Department of Computer Engineering

Faculty of engineering and technology

University of Buea

Département de Génie Informatique

Faculté de Génie et de Technologie

Université de Buea

COURSE CODE: CEF 440

COURSE TITLE: INTERNET PROGRAMMIMG AND MOBILE PROGRAMMING

TASK 4: SYSTEM MODELLING AND DESIGN

Presented by:

GROUP 16

Course instructor:

Dr Nkemeni Valery

Date: 26/05/2025

ABSTRACT

This project focuses on the development of an AI-powered mobile application aimed at helping car owners diagnose vehicle faults using smartphone-based tools. By utilizing computer vision and audio recognition, the system interprets dashboard warning lights and analyzes engine sounds to provide real-time diagnostic information. The objective is to offer an accessible, cost-effective, and user-friendly alternative to traditional diagnostic methods, thus improving vehicle maintenance and reducing reliance on professional mechanics for basic issues.

TABLE OF CONTENT

INTRODUCTION

Modern vehicles are equipped with sophisticated onboard diagnostic systems that notify drivers of issues via dashboard warning indicators. However, a significant number of car owners lack the technical understanding required to interpret these signals. This often results in delayed maintenance and exacerbated mechanical issues. Furthermore, early signs of mechanical problems such as abnormal engine noises are frequently overlooked due to the driver's inability to diagnose them.

Leveraging the capabilities of Artificial Intelligence (AI) and mobile technology, this project aims to design and implement a mobile application that empowers car owners with diagnostic tools that operate directly from their smartphones. By incorporating computer vision for image recognition and audio classification for sound diagnostics, the proposed system aims to provide a user-friendly, efficient, and accessible means of understanding car faults.

LITERATURE REVIEW

Several studies and existing tools have explored AI-based automotive diagnostics:

OBD-II Scanners: These devices plug into the vehicle's onboard diagnostics port to retrieve diagnostic trouble codes (DTCs). However, they often require a separate reader or smartphone app and may not interpret warnings in layman's terms.

Computer Vision in Automotive Maintenance: Research indicates that object detection models like YOLO and MobileNet can effectively recognize dashboard symbols with high accuracy.

Audio Fault Detection: Prior studies have demonstrated that convolutional neural networks (CNNs) and recurrent neural networks (RNNs) can be trained to classify engine sounds, identifying anomalies such as knocking, squealing, and hissing.

METHODOLOGY

❖ SYSTEM MODELLING AND DESIGN

System modeling is essential in software development as it provides a structured way to visualize, plan, and communicate how the system operates. It helps developers and stakeholders understand the system's components, their interactions, and behavior over time. Each diagram serves a specific purpose, contributing to a complete understanding of the system.

Importance of System Modelling

- ✓ **Clarifies requirements** – Helps define what the system should do by visualizing user needs and system functions.
- ✓ **Improves communication** – Provides a shared visual language for developers, stakeholders, and clients.
- ✓ **Defines system boundaries** – Clearly shows what is part of the system and what interacts externally (e.g., users, databases).
- ✓ **Identifies errors early** – Helps detect inconsistencies, missing logic, or flawed assumptions before development begins.
- ✓ **Supports better system design** – Guides the decomposition of the system into components and their interactions.
- ✓ **Aids documentation and maintenance** – Diagrams serve as long-term references for understanding and updating the system.
- ✓ **Enhances modularity and reusability** – Encourages development of reusable components and organized structure.
- ✓ **Facilitates simulation and testing** – Behavior models (like sequence or state diagrams) help simulate and test system workflows.
- ✓ **Supports team collaboration** – Ensures all team members have a consistent understanding of the system design.

1. CONTEXT DIAGRAM

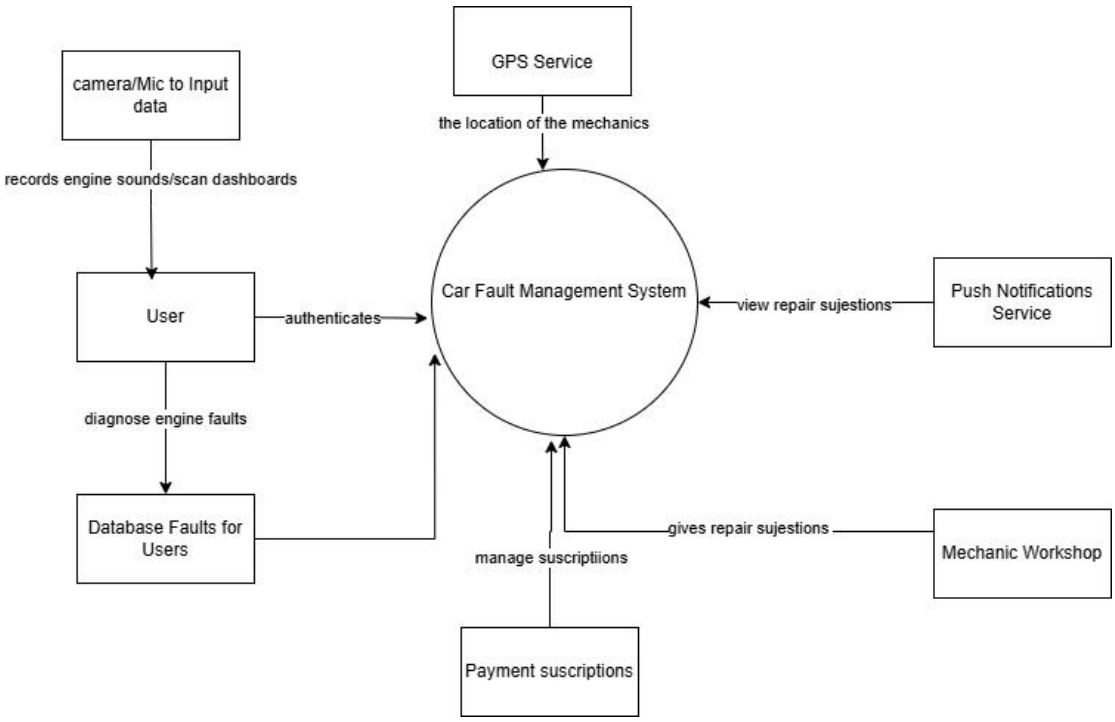
A **Context Diagram** is a **high-lvel visual representation** of a system that shows:

The **system as a single process or entity**.

All **external entities (actors or systems)** that interact with it.

The **data flows** between the system and those external entities.

- **Purpose:** Shows the overall system environment and its interactions with external entities. It helps in understanding the system boundaries.



DESCRIPTION

The context diagram describes system as a single process, shows all external entities interacting with the system, the data flow between the system and those external entities.

- **Main Elements of the App include**

System (center of the diagram): "Car Diagnostic Mobile App"

- **External Entities (outside of the system):**

User (main actor: vehicle owner/driver)

Mechanic Workshop

GPS Service (for location)

Payment Gateway (for subscriptions)

Push Notification Service

Camera & Microphone Hardware (as input sources)

- **Data Flows (between external entities and the system):**

1. The user can scan dashboard lights, record engine sounds, view detected faults, manage their account and settings, and make payments.

2. The camera and microphone provide input data such as images and engine sounds to the system.

3. The system sends alerts and notifications to the push notification service.

4. The system communicates with the GPS service to locate nearby mechanics.

5. The system exchanges payment information with the payment gateway for processing.

6. The system sends mechanic location data or optional appointment requests to nearby workshops.

7. The system provides scanned results, repair suggestions, and maintenance reminders back to the user.

2. DATA FLOW DIAGRAM (LEVEL 1)

A Data Flow Diagram (DFD) is a graphical representation of the flow of data within a system. It shows how data moves between processes, data stores, and external entities. DFDs are used in system analysis and design to visualize how information is input, processed, stored, and output. It has as key components: processes; data flows; data stores; external entities

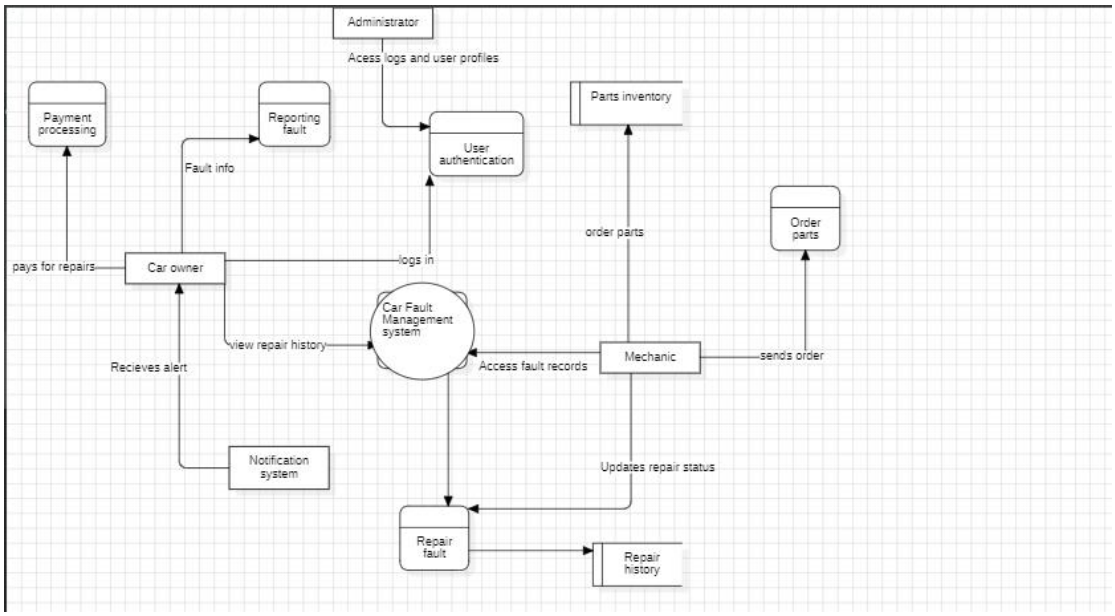
- **DFD Levels:**

Level 0 DFD (Context Diagram): Shows the entire system as a single process and its interactions with external entities.

Level 1, 2: Provide more detail, breaking down processes into sub-processes.

- **Purpose:**

DFDs help stakeholders understand how data moves in a system, identify inefficiencies, and design or improve systems effectively.



DESCRIPTION

This Data Flow Diagram (DFD) illustrates the Car Fault Management System, focusing on how data flows between different entities, processes, and data stores involved in reporting, inspecting, and repairing car faults.

- ✓ Car Owner (External Entity)

  Logs in to the system via the User Authentication process.

  Reports a fault using the Reporting Fault process, which sends fault info to the system.

  Pays for repairs via the Payment Processing system.

  Receives alerts through the Notification System.

  Views repair history fetched from the Car Fault Management System.

- ✓ **User Authentication (Process)**

  Authenticates the Car Owner and the Administrator when they log in.

  Allows access to appropriate components within the system.

- ✓ **Administrator (External Entity)**

  Logs in through User Authentication.

  Accesses logs and user profiles, likely for monitoring or administrative control.

- ✓ **Reporting Fault (Process)**

  Allows the Car Owner to submit fault information to the system for further diagnosis and repair handling.

- ✓ **Car Fault Management System (Central Process)**

  The central hub that:

  Handles login verification results.

  Stores and manages fault records.

  Allows the Mechanic to access fault records and update repair statuses.

  Lets the Car Owner view repair history.

  Triggers Notification System alerts to the Car Owner.

- ✓ **Mechanic (External Entity)**

  Accesses fault records from the Car Fault Management System.

  Updates repair statuses, which are stored in the Repair History data store.

  Orders parts from the Order Parts process

- ✓ **Repair Fault (Process)**

This process takes input from the Mechanic and processes fault repairs.

Outputs updated status to the Repair History data store.

- ✓ **Repair History (Data Store)**
  Stores completed repair records and updates.

  Can be queried by the Car Owner via the Car Fault Management System.

- ✓ **Order Parts (Process)**
  Receives part orders from the Mechanic.

  Sends requests to the Parts Inventory for fulfillment.

- ✓ **Parts Inventory (Data Store)**
  Stores available parts data.

  Supplies parts info when orders are made through the Order Parts process.

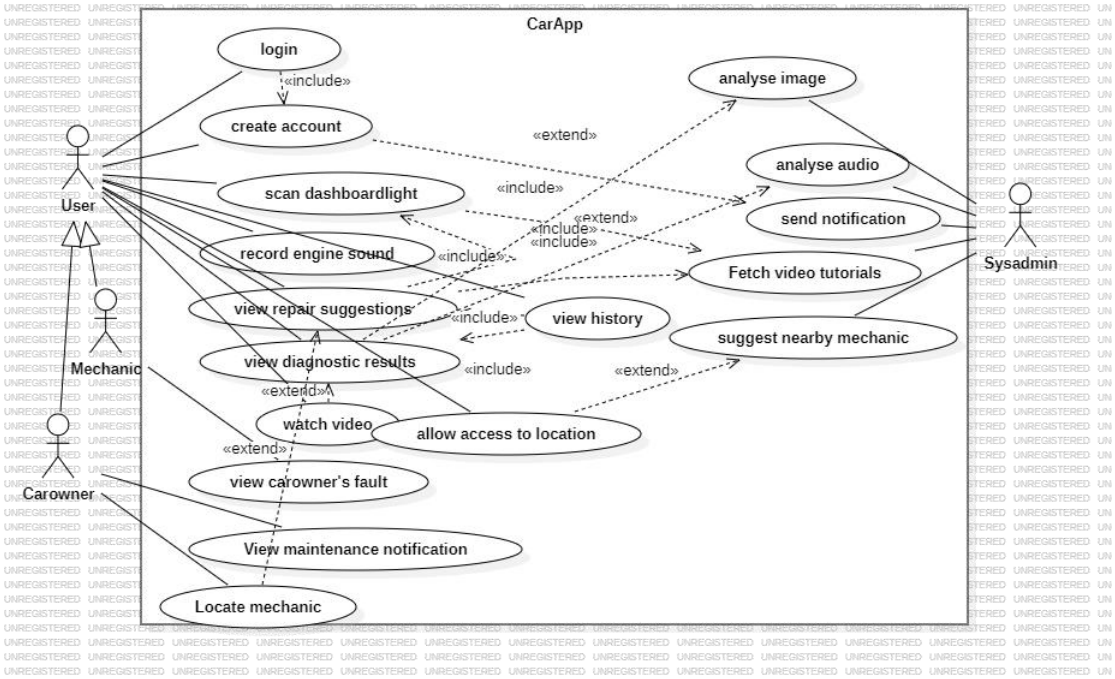
- ✓ **Payment Processing (Process)**
  Receives payment requests from the Car Owner for completed repair services.

- ✓ **Notification System (Process)**
  Sends alerts or status notifications to the Car Owner about fault progress or repair updates.

3. USE CASE DIAGRAM

A Use case diagrams provide a visual representation of how users/customers (actors) interact with the functionalities (use cases) of the system. It portrays the surface workings of the system alongside user involvement/participation with/within the system.

- **Purpose:** Identifies system functionalities and how different users (actors) interact with them.



Textual description of use case diagram

Actor	Use Case	Relationship	Associated Use Case	Description
User	Login	Include	Create Account	User logs in to access the CarApp. Creating an account is a prerequisite.
User	Create Account	-	-	User registers to use the app for the first time.
User	Scan Dashboard Light	Include	Analyse Image	User scans dashboard lights. The system analyzes the image.
User	Record Engine Sound	Include	Analyse Audio	User records engine sounds for diagnosis. The system

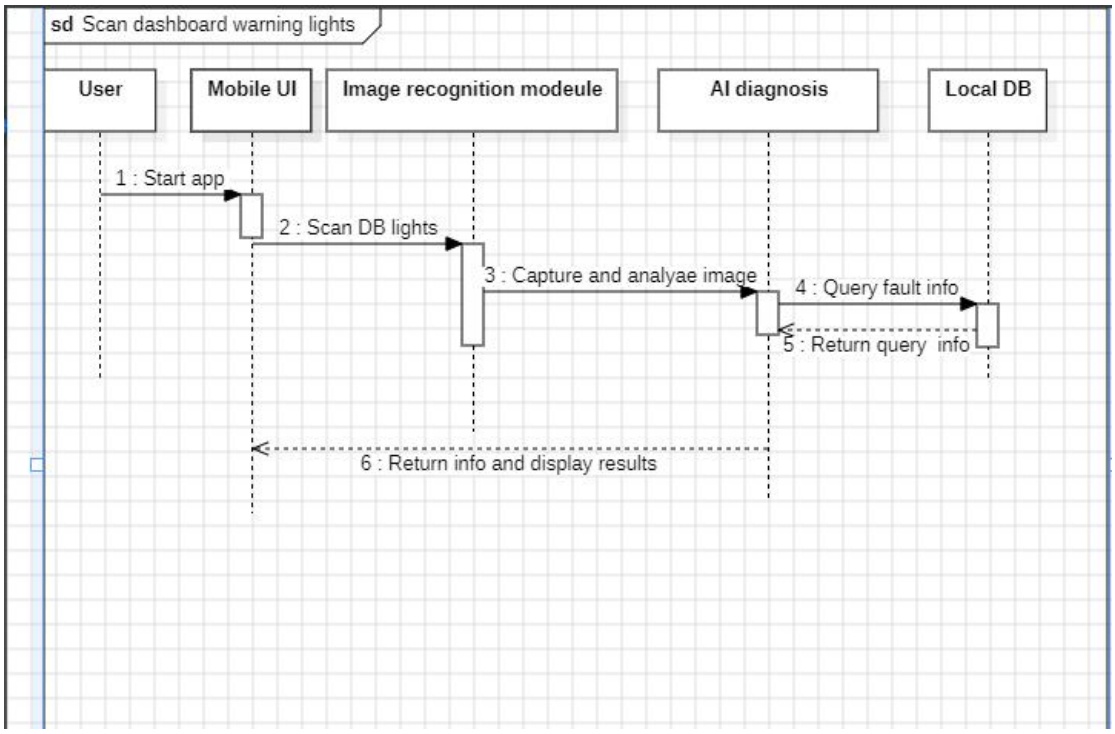
Actor	Use Case	Relationship	Associated Use Case	Description
				analyzes the audio.
User	View Repair Suggestions	Include	View Diagnostic Results	Displays potential repair solutions based on scan/audio analysis.
User	View Diagnostic Results	Extend	Watch Video	Shows diagnostic findings. May include tutorial video for further guidance.
User	Watch Video	Extend	Fetch Video Tutorials	Plays a video tutorial. Involves fetching video content.
User	View Car Owner's Fault	-	-	Displays the list of faults associated with the user's car.
User	View Maintenance Notification	-	Send Notification	Allows the user to see reminders or alerts related to car maintenance.
User	Locate Mechanic	Extend	Suggest Nearby Mechanic	User locates a nearby mechanic. The system suggests mechanics using GPS.
System	Analyse Image	-	-	Processes scanned dashboard images for diagnosis.
System	Analyse Audio	-	-	Processes recorded engine sounds for diagnosis.
System	Send Notification	-	-	Sends push notifications to the user (e.g., alerts, reminders).
System	Fetch Video Tutorials	-	-	Retrieves tutorial videos for the user.
System	Suggest Nearby Mechanic	-	Allow Access to Location	Suggests mechanics based on user's location. Requires location access.
User	View History	Include	View Repair Suggestions	Allows user to review past diagnostics and repairs.
System/User	Allow Access to Location	Extend	Suggest Nearby Mechanic	Grants permission to access user's location for mechanic suggestions.

4. SEQUENCE DIAGRAM

A sequence diagram is a type of interaction diagram that shows how objects operate with one another and in what order. It focuses on the sequence of messages exchanged between system components or actors to carry out a specific functionality or process.

- **Purpose:** Shows the sequence of interactions between system components or users to accomplish a task.

- **Scenario 1: User scans dashboard lights and receives diagnosis**



Participants (Lifelines):

User – The car owner using the mobile application.

Mobile UI – The user interface of the mobile app.

Image Recognition Module – The component responsible for processing the image of the dashboard.

AI Diagnosis – The intelligence layer that interprets the recognized symbols.

Local DB – A local database storing known faults and explanations.

Flow of Events:

Start App (User → Mobile UI):

The user launches the mobile application.

Scan DB Lights (Mobile UI → Image Recognition Module):

The user uses the camera to scan the dashboard lights.

The image is passed to the recognition module.

Capture and Analyze Image (Image Recognition Module → AI Diagnosis):

The module captures the image and sends it to the AI component for analysis and symbol interpretation.

Query Fault Info (AI Diagnosis → Local DB):

The AI module queries the local database to retrieve the meaning and possible implications of the detected dashboard symbols.

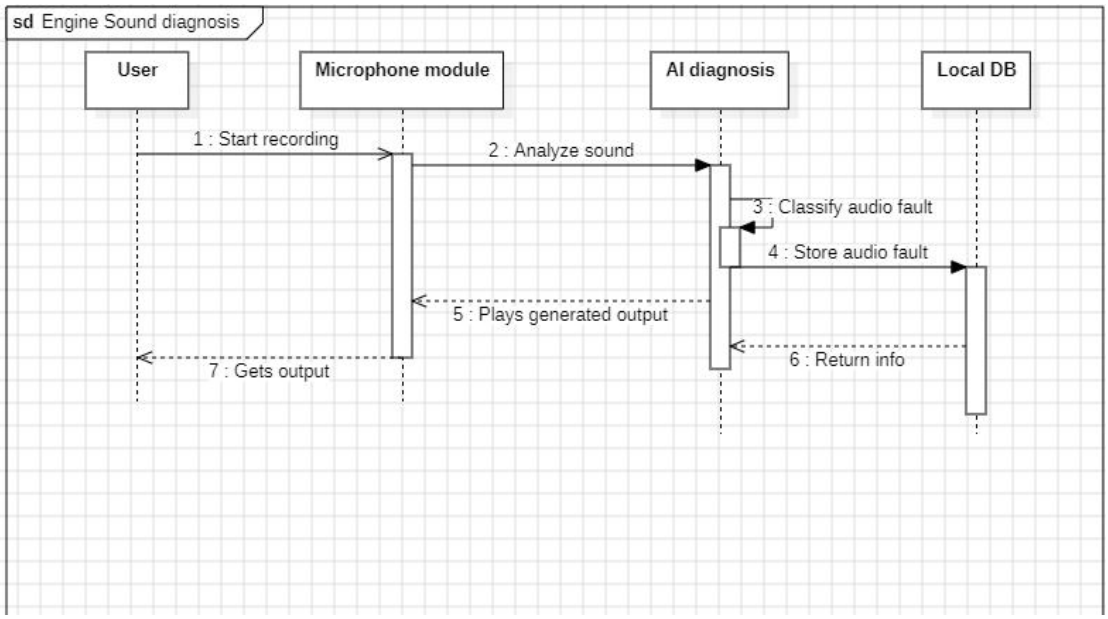
Return Query Info (Local DB → AI Diagnosis):

The database returns the matched information, including fault type, severity, and recommended actions.

Return Info and Display Results (AI Diagnosis → Mobile UI → User):

The processed diagnostic information is returned and displayed to the user in the mobile interface.

- **Scenario 2: Engine sound diagnosis**



Participants (Lifelines):

User – The individual using the mobile application to diagnose engine problems.

Microphone Module – Captures and processes the audio input from the user.

AI Diagnosis – The intelligence engine responsible for interpreting audio input and identifying faults.

Local DB – The local database that stores known faults and diagnostic information.

Flow of Events:

User → Microphone Module: Start recording

The user initiates the engine fault diagnosis by starting an audio recording of the engine's sound via the mobile app.

Microphone Module → AI Diagnosis: Analyze sound

Once the recording is completed, the microphone module processes and forwards the audio data to the AI diagnosis module for interpretation.

AI Diagnosis: Classify audio fault

The AI module uses machine learning models to classify the recorded engine sound and detect any fault patterns such as knocking or squealing.

AI Diagnosis → Local DB: Store audio fault

The identified fault is stored in the local database for reference and future tracking.

AI Diagnosis → Microphone Module: Plays generated output

A synthesized or textual summary of the diagnosis is generated and sent back to the microphone module, possibly for audio feedback.

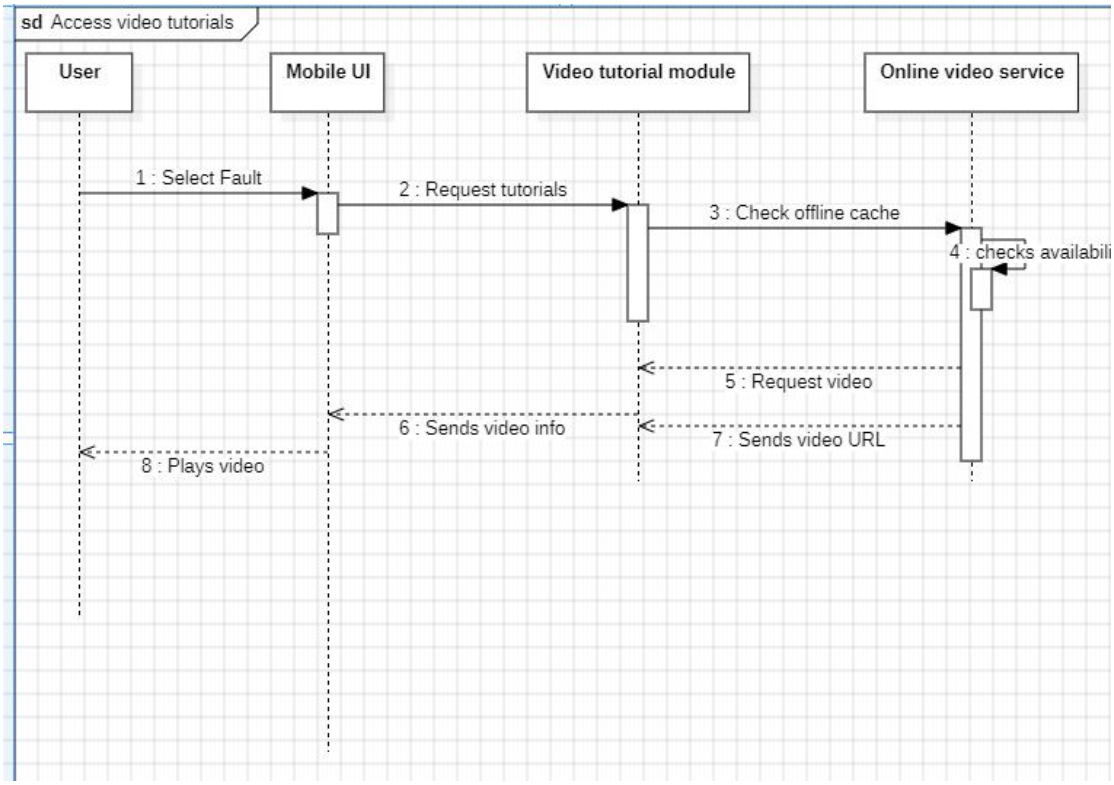
Local DB → AI Diagnosis: Return info

Additional information or historical context about the diagnosed fault is fetched from the local database and returned to the AI module.

Microphone Module → User: Gets output

The final diagnostic result is returned and displayed to the user through the mobile interface, including fault type and possible suggestions.

- **Scenario 3: Access video tutorials**



Participants (Lifelines):

1. **User** – The end-user who interacts with the system to access tutorials.
2. **Mobile UI** – The mobile application interface through which the user sends requests.
3. **Video Tutorial Module** – The system component responsible for managing tutorial content.
4. **Online Video Service** – The external service that hosts or streams the video content.

Flow of Events:

- ✓ **User selects a fault** (e.g., in a device or system they are troubleshooting) through the mobile UI.

  → **Message 1:** Select Fault

- ✓ The **Mobile UI** sends a request to the **Video Tutorial Module** to fetch relevant tutorials for the selected fault.

  → **Message 2:** Request tutorials

- ✓ The **Video Tutorial Module** first checks if the required tutorial is available in the **offline cache** (locally stored).

  → **Message 3:** Check offline cache

- ✓ If the video is not found locally, the system queries the **Online Video Service** to check availability.

  → **Message 4:** Check availability

- ✓ Once availability is confirmed, the **Video Tutorial Module** sends a request to the **Online Video Service** to fetch the video.

  → **Message 5:** Request video

  The **Online Video Service** responds by sending back the **video URL**.

  → **Message 6:** Sends video URL

- ✓ The **Video Tutorial Module** sends the video information (like title, description, and video URL) back to the **Mobile UI**.

  → **Message 7:** Sends video info

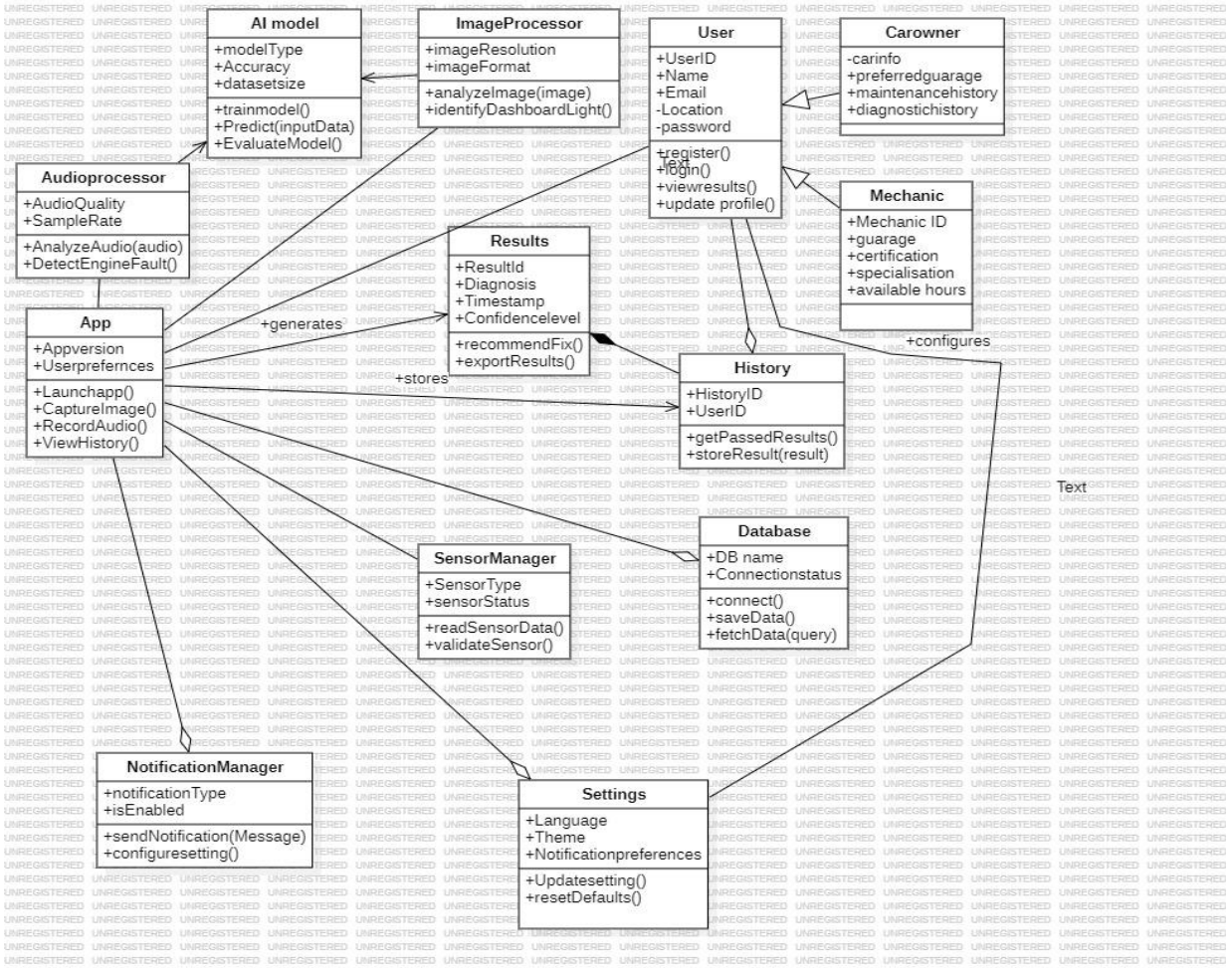
- ✓ The **User** then plays the video using the info provided.

  → **Message 8:** User plays video

5. CLASS DIAGRAM

The class diagram for the Car Fault Diagnosis Mobile Application presents a high-level overview of the system's structure by outlining the major classes involved, their attributes, methods, and the relationships between them. It represents the object-oriented design of the system, enabling effective planning and implementation of features that meet both user and technical requirements.

- **Purpose:** Displays the static structure of the system, including classes, attributes, methods, and relationships.



DESCRIPTION OF CLASSES

❖ **User (Abstract Class or Superclass)**

At the core of the system is the User class, which serves as a superclass for both CarOwner and Mechanic. This inheritance relationship allows shared attributes like userId, name, email, and location to be reused by both user types, ensuring consistency and reducing redundancy.

- ✓ **Attributes**: userId, name, email, location, password
- ✓ **Purpose:** A generic base class for all app users.
- ✓ **Relationships:** Inherited by **CarOwner** and **Mechanic.**

❖ **Car Owner (Inherits from User)**

It is a subclass of User and represents individuals who use the application to diagnose faults in
their personal vehicle

- ✓ **Attributes:** vehicleInfo, licensePlate, diagnosticHistory, preferredGarage
- ✓ **Purpose:** Represents a regular car user who uses the app for diagnosis.
- ✓ **Relationships:** Has a list of Diagnostic result entries, can schedule or contact a Mechanic.
- ✓

❖ **Mechanic (Inherits from User)**

It is a subclass of User, is designed for individuals who provide repair services

- ✓ **Attributes:** certification, specialization, Location, yearsOfExperience, repairLogs
- ✓ **Purpose**: Technical user who may help car owners understand or fix issues.
- ✓ **Relationships**: May receive service requests from CarOwner, may add or update RepairLog.

❖ **App**

It facilitates communication between the other classes. It is the primary class.

- ✓ **Attributes**: appVersion, userPreferences
- ✓ **Purpose**: Represents the mobile app system.
- ✓ **Relationships:** Coordinates interactions between users and processing modules.

❖ ImageProcessor

This class helps in analysing data collected from the sensor;camara through the AI model.

- ✓ **Attributes**: imageResolution, imageFormat
- ✓ **Methods**: analyzeImage(), identifyDashboardLight()
- ✓ **Purpose:** Processes images (e.g. of dashboard warning lights).
- ✓

❖ Audio-processor

This class helps in analyzing data collected from the sensor; microphone through the AI model.

- ✓ **Attributes:** audioQuality, sampleRate
- ✓ **Methods**: analyzeAudio(), detectEngineFault()**Purpose:** Processes engine sounds for fault analysis.
- ✓

❖ AIModel

This class is focused on delivering results after analysis from the two processors. Its primary focus is to ensure maximum accuracy in results.

- ✓ **Attributes**: modelType, accuracy, datasetSize
- ✓ **Methods:** trainModel(), predict(), evaluateModel()
- ✓ **Purpose**: Core AI engine behind diagnostics, used by both Image and Audio processors.

❖ DiagnosticResult

Stores and presents output of diagnostics.

- ✓ **Attributes:** resultId, diagnosis, confidenceLevel, timestamp
- ✓ **Purpose**: Holds the output of a diagnosis session.
- ✓ **Relationships:** Linked to CarOwner ,Generated by the app through AI processors.

❖ SensorManager

Manages device sensors; microphone and camera.

- ✓ **Attributes:** sensorType, sensorStatus
- ✓ **Methods:** readSensorData(), validateSensor()
- ✓ **Purpose:** Deals with hardware sensors like camera and mic

❖ **Database**

The class stores all data collected from all other classes

- ✓ **Attributes**: dbName, connectionStatus
- ✓ **Methods**: connect(), saveData(), fetchData()
- ✓ **Purpos**e: Responsible for data storage and retrieval.

❖ **Notification manager**

Handles system notifications to the user.

- ✓ **Attributes:** notificationType, isEnabled
- ✓ **Methods**: sendNotification(), configureSettings()
- ✓ **Purpose:** Sends alerts or updates to users.

❖ **Settings**

Allows users to adjust app configuration

- ✓ **Attributes**: language, theme, notificationPreferences
- ✓ **Methods**: updateSettings(), resetDefaults()
- ✓ **Purpose**: User app preferences.

2. **Relationships**

1. **Relationships**

Class From	Class To	Type	Description
User	CarOwner, Mechanic	Inheritance	Specialized user types
CarOwner	DiagnosticResult	Association (1..*)	A car owner can have multiple diagnostic records
Mechanic	RepairLog	Association (1..*)	Mechanics log repairs they perform
App	ImageProcessor, AudioProcessor	Uses	App sends data for AI analysis
ImageProcessor, AudioProcessor	AIModel	Association	AI used for inference
App	SensorManager	Aggregation	Reads sensor inputs
App	Database	Association	Persists data
App	NotificationManager	Aggregation	Sends user alerts
App	Settings	Association	Applies user preferences

5. DEPLOYMENT DIAGRAM

A Deployment Diagram is a type of UML diagram that shows the physical deployment of software components on hardware nodes. It illustrates how software systems are deployed across hardware infrastructure, such as servers, devices, and networks.

➢ Purpose:

• To model the physical architecture of a system.

• To show where and how software components are deployed.

• Useful for system administrators and network engineers for planning system infrastructure.

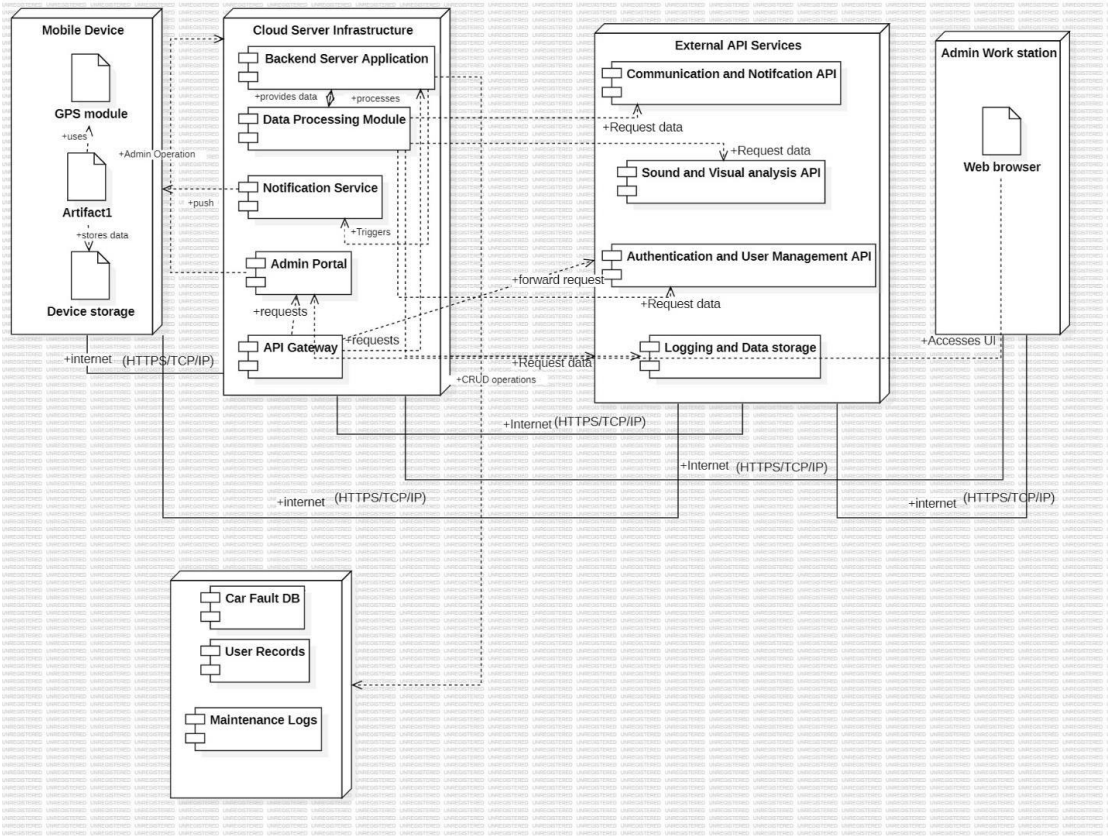
Deployment diagrams help in understanding system topology and planning real-world deployment scenarios.

❖ Key Elements of a Deployment Diagram:

1. Nodes – Represented by 3D boxes. These are physical or virtual hardware devices (e.g., servers, computers, mobile devices).

2. Artifacts – Represent the software components (e.g., applications, executables, databases) that are deployed on the nodes.

3. Communication Paths – Represented by lines between nodes. They show how nodes communicate with each other (e.g., through a network).



DESCRIPTION

This is a UML Deployment Diagram for a Car Fault Management System, showing how software components are distributed across hardware nodes, how they interact, and how data flows across the system. The diagram is made of:

1. Mobile Device Node

• **Hardware**: A user's mobile device (smartphone).

• **Deployed Components**:

➢ **GPS Module**: Used to detect vehicle location.

➢ **Artifact1**: Likely represents the compiled mobile application.

➢ **Device Storage**: Stores local logs, possibly image/sound data, temporary faults before sync.

• **Communication**:

➢ **Pushes** data to the cloud backend.➢ Uses **Internet (HTTPS/TCP/IP)** to communicate with Cloud Server Infrastructure.

➢ Interacts via **Admin Operation** path (e.g., login, profile updates).

2. Cloud Server Infrastructure Node

• Acts as the backend core where all data is processed and logic is executed.

• **Components Deployed**:

➢ **Backend Server Application**: Main logic handler. Receives data from devices, orchestrates communication.

➢ **Data Processing Module**: Processes raw input (sound, image, location, fault logs).

➢ **Notification Service**: Sends alerts/notifications based on detected faults.

➢ **Admin Portal**: Web interface used by admins (accessed via browser).

➢ **API Gateway**: A central access point for routing all API requests.

• **Interactions**:

➢ **Requests** flow from Admin Portal/API Gateway to other services or APIs.

➢ Triggers notifications to be sent when events occur.

➢ **Processes** data sent by the backend.

3. External API Services Node

• Contains third-party services used to enhance functionality.

• **APIs Deployed**:

➢ **Communication and Notification API**: Sends SMS, push notifications, or emails.

➢ **Sound and Visual Analysis API**: Uses AI or ML to analyze sounds/images from the vehicle.

➢ **Authentication and User Management API**: Handles sign-in, roles, and access control.

➢ **Logging and Data Storage**: Stores logs and tracks API activity for security/auditing.

• **Communication**:

➢ Cloud infrastructure forwards request to external APIs (e.g., for authentication, data
analysis).

➢ Some requests go directly from Admin Portal or the Mobile App via API Gateway.

4. Database Server Node

• Hosts persistent data storage for the system.

• **Databases**:

➢ **Car Fault DB**: Records of detected car issues.

➢ **User Records**: Information about car owners, technicians, admin accounts.

➢ **Maintenance Logs**: Historical data on repairs, diagnoses, and alerts.

• **Communication**:

➢ CRUD operations (Create, Read, Update, Delete) from backend to database.

➢ Connections happen over HTTPS/TCP/IP securely.

5. Admin Workstation Node

• **Hardware**: A desktop or laptop used by an admin.

• **Component**:

➢ **Web Browser**: Used to access the Admin Portal hosted in the Cloud Server.

• **Communication**:

➢ Accesses the UI via Internet (HTTPS), sends requests (e.g., view logs, modify users).

➢ May access APIs directly through the Admin Portal as well.

❖ Purpose of the Diagram:

This deployment diagram describes how software components (UI, services, databases) are distributed across physical devices (mobile, application server, and database server), including their interactions. It is useful for developers and system architects to understand the infrastructure setup and plan deployment and maintenance.

CONCLUSION

This project introduces an innovative, AI-driven solution to empower car owners with tools to independently diagnose vehicle issues. The mobile application integrates computer vision and audio recognition to analyze dashboard lights and engine sounds, providing actionable feedback and educational resources. With both offline and online functionalities, it ensures accessibility and continuous improvement through cloud-based updates.

Further enhancements may include integration with OBD-II adapters for real-time engine diagnostics, multilingual support, and AI-driven predictive maintenance features.