**Solution Approach: E-Commerce System (Backend Documentation)**

**Overview**

This document outlines the **backend architecture** of the **E-Commerce System** built using **Spring Boot**. The system provides RESTful APIs for:

* **User Authentication & Authorization** (JWT-based)
* **Product Management** (CRUD operations, filtering, search)
* **Order Processing** (order creation, payment integration with Razorpay)
* **Shopping Cart Management**
* **Admin Dashboard** (user, product, and order management)

The backend follows a **layered architecture** with:

* **Controllers** (API endpoints)
* **Services** (Business logic)
* **Repositories** (Database interactions)
* **Models** (Data entities)

**Backend Architecture**

**1. Authentication & User Management**

**Key Features:**

* **JWT-based authentication** (signup, login, token validation)
* **Role-based access control** (ADMIN/USER)
* **Password encryption** (BCrypt)

**Key Endpoints:**

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /auth/signup | POST | Register a new user |
| /auth/signin | POST | User login (returns JWT) |
| /api/users/profile | GET | Fetch user profile (JWT required) |
| /api/admin/users | GET | (ADMIN) List all users |

**Code Snippet:**

java

Copy

Download

@PostMapping("/signup")

public ResponseEntity<AuthResponse> createUserHandler(@Valid @RequestBody User user) throws UserException {

if (userRepository.findByEmail(user.getEmail()) != null) {

throw new UserException("Email already in use");

}

user.setPassword(passwordEncoder.encode(user.getPassword()));

User savedUser = userRepository.save(user);

String token = jwtTokenProvider.generateToken(new UsernamePasswordAuthenticationToken(savedUser.getEmail(), savedUser.getPassword()));

return ResponseEntity.ok(new AuthResponse(token, true));

}

**2. Product Management**

**Key Features:**

* **CRUD operations** (create, read, update, delete)
* **Filtering & Sorting** (by category, price, color, etc.)
* **Search functionality**

**Key Endpoints:**

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /api/products | GET | Get filtered products |
| /api/products/id/{id} | GET | Get product by ID |
| /api/products/search | GET | Search products |
| /api/admin/products | POST | (ADMIN) Create product |
| /api/admin/products/{id}/delete | DELETE | (ADMIN) Delete product |

**Code Snippet:**

java

Copy

Download

@GetMapping("/products")

public ResponseEntity<Page<Product>> findProducts(

@RequestParam String category,

@RequestParam List<String> color,

@RequestParam Integer minPrice,

@RequestParam Integer maxPrice

) {

Page<Product> products = productService.getAllProducts(category, color, minPrice, maxPrice);

return ResponseEntity.ok(products);

}

**3. Order & Payment Processing**

**Key Features:**

* **Order creation & management**
* **Razorpay integration** (payment gateway)
* **Order status tracking** (PLACED, CONFIRMED, SHIPPED, DELIVERED, CANCELLED)

**Key Endpoints:**

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /api/orders/ | POST | Create order (JWT required) |
| /api/orders/user | GET | Get user's order history |
| /api/payments/{orderId} | POST | Generate Razorpay payment link |
| /api/payments | GET | Handle payment callback |

**Code Snippet:**

java

Copy

Download

@PostMapping("/payments/{orderId}")

public ResponseEntity<PaymentLinkResponse> createPaymentLink(@PathVariable Long orderId) throws RazorpayException {

RazorpayClient razorpay = new RazorpayClient(apiKey, apiSecret);

JSONObject paymentLinkRequest = new JSONObject();

paymentLinkRequest.put("amount", order.getTotalPrice() \* 100);

paymentLinkRequest.put("currency", "INR");

PaymentLink payment = razorpay.paymentLink.create(paymentLinkRequest);

return ResponseEntity.ok(new PaymentLinkResponse(payment.get("short\_url"), payment.get("id")));

}

**4. Shopping Cart Management**

**Key Features:**

* **Add/remove items** from cart
* **Update quantities**
* **Cart persistence** (per user)

**Key Endpoints:**

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /api/cart/ | GET | Get user's cart |
| /api/cart/add | PUT | Add item to cart |
| /api/cart\_items/{id} | DELETE | Remove item from cart |

**Code Snippet:**

java

Copy

Download

@PutMapping("/cart/add")

public ResponseEntity<CartItem> addToCart(

@RequestBody AddItemRequest req,

@RequestHeader("Authorization") String jwt

) throws UserException, ProductException {

User user = userService.findUserProfileByJwt(jwt);

CartItem item = cartService.addCartItem(user.getId(), req);

return ResponseEntity.ok(item);

}

**5. Admin-Specific Operations**

**Key Features:**

* **User management** (view all users)
* **Product management** (create, update, delete)
* **Order management** (confirm, ship, cancel orders)

**Key Endpoints:**

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /api/admin/users | GET | List all users |
| /api/admin/products | POST | Create product |
| /api/admin/orders/{id}/confirmed | PUT | Confirm order |
| /api/admin/orders/{id}/cancel | PUT | Cancel order |

**Code Snippet:**

java

Copy

Download

@PutMapping("/orders/{orderId}/confirmed")

public ResponseEntity<Order> confirmOrder(@PathVariable Long orderId) throws OrderException {

Order order = orderService.confirmedOrder(orderId);

return ResponseEntity.ok(order);

}

**6. Ratings & Reviews API**

| **Endpoint** | **Method** | **Description** |
| --- | --- | --- |
| /api/ratings/create | POST | Submit a product rating/review (JWT required) |
| /api/ratings/product/{productId} | GET | Retrieve all ratings for a specific product |

**Key Features:**

* Users can rate products (1-5 stars)
* Fetch all ratings for product display
* JWT authentication for review submission

**Database Schema (Key Entities)**

| **Entity** | **Description** |
| --- | --- |
| **User** | Stores user details (email, password, role) |
| **Product** | Product details (name, price, category, etc.) |
| **Cart** | User's shopping cart |
| **CartItem** | Items in cart (product, quantity) |
| **Order** | Order details (user, products, status) |
| **PaymentDetails** | Payment info (Razorpay ID, status) |

**Conclusion**

The **Spring Boot backend** for this **E-Commerce System** provides:

* **Secure Authentication (JWT)** – Robust token-based security for user verification.
* **Efficient Product & Order Management** – Streamlined handling of inventory and customer orders.
* **Seamless Razorpay Payment Integration** – Smooth and secure payment processing.
* **Role-Based Admin Controls** – Granular access permissions for different admin levels.
* **Scalable & Maintainable Architecture** – Future-proof design for growth and easy updates.

This documentation covers the **core backend functionality**, ensuring a **robust, secure, and efficient** e-commerce platform.