Ecommerce Java Backend Project Document

1. Introduction

The Ecommerce Java Backend is a Spring Boot application designed to power an online shopping platform. It provides a comprehensive set of RESTful APIs to manage users, products, carts, orders, comments, and email notifications. The system is secure, scalable, and transactional, ensuring a seamless experience for both customers and administrators.

Purpose

This document outlines the project's objectives, functional requirements, and use cases to guide developers, stakeholders, and testers.

Scope

The project covers:

- User authentication and management.
- Product catalog management.
- Shopping cart and order processing.
- Product commenting system.
- Email notifications for order confirmation and email verification.
- Secure API access with JWT authentication.

2. System Overview

The backend is built using Spring Boot and follows a service-oriented architecture. It integrates with a relational database (MySQL/PostgreSQL) and uses Spring Security for authentication. The system supports two user roles: USER (customers) and ADMIN (store managers).

Key Components

- **UserService**: Handles user registration, login, email confirmation, and password management.
- **ProductService**: Manages product creation, updates, deletion, and retrieval.
- **CartService**: Manages shopping cart operations.
- **OrderService**: Processes orders and manages order status.
- **CommentService**: Handles product comments.
- **EmailService**: Sends email notifications.
- **JwtService**: Manages JWT token(access token and refresh token) generation and validation.

3. Functional Requirements

- Users can register, log in, confirm their email, and change their password.
- Users can browse products, add them to a cart, and place orders.
- Users can comment on products.
- Admins can manage products and update order statuses.
- The system sends email notifications for order confirmations and email verification.
- APIs are secured with JWT tokens.

4. Use Cases

Below are the primary use cases for the eCommerce backend, covering the interactions between actors USER and ADMIN

4.1. User Registration

- **Actor**: USER
- **Description**: A new user registers with an email and password.
- **Preconditions**: Email is not already in use.
- **Steps**:
    1. User sends a registration request with email and password.
    2. System validates the email is unique.
    3. System creates a user with a confirmation code and sends it via email.
    4. System saves the user with emailConfirmation=false.
- **Postconditions**: User is registered but cannot place orders until email is confirmed.

4.2. Email Confirmation

- **Actor**: USER
- **Description**: User confirms their email using a confirmation code.
- **Preconditions**: User is registered and has received a confirmation code.
  -**.Steps**:
    1. User submits email and confirmation code.
    2. System verifies the code matches the user's stored code.
    3. System sets emailConfirmation=true and saves the user.
- **Postconditions**: User can now place orders.

4.3. User Login

- **Actor**: USER, ADMIN
- **Description**: User logs in with email and password to receive JWT tokens.
- **Preconditions**: User is registered.
- **Steps**:
    1. User submits email and password.
    2. System authenticates credentials using Spring Security.
    3. System generates access and refresh JWT tokens.
    4. System revokes existing tokens and saves new tokens.
    5. System returns tokens to the user.
- **Postconditions**: User receives tokens for API access.

4.4. Change Password

- **Actor**: USER, ADMIN
- **Description**: User changes their password.
- **Preconditions**: User is logged in and provides the correct current password.
- **Steps**:
    1. User submits current and new passwords.
    2. System verifies the current password.
    3. System hashes and saves the new password.
- **Postconditions**: User’s password is updated.

4.5. Add Product

- **Actor**: ADMIN
- **Description**: Admin adds a new product to the catalog.
- **Preconditions**: Admin is logged in with a valid JWT token.
- **Steps**:
    1. Admin submits product details (name, description, price, quantity) and an optional image.
    2. System saves the image (if provided) and generates a URL.
    3. System creates and saves the product.
    4. System returns the created product details.

4.6. Update Product

- **Actor**: ADMIN
- **Description**: Admin updates an existing product.
- **Preconditions**: Product exists, and admin is logged in.
- **Steps**:
    1. Admin submits updated product details and an optional image.
    2. System validates the product exists.
    3. System updates the product and saves the new image (if provided).
    4. System returns the updated product details.
- **Postconditions**: Product is updated.

4.7. Delete Product

- **Actor**: ADMIN
- **Description**: Admin deletes a product from the catalog.
- **Preconditions**: Product exists, and admin is logged in.
- **Steps**:
    1. Admin submits the product ID.
    2. System validates the product exists.
    3. System deletes the product.
- **Postconditions**: Product is removed from the catalog.

4.8. View Products

- **Actor**: USER, ADMIN
- **Description**: User browses the product catalog.
- **Preconditions**: None (public endpoint).
- **Steps**:
    1. User requests a paginated list of products.
    2. System retrieves products without comments to avoid performance issues.
    3. System returns the product list.
- **Postconditions**: User receives a paginated product list.

4.9. Add to Cart

- **Actor**: USER
- **Description**: User adds a product to their cart.
- **Preconditions**: User is logged in, product exists, and sufficient stock is available.
- **Steps**:
    1. User submits product ID and quantity.
    2. System validates the user and product.
    3. System checks stock availability.
    4. System adds or updates the cart item and saves the cart.
    5. System returns the updated cart.
- **Postconditions**: Product is added to the user's cart.

4.10. Remove from Cart

- **Actor**: USER
- **Description**: User removes a product from their cart.
- **Preconditions**: User is logged in, and the product is in the cart.
- **Steps**:
    1. User submits the product ID.
    2. System validates the cart exists.
    3. System removes the product from the cart and saves it.
- **Postconditions**: Product is removed from the cart.

4.11. Clear Cart

- **Actor**: USER
- **Description**: User clears all items from their cart.
- **Preconditions**: User is logged in, and the cart exists.
- **Steps**:
    1. User requests to clear the cart.
    2. System validates the cart exists.
    3. System clears all items and saves the cart.
- **Postconditions**: Cart is empty.

4.12. Create Order

- **Actor**: USER
- **Description**: User creates an order from their cart.
- **Preconditions**: User is logged in, email is confirmed, cart is not empty, and sufficient stock is available.
- **Steps**:
    1. User submits address and phone number.
    2. System validates the user, email confirmation, and cart.
    3. System checks stock for all cart items.
    4. System creates an order with PREPARE status.
    5. System updates product quantities and clears the cart.
    6. System sends an order confirmation email.
    7. System returns the order details.
- **Postconditions**: Order is created, cart is cleared, and email is sent.

4.13. Update Order Status

- **Actor**: ADMIN
- **Description**: Admin updates the status of an order.
- **Preconditions**: Order exists, and admin is logged in.
- **Steps**:
    1. Admin submits the order ID and new status.
    2. System validates the order exists.
    3. System updates the order status and saves it.
    4. System returns the updated order.
- **Postconditions**: Order status is updated.

4.14. View Orders

- **Actor**: USER, ADMIN
- **Description**: User views their orders, or admin views all orders.
- **Preconditions**: User/admin is logged in.

- **Steps**:
    1. User/admin requests orders.
    2. System retrieves orders (filtered by user ID for USER role).
    3. System returns the order list.
- **Postconditions**: Orders are returned.

4.15. Add Comment

- **Actor**: USER
- **Description**: User adds a comment to a product.
- **Preconditions**: User is logged in, and product exists.
- **Steps**:
    1. User submits the product ID and comment details.
    2. System validates the user and product.
    3. System creates and saves the comment.
    4. System returns the comment details.
- **Postconditions**: Comment is added to the product.

4.16. View Comments

- **Actor**: USER, ADMIN
- **Description**: User views comments for a product.
- **Preconditions**: Product exists.
- **Steps**:
    1. User requests comments for a product ID.
    2. System retrieves comments.
    3. System returns the comment list.
- **Postconditions**: Comments are returned.

5. Non-Functional Requirements

Update soon…..

6. Assumptions

- The frontend is developed separately and consumes the backend APIs.
- A third-party SMTP service (e.g., Gmail) is used for emails.

- The database is pre-configured with the necessary schema.

7. Constraints

- No payment gateway integration in the current version.
- Limited to two roles (USER, ADMIN).
- Image uploads are stored locally (not cloud-based).

8. Dependencies

- Spring Boot 3.x
- Java 17
- MySQL/PostgreSQL
- Maven
- JavaMailSender
- JWT library for JWT handling

9. Risks and Mitigation

- **Risk**: Email delivery failures.
    - **Mitigation**: Log failures and allow manual resending of emails.
- **Risk**: Insufficient stock during order creation.
    - **Mitigation**: Validate stock before order creation and use transactions.
- **Risk**: Security vulnerabilities in JWT.
    - **Mitigation**: Use strong secrets, short-lived tokens, and refresh tokens.

10. Conclusion

The Ecommerce Java Backend provides a robust foundation for an online shopping platform. It supports essential Ecommerce features with a focus on security, scalability, and reliability. Future enhancements can include payment integration, product search, and advanced analytics.