**SRS For E-Commerce Product Catalog**

1. **Purpose**

The purpose of the E-Commerce Product Catalog system is to develop a centralized, user-friendly platform that allows customers to browse, search, and filter products with ease. The system aims to improve the shopping experience by offering detailed product information, high-quality images, and organized categories, enabling customers to make informed purchasing decisions. It will serve as the backbone of an online store’s inventory management by maintaining accurate and up-to-date product details, pricing, stock availability, and promotional offers. The platform will also allow administrators to efficiently add, update, and remove products, ensuring that the catalog always reflects the latest information. In addition, the system will integrate search and filter features to help users quickly locate products based on specific criteria such as category, price range, brand, and ratings.

By providing an intuitive and responsive design, the product catalog will be accessible across devices, including desktops, tablets, and smartphones, thereby ensuring maximum reach and usability. The platform will support high scalability to accommodate a growing number of products and users without affecting performance. The E-Commerce Product Catalog will also focus on enhancing security by implementing secure API endpoints and role-based authentication to protect sensitive product and user data. Overall, the system will act as a seamless link between the store’s inventory and its customers, supporting business growth and improving customer satisfaction.

**2. Scope**

The E-Commerce Product Catalog system will provide an organized, interactive, and scalable platform for managing and displaying products in an online store. It will enable customers to explore products through well-structured categories, advanced search options, and multiple filtering criteria such as price range, brand, ratings, and availability. The system will ensure that customers have access to accurate and updated product information, including descriptions, specifications, images, prices, discounts, and stock status. It will also include a product comparison feature, allowing users to evaluate multiple products side-by-side before making a purchase decision.

From the administrative perspective, the platform will provide a secure backend interface for adding, updating, deleting, and categorizing products. It will support bulk uploads and real-time inventory updates to maintain catalog accuracy. The system will be integrated with external modules such as shopping carts, payment gateways, and order management systems to ensure smooth operation within the overall e-commerce environment. The platform will be designed to handle high traffic and large product databases without compromising performance or loading speed. While the primary focus will be on product catalog management, features such as customer reviews, ratings, and recommendations will be incorporated to enhance user engagement and boost sales conversions.

**3. Non-Functional Requirements**

1. **Performance**
   * The system should load product listings within **2 seconds** for standard broadband users.
   * The platform must handle **up to 10,000 concurrent users** without significant performance degradation.
2. **Scalability**
   * The system must be able to scale horizontally and vertically to support a growing number of products and users.
   * It should support at least **1 million product entries** without affecting search or browsing performance.
3. **Security**
   * All data transmissions must be encrypted using **HTTPS/TLS protocols**.
   * The admin panel should require **multi-factor authentication (MFA)** for access.
   * User inputs must be validated to prevent **SQL injection, XSS, and CSRF attacks**.
4. **Usability**
   * The interface should be **user-friendly, intuitive, and responsive** across devices.
   * The system should follow **WCAG 2.1 accessibility standards** to ensure it can be used by people with disabilities.
5. **Availability**
   * The system must maintain **99.9% uptime** annually.
   * It should include backup and recovery mechanisms to restore the catalog within **1 hour** in case of a failure.
6. **Maintainability**
   * The system should be modular, allowing for easy updates and integration of new features without disrupting operations.
   * Documentation for both the frontend and backend should be maintained for developers.

**4. System Architecture**

**Tech Stack**

1. **Frontend** – **React.js**
   * Builds an interactive, responsive, and user-friendly interface for browsing products.
   * Communicates with the backend via REST API calls.
2. **Backend** – **Spring Boot**
   * Handles business logic, authentication, and product management.
   * Exposes secure REST APIs for frontend consumption.
3. **Database** – **MySQL**
   * Stores product details, categories, user accounts, and orders.
   * Supports efficient querying and indexing for fast search.
4. **Hosting/Deployment** (Optional for SRS)
   * **Backend**: Deployed on a cloud platform (AWS, Azure, or similar).
   * **Frontend**: Deployed via CDN for faster load times.

**5. Database Design**

**Entities & Attributes**

**1. User**

* user\_id (PK)
* username
* password
* email
* role (e.g., admin, customer)
* created\_at

**2. Product**

* product\_id (PK)
* name
* description
* price
* stock\_quantity
* category\_id (FK)
* image\_url

**3. Category**

* category\_id (PK)
* name
* description

**4. Order**

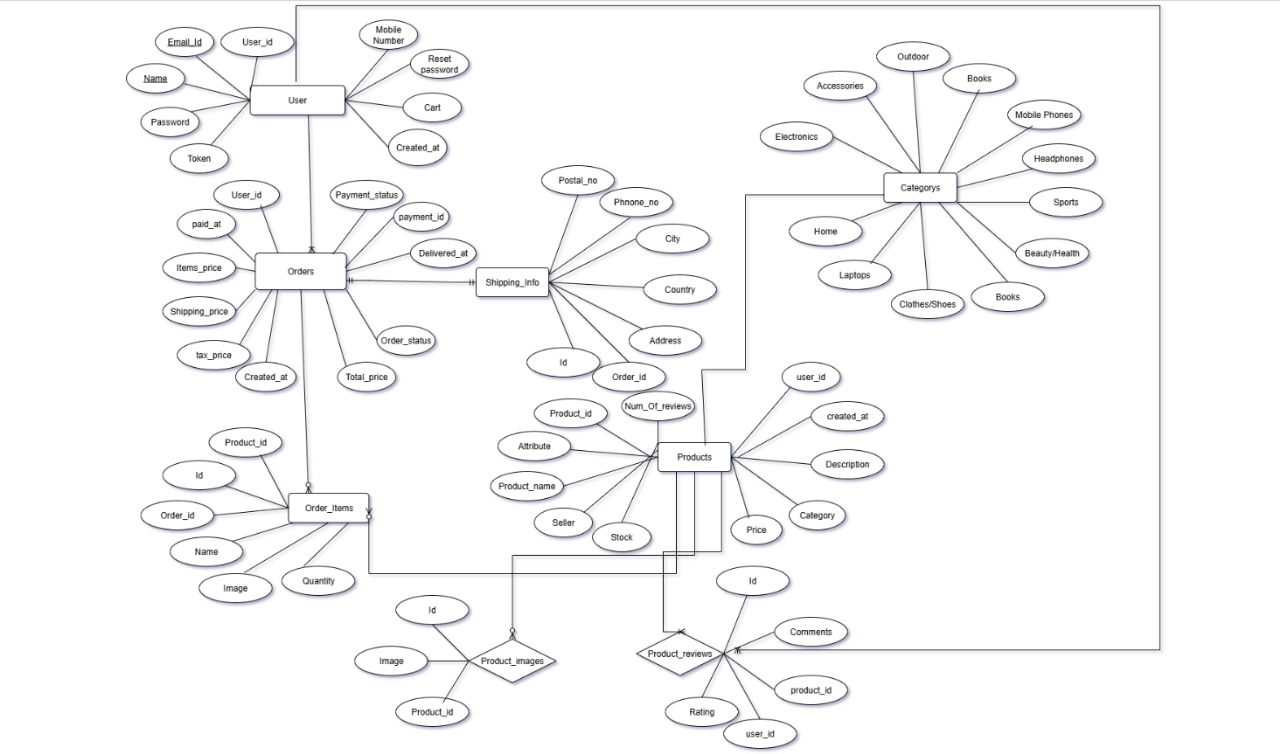
* order\_id (PK)
* user\_id (FK)
* order\_date
* total\_amount
* status (Pending, Shipped, Delivered)

**5. OrderItem**

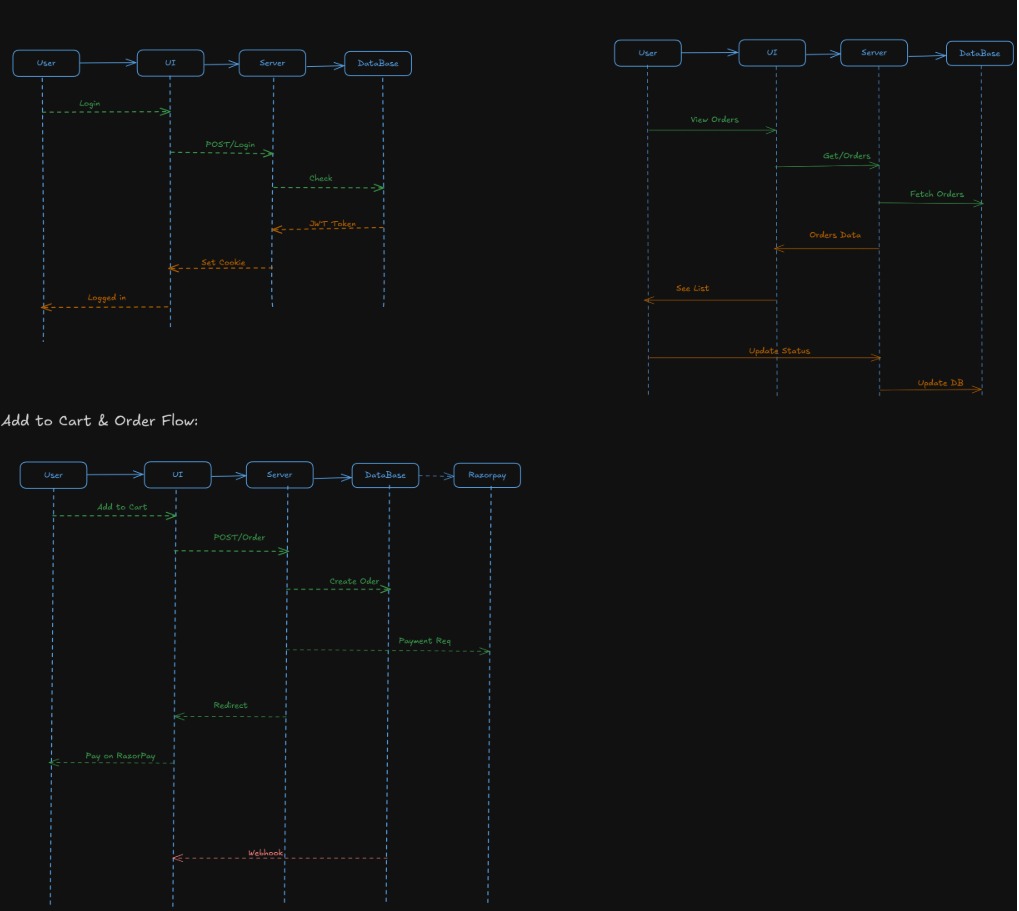
* order\_item\_id (PK)
* order\_id (FK)
* product\_id (FK)
* quantity
* price

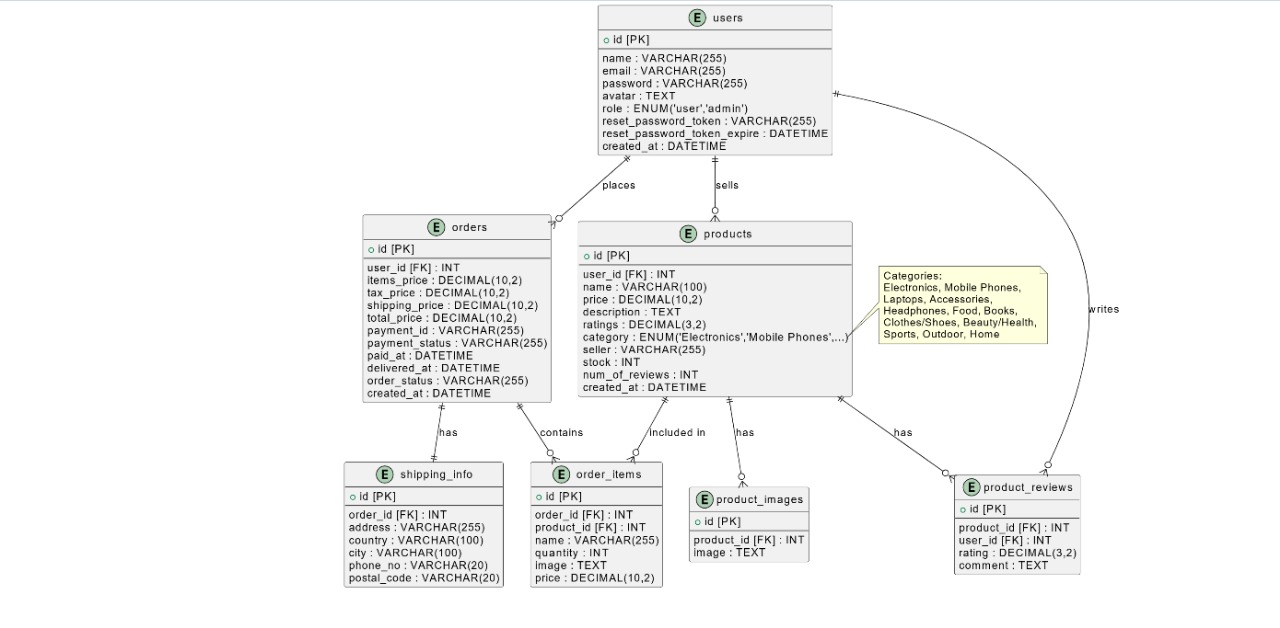
**6. Design Diagrams**

Tools: StarUML, Lucidchart, or draw.io.



ER Diagram



System flow Diagram

UML Diagram

**7. Implementation Details**

**Backend (Spring Boot)**

* Built using **Spring Boot** to handle all business logic, product management, authentication, and secure API communication with the frontend.
* Uses **Spring Data JPA** with Hibernate ORM to interact with the MySQL database for efficient CRUD operations.
* Implements **Spring Security** with **JWT (JSON Web Tokens)** for role-based authentication (Admin, Customer) and secure API access.
* Provides RESTful endpoints for product catalog operations, search filters, and integration with payment processing via **Stripe API**.

**Frontend (React with Vite)**

* Developed using **React.js** for a responsive and interactive user interface, bundled with **Vite** for faster development and build times.
* Utilizes **Bootstrap** for consistent UI design and responsive layouts.
* Integrates **Axios** for secure API requests to the backend.
* Includes features like advanced search, product filtering, sorting, and product comparison for improved customer experience.

**Database (MySQL)**

* MySQL is used to store product data, user accounts, categories, orders, and transaction history.
* Database schema is normalized to avoid redundancy and ensure data consistency.
* Indexed search fields for fast retrieval of product and category information.
* Supports relational mappings such as One-to-Many (Category → Products) and Many-to-Many (Orders ↔ Products).

**Payment & Email Services**

* Integrated with **Stripe Payment Gateway** for secure and smooth checkout experiences.
* Configured **Mailtrap SMTP** for testing transactional emails such as order confirmations and shipping updates.

**8. Testing Plan - Optional**

**1. Unit Testing**

* **Backend**: Use **JUnit** and **Mockito** to test individual service methods, repository queries, and business logic.
* **Frontend**: Use **React Testing Library** and **Jest** to test UI components, event handling, and state updates.

**2. Integration Testing**

* Test REST API endpoints using **Postman** or automated scripts to verify correct communication between frontend and backend.
* Validate data flow from frontend inputs → backend processing → MySQL database updates.

**3. Functional Testing**

* Verify all core features such as product search, filtering, cart management, checkout, and payment processing.
* Test role-based access (Admin vs. Customer) for restricted features like product management.

**4. Usability Testing**

* Ensure responsive design works seamlessly on desktop, tablet, and mobile devices.
* Conduct user feedback sessions to improve navigation, search, and checkout experience.

**5. Security Testing**

* Test authentication and authorization mechanisms (Spring Security + JWT).
* Perform input validation checks to prevent SQL injection, XSS, and CSRF attacks.

**6. Performance Testing**

* Use **JMeter** to simulate up to **10,000 concurrent users** and measure API response times.
* Monitor load times for product listing and search operations.

**9. Deployment**

**1. Backend Deployment**

* **Local Deployment**:
  + Run the Spring Boot application using Maven/Gradle on **Tomcat embedded server** (port 8080 by default).
  + MySQL database hosted locally or via Docker container.
* **Cloud Deployment (Optional)**:
  + Deploy backend to **AWS EC2**, **Azure App Service**, or **Heroku**.
  + Use **AWS RDS** or **Azure Database for MySQL** for cloud-hosted database.
  + Configure environment variables for database credentials, JWT secret, and API keys.

**2. Frontend Deploment**

* **Local Deployment**:
  + Build React app using **Vite** and run locally with npm run dev.
* **Production Deployment**:
  + Build optimized production files (npm run build).
  + Deploy via **Vercel**, **Netlify**, or **AWS S3 + CloudFront** for global CDN delivery.
  + Configure API base URLs to point to the deployed backend.

**3. Database Deployment**

* Initialize MySQL schema using provided **SQL scripts**.
* For cloud deployment, use **AWS RDS**, **Azure MySQL**, or **Google Cloud SQL** with proper network security groups.
* Enable automated daily backups and failover support for high availability.

**4. Additional Deployment Considerations**

* Enable **HTTPS** using SSL/TLS certificates for secure communication.
* Implement **CI/CD pipelines** (GitHub Actions, GitLab CI, or Jenkins) for automated builds, tests, and deployments.
* Maintain version control via **Git** with a structured branching strategy (main, dev, feature/\*).

**10. Appendix**

* **Spring Boot Documentation** – <https://spring.io/projects/spring-boot>
* **React Documentation** – https://react.dev/
* **MySQL Documentation** – https://dev.mysql.com/doc/
* **Bootstrap Documentation** – https://getbootstrap.com/
* **Axios Documentation** – https://axios-http.com/
* **Stripe Payment Gateway** – <https://stripe.com/docs>
* **Mailtrap SMTP Documentation** – https://mailtrap.io/docs/
* **JWT (JSON Web Tokens) Documentation** – <https://jwt.io/introduction/>

B. Glossary of Terms

|  |  |
| --- | --- |
| Term / Acronym | Definition |
| CRUD | Create, Read, Update, Delete – basic database operations. |
| JWT | JSON Web Token – a secure, compact way of transmitting authentication data between client and server. |
| REST API | Representational State Transfer – API communication style that uses HTTP methods like GET, POST, PUT, DELETE. |
| UI/UX | User Interface / User Experience – design principles for usability and visual appeal. |
| SMTP | Simple Mail Transfer Protocol – standard for sending email messages. |
| HTTPS/TLS | Secure communication protocol for encrypted data transfer between client and server. |