UNIVERSITY OF BUEA

REPUBLIC OF CAMEROON

PEACE-WORK-FATHERLAND

P.O. Box 63,
Buea, South West Region
CAMEROON
Tel : (237) 3332 21 34/3332 26 90
Fax: (237) 3332 22 72

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

SYSTEM MODELLING AND DESIGN OF A CAR FAULT DIAGNOSTIC APP

Project Title: Car Fault Diagnosis Mobile Application

Course Title: Internet Programming and Network Programming

Course Code: CEF 440

By:

BERINYUY CLETUS FE21A148
EFUETLATEH JOAN FE22A197
ETAPE NGABE FE22A210
ETIENDEM PEARL FE22A211
TATA GLEN FE22A309

2024/2025 Academic Year

Table of Contents

Table of Figures

Introduction

System modeling and design is a critical phase in the software development lifecycle, providing a structured blueprint that guides implementation. For the Car Fault Diagnosis Mobile Application, this phase ensures a clear understanding of how the system operates, how users interact with it, and how different components communicate. Through various diagrams, including context, data flow, use case, sequence, class, and deployment diagrams, we present a comprehensive visual representation of the system's functionality, architecture, and behavior. These models not only aid developers in building the system but also help stakeholders grasp the system's scope, processes, and design logic.

1. Context Diagram

The context diagram provides a high-level overview of the system, representing it as a single process and showing how it interacts with external entities. It illustrates the boundaries of the system and the information that flows between the system and external actors. This diagram is useful for understanding the scope of the application and identifying primary sources and destinations of data.

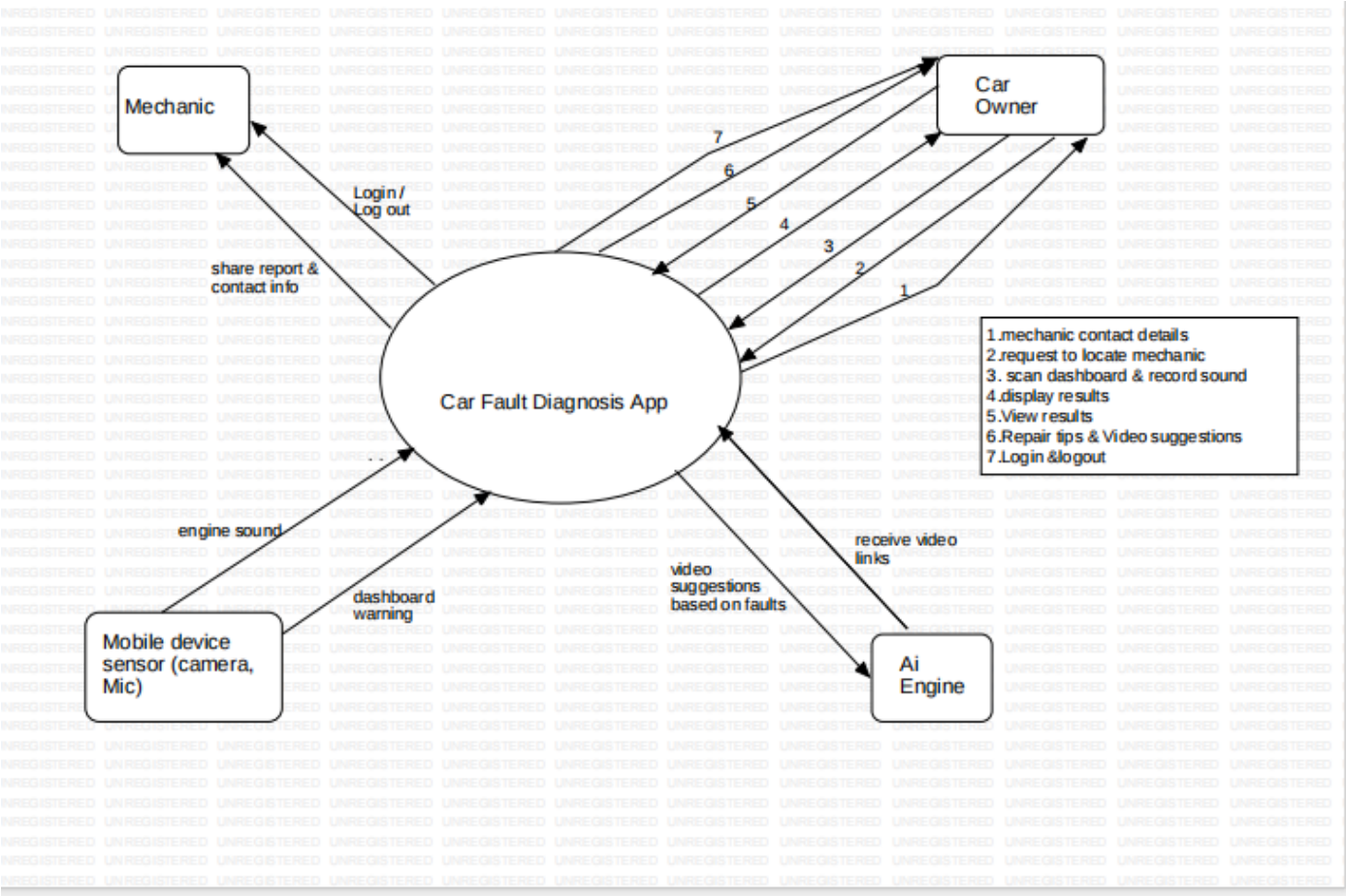

Figure 1: Context Diagram

2. Data Flow Diagram (DFD) - Level 1

The Level 1 DFD breaks down the single process shown in the context diagram into multiple subprocesses. It shows how data moves through the system via input, processing, and storage. This diagram offers a more detailed view of the system's functionality and helps in identifying redundant processes, potential bottlenecks, and opportunities for optimization.

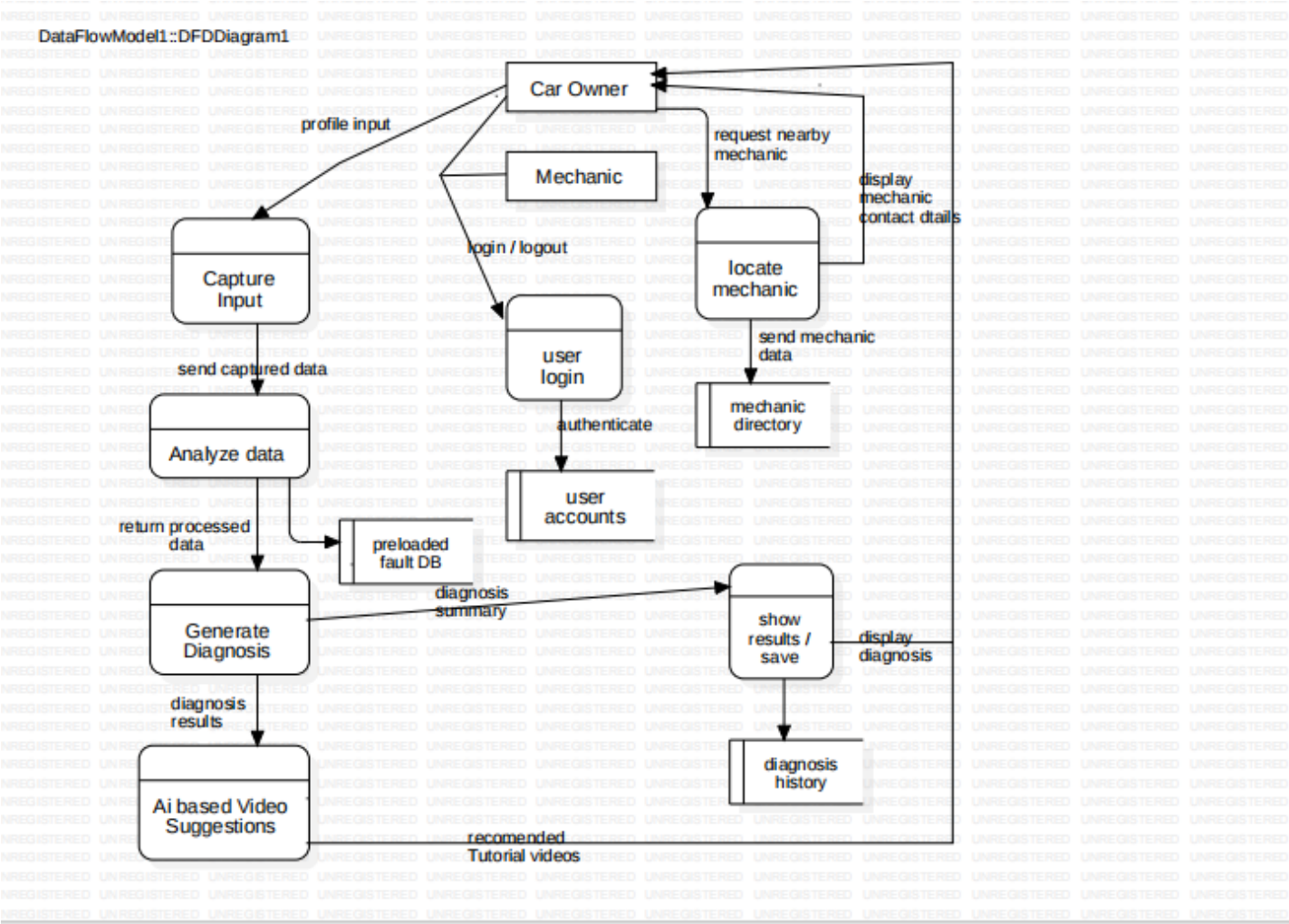

Figure 2: Data Flow Diagram

3. Use Case Diagram

This use case diagram visually represents the functional requirements of our system. It illustrates the interactions between users and the system, highlighting the various use cases that fulfill user needs.

The primary actors are the **Car User** and the **Mechanic**. The app allows users to register and log in, providing them access to various features. Key functionalities include:

- **Scanning Dashboard Lights**: Users can interpret warning lights on their car's dashboard.
- **Analyzing Engine Sounds**: The app can record and analyze sounds to diagnose issues.
- **Receiving Diagnoses**: Based on the data collected, users receive feedback on potential problems.
- **Getting Fix Recommendations**: The app suggests solutions for the identified issues.
- **Accessing Video Tutorials**: Users can view helpful videos for troubleshooting.
- **Contacting Nearby Mechanics**: Finally, users can easily find and reach out to local mechanics if needed.

In summary, this use case diagram helps us understand how our app serves users by outlining essential interactions and functionalities, ultimately enhancing the car diagnosis experience."

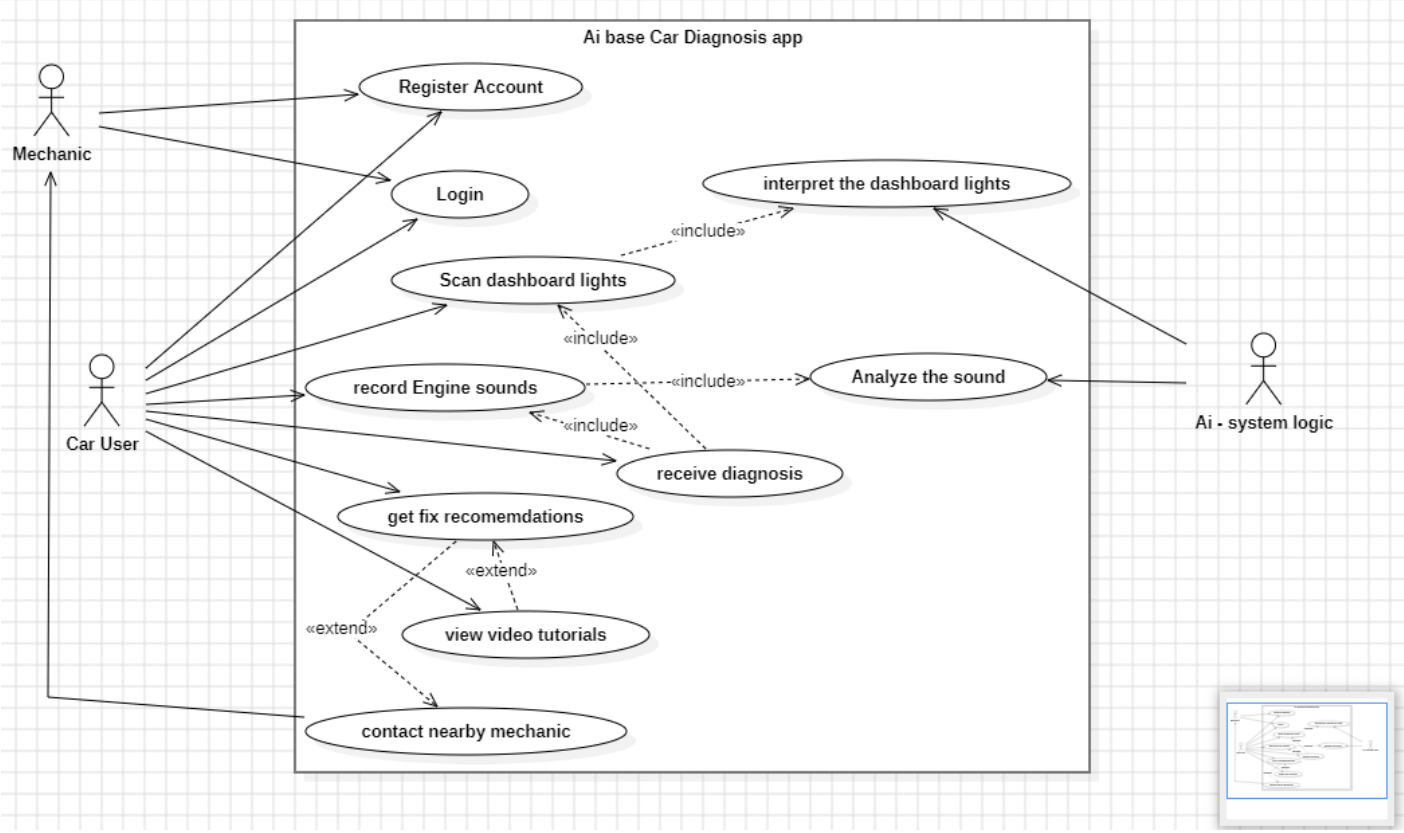

Figure 3: Use Case Diagram

4. Sequence Diagrams

1.0 Introduction

The sequence diagrams illustrate the step-by-step interactions between the user and various components of the Car Fault Diagnosis Mobile Application. These diagrams focus on how the system processes user actions like scanning dashboard lights, recording engine sounds, logging in, and locating mechanics. Each diagram highlights the flow of control, messages, return responses, and conditions for both successful and failed operations.

1.1 Scan Dashboard Light and Receive Diagnosis

This sequence shows how the user captures a dashboard light using the camera. The image is processed, analyzed, and a diagnostic result is returned. It includes a condition for when the image is unclear.

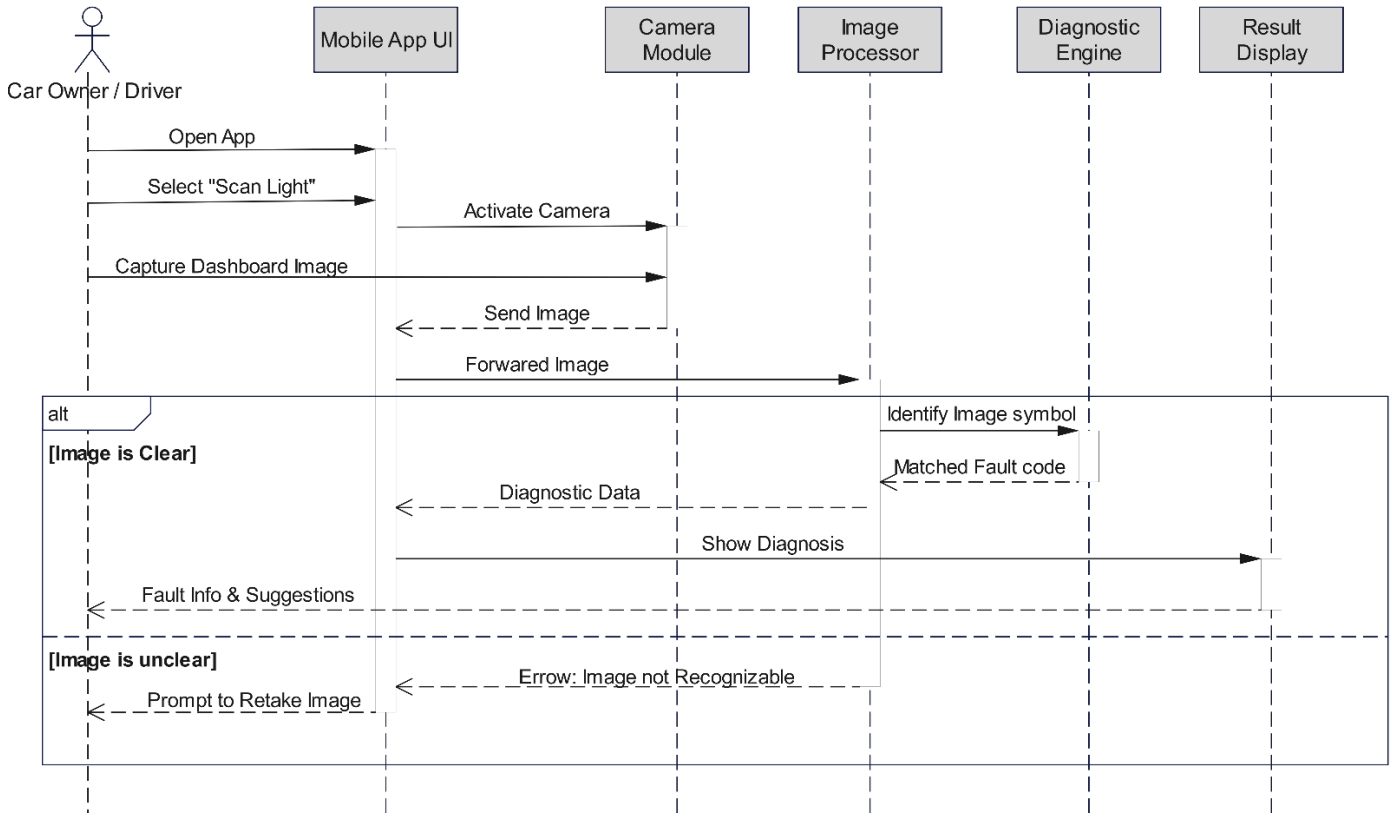

Figure 4: Sequence Diagram for Dashboard Light Scan and Diagnosis

1.2 Record Engine Sound and Receive Diagnosis

This diagram shows the flow of recording engine sound, processing the audio, and receiving results. It includes a branch for when the sound quality is too poor to analyze.

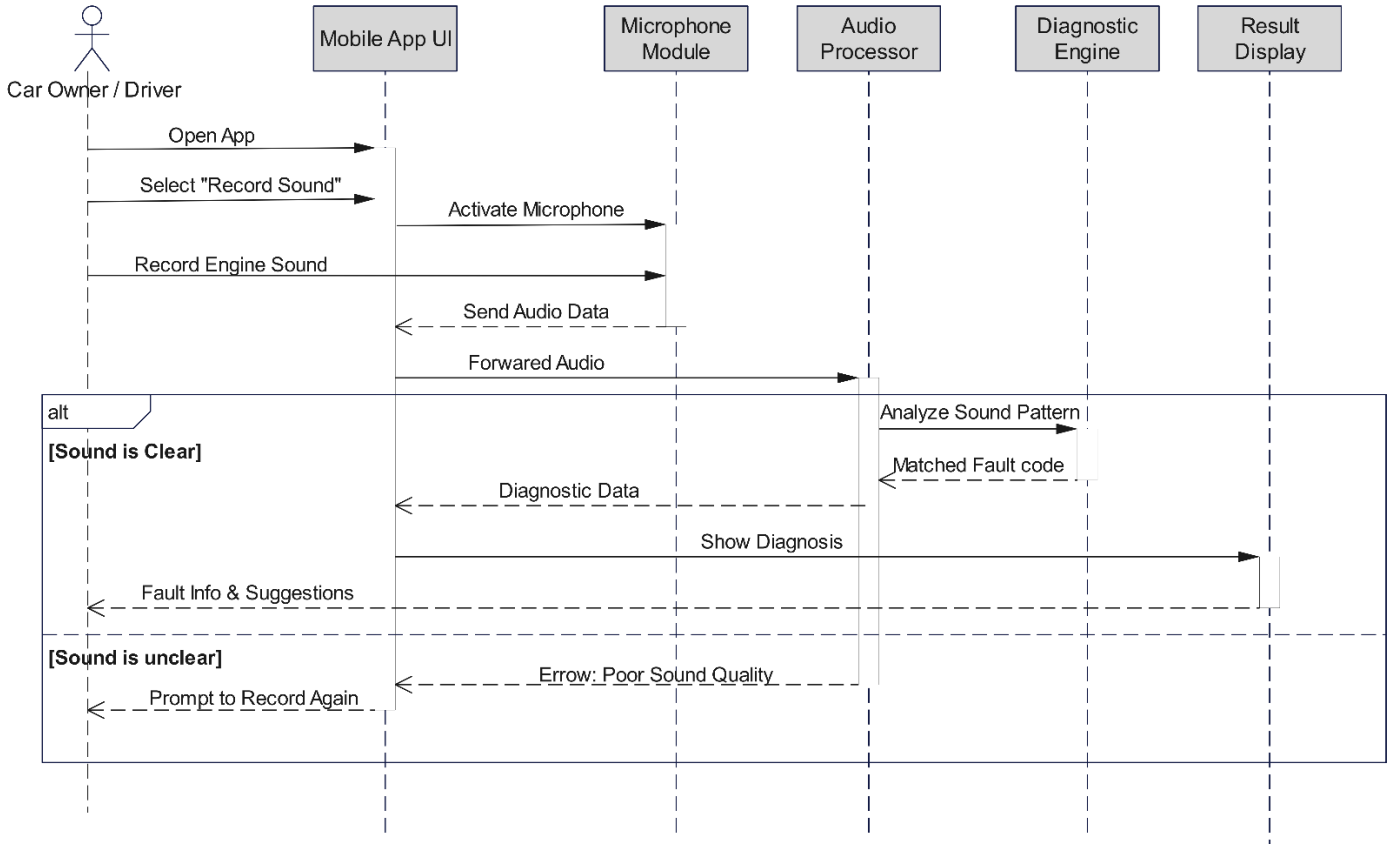

Figure 5: Sequence Diagram for Engine Sound Record and Diagnosis

1.3 Login, and Sign Up

This diagram captures the process for authenticating users. It includes the login flow, handling invalid credentials, and a follow-up sign-up process if the user doesn't have an account.

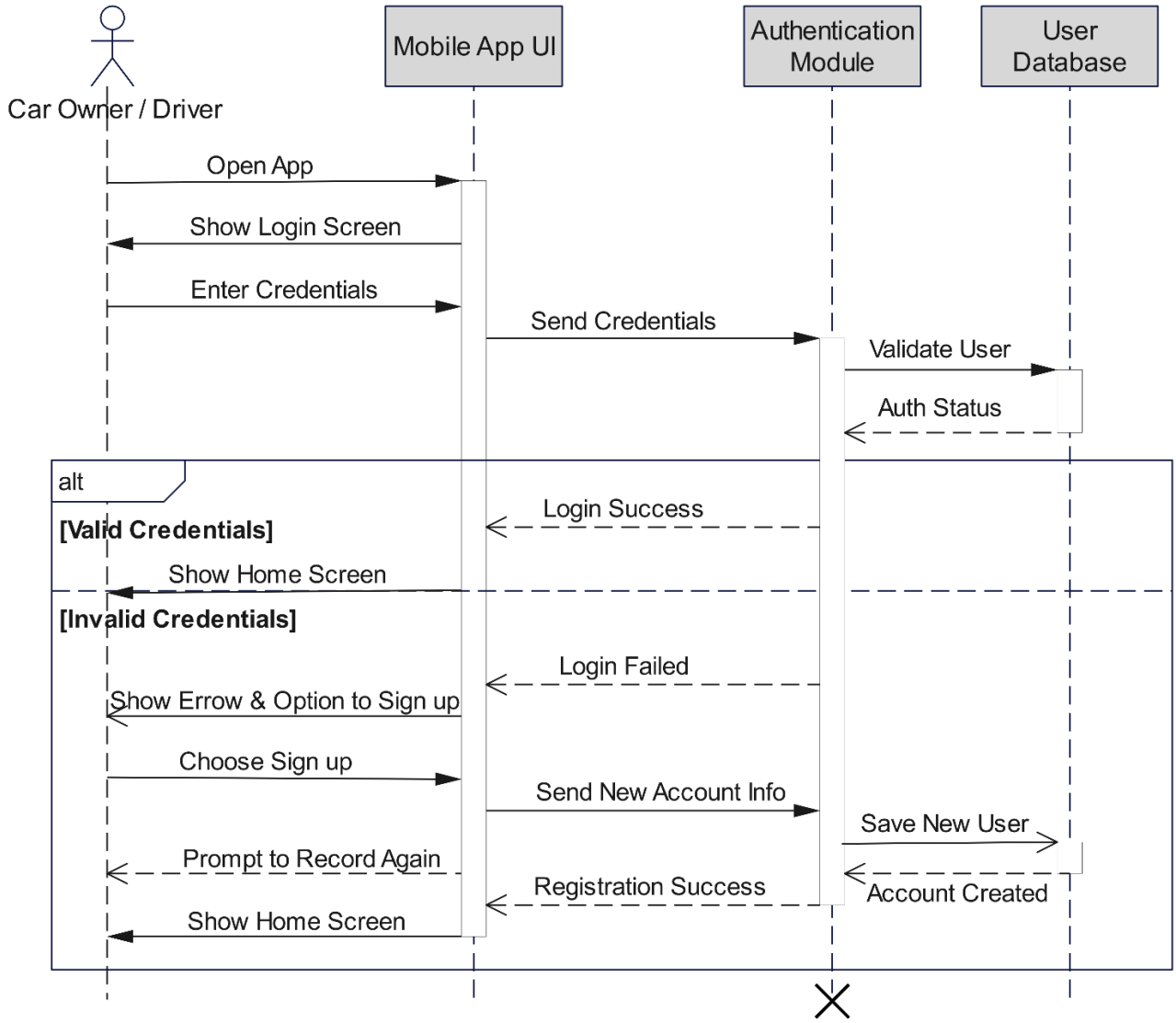

Figure 6: Sequence Diagram For Login, and Sign Up

1.4 Locate and Contact Mechanic

This sequence represents how a user can locate nearby mechanics through the app. It uses GPS to fetch the current location, searches a mechanic directory, and allows the user to view or contact one. It also handles the case where location access is denied or fails.

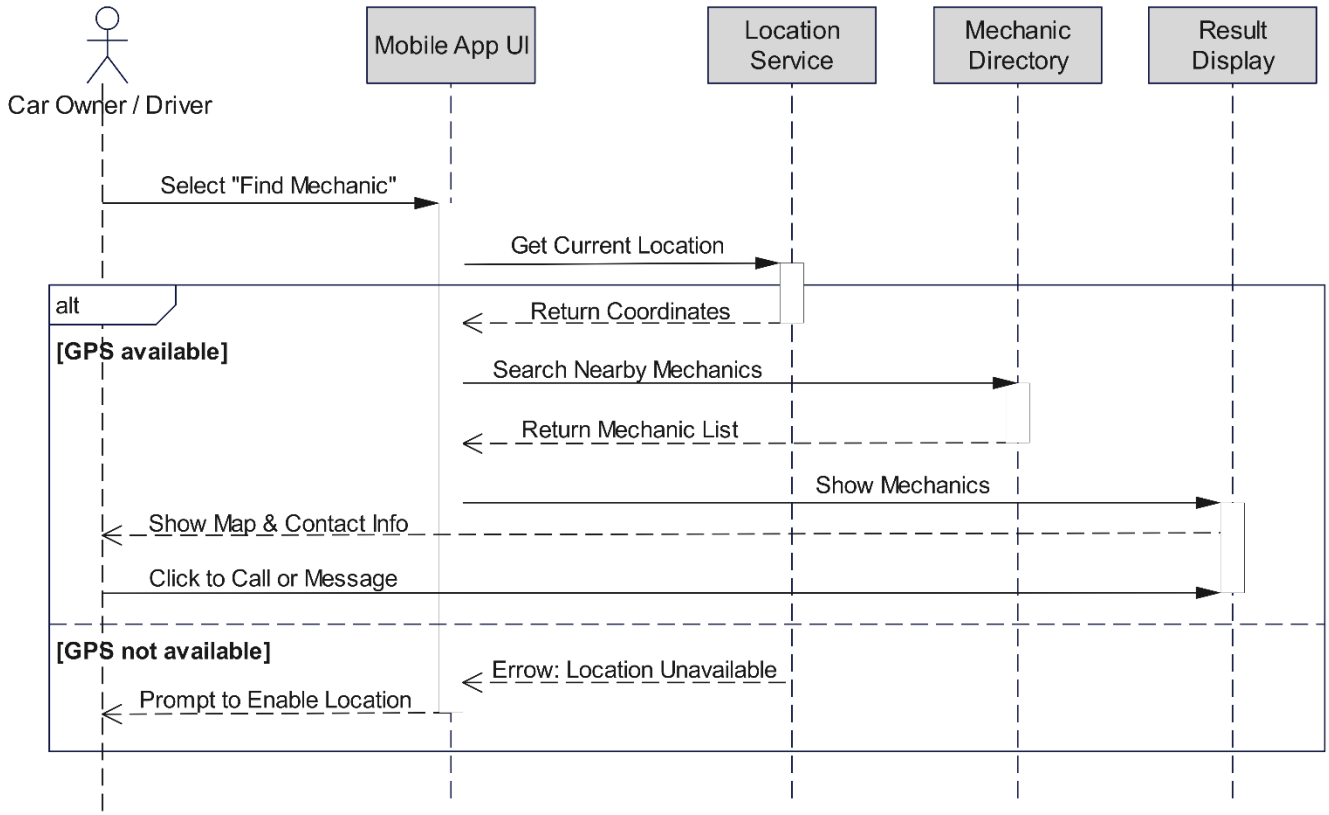

Figure 7: Sequence for Locating and Contacting a Mechanic

These sequence diagrams provide a detailed visualization of key functionalities within the mobile app, helping the development team and stakeholders understand how system components interact during each use case. They also highlight alternate paths to improve system robustness.

5. Class Diagram

1.0 Introduction

The class diagram provides a high-level view of the structure of the Car Fault Diagnosis Mobile Application. It illustrates the key classes (objects) in the system, their attributes and methods, as well as the relationships between them. This diagram helps developers understand how data is organized, how different components of the system interact, and how responsibilities are distributed across the application.

1.1 Class Overview

- **User**: Represents a registered user of the app. Users can log in, submit diagnosis requests, contact mechanics, and provide feedback.

- **AuthManager**: Handles login, logout, and user registration processes.
- **DiagnosisRequest**: Captures input from the user in the form of images or sound for diagnostic purposes.
- **DiagnosticEngine**: Contains the core logic for analyzing user input (image or sound) and generating diagnosis results.
- **DiagnosisResult**: Stores the output from the diagnostic engine, including fault name, suggestions, and urgency.
- **History**: Maintains past diagnostic results for each user and allows retrieval or deletion of past records.
- **Mechanic**: Represents a mechanic that users can contact or locate through the app.
- **Feedback**: Stores user-submitted messages or suggestions about the app experience.

1.2 Class Relationships

- A User uses the AuthManager to authenticate into the system.
- A User can submit one or more DiagnosisRequest instances, which are then processed by the DiagnosticEngine.
- The DiagnosticEngine generates a DiagnosisResult, which is displayed and can be stored in History.
- The History class is composed of multiple DiagnosisResult objects, meaning that each history record tightly holds its related result.
- A User can view and manage their history, and also has the option to contact or locate Mechanics listed in the system.
- A User may also send Feedback, which is stored for review or future improvements.

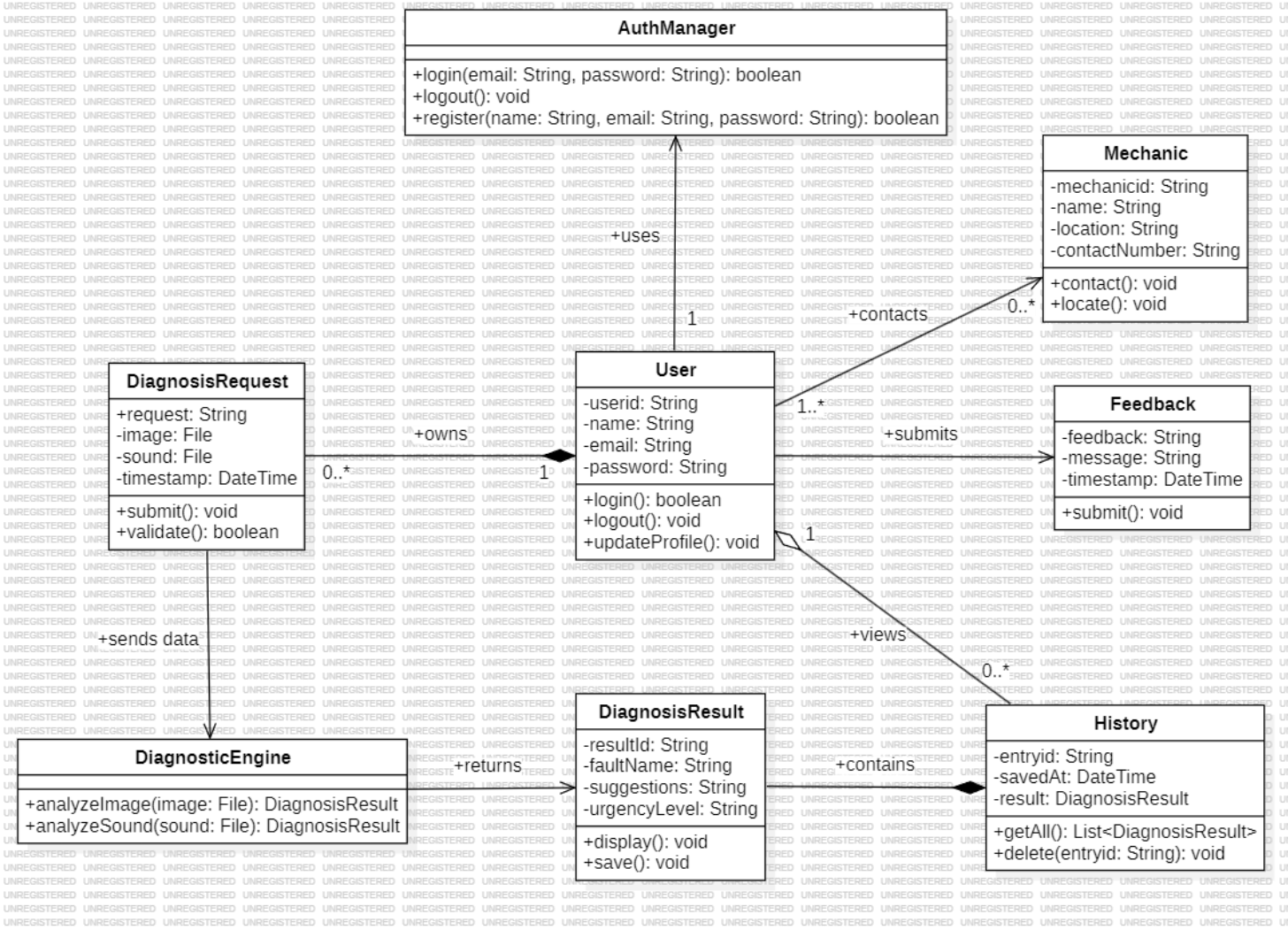

Figure 8: Class Diagram

The class diagram clearly defines the core components of the mobile application and their interactions. It ensures proper organization of data, simplifies code design, and supports efficient implementation and maintenance of the system. Understanding the relationships and responsibilities of each class is essential for building a robust and scalable app architecture.

6. Deployment Diagram

The deployment diagram illustrates the physical architecture of the system, showing how software components are distributed across hardware environments. It maps the system's software artifacts onto hardware nodes, such as mobile devices and cloud servers. This diagram helps in planning infrastructure needs and understanding system scalability and performance.

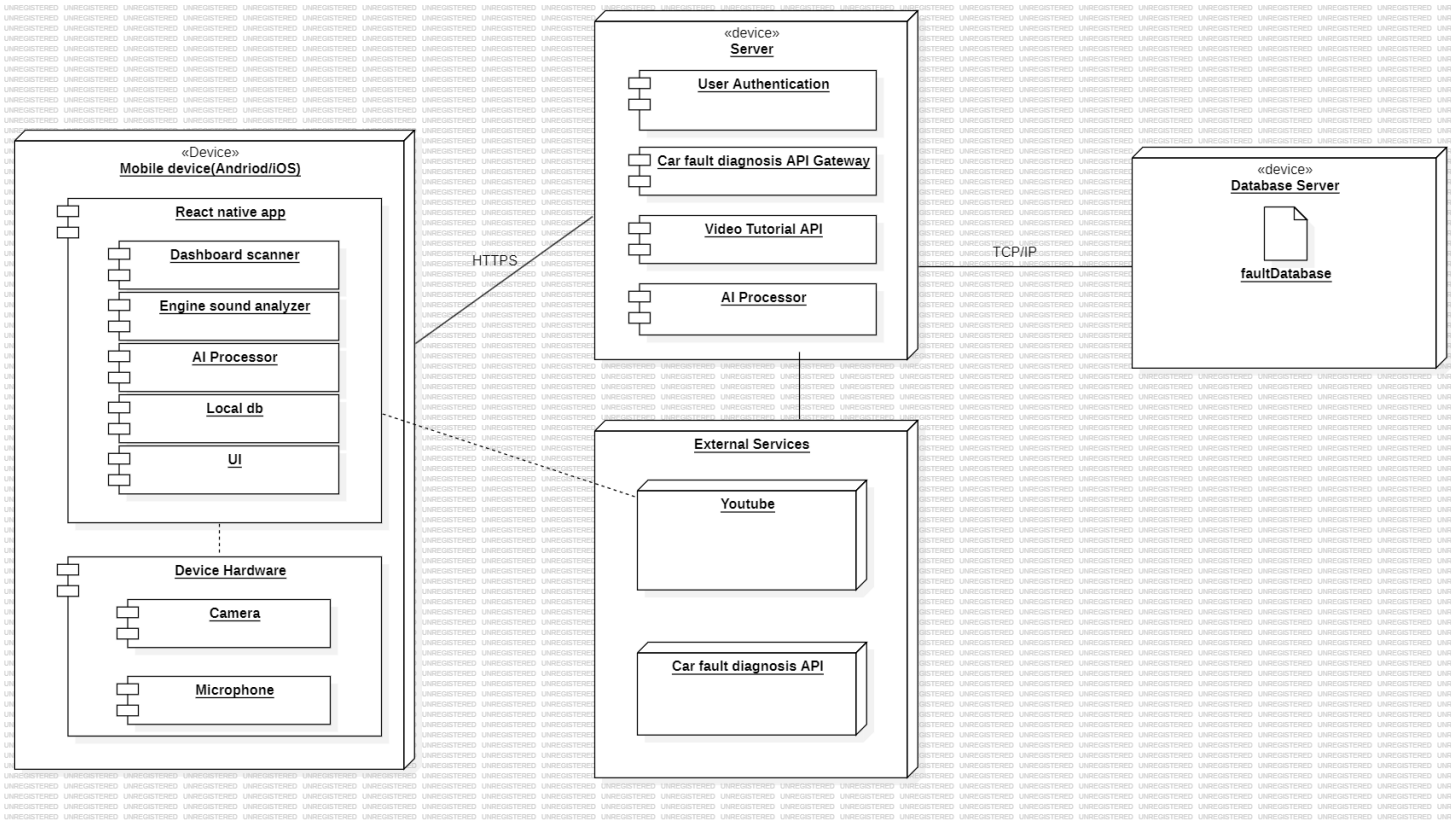

Figure 9: Deployment Diagram

Conclusion:

The system modeling and design phase structured the app into clearly defined interactions, processes, and data flows. With detailed diagrams including Context, DFD, Use cases, and Sequences, the architecture ensures clarity, modularity, and future scalability. These models serve as a foundation for coding, testing, and deployment.