**Review and Analysis of Functional Requirements for Car Diagnostic Mobile Application**

# 1.0 Introduction

This report outlines the requirements for a car diagnostic mobile application based on user responses gathered through a structured survey. The report categorizes the requirements into functional and non-functional aspects. It follows a two-part structure for each:

1. Review of raw user responses,
2. Analysis (based on completeness, clarity, technical feasibility, and dependencies relationships) of requirements.

## 1.1 Review of raw user responses

### 1.1.1 Raw Functional Requirements from User Responses

* Scan and interpret dashboard warning lights
* Record and analyze engine sounds
* Provide automated explanations and repair suggestions
* Provide access to video tutorials for repairs
* Display instant diagnosis results after analysis
* Offer detailed repair recommendations based on diagnosis
* Send maintenance reminders based on vehicle condition or schedule
* Log maintenance records and maintenance cost
* Support live chat with a mechanic or support representative
* Provide step-by-step guides for vehicle troubleshooting and repair
* Show maintenance alerts when critical thresholds are detected
* Diagnose all common and critical faults in a car
* Perform predictive maintenance using machine learning to anticipate potential issues
* Enable community forums and support interaction among users
* Provide early detection and prediction of faults before they escalate

### 1.1.2 Raw Non-Functional Requirements from User Responses

* Accuracy of the diagnosis
* Complexity of the technology
* High data usage or cost
* Zero ads display
* Efficiency

## 1.2 Analysis of Requirements

The requirements were assessed for:

* **Completeness:** whether enough information is provided--**Do the requirements cover all aspects of what the system needs to do?**
* **Clarity:** whether the requirement is easy to understand--**Is each requirement clearly expressed without ambiguity?**
* **Technical feasibility:** whether it is realistically implementable--Can these requirements be implemented with available technology?
* **Dependency relationships:** whether one requirement depends on others--How do requirements relate to and impact each other?

The results are summarized and refined below.

### 1.2.1 Finalized Functional Requirements with Explanations

1. **Scan and interpret dashboard warning lights**

Use image recognition to detect and explain dashboard indicators.

1. **Record and analyze engine sounds**

Capture engine noises via microphone and identify issues like misfires using ML algorithms.

1. **Provide automated explanations and repair suggestions**

Generate readable reports with likely causes and potential solutions based on detected issues.

1. **Provide access to video tutorials for repairs**

Offer repair videos for common problems to guide users through fixing them.

1. **Display instant diagnosis results after analysis**

Show a quick summary of diagnostics as soon as analysis completes.

1. **Send maintenance reminders based on vehicle condition or schedule**

Notify users of upcoming maintenance based on mileage or time intervals.

1. **Log maintenance records and maintenance cost**

Maintain a history of repairs and cost estimates for future reference and analytics.

1. **Support live chat with a mechanic or support representative**

Provide real-time chat with experts or AI assistant for consultation.

1. **Show maintenance alerts when critical thresholds are detected**

Send warnings when certain engine or system values exceed safe limits.

1. **Perform predictive maintenance using machine learning to anticipate potential issues**

Use historical data trends to forecast problems before they occur.

1. **Enable community forums and support interaction among users**

Allow users to post, answer questions, and interact about vehicle issues and repairs.

1. **Provide early detection and prediction of faults before they escalate**

Continuously monitor and alert users when early signs of potential problems are identified.

### 1.2.2. Finalized Non-Functional Requirements with Explanations

1. **Accuracy of the diagnosis**

The app should provide diagnostic results with at least 90% accuracy under typical conditions.

1. **Complexity of the technology**

The UI/UX design must simplify complex diagnostic operations for non-technical users.

1. **Data usage and cost optimization**

The app should minimize bandwidth consumption by using lightweight data formats and caching.

1. **Zero advertisement policy**

No in-app advertisements shall be displayed to maintain focus and professionalism.

1. **Efficiency**

App functions should complete within acceptable timeframes

# 2.0 Identify Issues in Requirements

2.1 Introduction

High‑quality, unambiguous requirements are the foundation of any successful software project. To ensure our car‑fault diagnostic app meets user and technical needs, we conducted structured peer reviews and walkthroughs with the development team, systematically flagging and clarifying problematic statements before design and implementation began.

**2.2 Summary of Identified Issues**

* **Inconsistencies:**   
  We discovered overlapping repair‑guidance requirements. For example, “Offer detailed repair recommendations based on diagnosis” and “Provide step‑by‑step guides for vehicle troubleshooting and repair” both duplicate “**Provide automated explanations and repair suggestions,**” leading to redundant functionality that must be consolidated.
* **Ambiguities:**   
  The requirement “**Diagnose all common and critical faults in a car**” is overly broad—its scope and the definition of “common” versus “critical” faults vary by make, model, and region. We must specify which fault categories are supported to avoid misinterpretation.
* **Missing Information:**

Our review found no gaps in the documented requirements. All stakeholder needs appeared to be captured; instead, the focus is on refining existing statements for clarity and consistency.

**2.3 Conclusion**

By identifying and resolving these inconsistencies and ambiguities early, through collaborative reviews and targeted stakeholder queries, we strengthen the SRS, reduce downstream rework, and pave the way for a development phase that is both efficient and aligned with user expectations.