Software Requirements Specification (SRS)

Project Name: Auto Spare Parts Management Application
Version: 1.0
Date: 28.11.2024
Author: Mohsen S. Beigi

---

1. Introduction

1.1 Purpose

The Auto Spare Parts Management Application *(name not choosen yet)* is a full-stack web application designed to streamline inventory management, sales, and customer engagement for auto spare parts. The system will provide an intuitive user interface for managing spare parts and ensure seamless backend operations with reliable data handling.

1.2 Scope

This application will allow users to:

- Add, update, and delete spare part records.
- View inventory with filtering and searching capabilities.
- Handle customer orders and sales transactions.
- Integrate analytics and reporting features. The backend will manage data storage and processing, while the frontend will deliver an engaging user experience.

1.3 Definitions, Acronyms, and Abbreviations

- **SRS**: Software Requirements Specification
- **UI**: User Interface
- **API**: Application Programming Interface
- **CRUD**: Create, Read, Update, Delete
- **CI/CD**: Continuous Integration/Continuous Deployment

1.4 References

- React Documentation
- Spring Framework Documentation
- PostgreSQL Documentation

---

2. System Overview

The application consists of a **backend** for managing business logic and data processing and a **frontend** for interacting with end users. It will be containerized for easy deployment and scalability.

---

3. Functional Requirements

3.1 Backend

1. **Programming Language:** Java
2. **Framework:** Spring (Spring Boot for simplification)
3. **Database:** PostgreSQL
    - Store inventory data, customer details, and order transactions.
4. **Containerization:** Docker
    - Ensure consistent development and production environments.
5. **Database Version Control:** Liquibase
    - Manage and track database schema changes effectively.
6. **APIs:**
    - Provide RESTful APIs for frontend communication.
    - Endpoints to handle CRUD operations for inventory and orders.
7. **Authentication:**
    - Use JWT or OAuth2 for secure user authentication.

3.2 Frontend

1. **Programming Language:** TypeScript
2. **Library:** React
3. **UI Framework:** ShadCN
    - Build responsive and accessible UI components.
4. **State Management:** Redux (with Redux Toolkit)
    - Manage global state efficiently.
5. **Features:**
    - Dashboard with inventory overview.
    - Forms for adding and editing spare parts.
    - Search and filter functionality for inventory.
    - Order management interface.

---

4. Non-Functional Requirements

1. **Performance:**
    - Backend APIs should respond within 200ms for 95% of requests.
    - Frontend should maintain a Time to Interactive (TTI) under 2 seconds.
2. **Scalability:**
    - The application should handle up to 10,000 concurrent users.
3. **Security:**
    - Use HTTPS for secure communication.
    - Store sensitive data (e.g., passwords) using strong encryption.
4. **Maintainability:**

    - Modular architecture for easy updates and feature additions.

5. **Portability:**
    - Deployable on any Docker-compatible infrastructure.

---

5. System Design

5.1 Backend Design

- **Architecture:** Layered architecture (Controller, Service, Repository).
- **Database Schema:**
    - Tables: `Inventory`, `Orders`, `Customers`, `Users`.
    - Relationships:
        - `Orders` table references `Inventory` and `Customers`.

5.2 Frontend Design

- **Architecture:** Component-based structure using React.
- **Routing:** React Router for managing navigation.
- **State Management:** Redux for global state, with slices for `Inventory`, `Orders`, and `User`.

5.3 API Contracts

- Example Endpoint:
    - **GET** `/api/inventory` - Fetch all inventory items.
    - **POST** `/api/orders` - Create a new order.

---

6. Constraints

- Development must be completed within [Insert Timeline].
- Deployment must support Linux-based Docker hosts.

---

7. Assumptions

- All users will have internet access.
- The client will provide hosting infrastructure.
- Initial database schema will be shared by the client.

---

8. Acceptance Criteria

1. Users can add, update, and delete spare parts through the frontend.
2. Orders and customer details are saved and retrieved correctly from the backend.
3. The application is deployable using Docker without manual configuration.
4. UI is responsive and works across major browsers.

---

9. Appendix

Include diagrams like architecture, database schema, or API workflows if necessary.