interacts with the system by generating reports and receiving alerts.
* **System Admin**: Manages API integrations and configurations.
* **Third Party Services**: Supplies external data like weather and road signs.

**Processes**:

* **1.0 Manage Reports**: Handles the creation and management of reports.
* **2.0 Manage Road Signs**: Oversees the data related to road signs.
* **3.0 Generate Alerts**: Triggers alerts based on certain conditions.
* **4.0 Generate Route Suggestions**: Provides users with suggested routes based on current data.
* **5.0 Fetch External data:** Gets the data from Eternal systems through APIs
* **6.0 Manage API Integration:** Gets API keys and ensures the integration in the codebase.

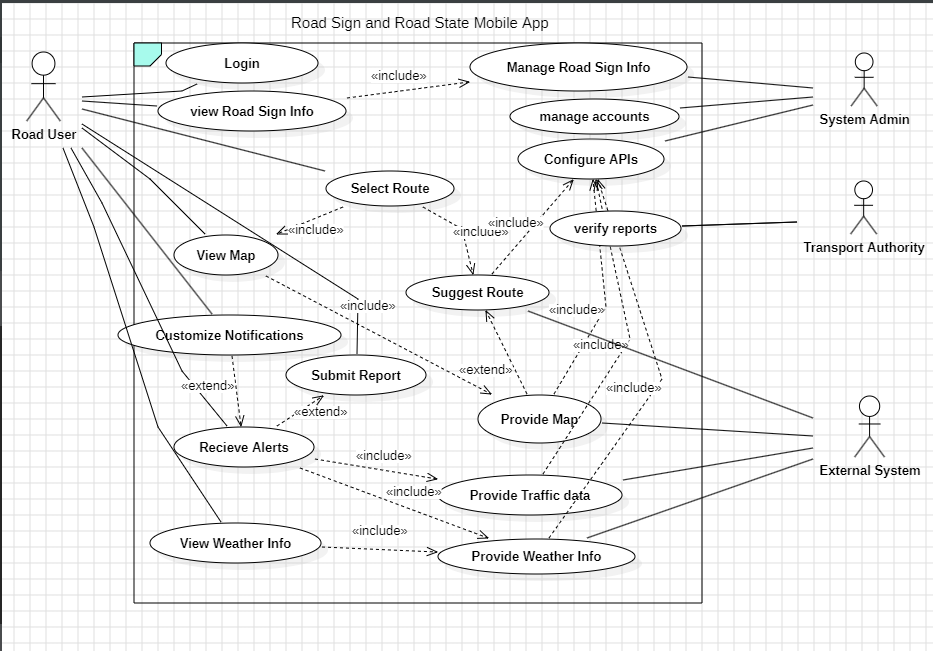
**Data Flows**:

* **User to Reports**: Users submit reports, which then flow into the "Manage Reports" process.
* **Reports to User**: The system generates and sends back reports to the user.
* **Admin to API Keys**: The system admin manages API keys, allowing integration with third-party services.
* **Third Party Services to Road Signs**: Supplies data related to road signs to the system.
* **Weather Data**: Third-party services provide weather data, which is used in various processes.
* **Route Suggestion**: The system uses data to generate route suggestions, which are then sent back to the user.

From this DFD, it illustrates the flow of information within the system, detailing how data is processed and managed among users, admins, and third-party services.

**3. Use case Diagram:**

A use case diagram is a visual representation of the interactions between users (or actors) and a system. It outlines the functionalities the system offers and how different actors interact with those functionalities. Use case diagrams help stakeholders understand the system's requirements and the relationships between users and functions.



### Use Cases

* **Login**: Users authenticate their identity to securely access the application.
* **View Road Sign Info**: Users can access and view information about road signs in their vicinity, enhancing awareness and safety.
* **Manage Road Sign Info**: Admins have the ability to add, update, or remove road sign data, ensuring all information is current and accurate for users.
* **Configure APIs**: Admins are responsible for managing the configuration of APIs that integrate external data sources into the application.
* **Select Route**: Users can select a route based on their preferences and available options, facilitating personalized navigation experiences.
* **Suggest Route**: The system generates route suggestions for users based on current data from road signs, traffic conditions, and user preferences.
* **Receive Alerts**: Users receive timely notifications about important updates, such as road hazards or traffic changes, keeping them informed while traveling.
* **Submit Report**: Users can submit reports about road conditions or sign issues, contributing to community engagement and safety.
* **View Map**: Users can visualize their location and nearby road signs through a map interface, aiding in navigation.
* **Provide Map**: The system supplies essential map data for various use cases, enabling features like viewing and route selection.
* **View Weather Info**: Users can access current weather conditions that may affect travel, promoting safety and informed decision-making.
* **Provide Traffic Data**: The system retrieves traffic data from external sources to enhance route suggestions and user awareness.
* **Verify Reports**: The transport authority reviews and verifies reports submitted by users to ensure they are accurate and take necessary actions based on this information.
* **Manage Accounts**: Admins can create, update, or deactivate user accounts, maintaining control over user access and security within the application.

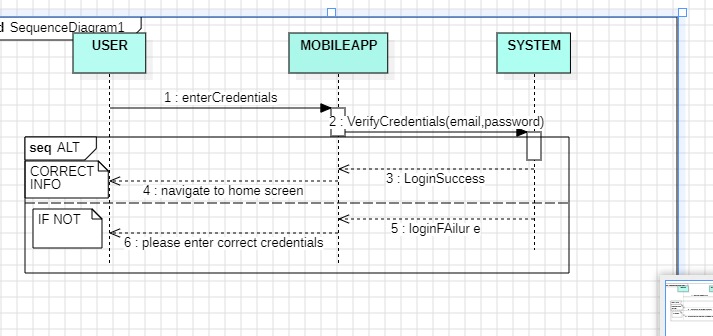
This description of actors and use cases, provides an understanding of their interactions within the Road Sign and Road State Notification Mobile Application and can easily model the behaviour of the application.

**4 Sequence Diagram:**

A sequence diagram illustrates how objects interact in a particular scenario of a use case, focusing on the order of messages exchanged. This diagram shows the process of submitting a report in the Road Sign and Road State Notification Mobile Application.

1. **Sequence Diagram for User Login**

This sequence diagram illustrates the login process for a user within a mobile application, detailing the interactions between the user, the mobile app, and the system.



#### Actors and Components:

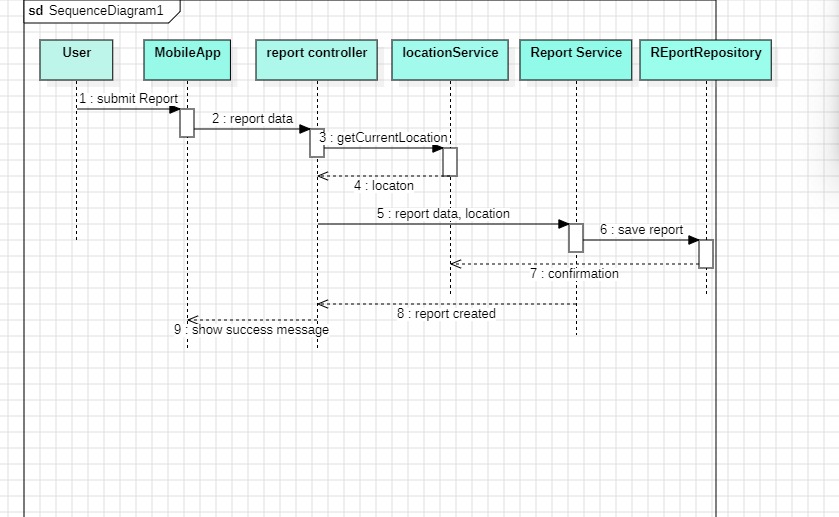
* **User**: The individual attempting to log in to the mobile application.
* **Mobile App**: The interface through which the user enters their credentials.
* **System**: The backend that verifies the user’s credentials.

#### Sequence of Events:

* **Enter Credentials**:The user inputs their email and password into the mobile app.
* **Verify Credentials**:The mobile app sends the entered credentials to the system for verification.
* **Alternative Flow (ALT)**: The system checks if the provided information is correct:
* **Correct Info**: If the credentials are valid, the flow proceeds to the next step.
* **If Not**: If the credentials are invalid, the system prompts the user to enter the correct credentials.
* **Navigate to Home Screen**: If the credentials are correct, the user is directed to the home screen of the application.
* **Login Failure**: If the credentials are incorrect, the app displays a login failure message, prompting the user to correct their input.

This sequence diagram shows the flow of actions involved in the login process for a user in the mobile application. It outlines how the user interacts with the application and the system's response based on the validity of the entered credentials, ensuring clarity in the login procedure.

### Sequence Diagram for user reports



#### Actors and Components:

* **User**: The individual submitting the report.
* **Mobile App**: The application interface used by the user to submit the report.
* **Report Controller**: Manages the report submission process and acts as an intermediary between the app and services.
* **Location Service**: Retrieves the user’s current location.
* **Report Service**: Handles the business logic related to reports.
* **Report Repository**: A data storage component where reports are saved.

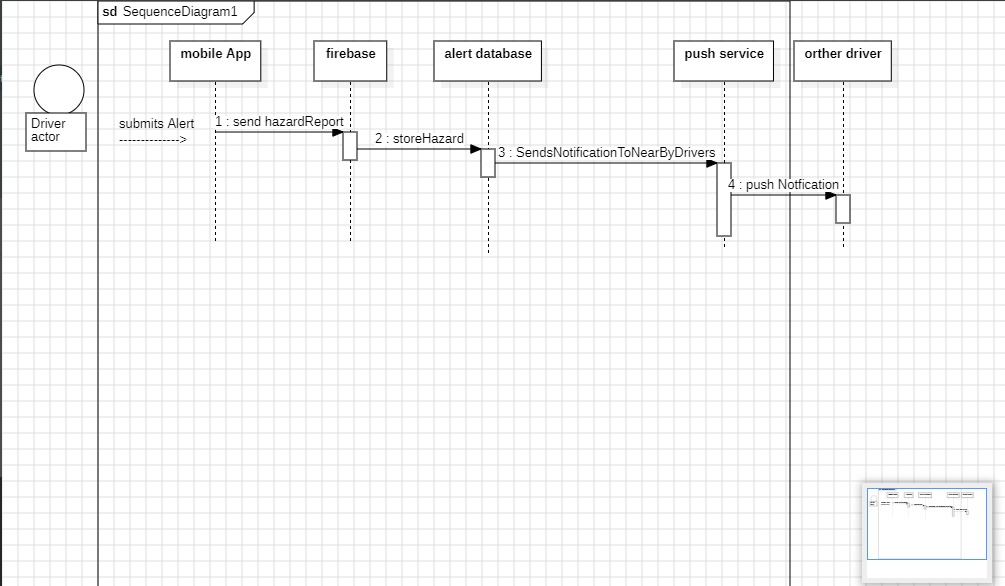
#### Sequence of Events:

* **Submit Report**:The user initiates the process by submitting a report through the mobile app.
* **Report Data**:The mobile app sends the report data to the report controller for processing.
* **Get Current Location**:The report controller requests the current location from the location service to include it in the report.
* **Location**:The location service retrieves and returns the user's current location to the report controller.
* **Report Data, Location**:The report controller then compiles the report data along with the current location.
* **Save Report**:The report controller sends the compiled report data to the report service to save it.
* **Confirmation**:The report service saves the report in the report repository and returns a confirmation to the report controller.
* **Show Success Message**:Finally, the report controller instructs the mobile app to show a success message to the user, indicating that the report has been successfully submitted.

This sequence diagram demonstrates the flow of interactions involved in submitting a report within the application, showing the roles of different components and the order in which messages are exchanged. It provides a clear understanding of the process from the user's action to the confirmation of report submission.

**C. Sequence Diagram for Alerts**

This sequence diagram outlines the process of alerts within the mobile application, detailing the interactions between various components involved in the alert system.



#### Actors and Components:

* **Driver Actor**: The individual who submits an alert about a hazard.
* **Mobile App**: The application interface used by the driver to send the hazard report.
* **Firebase**: A backend service that handles data storage and communication.
* **Alert Database**: Stores hazard reports submitted by users.
* **Push Service**: Responsible for sending notifications to other drivers.
* **Other Driver**: Represents other users who might receive the hazard notification.

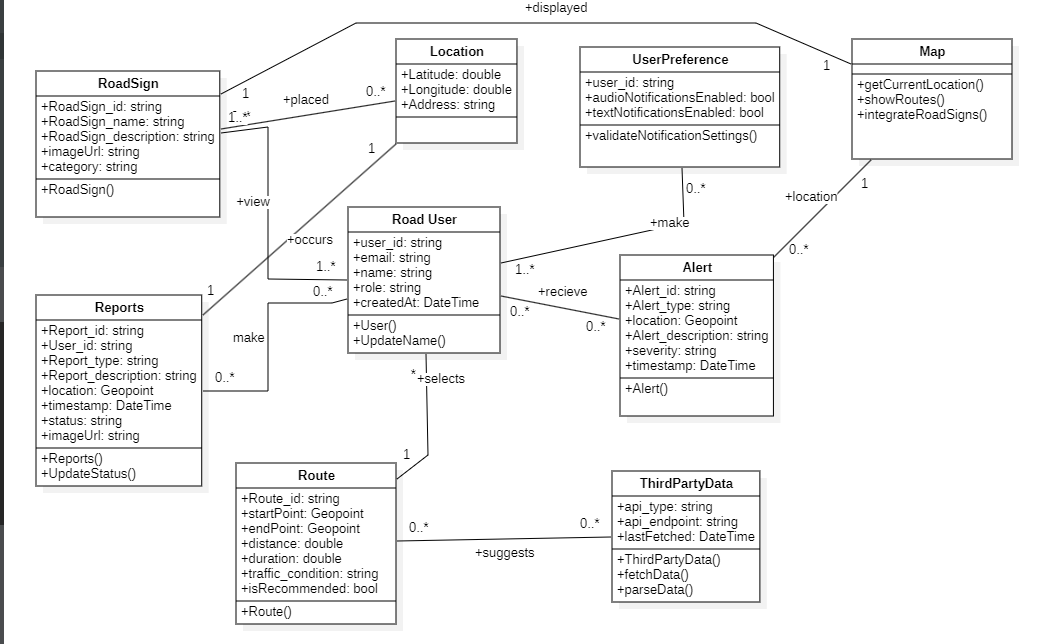
#### Sequence of Events:

* **Send Report**:The driver submits an alert through the mobile app, indicating a hazard on the road.
* **Store Hazard**:The mobile app sends the hazard report to Firebase, which stores the report in the alert database.
* **Send Notification to Nearby Drivers**:The alert database processes the report and triggers the push service to notify other nearby drivers about the hazard.
* **Push Notification**: The push service sends a notification to the other drivers, informing them of the reported hazard.

This sequence diagram demonstrates the flow of interactions involved when a driver submits a hazard alert in the mobile application. It highlights how the alert is processed, stored, and communicated to other drivers, ensuring that relevant information is disseminated promptly to enhance road safety.

**5. Class Diagram:**

A class diagram is a type of static structure diagram that represents the classes, attributes, operations (or methods), and the relationships among objects in a system. It provides a visual representation of the system's structure and serves as a blueprint for the application's architecture.



This class diagram illustrates the key classes involved in the Road Sign and Road State Notification Mobile Application, highlighting their attributes, methods, and relationships.

#### Classes and Their Attributes:

* **RoadSign**:

**Attributes**:

* id: string - Unique identifier for each road sign.
* category: string - Type/category of the road sign.
* placed: bool - Indicates whether the sign is currently placed.

**Methods**:

* validateLocation() - Validates the location of the road sign.
* **Location**:

**Attributes**:

* latitude: double - Latitude coordinate for the location.
* longitude: double - Longitude coordinate for the location.
* address: string - Physical address associated with the location.

**Methods**:

* getCurrentLocation() - Retrieves the current location.
* integrateRoadSigns() - Integrates road sign data with the location.
* **UserPreference**:

**Attributes**:

* notificationsEnabled: bool - Indicates if the user has enabled notifications.

**Methods**:

* validateNotificationsEnabled() - Validates the user's notification settings.
* **RoadUser**:

**Attributes**:

* name: string - User's name.
* email: string - User's email address.
* lastLogin: DateTime - Timestamp of the user's last login.

**Methods**:

* receive() - Method for the user to receive alerts.
* selects() - Allows the user to select options or routes.
* **Reports**:

**Attributes**:

* reportId: string - Unique identifier for each report.
* description: string - Description of the reported issue.
* timestamp: DateTime - Time when the report was created.

**Methods**:

* update() - Updates the report information.
* **Alert**:

**Attributes**:

* alertId: string - Unique identifier for the alert.
* description: string - Description of the alert.
* severity: string - Severity level of the alert.

**Methods**:

* send() - Sends the alert to users.
* **Route**:

**Attributes**:

* routeId: string - Unique identifier for the route.
* startPoint: GeoPoint - Starting point of the route.
* endPoint: GeoPoint - Ending point of the route.
* trafficCondition: string - Current traffic condition on the route.

**Methods**:

* suggest() - Suggests the best route based on current conditions.
* **ThirdPartyData**:

**Attributes**:

* apiKey: string - API key for accessing third-party services.
* lastUpdated: DateTime - Timestamp of the last data update.

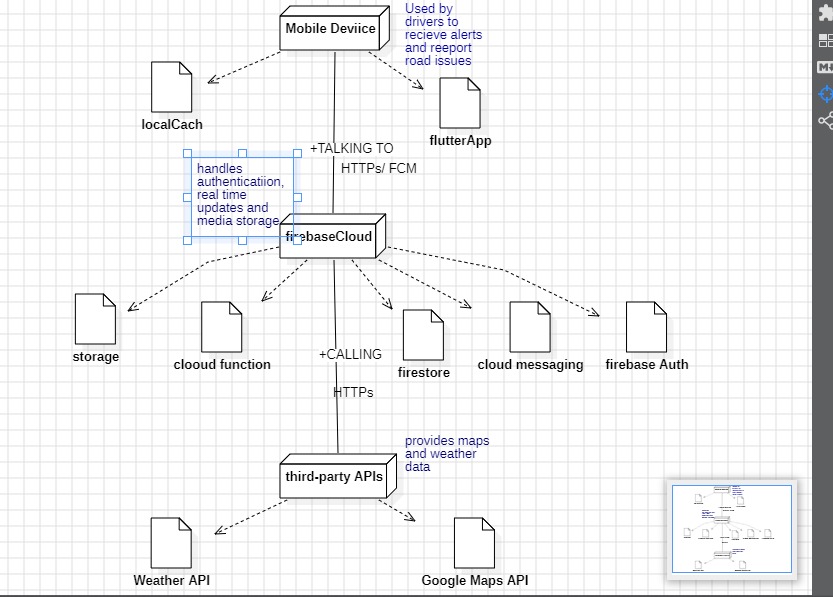
**Methods**:

* parseData() - Parses data received from third-party services.

This class diagram provides a detailed view of the system's structure, showcasing the key classes, their attributes, methods, and how they relate to one another within the Road Sign and Road State Notification Mobile Application. It serves as a foundational blueprint for understanding the application’s architecture and functionality.

**6. Deployment Diagram:**

A deployment diagram illustrates the physical deployment of artifacts on nodes. It shows how software components are distributed across hardware and how they interact with each other and external systems.



This deployment diagram represents the architecture of the Road Sign and Road State Notification Mobile Application, highlighting the components and their interactions.

#### Key Components:

* **Mobile Device**:
* **Description**: The platform used by drivers to receive alerts and report road issues.
* **Connections**:
* **Local Cache**: Stores data locally for quick access.
* **Firebase Cloud**:
* **Description**: The backend service that handles various functionalities.
* **Functions**:
* **Handles authentication**: Manages user login and authentication.
* **Real-time updates**: Provides live updates for alerts and reports.
* **Media storage**: Stores media files related to reports and alerts.
* **Connections**:
* **Talking to HTTPs/FCM**: Communicates via HTTP and Firebase Cloud Messaging.
* **Third-Party APIs**:
* **Description**: External services that provide additional data.
* **Connections**:
* **Weather API**: Supplies current weather information.
* **Google Maps API**: Provides mapping and location services.

This deployment diagram provides a clear view of how the components of the Road Sign and Road State Notification Mobile Application are deployed. It illustrates the relationships between the mobile device, Firebase Cloud, and third-party APIs, showing how they work together to deliver functionality and data to users.