Software Requirements Specification (SRS) for Full Node.js E-Commerce System

1. Introduction

1.1 Purpose

This document outlines the Software Requirements Specification (SRS) for the Full Node.js E-Commerce System, a web-based platform designed to facilitate online shopping. The SRS defines the functional and non-functional requirements, scope, and constraints to guide developers, stakeholders, and testers in implementing and validating the system.

1.2 Scope

The Full Node.js E-Commerce System is a web application built using Node.js, Express, and MongoDB, following the MVC architecture. It provides a platform for users to browse products, manage shopping carts, place orders, and perform administrative tasks such as product and user management. Key features include:

- User authentication (registration, login, password management).
- Product management (CRUD operations for products).
- Order processing and cart management.
- Image processing for product images (uploading and resizing).
- Role-based access control (admin and regular users).

The system aims to provide a scalable, secure, and user-friendly e-commerce experience.

1.3 Definitions, Acronyms, and Abbreviations

- **API**: Application Programming Interface.
- **MVC**: Model-View-Controller architecture.
- **JWT**: JSON Web Token, used for authentication.
- **MongoDB**: NoSQL database used for storing users, products, and orders.
- **Multer**: Middleware for handling file uploads.
- **Sharp**: Library for image processing.
- **CRUD**: Create, Read, Update, Delete operations.

1.4 References

- GitHub Repository: https://github.com/Ahmed2222002/Full-NodeJs-E-Commerce-system-
- IEEE SRS Template (standard for SRS structure).

2. Overall Description

2.1 Product Perspective

The system is a standalone web application interfacing with a MongoDB database for data storage and retrieval. It uses RESTful APIs to handle client-server communication, with Express.js as the backend framework. The system integrates with Multer and Sharp for image uploads and processing, and JWT for secure authentication.

2.2 Product Functions

- **User Management**:
    - Register new users with email and password.
    - Login and logout functionality with JWT-based authentication.
    - Update user profile (e.g., name, email, password).
    - Admin-specific functions to manage users (view, delete, update roles).
- **Product Management**:
    - Create, read, update, and delete products (admin only).
    - Upload and resize product images using Multer and Sharp.
    - Categorize products and filter by category or price.
- **Cart and Order Management**:
    - Add/remove products to/from the shopping cart.
    - View cart details and calculate total price.
    - Place orders and view order history.
- **Image Processing**:
    - Upload multiple images for products (e.g., cover image and additional images).
    - Resize images to 500x500 pixels with 90% JPEG quality.
- **Search and Filtering**:
    - Search products by name or description.
    - Filter products by category, price, or availability.

2.3 User Classes and Characteristics

- **Regular Users**:
    - Can register, log in, browse products, manage carts, and place orders.
    - Need an intuitive and responsive interface.
- **Admin Users**:
    - Have full access to manage products, users, and orders.

    - Require a dashboard for administrative tasks.
- **Guests**:
    - Can browse products and view public information without authentication.

2.4 Operating Environment

- **Backend**: Node.js with Express.js, MongoDB for data storage.
- **Deployment**: Can be hosted on cloud platforms like Heroku, AWS, or Render.
- **Browsers**: Compatible with modern browsers (Chrome, Firefox, Safari).

2.5 Design and Implementation Constraints

- The system uses MongoDB, limiting database choice to NoSQL.
- Image uploads are processed using Multer and Sharp, requiring proper configuration for file handling.
- JWT-based authentication requires secure storage of the JWT secret key.
- The system assumes a stable internet connection for API requests.

2.6 Assumptions and Dependencies

- MongoDB is properly configured and accessible.
- Environment variables (e.g., JWT_SECRET_KEY, JWT_EXPIRATION) are set correctly.
- The frontend (if implemented) supports RESTful API integration.
- Users have modern browsers for accessing the web application.

3. Functional Requirements

3.1 User Management

- **FR1.1**: The system shall allow users to register with a unique email, password, and name.
- **FR1.2**: The system shall authenticate users using JWT upon login, with tokens expiring based on the configured duration (JWT_EXPIRATION).
- **FR1.3**: The system shall allow users to update their profile information (name, email, password).
- **FR1.4**: The system shall allow admins to view, update, or delete user accounts.
- **FR1.5**: The system shall support password hashing for secure storage.

3.2 Product Management

- **FR2.1**: The system shall allow admins to create products with details (name, description, price, category, images).
- **FR2.2**: The system shall allow admins to update or delete existing products.

- **FR2.3**: The system shall support uploading a cover image and multiple additional images for each product.
- **FR2.4**: The system shall resize uploaded images to 500x500 pixels with 90% JPEG quality using Sharp.
- **FR2.5**: The system shall allow users to browse products with pagination and sorting options (e.g., by price or category).

3.3 Cart and Order Management

- **FR3.1**: The system shall allow users to add products to their cart and view cart contents.
- **FR3.2**: The system shall calculate the total price of items in the cart, including any applicable taxes or discounts.
- **FR3.3**: The system shall allow users to place orders, storing order details (products, total price, date) in MongoDB.
- **FR3.4**: The system shall allow users to view their order history.
- **FR3.5**: The system shall allow admins to view and manage all orders.

3.4 Search and Filtering

- **FR4.1**: The system shall provide a search function to find products by name or description.
- **FR4.2**: The system shall allow filtering products by category, price range, or availability.

4. Non-Functional Requirements

4.1 Performance

- **NFR1.1**: The system shall handle up to 1000 concurrent users with a response time of less than 2 seconds for API requests.
- **NFR1.2**: Image resizing shall complete within 1 second per image.

4.2 Security

- **NFR2.1**: The system shall use HTTPS for all API communications.
- **NFR2.2**: User passwords shall be hashed using a secure algorithm (e.g., bcrypt).
- **NFR2.3**: JWT tokens shall be securely stored and validated to prevent unauthorized access.

4.3 Scalability

- **NFR3.1**: The system shall support horizontal scaling by adding more Node.js instances.
- **NFR3.2**: MongoDB shall be configured to handle increasing data volumes (e.g., sharding if needed).

4.4 Usability

- **NFR4.1**: The system shall provide a user-friendly interface (for future frontend implementation) with clear navigation.
- **NFR4.2**: API responses shall include meaningful error messages for invalid requests.

4.5 Reliability

- **NFR5.1**: The system shall achieve 99.9% uptime when deployed.
- **NFR5.2**: The system shall handle invalid inputs gracefully without crashing.

5. System Architecture

5.1 Overview

The system follows the MVC architecture:

- **Model**: MongoDB schemas (productModel.js, userModel.js, orderModel.js) for data management.
- **Controller**: Logic for handling requests (e.g., product creation, user authentication).
- **View**: To be implemented (frontend not included in the repository).
- **Routes**: RESTful API endpoints defined in routes folder (e.g., /api/products, /api/users).

5.2 Data Flow

- Users send HTTP requests to API endpoints.
- Express routes direct requests to appropriate controllers.
- Controllers interact with MongoDB models to perform CRUD operations.
- Responses are sent back to the client in JSON format.

6. External Interface Requirements

6.1 User Interfaces

- The backend provides RESTful APIs; the frontend (if implemented) will handle UI rendering.
- Admin dashboard for managing products, users, and orders (to be implemented).
- User interface for browsing products, managing carts, and viewing orders.

6.2 Software Interfaces

- **Node.js v16+**: Backend runtime environment.
- **Express.js**: Framework for handling HTTP requests.
- **MongoDB**: Database for storing user, product, and order data.
- **Multer**: Middleware for file uploads.
- **Sharp**: Library for image processing.

- **JWT**: Library for authentication (jsonwebtoken).

6.3 Communication Interfaces

- **HTTP/REST**: For client-server communication.
- **MongoDB Driver**: For database connectivity.

7. Constraints

- The system is limited to MongoDB as the database, requiring NoSQL expertise.
- Image uploads are constrained by Multer's configuration (e.g., file size limits).
- The system assumes a stable internet connection for API and database access.

8. Assumptions

- A frontend will be developed separately to consume the backend APIs.
- Environment variables (e.g., MONGO_URI, JWT_SECRET_KEY) are correctly configured.
- The system will be deployed on a platform supporting Node.js and MongoDB.

9. Appendices

- **GitHub Repository**: https://github.com/Ahmed2222002/Full-NodeJs-E-Commerce-system-
- **Sample API Endpoints**:
    - **POST /api/users/register:** Register a new user.
    - **POST /api/products:** Create a new product (admin only).
    - **GET /api/products:** Retrieve all products with pagination.
    - **POST /api/orders:** Place a new order.