REPUBLIC OF CAMEROON

Peace-Work-Fatherland

MINISTRY OF HIGHER EDUCATION

UNIVERSITY OF BUEA

REPUBLIQUE DU CAMEROUN

Paix-Travail-Patrie

MINISTRE DE L'ENSEGNEMENT SUPERIEUR

UNIVERSITE DE BUEA

UNIVERSITY OF BUEA

FACULTY OF ENGINEERING AND TECHNOLOGY

SYSTEM MODELLING AND DESIGN

TASK 4 BY GROUP 25

COURSE INSTRUCTOR: **Dr NKEMENI VALERY**

COURSE CODE/TITLE: **CEF 440/ INTERNET PROGRAMMING(J2EE) AND MOBILE PROGRAMMING.**

ACADEMIC YEAR 2024/2025

Table of Contents

1. Introduction

This report presents the comprehensive system modeling and design for the Automotive Diagnostic Mobile Application. The system is designed to help car owners diagnose vehicle problems through dashboard warning light recognition, engine sound analysis, and provide connectivity to professional mechanics. The modeling approach follows industry-standard UML practices to ensure clear communication of system architecture and functionality.

1.1 Project Scope

The automotive diagnostic application addresses the growing need for accessible vehicle diagnostics tools that can help car owners understand their vehicle's condition and make informed decisions about maintenance and repairs. The system integrates advanced AI-powered diagnostic capabilities with user-friendly interfaces and professional service connectivity.

1.2 Design Objectives

- Provide accurate and timely vehicle diagnostics
- Enable non-technical users to understand vehicle problems
- Connect car owners with professional mechanics
- Maintain comprehensive vehicle history and maintenance records
- Support offline functionality for critical diagnostic features

2. Context Diagram

A Context Diagram is a high-level visual tool used in systems analysis, software engineering, and business process modeling to illustrate the scope of a system and its interactions with external entities. It serves as the foundation for understanding system boundaries.

a) Introduction:

Represents the entire system as a single process (black box).

Shows external entities (actors, users, or systems) that interact with it.

Displays data flows (inputs and outputs) between the system and external components.

Importance of context diagram.

✔ Clearly defines system boundaries (what’s inside vs. outside).

✔ Helps stakeholders (non-technical & technical) understand the system at a glance.
✔ Serves as a starting point for detailed system design.

b) Key Components

A Context Diagram consists of three main elements:

Component	Description	Example
Central System (Process)	The main system being analyzed, represented as a single box.	"car fault diagnosis application"
External Entities (Actors)	People, organizations, or systems that interact with the central system.	"drivers," "car owners," "dashboard lights and signals" “engine sound”
Data Flows (Arrows)	Movement of information between the system and external entities.	"scanning of dashboard light and signals,"

c) How we Created the Context Diagram?

Step 1: Identify the System

Define the central process that is a car fault diagnosis app

Step 2: List External Entities

Who or what interacts with the system? (e.g., "drivers," "car dashboard," "engine").

Step 3: Define Data Flows

What information is exchanged? (e.g scanning dashboard lights)

Identification of Components of the Context Diagram

- **Central System (Process):**

- "Car Fault Diagnosis App" – The core system that processes inputs (dashboard lights, engine sounds) and outputs diagnostics.

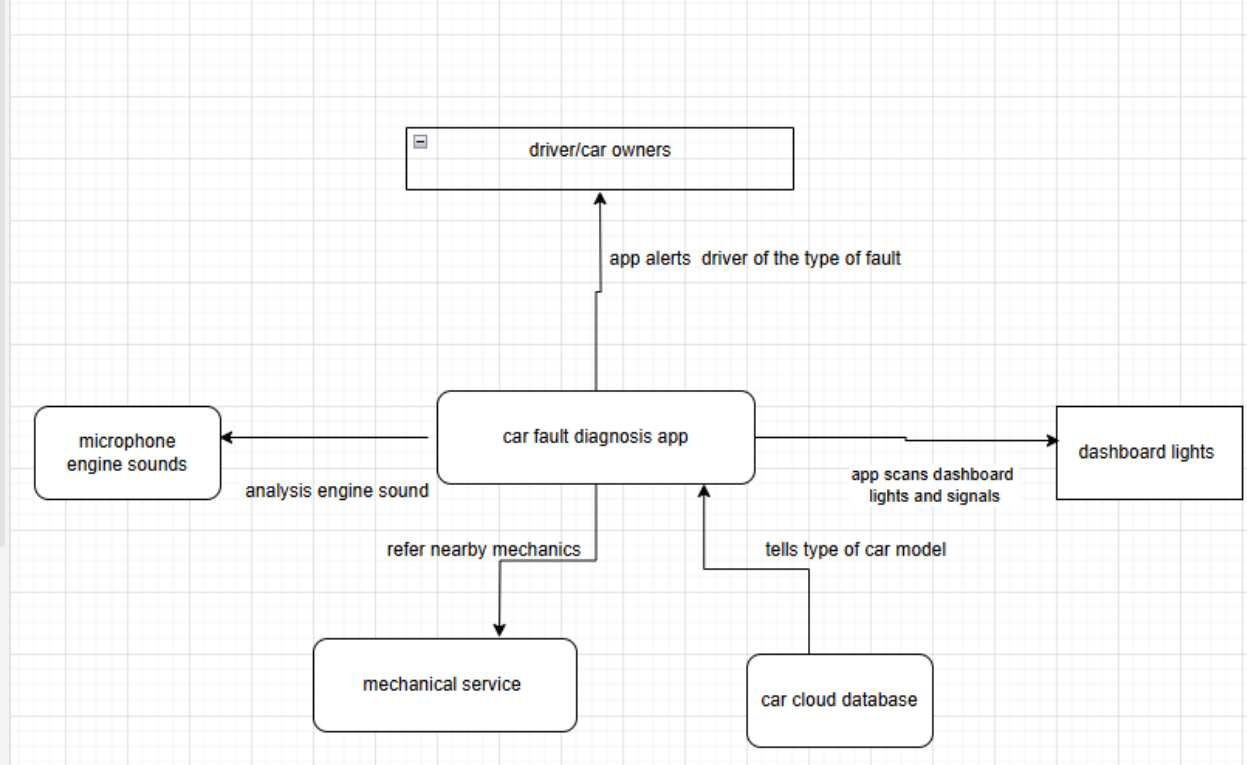
- **External Entities (Actors):**

- Driver – Interacts with the app (inputs queries, receives alerts).
- Vehicle Dashboard (OBD-II Port/Sensors) – Provides real-time error codes and light signals.
- Microphone (Engine Sound Analyzer) – Captures engine noise for anomaly detection.
- Mechanic/Service Center – Receives diagnostic reports for repairs.
- Vehicle Database (Cloud/API) – Stores fault patterns, repair manuals, and known issues.

d) Data Flows (Inputs & Outputs):

From → To	Data Flow Description
Driver → Car Fault Diagnosis App	"Scan Request," "Manual Input (Symptoms)"
Vehicle Dashboard → App	"OBD-II Error Codes," "Dashboard Light Signals"
Microphone → App	"Engine Sound Recording"
App → Driver	"Fault Report," "Recommended Actions," "Emergency Alert"
App → Mechanic/Service Center	"Diagnostic Report," "Repair Suggestions"
Vehicle Database → App	"Known Error Patterns," "Repair Manuals"

e) Visual Representation, drawn using draw.io



f) Explanation of Interactions

1- Driver Interaction:

Requests a scan or manually inputs symptoms (e.g., "Check Engine Light is ON").

Receives a diagnostic report (e.g., "Fault: Oxygen Sensor Failure – Recommended: Replace Sensor").

2- Vehicle Dashboard (OBD-II/Sensors):

Sends real-time error codes (e.g., P0420 – Catalyst System Efficiency Below Threshold).

Detects dashboard warning lights (e.g., ABS, Oil Pressure).

3- Microphone (Sound Analysis):

Records engine noise to detect abnormal sounds (e.g., knocking, misfiring).

Matches patterns with known fault signatures.

4- Vehicle Database (Cloud/API):

Provides historical fault data and repair guides.

Updates the app with new error patterns.

5- Mechanic/Service Center:

Receives automated diagnostic reports for faster servicing

conclusively

This context diagram effectively captures the high-level interactions of a Car Fault Diagnosis App.

3. Data Flow Diagram (DFD)

The DFD describes the flow of information within a **mobile application** designed for **car fault diagnosis**. It shows how a **Car Owner (user)** interacts with various components of the app to diagnose engine and dashboard issues using **audio recordings** and **camera scans**. The system leverages **AI models**, **local data**, and **online resources** to provide diagnostic results and tutorials.

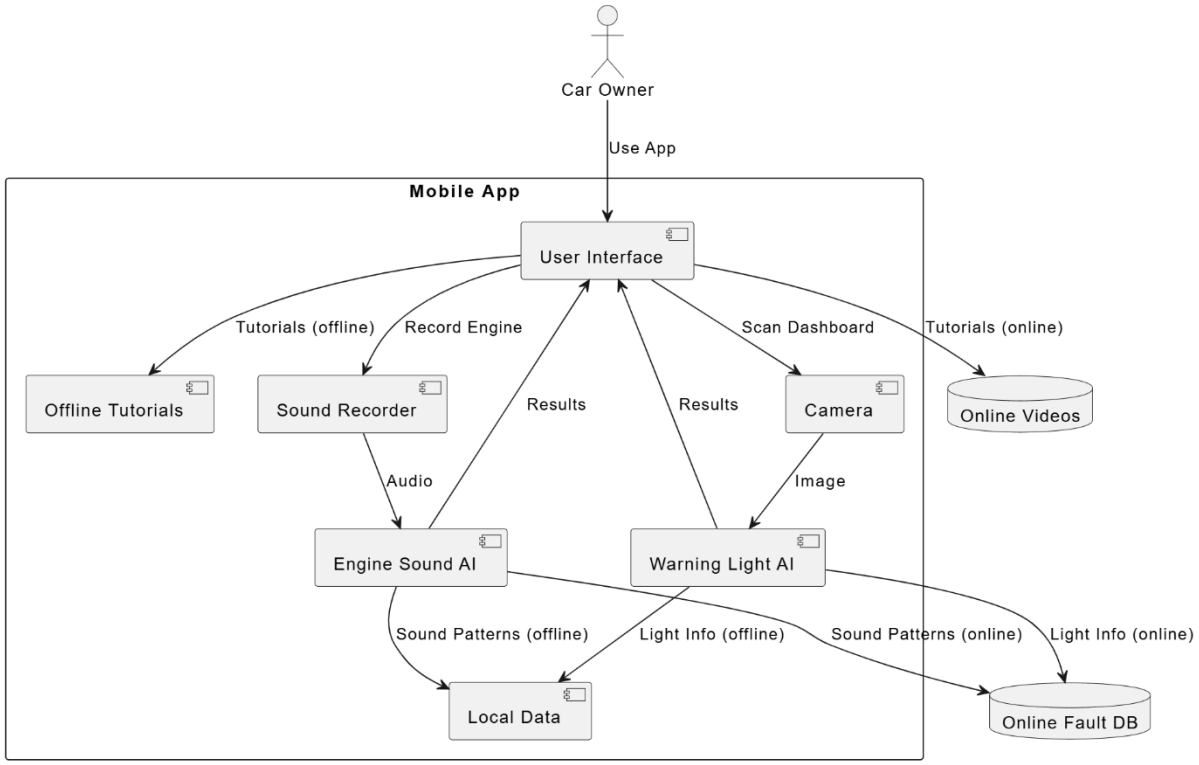
1. External Entity

- **Car Owner:** The end-user of the mobile application.

  Initiates the process by using the app to record engine sounds, scan dashboard lights, or view tutorials.

2. Main Process: Mobile App

The mobile app acts as the central process that houses several subcomponents, each performing a specific task:



3. Key Components and Their Roles

3.1 User Interface

- Acts as the central interaction point for the car owner.
- Connects to all other modules, directing user inputs (e.g., record, scan, tutorial) to appropriate components.
- Displays diagnostic **results** returned from AI modules.

3.2 Sound Recorder

- Allows the user to **record engine sounds**.
- Passes audio data to the **Engine Sound AI** for analysis.

3.3 Camera

- Used to **scan the dashboard** and capture **images of warning lights**.
- Sends image data to the **Warning Light AI** module for interpretation.

3.4 Engine Sound AI

- Processes recorded audio to identify **fault patterns**.
- Relies on **Local Data** and **Online Fault DB** to interpret sound patterns.

- Returns fault diagnosis results to the **User Interface**.

3.5 Warning Light AI

- Analyzes dashboard images to interpret **warning light meanings**.
- Uses data from both **Local Data** and **Online Fault DB** for reference.
- Sends diagnostic results to the **User Interface**.

3.6 Local Data

- Stores sound patterns and light information **offline**.
- Acts as a quick-access knowledge base for the AI modules when internet is unavailable.

3.7 Online Fault DB

- A cloud-based database containing a large set of sound patterns and light indicators.
- Supports the AI modules with **updated** and **comprehensive** diagnostic knowledge.

3.8 Offline Tutorials

- Provides educational content on using the app and basic car maintenance.
- Can be accessed via the **User Interface** without internet

3.9 Online Videos

- Hosts **tutorial content** online for advanced or updated learning.
- Accessed via the **User Interface** when an internet connection is available.

4. Data Flows

The diagram includes the following major data flows:

From	To	Data
Car Owner	User Interface	Use App
User Interface	Sound Recorder	Record Engine
User Interface	Camera	Scan Dashboard
Sound Recorder	Engine Sound AI	Audio
Camera	Warning Light AI	Image
Engine Sound AI	User Interface	Results
Warning Light AI	User Interface	Results

Engine Sound AI	Local Data	Sound Patterns (offline)
Warning Light AI	Local Data	Light Info (offline)
Engine Sound AI	Online Fault DB	Sound Patterns (online)
Warning Light AI	Online Fault DB	Light Info (online)
User Interface	Offline Tutorials	Tutorials (offline)
User Interface	Online Videos	Tutorials (online)

5. System Functionality Summary

Functionality	**Description**
Engine Sound Diagnosis	Uses audio recordings analyzed via AI to detect engine faults.
Warning Light Detection	Uses image recognition to identify dashboard warnings.
Offline Functionality	Supports diagnosis using local datasets and offline tutorials.
Online Connectivity	Enhances diagnosis with an online fault database and tutorial videos.
User-Friendly UI	Central access point for all app features, making the experience smooth and guided.

6. Strengths of the System

- **AI Integration**: Efficient fault detection using machine learning.
- **Hybrid Data Sources**: Uses both local and cloud-based data for flexibility and robustness.
- **User Accessibility**: Supports both online and offline usage, increasing usability in low-connectivity environments.
- **Educational Support**: Offers tutorials to educate users on diagnostics and car maintenance.

This DFD effectively captures the architecture and workflow of the **Car Fault Diagnosis Mobile Application**. By combining AI-based diagnostics with user-friendly design and both online/offline capabilities, the app serves as a powerful tool for car owners to independently monitor and troubleshoot vehicle issues.

4. Use Case Diagram

a) Actor Identification

Primary Actors:

- **Car Owner**: Performs vehicle diagnostics, manages vehicle information, and accesses maintenance resources

Secondary Actors:

- **System Administrator**: Manages system configuration, user accounts, and knowledge base updates
- **System**: Manages the system configuration also known as the app itself.
- **External Services**: Provides third-party integrations for payments, maps, and data validation

b) Use Case Packages

1) Diagnostic Features Package

This package contains the core functionality for vehicle problem identification and analysis.

Key Use Cases:

- **Recognize Dashboard Warning Lights**: Process uploaded images to identify warning lights and provide explanations
- **Analyze Engine Sounds**: Process audio recordings to detect potential engine problems
- **Perform Problem Diagnosis**: Combine multiple inputs to provide comprehensive vehicle diagnosis
- **Estimate Repair Urgency**: Classify problems by severity and recommend action timelines

Include Relationships:

- Warning light recognition includes image upload and processing
- Sound analysis includes audio recording and pattern matching
- Problem diagnosis includes symptom correlation and confidence rating

2) User Management Package

Handles user authentication, profile management, and account security.

Key Use Cases:

- **Create User Account**: New user registration with profile setup
- **Login to System**: User authentication with biometric support
- **Manage User Profile**: Update personal information and preferences
- **Reset Password**: Secure password recovery process

3) Vehicle Management Package

Manages vehicle information, maintenance history, and service tracking.

Key Use Cases:

- **Add Vehicle Information**: Register new vehicles with VIN validation
- **Track Maintenance Records**: Monitor service history and upcoming maintenance
- **Generate Maintenance Reminders**: Automated alerts based on mileage and time intervals
- **Export Vehicle Data**: Share vehicle history with mechanics or for personal records

4) Support Features Package

Provides additional services including mechanic connectivity and knowledge resources.

Key Use Cases:

- **Find Nearby Mechanics**: Location-based search for qualified service providers
- **Share Diagnostic Results**: Send diagnostic reports to selected mechanics
- **Access Knowledge Base**: Search common problems and DIY repair guides
- **Use Offline Diagnostics**: Core diagnostic functions available without internet

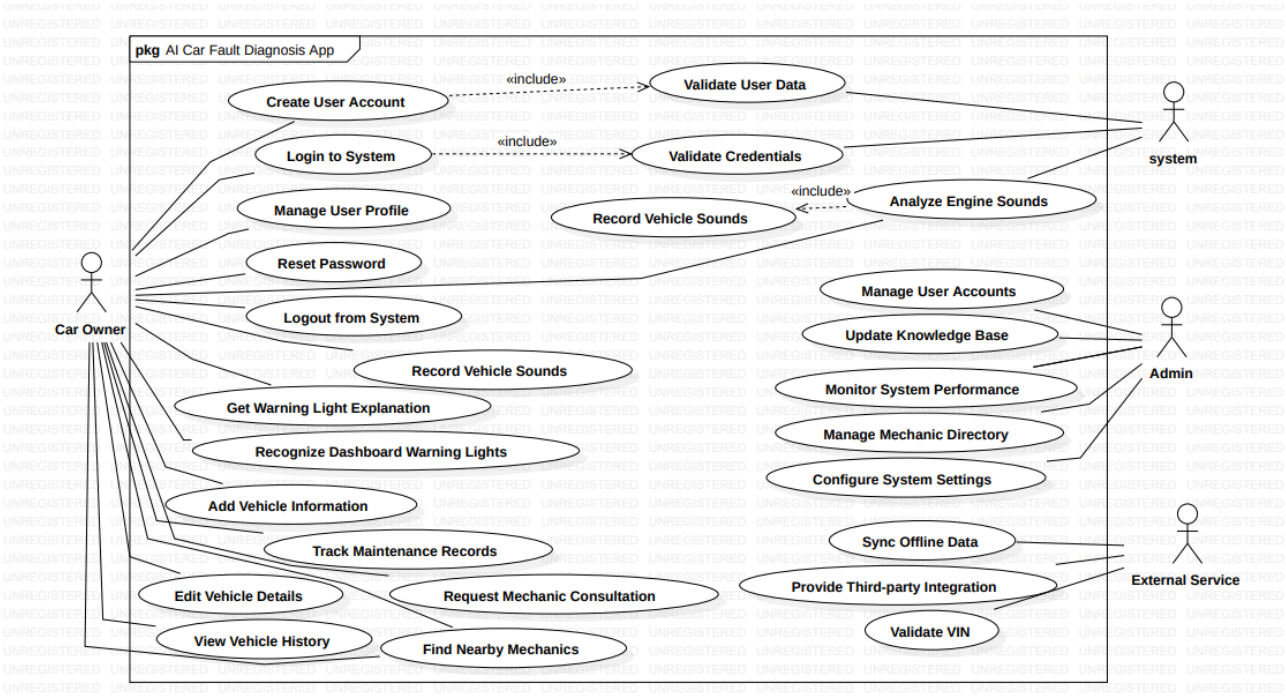
Relationships Analysis

Extend Relationships:

- Biometric authentication extends standard login for enhanced security
- Mechanic consultation extends diagnostic result sharing for professional advice
- Advanced diagnostics extends basic problem diagnosis for complex issues

Generalization Relationships:

- Audio and visual diagnostics generalize to diagnostic processing
- Different user types generalize to system user base class

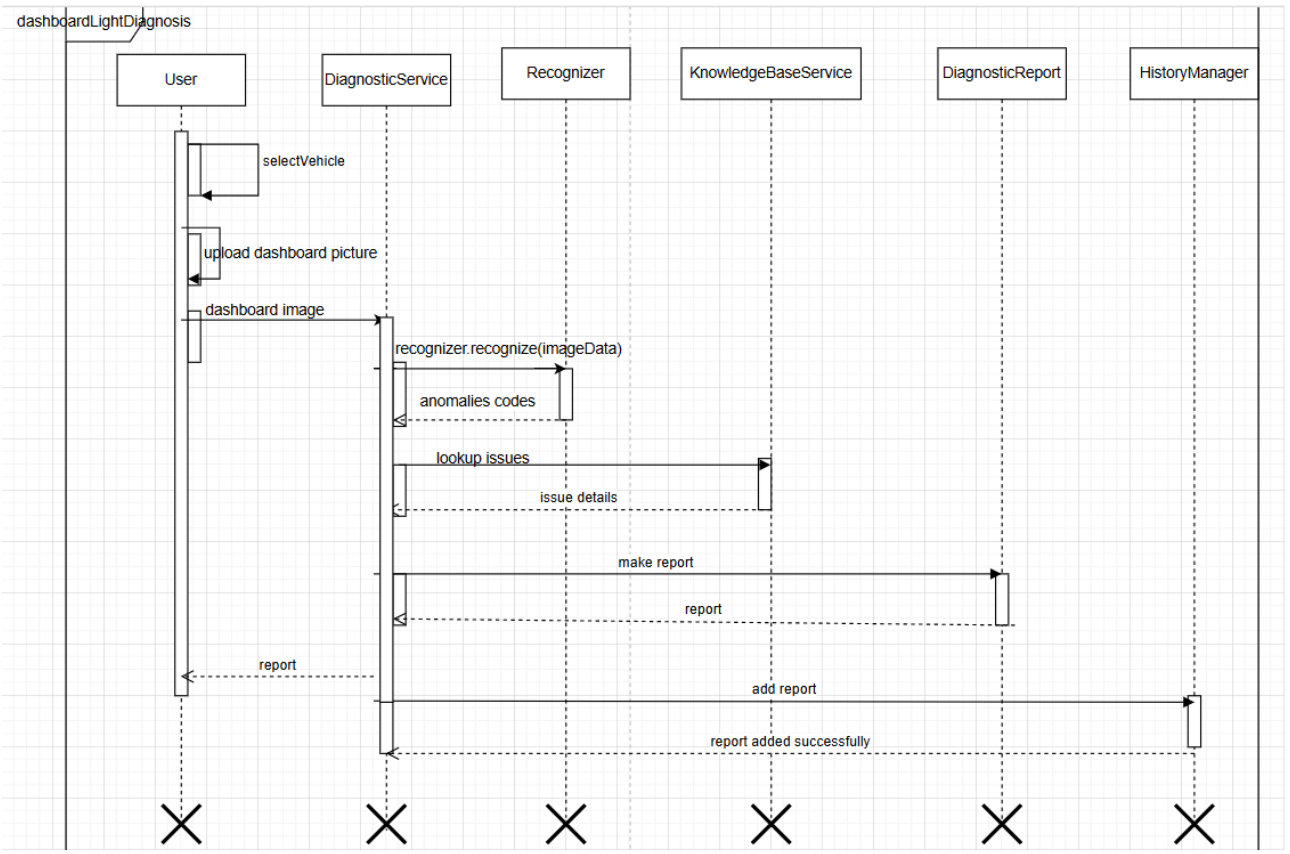


5. Sequence Diagram

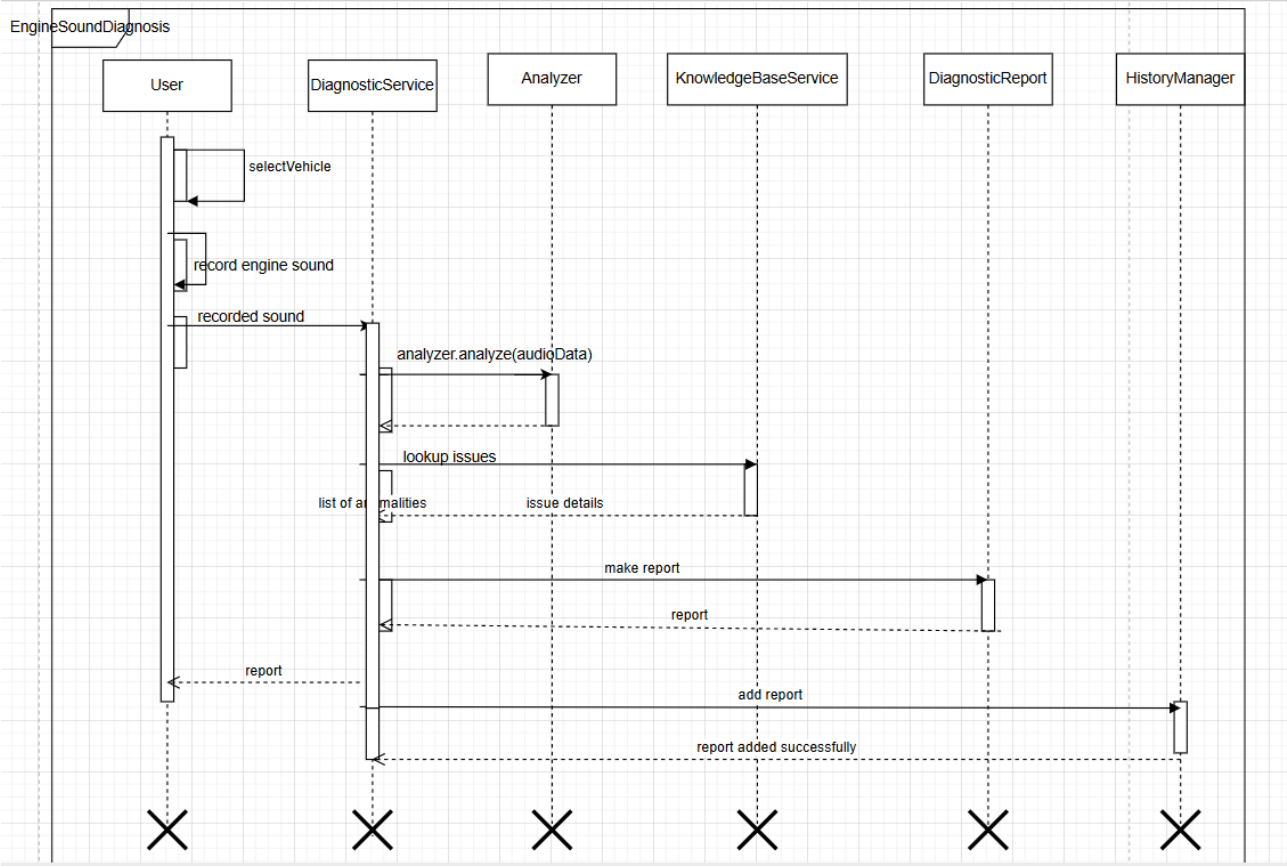
A sequence diagram is a type of interaction diagram in the Unified Modeling Language (UML) that models the dynamic behavior of a system by showing how and in what order a set of objects exchange messages over time.

The sequence diagram for the two main functions (Dashboard Light Scanning and Sound Engine Analysis) of the system are given below.

Dashboard Light Scanning Sequence diagram



Sound Engine Analysis sequence diagram



6. Class Diagram

A class diagram is a type of static structural diagram in the Unified Modeling Language (UML) that provides a blueprint of a system's object-oriented architecture. It represent the static interaction of the system. The different classes involved in the system are given below providing their properties and methods.

User

Properties

- **userId: String:** Unique identifier for the user.
- **name: String:** The user's full name.
- **email: String:** The user's login address and contact point for notifications.
- **passwordHash: String:** A secure hash of the user's password.

Methods

- **login(): boolean:** Verifies credentials and returns true if authentication succeeds.
- **logout(): void:** Ends the user's session.
- **addVehicle(v: Vehicle): void:** Associates a new Vehicle with this user.

- **removeVehicle(v: Vehicle): void:** Disassociates a Vehicle from the user.

Vehicle

Properties

- **vin: String:** Vehicle Identification Number.
- **make: String:** Manufacturer of the vehicle.
- **model: String:** Specific model name.
- **year: int:** Manufacture year.

Methods

- **getDetails(): String:** Returns a human-readable summary of the vehicle.
- **setReminder(m: MaintenanceReminder): void:** Attaches a maintenance reminder to this vehicle.

WarningLightRecognizer

Properties

- **modelPath: String:** Path to the ML model used for image recognition.

Methods

- **recognize(imageData: ImageData): List<WarningLight):** Detects warning lights in a dashboard image.

EngineSoundAnalyzer

Properties

- **config: AnalysisConfig:** Parameters used for audio analysis.

Methods

- **analyze(audioData: AudioData): List<SoundAnomaly):** Processes engine noise and returns anomalies.

DiagnosticService

Methods

- **diagnose(imageData: ImageData, audioData: AudioData): DiagnosticReport:** Produces a consolidated diagnostic report.

DiagnosticReport

Properties

- **timestamp: Date:** When the diagnosis was performed.
- **severity: SeverityLevel:** Seriousness of the issues found.
- **causes: List<String>:** List of probable causes.
- **confidence: float:** Confidence level of the findings.

Methods

- **generateSummary(): String:** Compiles a brief narrative of the findings.

HistoryManager

Properties

- **records: List<DiagnosticReport>:** Log of all past diagnostics.

Methods

- **addReport(r: DiagnosticReport): void:** Adds a new report to the history.
- **getHistory(): List<DiagnosticReport>:** Retrieves past reports.

MechanicService

Methods

- **findNearby(location: GeoLocation): List<Mechanic>:** Finds nearby mechanics.
- **shareReport(mech: Mechanic, report: DiagnosticReport): boolean:** Sends the report to a mechanic.

Mechanic

Properties

- **name: String:** Mechanic's or business name.
- **address: String:** Physical workshop address.
- **contactInfo: String:** Phone or email contact.

Methods

- **contact(): String:** Returns formatted contact information.

KnowledgeBaseService

Properties

- **dbConnection: DatabaseConnection:** Connection to the issue-and-repair database.

Methods

- **lookupIssue(id: String): IssueDetail:** Fetches issue details from the database.
- **getDIYGuides(id: String): List<Guide>:** Returns troubleshooting guides.

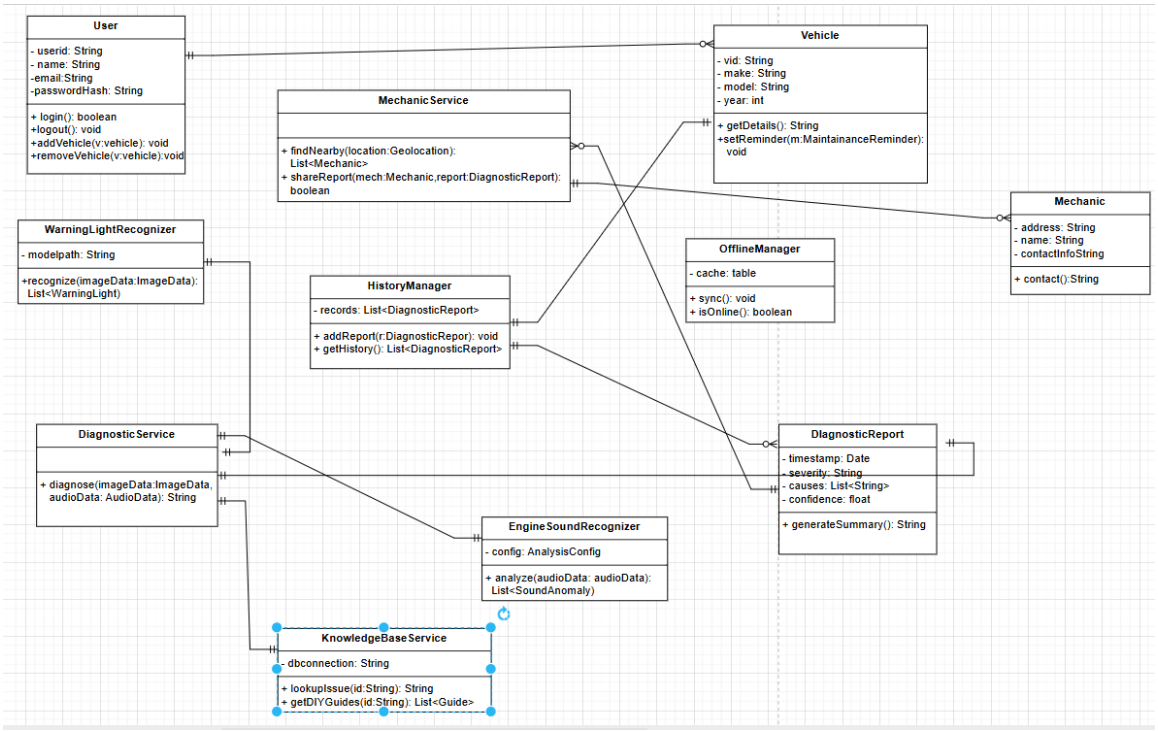
OfflineManager

Properties

- **cache: CacheStore:** Local storage for caching data offline.

Methods

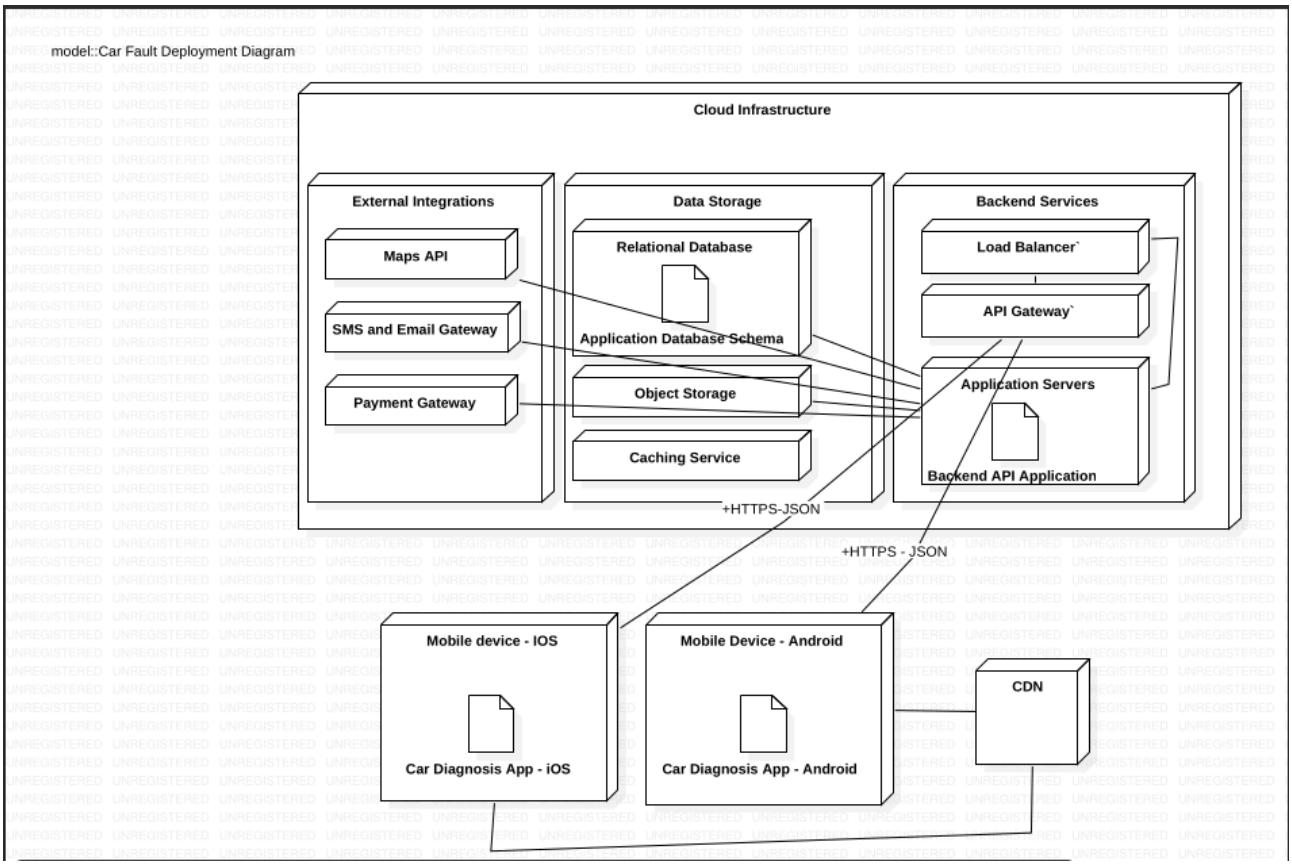
- **sync(): void:** Syncs cached data when online.
- **isOnline(): boolean:** Checks network status.



7. Deployment Diagram

This part provides a comprehensive overview of the deployment architecture for a Car Fault Diagnosis system. The architecture supports mobile platforms (iOS and Android), cloud-based backend services, external integrations, and optimized performance through caching and object storage. The primary goal of this system is to allow users to diagnose vehicle faults via a mobile application that communicates with cloud-based infrastructure through secure HTTPS connections.

1. System Components



1.1 Mobile Clients

- **Mobile Device – iOS**
    - Runs the **Car Diagnosis App – iOS**.
    - Communicates with backend services via **HTTPS-JSON**.
- **Mobile Device – Android**
    - Runs the **Car Diagnosis App – Android**.
    - Communicates with backend services via **HTTPS-JSON**.

These applications are the user-facing components that enable users to input data and receive diagnostic results.

1.2 Cloud Infrastructure

A. Data Storage Layer

- **Relational Database**
    - Stores structured application data (e.g., user data, diagnostic results).

- Contains the **Application Database Schema**.

- **Object Storage**
    - Manages unstructured data (e.g., images, logs, documents).
- **Caching Service**
    - Provides temporary storage for frequently accessed data to improve performance and reduce latency.

B. Backend Services

- **Load Balancer**
    - Distributes incoming requests across multiple **Application Servers** for scalability and fault tolerance.
- **API Gateway**
    - Acts as a single entry point for all client requests.
    - Handles routing, authentication, and rate limiting.
- **Application Servers**
    - Host the **Backend API Application** which contains the core business logic and processes requests from mobile apps.

C. External Integrations

- **Maps API**
    - Provides location services, possibly to map repair stations or user locations.
- **SMS and Email Gateway**
    - Sends notifications, alerts, or results to users via SMS and email.
- **Payment Gateway**
    - Facilitates transactions, possibly for premium diagnostics or professional assistance.

1.3 CDN (Content Delivery Network)

- Used to deliver static content such as images, JavaScript, and style sheets.
- Enhances performance and load times by serving content from geographically distributed servers.

2. Communication and Data Flow

- **Mobile clients (iOS & Android)** interact with the **API Gateway** via **HTTPS-JSON** requests.
- The **API Gateway** forwards requests to the appropriate **Application Servers**.
- **Application Servers** may:
    - Fetch data from the **Relational Database**.
    - Retrieve/store files in **Object Storage**.
    - Use the **Caching Service** for performance.
    - Call **External Integrations** (e.g., SMS, Maps).
- The **CDN** handles the distribution of static assets to the mobile clients.

3. Security Considerations

- All communications between clients and servers are encrypted using **HTTPS**.
- API Gateway can implement authentication and rate limiting to secure the backend.
- Object Storage and Databases should use role-based access controls and encryption at rest.

4. Scalability and Availability

- **Load Balancer** ensures high availability and distributes load efficiently.
- The use of **Caching** and **CDN** supports scalable content delivery and reduces server load.
- Cloud-native services (database, storage) allow elastic scaling.

This deployment architecture demonstrates a modular, scalable, and secure system for diagnosing car faults using mobile applications. It leverages cloud infrastructure to offer robust backend services, integrates with essential third-party APIs, and ensures high performance through caching and CDN distribution.

8. Summary and Recommendations

System Architecture Strengths

Comprehensive Functionality: The system design covers all functional requirements from basic diagnostic features to advanced mechanic connectivity and offline capabilities.

Scalable Architecture: The cloud-based deployment with microservices architecture ensures the system can handle growth in users and data volume effectively.

User-Centric Design: The use case analysis demonstrates clear focus on user needs, with intuitive workflows for both car owners and mechanics.

Security Focus: Multi-layered security approach with encryption, authentication, and secure communication protocols.

Implementation Recommendations

Phase 1: Core Development (Week 1-6)

- Implement basic user management and authentication
- Develop core diagnostic features (warning lights and sound analysis)
- Create mobile application with offline capabilities
- Set up basic cloud infrastructure

Phase 2: Advanced Features (Months 7-12)

- Integrate mechanic connectivity features
- Implement comprehensive knowledge base
- Add vehicle history and maintenance tracking
- Deploy full cloud architecture with scaling

Phase 3: Enhancement and Optimization (Months 13-18)

- Advanced AI model improvements
- Performance optimization and caching
- Additional integrations and partnerships
- Advanced analytics and reporting

Technical Considerations

AI Model Training: Continuous improvement of diagnostic accuracy through machine learning model updates and training data expansion.

Performance Monitoring: Implement comprehensive monitoring to ensure response time requirements are met, particularly the 3-second warning light recognition target.

Data Privacy Compliance: Ensure adherence to data protection regulations (GDPR, CCPA) with proper consent mechanisms and data handling procedures.

Quality Assurance: Implement rigorous testing protocols for diagnostic accuracy, including validation against known vehicle problems and professional mechanic verification.

Risk Mitigation

Accuracy Risks: Implement confidence ratings, multiple validation methods, and clear disclaimers about diagnostic limitations.

Dependency Risks: Design fallback mechanisms for external service failures and maintain critical functionality offline.

Scalability Risks: Plan for gradual rollout with load testing and performance monitoring to identify bottlenecks early.

Security Risks: Regular security audits, penetration testing, and compliance reviews to maintain data protection standards.

Conclusion

The comprehensive system modeling and design presented in this report provides a solid foundation for developing the automotive diagnostic mobile application. The UML diagrams offer clear visualization of system structure and behavior, facilitating effective communication among development teams and stakeholders. The modular architecture ensures maintainability and extensibility while meeting all specified functional and non-functional requirements.

The proposed system balances sophisticated diagnostic capabilities with user-friendly interfaces, creating value for both individual car owners and professional mechanics. With proper implementation following the outlined design, the application will provide reliable, accurate, and accessible vehicle diagnostics while maintaining high standards of performance and security.