**Republic of Cameroon**
**Peace-Work-Fatherland**
**Ministry of Higher Education**
**University of Buea**
**Faculty of Engineering and Technology**
**Department of Computer Engineering**

**République du Cameroun**
**Paix-Travaille-Patrie**
**Ministère de l'enseignement supérieures**
**Université de Buea**
**Faculty of Engineering and Technology**
**Department of Computer Engineering**

# Car Fault Diagnostics App: Software Requirement Specification Document

Course Instructor: Dr Nkemeni Valery

**Group21 Members**

| | |
|---|---|
| **1- Angi Atangwa Valerine** | **FE22A148** |
| **2- Av'Nyehbeuhkonh Ezekiel Hosana** | **FE22A160** |
| **3- Beng Dze Meinoff** | **FE22A173** |
| **4- Jong Turkson Junior** | **FE22A228** |
| **5- Nyuyesemo Calglain** | **FE22A287** |

Table of Contents

# I – INTRODUCTION

## I.1 Purpose

The purpose of this document is to define the software requirements for the Car Fault Diagnostic App. This mobile application is designed to assist vehicle owners and mechanics in identifying and diagnosing faults in vehicles by interfacing with the car's onboard diagnostics (OBD-II) system. The SRS will serve as a guideline for developers, testers, and stakeholders involved in the development and deployment of the application.

## I.2 Scope

The Car Fault Diagnostic App provides real-time monitoring and fault detection capabilities by connecting to a vehicle's OBD-II port using Bluetooth or Wi-Fi. It interprets diagnostic trouble codes (DTCs), displays readable fault descriptions, suggests possible causes, and recommends solutions or next steps. The app aims to empower users with actionable insights without requiring advanced automotive knowledge. Features include fault history logs, live engine parameter viewing, maintenance reminders, and a user-friendly dashboard.

## I.3 Definitions, Acronyms, and Abbreviations

OBD-II: On-Board Diagnostics version 2

DTC: Diagnostic Trouble Code

ECU: Engine Control Unit

UI: User Interface

App: Mobile application

# II– SYSTEM OVERVIEW

### 1) Problem statement and proposed solutions

#### a) Problem Statement

Car owners often struggle to interpret dashboard warning lights and engine sounds, leading to delayed maintenance, unnecessary mechanic visits, and inflated repair costs. Many rely on mechanics even for minor issues due to a lack of accessible, user-friendly diagnostic tools. Existing solutions either require technical expertise, lack multilingual support, or fail to integrate critical features like sound analysis and offline functionality.

#### b) Proposed Solution

A mobile application that empowers non-technical users to:

- **Diagnose Issues**: Scan dashboard lights/signs and analyze engine sounds via smartphone sensors.
- **Receive Guidance**: Get explanations of warnings, repair tutorials, and maintenance reminders.
- **Connect with Experts**: Locate nearby mechanics/garages and request professional advice.
- **Work Offline**: Access core features (tutorials, diagnostic history) without internet.

**2) System core Features and Purpose**

**a) Core Features:**

- Dashboard light/sign recognition via camera.
- Engine sound analysis using ML models.
- Multilingual support (English, French, local languages).
- Maintenance logs and reminders.
- Explanation of warnings, repair tutorials, and maintenance reminders.
- Geolocation-based mechanic/garage suggestions.
- Offline functionality for critical tasks.

**Users:** Car owners, drivers, mechanics (secondary stakeholders).

**b) Purpose:**

- Reduce dependency on mechanics for minor issues.
- Lower maintenance costs and extend vehicle lifespan.
- Educate users about car diagnostics through tutorials and expert advice.
- Provide a scalable, intuitive tool compatible with Android and iOS devices.

## II – TOOLS AND TECHNOLOGY TO BE USED

The technology stack is selected based on cross-platform compatibility, scalability, and alignment with the project's functional requirements (like offline support, machine learning, geolocation).

| Layer | Technology/Tools | Purpose |
|---|---|---|
| Frontend | Flutter (Dart) | Cross-platform UI development for Android and iOS users. |
| Backend | Firebase (Authentication, Firestore, Cloud Functions) | User authentication, real-time data storage, and serverless backend logic. |
| Machine Learning | TensorFlow Lite, PyTorch (for model training) | On-device sound recognition (engine noises) and dashboard light detection. |
| Database | Firestore (NoSQL), Hive/SQFlite (local storage) | Cloud-based and offline storage for diagnostic history and user profiles. |
| APIs | Node.js + Express (REST API), API's for image and sound recognition for dashboard and engine sound analysis. | Integration with third-party services (e.g, mechanics' databases, maps). |
| Geolocation | Google Maps API, Flutter Geolocator Plugin | Locate nearby garages and mechanics based on user's location. |
| Image Processing | OpenCV, TensorFlow (for image recognition) | Analyze dashboard lights/signs via camera input. |

| | | |
|---|---|---|
| Offline Functionality | Hive, SQFlite | Enable app usage without internet (e.g., cached tutorials, basic features). |
| State Management | Provider/Riverpod | Manage app state efficiently across screens. |
| CI/CD | GitHub Actions, Codemagic | Automated testing and deployment for Android and iOS. |

## III – USER CLASSES AND CHARACTERISTICS

This system has different types of users (primary and secondary) with different roles and purpose of using the app, below are the different users with their needs and motivations;

**a. Car Owners**

**Profile:**

- Primary users with limited technical knowledge of vehicles.
- May speak French, English, or local languages (e.g., Pidgin).
- Often rely on affordable, older car models (e.g., used Toyota Corollas, Avensis).

**Needs:**

- Simple interface with minimal text (icon-driven design).
- Offline functionality for areas with poor connectivity.
- Repair recommendations tailored to locally available spare parts and tools.

**Motivation**: Avoid frequent mechanic visits, save costs, and ensure roadworthiness

**b. Mechanics (Partners)**

**Profile:**

- Garage owners.
- May lack formal training but have empirical repair knowledge.

**Needs:**

- Access to the app's diagnostic results to validate their manual inspections.
- Support for remote diagnostics and help in understanding specific fault codes or symptoms.

**Motivation**: Provide better service, adopt modern tools without heavy investment.

**c. App Administrators**

**Profile**:

- Maintain the app, update machine learning models, manage backend databases, and ensure data privacy
- Managing content and partnerships.

**Needs**: Easy backend tools, low-cost cloud/server solutions, and monitoring dashboards.

**Responsibilities**:

- Curate region-specific fault databases (e.g., common issues in Cameroonian car models).
- Partner with mechanics to validate diagnoses.
- Ensure multilingual support (French + Pidgin/English).

## IV – SPECIFIC REQUIREMENTS

This refers to the detailed, precise, and actionable statements that describe how a software system must behave or perform to meet its goals. They translate broad system objectives exact instructions that developers, testers and stakeholders can follow and very.

These requirements are divided into several categories:

1) **Functional Requirements**

   These define the functions or services the software must provide that is,

   - what the system should do
   - How it should respond to inputs
   - How it should behave in particular situations.

Below we have the requirements with their respective specifications

| | |
|---|---|
| **User Authentication and Authorization** | The application should allow users to register and log in securely. Based on the role (car owner, driver, or mechanic), access to features will be granted. |
| **Dashboard Light Recognition** | Users can capture images of dashboard indicators with their camera. These images are compared to a database to return matching results with explanations. Users could also scan dashboard signals. |
| **Engine Sound Analysis** | Records sound from the engine using the phone's microphone and classifies issues using ML-trained datasets. |
| **Diagnostic Explanation and Repair Guidance** | Provides issue-specific explanations and steps to resolve them. Includes videos and images. |
| **Maintenance Log and Reminders** | Logs vehicle service history and sets maintenance reminders. Notifies users via app alerts. |
| **Expert Advice and Nearby Mechanic Locator** | Displays nearby garages using GPS and allows users to contact mechanics in-app. |
| **Offline Functionality** | Downloads and caches essential data like tutorials and history so they are available offline. |
| **Diagnostic History Tracking** | Saves results from previous scans along with timestamps and action taken. |
| **Lessons and Tutorials** | Offers interactive lessons with multimedia to educate users about car diagnostics. |

**2) Non-Functional Requirements**

These requirements define the quality attributes of the system that is, how it performs rather than what it does.

Below we have the requirements with their specifications

| Requirement | Specification |
|---|---|
| Performance | App should respond within **2 seconds** for scans. |
| Privacy | Process audio/images **on-device**; minimal data collection. |
| Usability | Simple UI with **walkthrough guides** for new users. |
| Reliability | **95% accuracy** in light/sound recognition. |
| Scalability | Support **10,000+ users** without slowdowns. |
| Offline Capability | Core features work **without internet**. |
| Security | Encrypt user data (maintenance logs, location). |
| Responsiveness | Supports dynamic resizing for phones and tablets in both portrait and landscape orientations. |
| Accessibility | Meets accessibility standards; supports screen readers and high contrast modes. |

# V- VERIFICATION AND VALIDATION OF REQUIREMENTS

## 1) Functional Requirement Analysis

### i. User Authentication and Authorization

- **Technique Used**: Inspection, Peer Review

- **Result**: Requirement is complete, necessary, feasible, and verifiable. Login and role-based access control are standard implementations. No ambiguity found.

**ii. Dashboard Light Recognition**

- **Technique Used**: Walkthrough, Checklist
- **Result**: Verified as necessary and correct. Completeness was flagged to ensure the database of symbols includes local variants. Improvement suggested in database coverage.

**iii. Engine Sound Analysis**

- **Technique Used**: Peer Review, Feasibility Check
- **Result**: Found feasible using pre-trained ML models. Completeness confirmed, effectiveness dependent on training data quality.

**iv. Diagnostic Explanation and Repair Guidance**

- **Technique Used**: Walkthrough, Inspection
- **Result**: Complete and correct. Stakeholders validated usefulness and clarity. Verifiable via content tests.

**v. Maintenance Log and Reminders**

- **Technique Used**: Checklist, Peer Review
- **Result**: Verified as consistent and complete. Efficiency verified through time-based reminders and test cases.

**vi. Expert Advice and Nearby Mechanic Locator**

- **Technique Used**: Feasibility Analysis, Walkthrough
- **Result**: Location-based functions confirmed feasible. Stakeholder walkthrough validated necessity.

**vii. Offline Functionality**

- **Technique Used**: Checklist, Peer Review

- **Result**: Offline accessibility validated as effective and necessary for rural users. Verifiable through simulation tests.

**viii. Multilingual Support**

- **Technique Used**: Inspection, Checklist
- **Result**: Requirement is necessary and effective. Completeness depends on language translation coverage.

**ix. Diagnostic History Tracking**

- **Technique Used**: Walkthrough, Peer Review
- **Result**: Confirmed as verifiable and feasible. Completeness validated for user scenarios.

**x. Lessons and Tutorials**

- **Technique Used**: Walkthrough, Inspection
- **Result**: Tutorials validated as effective and complete. Verification possible through usability testing.

**SUMMARY POINTS**

| | |
|---|---|
| **Completeness** | Most requirements are complete. A few (e.g., dashboard light database, multilingual coverage) require expansion to ensure comprehensive coverage across models and regions. |
| **Feasibility** | All requirements were confirmed feasible using current mobile technologies (e.g., ML models, GPS, local caching). No critical implementation barriers were identified. |
| **Necessity** | Every requirement was validated as necessary based on stakeholder input. They address real user pain points like lack of knowledge, high repair costs, and limited offline tools. |
| **Verifiability** | All requirements can be verified through functional testing, simulated use cases, or by measuring performance criteria. Each has clear test paths. |

| | |
|---|---|
| **Effectiveness** | Requirements are effective in solving the identified problem. Particularly strong are sound analysis and offline functionality. Some effectiveness (e.g., sound detection accuracy) will depend on data quality during implementation. |

**2)Non-Functional Requirement verification and validation**

| **Quality Attribute** | **Result Summary** |
|---|---|
| **Completeness** | All non-functional requirements are well-defined. They address performance, usability, security, and accessibility. No major omissions were found. |
| **Feasibility** | Every non-functional requirement is feasible with current mobile development frameworks (e.g., Flutter, React Native, Android SDK). Especially achievable are responsiveness and encryption standards. |
| **Necessity** | Each requirement serves a critical role in ensuring the app is usable, scalable, and secure. Stakeholders emphasized usability and offline reliability as essential for adoption. |
| **Verifiability** | All are testable using system benchmarks, usability tests, accessibility audits, and security checks. Each can be validated during system testing phases. |
| **Effectiveness** | Non-functional requirements significantly contribute to user satisfaction, trust, and engagement. Effective UI, fast response times, and offline access were seen as major usability enhancers. |

Each requirement has been individually verified and validated using a structured technique. All functional and non-functional requirements passed the analysis for completeness, correctness, feasibility, and verifiability. This ensures readiness for system design and further development.

# VI – VERIFICATION AND VALIDATION OF TECHNIQUES USED

### 1. Review and Analysis of Gathered Requirements

**Sources of Requirements**

The project requirements were gathered using a mix of the following techniques:

- **Surveys** (Google Forms – CSV responses analyzed)
- **Questionnaires** (targeted at drivers in Molyko and Mile 17)
- **Stakeholder Interviews** (with Dr. Valery and group brainstorming)
- **Reverse Engineering** (existing diagnostic apps like FIXD, Torque Pro)

**Key Insights from Survey and PREVIOUS REQUIREMENT DOCUMENT:**

- **Demographics:** Majority are aged 18–34, both male and female, and own cars.
- **Pain Points:**
  - Difficulty interpreting dashboard lights.
  - Lack of confidence in diagnosing issues.
  - Reluctance to rely solely on apps due to fear of misdiagnosis.
- **User Needs:**
  - App that can scan and interpret dashboard lights.
  - Suggestions for repair steps and tutorial links.
  - Sound-based engine fault detection.
  - Offline functionality for low connectivity areas.
  - Notifications and logs for maintenance tracking.
- **Existing System Gaps:** Most systems require OBD2 connection and lack user-centric features like offline mode, sound detection, and in-app tutorials.

## 2. Identified Inconsistencies and Ambiguities

### 2.1 Inconsistencies

1. **Feature Prioritization**
   - The survey indicates strong demand for **sound recognition** and **offline functionality**, but these are not explicitly prioritized in the previous requirement documents.
   - Some users prefer **visual representations** of warning lights, while others prefer **text descriptions** – this inconsistency should be addressed in UI design.
2. **Diagnosis Methods**
   - Most survey respondents **consult mechanics**, but the previous requirement document emphasizes **self-diagnosis via mobile apps**. This suggests a need for **professional integration** (e.g., mechanic recommendations).
3. **Privacy Concerns**
   - The previous requirement document mentions **on-device processing** for privacy, but the survey does not explicitly ask about privacy preferences.

### 2.2 Ambiguities

1. **Offline Functionality**
   - Unclear whether this includes **full diagnostic capabilities** or just **basic reference materials**.
   - Needs clarification on **how much data is stored locally**.
2. **Sound Recognition**
   - No details on **how sounds are captured** (microphone quality, background noise handling).
   - No mention of **accuracy expectations**.
3. **Severity Indicator**
   - Unclear how severity is determined (e.g., **AI-based risk assessment** or **predefined rules**).

### 3. Technical Feasibility Assessment

| Feature | Feasibility | Challenges |
|---|---|---|
| **Dashboard Light Scanning** | High | Requires OBD2 integration or image recognition. |
| **Sound Recognition** | Medium | Noise filtering, ML model accuracy. |
| **Offline Functionality** | High | Storage limitations for diagnostic databases. |
| **Maintenance Reminders** | High | Simple push notifications. |
| **Multilingual Support** | Medium | Translation accuracy, UI adaptation. |
| **Nearest Mechanic Finder** | Medium | Relies on GPS and mechanic database. |

**Key Feasibility Notes:**

- **OBD2 Integration:** Required for real-time diagnostics but may need hardware.
- **Machine Learning Models:** Needed for sound/image recognition; requires training data.
- **Offline Mode:** Possible but may limit diagnostic depth.

## VII – KEY USE CASES FOR THE APPLICATION

**Use Case 1: Dashboard Warning Light Scanning**

**Actor: Car Owner**

**Scenario**:

- A driver notices a warning light and opens the app to scan it using the camera.
- App detects issue.

**Flow:**

1. App identifies the light using offline computer vision (pre-trained model).
2. Displays explanation: “Problem with engine – check the spark plugs or the air filter.”
3. Suggests urgency level (“High – Visit mechanic within 48 hours”).
4. Links to a video tutorial by a mechanic on cleaning air filters.

**Use Case 2: Engine Sound Analysis**

**Actor: Car Owner**

**Scenario:**

- User records engine noise while idling.
- App detects a knocking sound (indicative of engine misfire).

**Flow:**

1. App filters background noise (street vendors, traffic) using spectral analysis.
2. Matches audio to preloaded fault patterns (trained on common engine sounds).
3. Recommends e.g.: “Problem with spark plugs – Buy spare parts from [local vendor] at mile 17.”
4. Provides offline access to step-by-step replacement guides.

**Use Case 3**: Video Tutorial Access

**Actor**: Car Owner

**Scenario**: User gets a diagnostic result and wants help understanding the issue.

**Outcome**: The app provides YouTube links or offline embedded videos.

## VII – CONCLUSION

This document outlines the overall system design, specific functional and non-functional requirements, user interface guidelines, external interface definitions, and assumptions for the Car Fault Diagnostic App. It is intended to ensure all stakeholders share a common understanding of the application's objectives and operational requirements.

## VIII – REFERENCES


- ISO15031: Communication between vehicle and external diagnostic tools
- SAEJ1979: Standard for OBD-II diagnostic services
- IEEE830-1998: Recommended Practice for Software Requirements Specifications