E-Commerce Backend System — Documentation

Overview

This project is a **production-grade backend system** for an **E-Commerce platform** built with **Java Spring Boot**. It manages **Users, Products, Carts, Orders, Payments, and Inventory** with **JWT-based authentication** for secure access.

The backend exposes **REST APIs** that can be consumed by a web or mobile frontend.

Objectives

- Implement a **modular and layered architecture** (Controller → Service → Repository → Entity)
- Enable **secure authentication & authorization** using **JWT**
- Manage **CRUD operations** for users, products, carts, and orders
- Simulate **payments and inventory management**
- Ensure **role-based access** (ADMIN & CUSTOMER)

Technical Stack

Component	Technology
Language	Java 17+
Framework	Spring Boot 3+
ORM	Spring Data JPA / Hibernate
Database	MySQL
Build Tool	Maven
Testing	JUnit 5, Mockito
Security	Spring Security (JWT Authentication)
Logging	SLF4J / Logback

Architecture

```
com.incture.e_commerce
 ├ 📁controller       → REST Controllers (API endpoints)
 ├ 📁service          → Business logic layer
 ├ 📁repository       → Data access layer (JPA)
```

```
├ 📁entity          → Database entities (User, Product, Order, etc.)
├ 📁dto             → Request/Response DTOs
├ 📁exception       → Custom exceptions
└ 📁config          → Security & JWT configuration
```

Database Schema

Entity	Relationships
User	`1:N` → Cart, Orders
Product	`1:N` → CartItem, OrderItem
Cart	`1:N` → CartItem
CartItem	`N:1` → Cart, Product
Order	`1:N` → OrderItem
OrderItem	`N:1` → Order, Product

Authentication — JWT (JSON Web Token)

All protected routes require a valid JWT token.

Register (Public)

POST `/api/auth/register`

```
{
  "name": "Keerthana",
  "email": "keerthana@example.com",
  "password": "password123"
}
```

Response:

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6..."
}
```

Login (Public)

POST `/api/auth/login`

```
{
  "email": "keerthana@example.com",
```

```
  "password": "password123"
}
```

Response:

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6..."
}
```

Use this token in all protected routes:

```
Authorization: Bearer <token>
```

User Management APIs

Method	Endpoint	Access	Description
POST	`/api/auth/register`	Public	Register new user
POST	`/api/auth/login`	Public	Login user (returns JWT)
GET	`/api/users/me`	Authenticated	Get logged-in user profile
GET	`/api/users/{id}`	ADMIN	Get user by ID
PUT	`/api/users/{id}`	ADMIN	Update user
DELETE	`/api/users/{id}`	ADMIN	Delete user
GET	`/api/users`	ADMIN	List all users

Product Management APIs

Method	Endpoint	Access	Description
POST	`/api/products`	ADMIN	Add new product
GET	`/api/products`	Public	Get all products (pagination supported)
GET	`/api/products/{id}`	Public	Get product by ID
PUT	`/api/products/{id}`	ADMIN	Update product details
DELETE	`/api/products/{id}`	ADMIN	Delete product

Example: Create Product

POST `/api/products`

Headers:

```
Authorization: Bearer <admin_token>
Content-Type: application/json
```

Body:

```
{
  "name": "Apple iPhone 15",
  "description": "Latest iPhone model with A17 chip",
  "price": 89999.99,
  "stock": 15,
  "category": "Mobiles",
  "imageUrl": "https://example.com/iphone15.jpg"
}
```

Cart Management APIs

Method	Endpoint	Access	Description
POST	`/api/cart/add/{productId}`	Authenticated	Add product to logged-in user's cart
PUT	`/api/cart/update/{productId}`	Authenticated	Update product quantity in cart
DELETE	`/api/cart/remove/{productId}`	Authenticated	Remove product from cart
GET	`/api/cart`	Authenticated	Get logged-in user's cart

Example: Add Item to Cart

```
POST /api/cart/add/1?qty=2
Authorization: Bearer <token>
```

Order Management APIs

Method	Endpoint	Access	Description
POST	`/api/orders/checkout/{userId}`	Authenticated	Checkout and create order from cart
GET	`/api/orders/{id}`	Authenticated	Get order details
GET	`/api/orders`	Authenticated	List all user orders
PUT	`/api/orders/{id}/status`	ADMIN	Update order status (e.g., SHIPPED, DELIVERED)

Payment Simulation

- During checkout, payment is simulated automatically.
- `Order.paymentStatus = "PAID"`
- On success → Cart is cleared and product stock decreases.
- On failure (optional extension) → Order is cancelled.

Inventory Management

- Product stock reduces automatically on successful checkout.
- Orders for products with insufficient stock are rejected.

Running Locally

Prerequisites

- Java 21
- MySQL running locally
- Maven installed

Setup Database

Create the database:

```
CREATE DATABASE ecommerce_db;
```

Configure `application.properties`

```
spring.datasource.url=jdbc:mysql://localhost:3306/ecommerce_db
spring.datasource.username=root
spring.datasource.password=YOUR_PASSWORD
spring.jpa.hibernate.ddl-auto=update
jwt.secret=h7kL9v3Qz8Rj2Yp5Sx1uV4eN0bC6mTqW
```

Run Application

```
mvn spring-boot:run
```

Testing with Postman

Import Postman Collection

Use `/api/auth/login` to generate a JWT, then:

- Copy the `token` from response.
- Add Header:

```
Authorization: Bearer <token>
```

- Test secured routes (`/api/products`, `/api/cart`, `/api/orders`).

Public routes: `/api/auth/register`, `/api/auth/login`, `/api/products`

Example Test Flow

1 Register Admin → `/api/auth/register` 2 Login → get token 3 Add Product → `/api/products` (use token) 4 Add to Cart → `/api/cart/add/{productId}` (use token) 5 Checkout → `/api/orders/checkout/{userId}` (use token) 6 View Order → `/api/orders/{id}` 7 (Admin) Update Order Status → `/api/orders/{id}/status`
