UNIVERSITY OF BUEA
FACULTY OF ENGINEERING
AND TECHNOLOGY

REPUBLIC OF CAMEROON
PEACE-WORK-FATHERLAND

COURSE TITLE: INTERNET PROGRAMMING AND MOBILE PROGRAMMING

COURSE CODE: CEF440

COURSE INSTRUCTOR: Dr. VALERY NKEMENI

GROUP NUMBER: 18

TASK 4: SYSTEM MODELLING AND DESIGN

Group 18 Members

S/N	NAME	MATRICULE
1	Sindze Djam Alan Wilfried	FE20A105
2	Enjema Ndive Peace	FE22A203
3	Ngwinkem Ketty Nerita	FE22A269
4	Njamutoh Shalom Brenda Tasah	FE22A271
5	Tabi Atem Rogan Essembion	FE22A300

22/05/2025

A) Context Diagram Explanation

1. Introduction

This document explains the context diagram for a Car Fault Diagnosis Mobile Application. The diagram illustrates the interaction between the main system and various external entities. It highlights the flow of information in and out of the system.

2. Central System: Car Fault Diagnosis System

The Car Fault Diagnosis System is the core component of the application. It processes input data from different sources, analyzes the fault codes, provides diagnostic reports, repair tips, and cost estimates, and communicates with multiple external entities.

3. External Entities and Interactions

➢ Car Owner/Driver

- Input: Dashboard images, engine sounds, user preferences.
- Output: Diagnosis, repair tips, cost estimates, notifications, video links.

➢ Mechanic

- Input: Feedback on fault accuracy, expert video content.
- Output: Diagnostic reports, client fault history.

➢ OBD-II Device

- Input: Sends fault codes to the system.

➢ YouTube

- Output: Provides repair and maintenance videos based on faults.

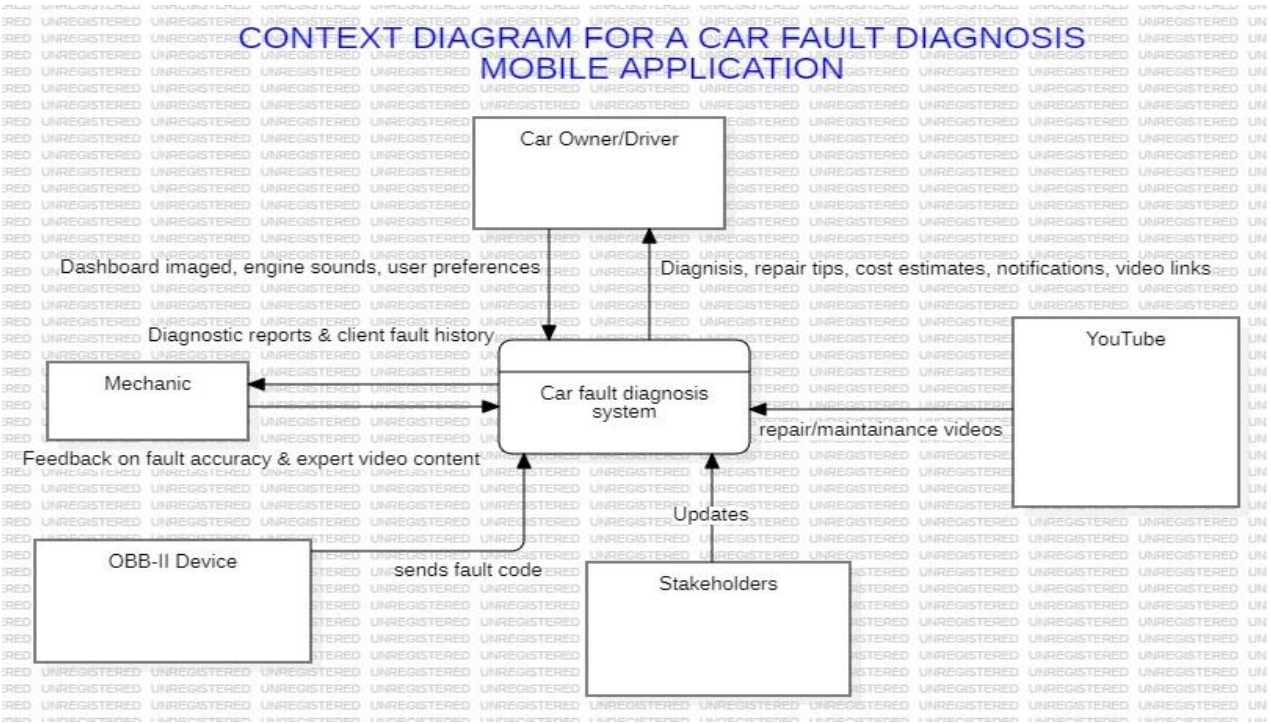
➢ Stakeholders

- Input/Output: Provide and receive system updates or reports.

4. Summary of Relationships

External Entity	Type of Interaction	Data/Information Exchanged
Car Owner/Driver	Bidirectional	Inputs: dashboard images, engine sounds, preferences; Outputs: diagnosis, repair tips, cost, notifications
Mechanic	Bidirectional	Inputs: feedback, videos; Outputs: reports, history
OBD-II Device	Input only	Sends fault codes
YouTube	Output only	Provides relevant videos
Stakeholders	Bidirectional	Inputs/Outputs: System updates, reports

Below is the context diagram



B) Level 1 Data Flow Diagram.

A Data Flow Diagram (DFD) is a graphical representation used to visualize the flow of data through a system. It helps in understanding how input data is transformed into output results through various processes. A DFD breaks down a system into smaller modules and shows how data moves between them, as well as between external entities and data stores. DFDs are commonly used in system analysis and design to model the functional aspects of a system.

1. Introduction

This document provides a detailed explanation of the Level 1 Data Flow Diagram (DFD) for a Car Fault Diagnostic Mobile Application. It outlines the system's processes, external entities, data stores, and the flow of information among them.

2. External Entity

- **User**

The user interacts with the system by providing input such as vehicle symptoms or triggering an automatic diagnostic. They receive diagnostic results, repair tips, code interpretations, and maintenance suggestions.

3. System Processes

3.1 Perform Automatic Diagnostic

Automatically retrieves data from the OBD-2 device and forwards the fault codes to interpretation and diagnostic units.

3.2 Perform Manual Diagnostic

Accepts user-input symptoms and forwards them to the diagnostic unit for fault analysis.

3.3 Send OBD Code for Interpretation

Sends retrieved OBD-2 codes to the OBD-2 Code Database for interpretation.

3.4 Perform Diagnostic

Analyzes fault codes and symptoms from both automatic and manual processes to determine possible faults.

3.5 Display OBD-2 Code Interpretation

Displays human-readable interpretations of OBD-2 fault codes using information from the OBD-2 Code Database.

3.6 Display Maintenance Book

Displays relevant maintenance instructions based on fault diagnosis using information from the Maintenance Book Database.

3.7 Repair Tutorials

Fetches and displays video repair tutorials relevant to the diagnosed faults using the YouTube API.

4. Data Stores

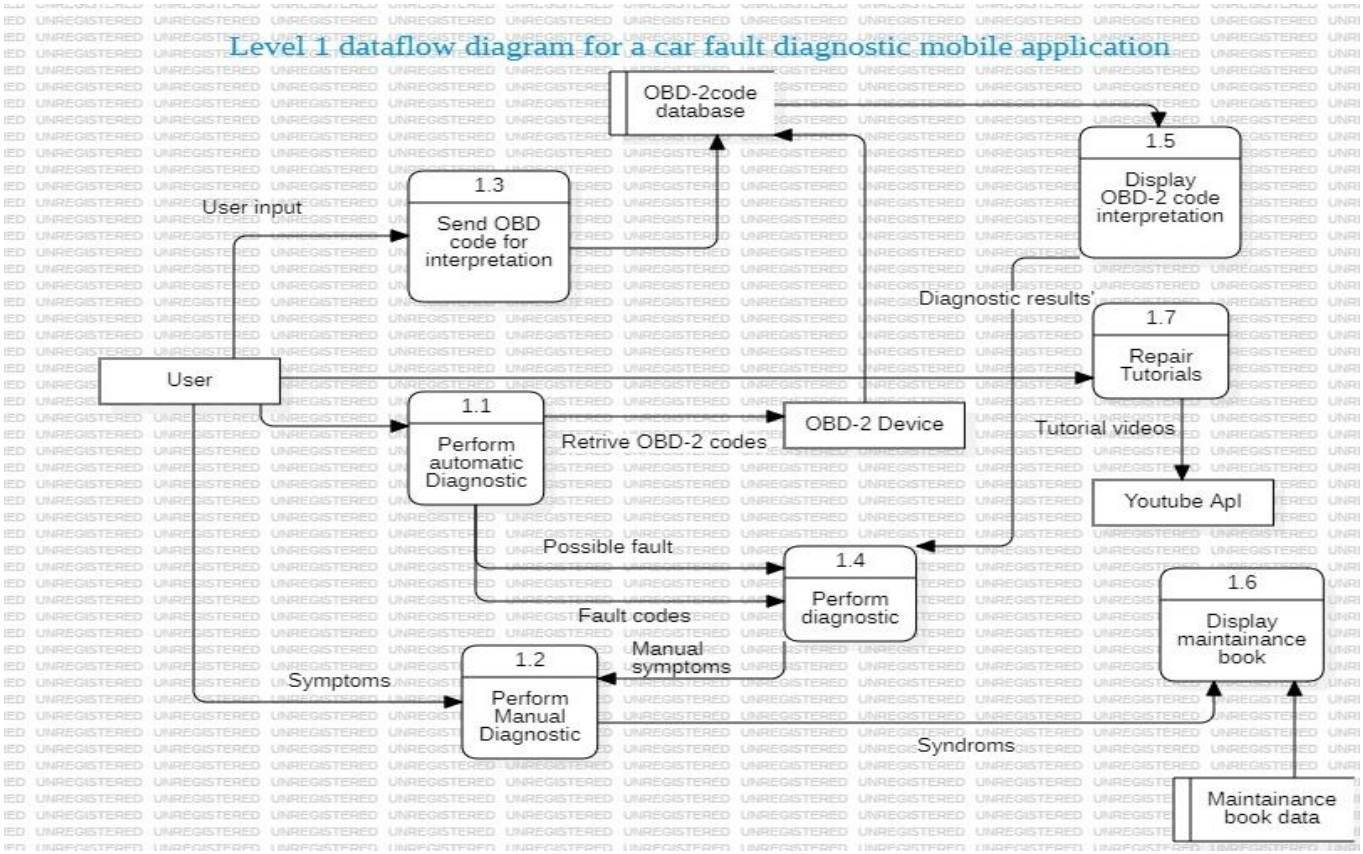
- **OBD-2 Code Database**

Stores fault code definitions and descriptions used for code interpretation.

- **Maintenance Book Data**

Contains repair and maintenance procedures matched to different fault types.

Below is a level1 data flow diagram



5. Data Flow Summary

From	To	Data
User	1.1, 1.2	Inputs (trigger or symptoms)
1.1	OBD-2 Device	Request codes
OBD-2 Device	1.1, 1.4	Fault codes
1.3	OBD-2 Code Database	Code lookup
OBD-2 Code Database	1.5	Code meanings
1.4	1.5, 1.6, 1.7	Diagnosis results
1.6	Maintenance Book Data	Maintenance info
1.7	YouTube API	Video tutorials

C) Use Case Diagram.

1. Introduction to Use Case Diagrams

A Use Case Diagram is a visual representation used in system analysis to show the interactions between users (actors) and a system to achieve specific goals (use cases). It identifies the functionality provided by a system in terms of how external entities (like users) interact with it. Use case diagrams help stakeholders understand system behavior, scope, and user requirements.

2. Overview of the Use Case Diagram

The diagram models the interactions between a user and the Car Fault Mobile Application. It includes use cases related to diagnosing car faults, accessing repair and maintenance content, and interpreting diagnostic data such as OBD-2 codes, dashboard warning lights, and engine sounds.

3. Actors and Use Cases

Primary Actor: User

The user is the central actor in the diagram. They interact with the mobile application to perform various diagnostic and informational tasks. The user initiates all use cases.

4. Use Cases and Relationships

➢ Diagnose Car Issue

Main use case that allows the user to identify vehicle faults using automatic or manual diagnostics.

✧ Automatic Diagnostic

Automatically retrieves and interprets OBD-2 codes via Bluetooth to identify faults.

● Get OBD-2 Codes via Bluetooth

Obtains diagnostic data from the car's OBD-2 system through Bluetooth communication.

● Request Bluetooth Access

Requests the necessary Bluetooth permissions to communicate with the vehicle.

✧ Manual Diagnostic

Allows the user to input various symptoms manually to help identify possible faults.

● Indicate Vehicle Won't Start

A specific manual symptom input representing a failure to start.

● Input Hepatic Symptoms

Allows users to input symptoms based on physical feedback (e.g., vibration).

● Input Auditory Symptoms

Allows users to describe or upload engine-related noises.

● Input Visual Symptoms

Users can input visual clues like smoke or fluid leaks.

● Access Repair Tutorials

Gives users access to repair guides and videos to fix identified faults.

● Refer to YouTube Videos

Includes links to relevant repair tutorial videos for identified faults.

● Access Maintenance Book

Displays maintenance information related to diagnosed issues.

● Interpret OBD-2 Code

Users can enter a code manually for interpretation.

● Enter OBD-2 Code

Used when the user has a known fault code they want interpreted.

● Interpret Dashboard Warning Light

Explains the meaning of specific dashboard lights.

● Snap Dashboard Warning Light

User captures a photo of a warning light for interpretation.

- **Interpret Engine Sound**

Analyzes uploaded engine sounds for possible issues.

- **Record Engine Sound**

Records audio from the engine for analysis.

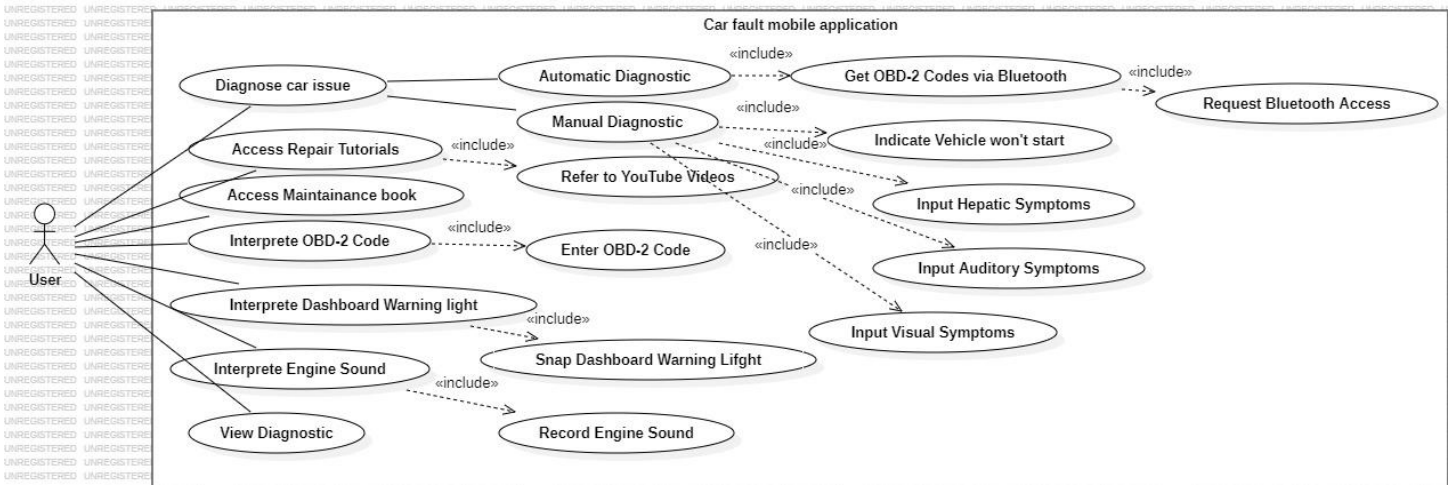
- **View Diagnostic**

Presents the final diagnostic results to the user.

5. Relationships (<<include>>)

The <<include>> relationships indicate that one use case is a mandatory part of another use case. For example, performing an Automatic Diagnostic includes Getting OBD-2 Codes via Bluetooth, which in turn includes Requesting Bluetooth Access. This shows a clear dependency and ensures that preconditions are met.

Below is the use case diagram



D) Sequence Diagram

➢ Introduction

This sequence diagram illustrates the interactions between different system components and users for various automotive diagnostic and repair functionalities. It depicts the order of messages exchanged between objects to perform specific functions, showing the lifespan of objects and the messages they send and receive.

➢ Parameters (Lifelines/Objects)

The horizontal elements at the top, extending vertically, represent the participants (objects or actors) involved in the interaction. Their vertical lines below them are 'lifelines,' indicating their existence over time.

- User (Actor): Represents an end-user or technician who initiates various diagnostic and repair processes.
- User (General Actor): A more abstract representation of any user interacting with the system. It serves as a placeholder.
- Mobile App (System Component): The primary user interface that interacts with various backend systems and devices.
- Bluetooth Device (System Component): Connects the Mobile App to the OBD2 Scanner using Bluetooth.
- OBD2 Scanner (System Component): Interfaces with the vehicle's OBD system to retrieve diagnostic data.
- Server/Backend (System Component): Handles data processing, interpretation, storage, and diagnosis.
- YouTube API (External System): Fetches and displays relevant repair video tutorials from YouTube.

➢ Relationships and Interactions (Messages)

The horizontal arrows between lifelines represent messages passed between objects. The arrow direction indicates the sender and receiver, and the top-to-bottom order shows the sequence of events.

1. Automatic Diagnostic

- User initiates diagnostic via Mobile App.
- App requests and receives Bluetooth permission from User.
- App connects to Bluetooth Device and OBD2 Scanner.
- OBD2 Scanner retrieves and sends codes to Server.
- Server interprets codes and sends results to Mobile App.
- App displays interpreted results to User.

2. Manual Diagnostic

- User selects Manual Diagnostic option.
- App asks symptom-related questions.
- User inputs responses.
- Responses sent to Server for analysis.
- Server returns possible issues.
- App displays issues to User.

3. Repair Tutorials

- User opens Repair Tutorials section.
- App queries YouTube API for relevant videos.
- API returns video links.
- App displays tutorials to User.

4. Maintainable Book

- User opens Maintainable Book.
- App requests maintenance info from Server.
- Server returns data.
- App displays info to User.

5. OBD2 Code Lookup

- User navigates to OBD2 code lookup.
- User inputs specific code (e.g., P0300).
- Code sent to Server for interpretation.
- Server returns explanation.
- App displays the interpretation.

6. Dashboard Light Detection (Camera)

- User uses camera to capture dashboard light.
- Image sent to Server.
- Server analyzes and identifies the warning.
- App displays interpretation.

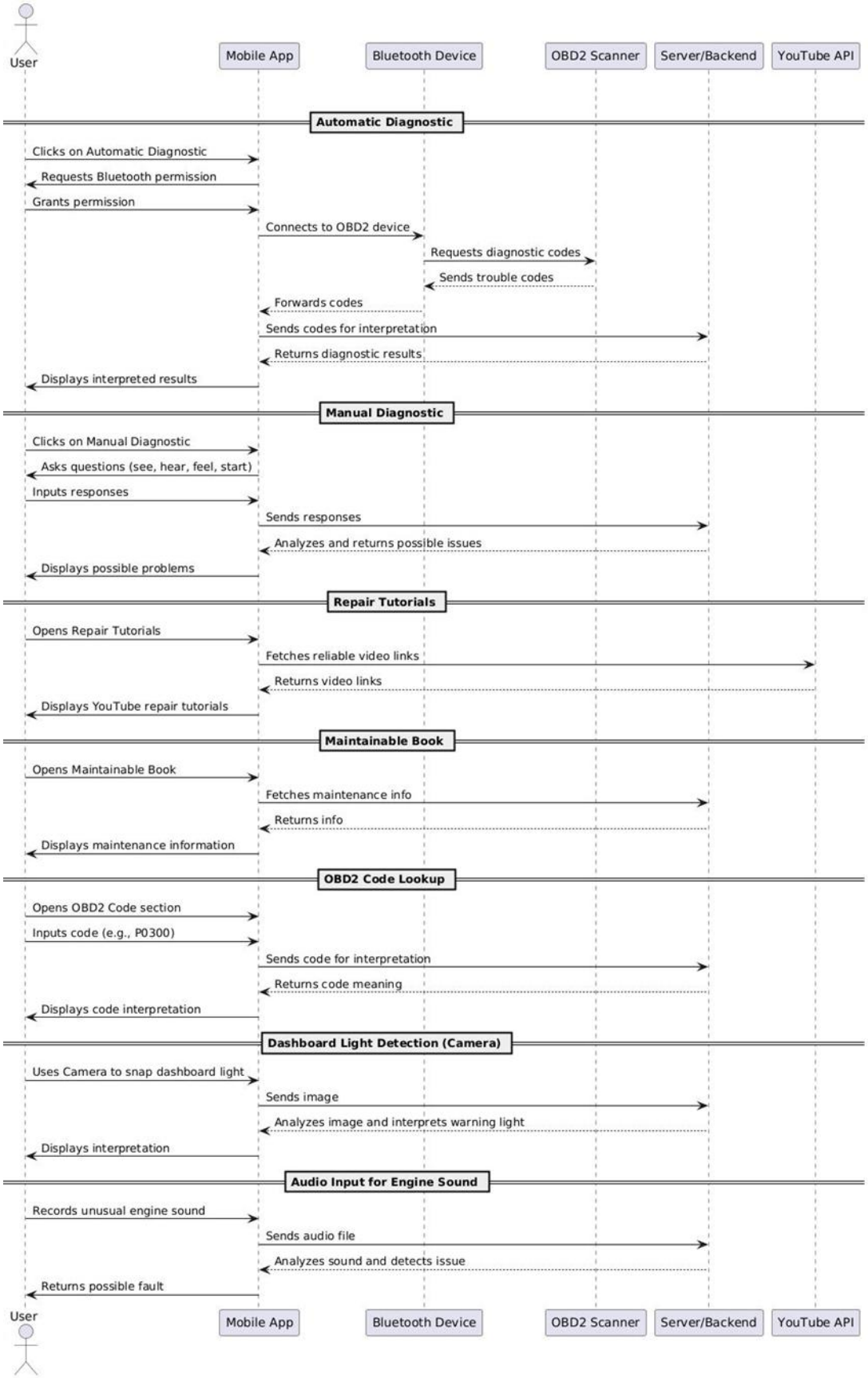
7. Audio Input for Engine Sound

- User records engine sound using app.
- Audio sent to Server.
- Server analyzes sound and detects issue.
- App displays identified fault.

➢ Overall Purpose of the Diagram

This sequence diagram provides a comprehensive, chronological visualization of interactions and message flows between components of an automotive diagnostic system. It serves to outline system responsibilities, highlight communication paths, and help developers understand real-time behavior, potential inefficiencies, and key functional requirements during diagnostic and repair operations.

Below is the overall sequence diaram for all the scenarios



E) Class Diagram

➢ Introduction

A Class Diagram is a fundamental building block in object-oriented programming and analysis within the Unified Modeling Language (UML). It provides a static, structural view of a system by showing its classes, their attributes (properties), operations (methods), and the relationships between them.

◆ **Key elements of a Class Diagram include:**

➢ Classes: Represented by rectangles divided into three compartments:

- * Name: The name of the class.
- * Attributes: The data or properties that an object of the class holds (e.g., userID, name).
- * Operations (Methods): The functions or behaviors that an object of the class can perform (e.g., login(), diagnoseCar()).

➢ Relationships:

- * Association: A general relationship between two classes, indicating that objects of one class are connected to objects of another.
- * Aggregation: A 'has-a' relationship where one class is part of another, but can exist independently.
- * Composition: A strong 'has-a' relationship where the part cannot exist independently of the whole.
- * Generalization (Inheritance): An 'is-a' relationship where one class inherits attributes and operations from another.
- * Dependency: A 'uses-a' relationship where one class depends on another.

➢ Detailed Class Description

This diagram models the static structure of an 'AI Car Fault Diagnosis Mobile App,' illustrating the various classes that make up the application, their internal structure (attributes and operations), and how they relate to each other.

◆ User

- * userID: String - A unique identifier for the user.
- * name: String - The name of the user.
- * email: String - The user's email address.
- * preferences: Preferences - An attribute representing an instance of the Preferences class.
- * diagnoseCar() - Initiates a car diagnosis.

➢ Preferences

- * preferredMode: String - The user's preferred diagnostic mode.
- * language: String - The user's preferred language.

◆ DiagnosticSystem

- * startAutomaticDiagnosis() - Initiates the automatic diagnostic process.
- * startManualDiagnosis() - Initiates the manual diagnostic process.
- * interpretOBD2Code(code: String): String - Returns code interpretation.
- * processCameraInput(image: Image): String - Interprets dashboard images.
- * processAudioInput(audio: Audio): String - Interprets engine sounds.

➢ Relationships between Classes

The lines and arrows connecting the classes define their relationships:

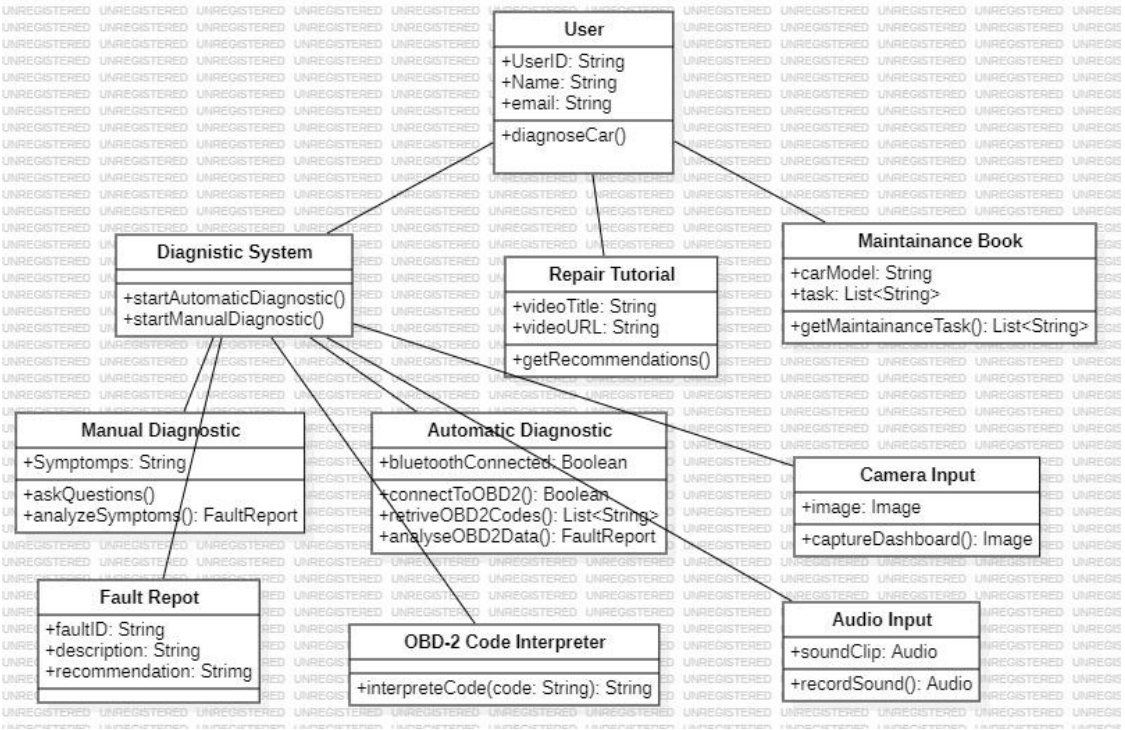
- * Dependency (Dashed Arrow): Indicates that one class uses or depends on another class.
- * Association (Solid Line): A general relationship.
- * Generalization, Composition/Aggregation are conceptually used.

➢ Overall Purpose of the Diagram

This Class Diagram provides a blueprint of the AI Car Fault Diagnosis Mobile App's software architecture. It defines:

- * What components exist
- * What data they hold
- * What actions they can perform
- * How they are connected

Below is the class diagram



F) Deployment DiAgram

Introduction

A Deployment Diagram in UML (Unified Modelling Language) is a structural diagram that illustrates the physical execution architecture of a system. It shows the configuration of run-time processing nodes and the components that live on them. These nodes can be physical machines, virtual machines, or other execution environments. The connections between nodes represent the communication paths (e.g., network connections) over which components interact. Deployment diagrams are crucial for understanding the physical distribution of a system and for planning its deployment, scaling, and maintenance.

Detailed Explanation of the Diagram

1. Nodes (Physical Devices/Environments):

The diagram identifies three primary nodes, each representing a distinct physical or logical computing environment:

- **Car Owners Smart Phone:** This node represents the user's mobile device, likely a smartphone, where the primary user interface and some local processing occur.
- **Backend Server:** This node represents a remote server infrastructure that handles more complex processing, data storage, and external integrations.
- **OB-2 Device:** This node represents a physical device connected to a car's On-Board Diagnostics (OBD-II) port, used for accessing vehicle data.

2. Artefacts (Software Components/Executables):

Within each node, various artefacts (software components or executables) are deployed. These are typically represented by rectangles with the artefact name inside.

2.1. Artefacts on "Car Owners Smart Phone":

- **Mobile Application:** This is the core application running on the smartphone. It's the user-facing part of the system.
- **Camera:** Represents the smartphone's camera hardware and its associated software interface, used for capturing images.
- **Microphone:** Represents the smartphone's microphone hardware and its associated software interface, used for recording audio.
- **Bluetooth Module:** Represents the smartphone's Bluetooth hardware and its associated software for wireless communication.

2.2. Artefacts on "Backend Server":

- **Diagnostic Engine:** This artefact on the server is responsible for processing diagnostic information, likely from various sources, to identify issues.
- **Warning Light Classifier:** This artefact specializes in interpreting images of dashboard warning lights to identify their meaning.
- **OBD-2 Code Database:** This artefact is a database storing meanings and details for various OBD-II diagnostic trouble codes.
- **Engine Sound Analysis:** This artefact is responsible for analysing engine sounds to detect anomalies or identify problems.

2.3. Artefacts on "OB-2 Device":

- **OBD-2 Interface:** This artefact represents the software and hardware interface within the OB-2 Device that allows it to communicate with the car's OBD-II system.

3. Communication Paths (Relationships):

The dashed lines with open arrowheads represent communication paths or dependencies between nodes and artefacts, indicating how they interact. The labels on these lines describe the nature of the communication.

3.1. Between "Car Owners Smart Phone" and "Backend Server":

- **"send data for interpretation"**: The Mobile Application on the smartphone sends data to the Backend Server for interpretation. This is a general communication path, likely for various types of data.
- **"snap dashboard lights"**: The Camera on the smartphone captures images of dashboard lights and sends them to the Backend Server. This implies the Warning Light Classifier on the server will process these images.
- **"record engine, sounds"**: The Microphone on the smartphone records engine sounds and sends them to the Backend Server. This indicates the Engine Sound Analysis artefact on the server will process these sounds.

3.2. Between "Car Owners Smart Phone" and "OB-2 Device":

- **"request OBD codes"**: The Bluetooth Module on the smartphone initiates a request to the OB-2 Device to retrieve OBD codes from the vehicle. This implies a wireless Bluetooth connection.

3.3. Within "Backend Server" (Internal Communication):

- **"interpret light images"**: The Warning Light Classifier artefact on the Backend Server receives light images (likely from the Mobile Application) and interprets them. The output of this interpretation might be used by the Diagnostic Engine.
- **"analyse engine sounds"**: The Engine Sound Analysis artefact on the Backend Server receives engine sound data (from the Mobile Application) and performs analysis. The results of this analysis might feed into the Diagnostic Engine.
- **"fetch code meaning"**: The Diagnostic Engine on the Backend Server queries the OBD-2 Code Database to retrieve the meaning of specific OBD-II codes. This indicates that the Diagnostic Engine relies on the database for code interpretation.

3.4. Between "Backend Server" and "External Services":

- **"Fetch repair tutorials"**: The Backend Server (specifically, it's implied that the Diagnostic Engine or another server-side component would initiate this) communicates with External Services to fetch repair tutorials.
- **"YouTube API"**: This is a specific External Service artefact that the Backend Server interacts with. This implies that the fetched repair tutorials might often come from YouTube via its API.

3.5. From "OB-2 Device" (Communication with Car):

- **"OBD-2 Interface"**: This artefact within the OB-2 Device is implicitly responsible for communicating with the car's OBD-II port to retrieve diagnostic data. The diagram doesn't show the car itself as a node, but the function of the OB-2 Device implies this interaction.

4. Key Relationships and Flows:

- **User Interaction (Smartphone):** The Mobile Application serves as the primary interface for the car owner. It uses the smartphone's Camera and Microphone to gather sensory data (dashboard lights, engine sounds).
- **Data Upload to Server:** Sensory data and diagnostic requests are sent from the smartphone to the Backend Server for more intensive processing.
- **Server-Side Diagnostics:** The Backend Server houses the core intelligence, with components like Diagnostic Engine, Warning Light Classifier, and Engine Sound Analysis working together.
- **External Knowledge Base:** The Backend Server leverages external services like YouTube API to provide additional information (e.g., repair tutorials).
- **OBD-II Integration:** The OB-2 Device acts as a bridge between the car's diagnostic system and the smartphone (via Bluetooth), allowing the mobile application to request and retrieve OBD codes.
- **Data Interpretation and Feedback:** The processed information (e.g., warning light meanings, diagnostic code interpretations, repair tutorials) would then be sent back to the Mobile Application on the smartphone for the user to view.

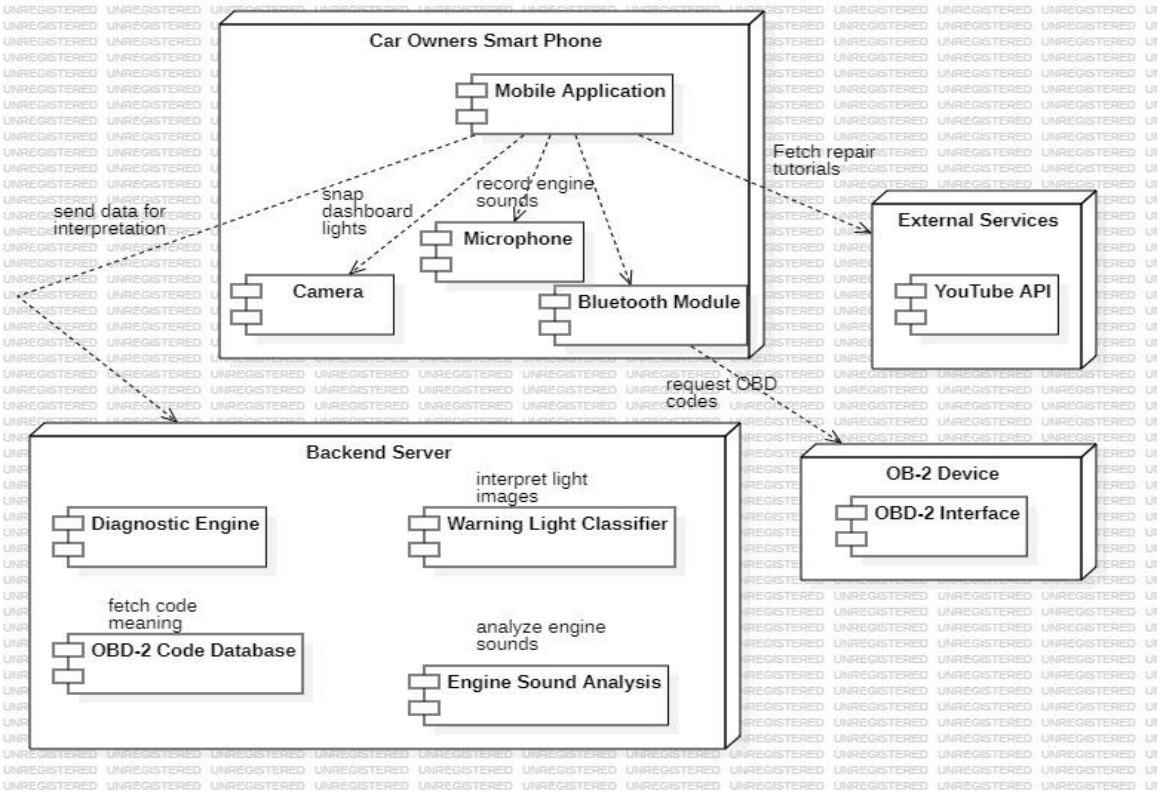
Summary of System Functionality (Inferred from Diagram):

The system allows car owners to:

1. **Diagnose Warning Lights:** By taking a picture of dashboard warning lights with their smartphone, the image is sent to the backend server, classified, and the meaning is returned to the user.
2. **Analyse Engine Sounds:** By recording engine sounds with their smartphone, the audio is sent to the backend server for analysis, potentially identifying issues.
3. **Retrieve OBD-II Codes:** Connect to an OB-2 device via Bluetooth to fetch diagnostic trouble codes directly from their vehicle.
4. **Get Code Meanings:** The fetched OBD-II codes are sent to the backend server, which uses a database to provide detailed explanations.
5. **Access Repair Information:** The backend server can fetch relevant repair tutorials from external services (like YouTube) based on diagnostic findings.

This detailed breakdown provides a comprehensive understanding of the physical architecture and interactions within the depicted "Smart Car Owners Application" system.

Below is the deployment diagram



G)Conclusion

In designing a car fault detection mobile application, diagrams play a vital role in visualizing and understanding system behavior. The **Context Diagram** outlines the app's interaction with external entities like users and diagnostic servers. The **Dataflow Diagram** maps how fault data moves through the system. The **Use Case Diagram** identifies user actions such as scanning or viewing reports. The **Sequence Diagram** details interactions between user, app, and backend during operations like fault analysis. The **Class Diagram** structures app components, while the **Deployment Diagram** shows how mobile and cloud elements are organized. Together, these diagrams ensure a clear, efficient app design.