Grocery Booking Web Application Overview

Contents

[1. Brief Introduction: 1](#_Toc151116672)

[1.2. Purpose: 1](#_Toc151116673)

[2. Key Features 1](#_Toc151116674)

[Project Structure 2](#_Toc151116675)

[Package Descriptions: 2](#_Toc151116676)

[Technologies Used: 3](#_Toc151116677)

[Database Schema Overview 4](#_Toc151116678)

[Collections 4](#_Toc151116679)

[Database Schema Relation Diagram 6](#_Toc151116680)

[User Roles and Permissions 7](#_Toc151116681)

[**1.** User Role: Customer 7](#_Toc151116682)

[2. User Role: Seller 8](#_Toc151116683)

[Authentication and Authorization Summary: 9](#_Toc151116684)

[1. Authentication Configuration (SecurityConfig class): 9](#_Toc151116685)

[2. Custom Authentication Success Handler 9](#_Toc151116686)

[Diagram Role Based Login And Operation 10](#_Toc151116687)

[**Endpoints and API Documentation** 10](#_Toc151116688)

[RestAPIS 15](#_Toc151116689)

1. Brief Introduction:

The grocery booking web application is a comprehensive online platform designed to streamline the process of purchasing groceries. It provides users with a convenient and efficient solution for browsing, selecting, and ordering a wide range of grocery items. Through an intuitive and user-friendly interface, customers can access an extensive catalog of products, manage their shopping carts, and complete transactions securely.

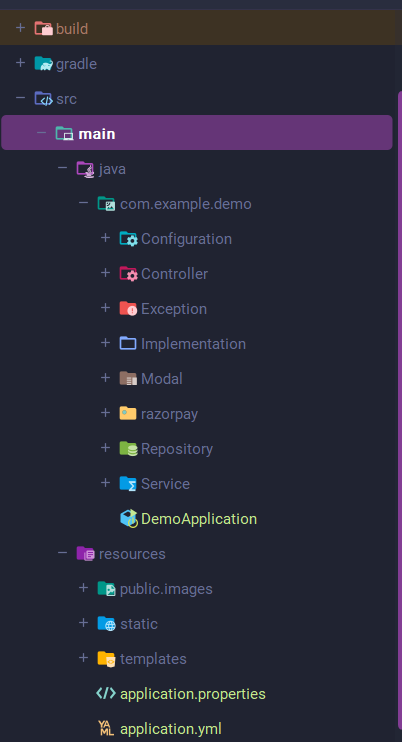
## 1.2. Purpose:

The primary purpose of the grocery booking web application is to offer users a convenient and time-saving alternative to traditional in-store grocery shopping. By leveraging the power of e-commerce, the application aims to simplify the entire grocery procurement process, providing users with a seamless and personalized experience from product selection to order fulfillment.

2. Key Features**:**

* **Product Catalog:**
  + An extensive catalog of diverse grocery items, categorized for easy navigation.
  + Detailed product information, including descriptions, prices, and images.
* **User Accounts:**
  + User registration and authentication for personalized shopping experiences.
  + User profiles to store preferences, order history, and delivery information.
* **Shopping Cart Management:**
  + Intuitive shopping cart functionality to add, remove, and modify selected items.
  + Real-time cart updates to reflect quantity changes and product availability.
* **Secure Checkout:**
  + A secure and straightforward checkout process with multiple payment options.
  + Integration with payment gateways to ensure financial transactions are safe and reliable.

# Project Structure



## Package Descriptions:

1. config Package:
   * **Responsibility:** Contains configuration classes for Spring Boot, such as security configuration (SecurityConfig) and web configuration (WebConfig).
2. **controller Package:**
   * **Responsibility:** Houses controllers that handle HTTP requests and define the application's endpoints.
   * **Controllers:**
     + CustomerController: Handles customer-related actions.
     + AdminController: Manages administrative tasks.
     + HomeController: Manages home page and general navigation.
3. **model Package:**
   * **Responsibility:** Defines domain models or entities representing the core data structure of the application.
   * **Models:**
     + User: Represents user information.
     + Order: Represents an order placed by a customer.
     + Product: Represents grocery products.
4. **repository Package:**
   * **Responsibility:** Contains Spring Data repositories for data access.
   * **Repositories:**
     + UserRepository: Handles database operations related to users.
     + OrderRepository: Manages orders in the database.
     + ProductRepository: Manages product data in the database.
5. **service Package:**
   * **Responsibility:** Houses service classes that contain business logic and act as an intermediary between controllers and repositories.
   * **Services:**
     + UserService: Provides user-related business logic.
     + OrderService: Contains order-related business logic.
     + ProductService: Manages product-related business logic.
6. **GroceryApplication Class:**
   * **Responsibility:** The main entry point for the Spring Boot application. Contains the main method to run the application.

# Technologies Used:

1. **Spring Framework:**
   * Data: MongoDB (spring-boot-starter-data-mongodb)
   * Web: Thymeleaf (spring-boot-starter-thymeleaf), Mail (spring-boot-starter-mail), Security (spring-boot-starter-security)
2. **Dropwizard:**
   * Core: dropwizard-core (1.0.2)
   * Views: dropwizard-views (1.0.2), dropwizard-views-freemarker (1.0.2)
   * Jersey Integration: dropwizard-jersey (4.0.0)
3. **Persistence:**
   * Hibernate: hibernate-gradle-plugin (5.6.15.Final)
4. **Web Scraping:**
   * Jsoup: jsoup (1.7.2)
5. **Database:**
   * Derby: derby (10.12.1.1)
6. **OAuth and Security:**
   * Spring Security OAuth: spring-security-oauth2
7. **Payment Gateway:**
   * Razorpay: razorpay-java (1.3.1)
8. **HTTP Requests:**
   * OkHttp: okhttp
9. **Developer Tools:**
   * Lombok: lombok, Spring Boot DevTools: spring-boot-devtools

# Database Schema Overview

The grocery booking web application utilizes a MongoDB database to store information related to users, orders, products, and various other entities. The following sections provide an overview of the database schema, detailing the collections, fields, relationships, and specific considerations.

## Collections

**1. checkOut Collection**

* **Fields:**
  + id (ObjectId): Unique identifier for the checkOut entry.
  + userInfo (DBRef): Reference to user information.
  + processed (boolean): Flag indicating whether the order has been processed.
  + sameAddress (Boolean): Flag indicating whether the delivery address is the same as the user's address.
  + totalAmount (Double): Total amount for the order.
  + orderitem (List<OrderItem>): List of order items.
  + saveInfo (Boolean): Flag indicating whether user information should be saved.
  + savedAmount (Double): Amount saved through discounts or promotions.
  + TotalItems (Integer): Total number of items in the order.
  + coupon (Coupon): Reference to a coupon associated with the order.
  + Orderdate (Date): Date of the order.
  + productNames (List<String>): List of product names in the order.
  + shipment (Shipment): Reference to shipment information.

**2. Coupon Collection**

* **Fields:**
  + id (ObjectId): Unique identifier for the coupon.
  + couponCode (String): Code associated with the coupon.
  + discountPercentage (Double): Percentage discount offered by the coupon.
  + discountAmount (Double): Amount discounted by the coupon.

**3. Email Collection**

* **Fields:**
  + from (String): Sender's email address.
  + to (String): Recipient's email address.
  + subject (String): Subject of the email.
  + body (String): Body of the email.

**4. Image Collection**

* **Fields:**
  + id (ObjectId): Unique identifier for the image.
  + FileName (String): Name of the file.
  + contentType (String): Type of content (e.g., image/jpeg).
  + fileData (byte[]): Binary data representing the file.

**5. Order Collection**

* **Fields:**
  + id (ObjectId): Unique identifier for the order.
  + user (DBRef): Reference to user information.
  + orderItems (List<OrderItem>): List of order items.
  + totalAmount (double): Total amount for the order.
  + orderDate (LocalDateTime): Date and time of the order.

**6. OrderImage Collection**

* **Fields:**
  + id (ObjectId): Unique identifier for the order image.
  + FileName (String): Name of the file.
  + contentType (String): Type of content (e.g., image/jpeg).
  + fileData (byte[]): Binary data representing the file.

**7. OrderItem Collection**

* **Fields:**
  + id (String): Unique identifier for the order item.
  + product (DBRef): Reference to the associated product.
  + productName (List<String>): List of product names.
  + quantity (int): Quantity of the product in the order.
  + userInfo (UserInfo): User information associated with the order item.
  + coupon (Coupon): Reference to a coupon associated with the order item.
  + price (double): Price of the product.
  + totalPrice (double): Total price for the quantity.

**8. OrderReturn Collection**

* **Fields:**
  + id (String): Unique identifier for the order return.
  + productNames (List<String>): List of product names in the return.
  + name (String): Name of the person initiating the return.
  + email (String): Email address of the person initiating the return.
  + product (List<String>): List of products being returned.
  + productId (String): Identifier for the returned product.
  + quantity (int): Quantity of the returned product.
  + complaintMessage (String): Message describing the reason for the return.
  + reason (String): Categorized reason for the return.
  + status (String): Current status of the return.
  + Orderimages (List<OrderImage>): List of images associated with the return.
  + TotalAmount (Double): Total amount associated with the return.

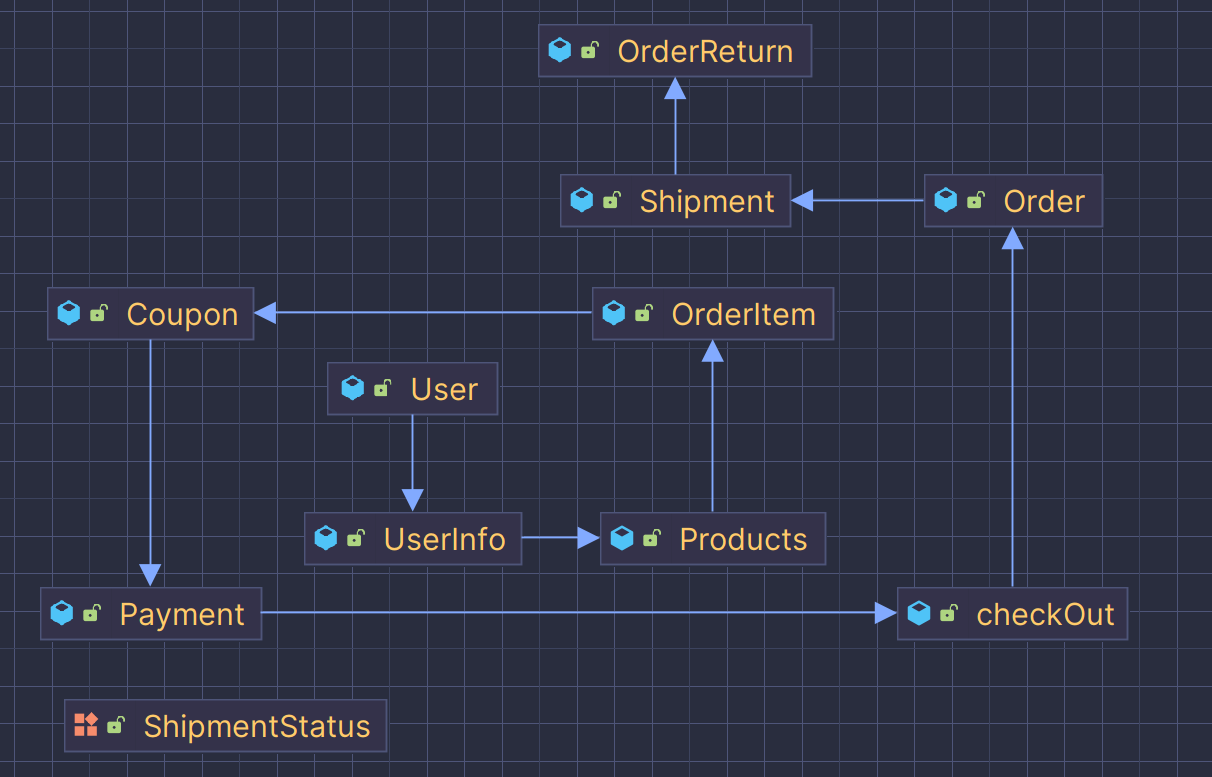
**9. Products Collection**

* **Fields:**
  + id (ObjectId): Unique identifier for the product.
  + title (String): Title of the product.
  + quantity (Integer): Available quantity of the product.
  + category (List<String>): List of categories associated with the product.
  + description (String): Description of the product.
  + brand (String): Brand of the product.
  + price (double): Price of the product.
  + images (List<Image>): List of images associated with the product.
  + product (DBRef): Reference to the parent product (if applicable).
  + orderDate (Date): Date of the order.

**10. Shipment Collection**

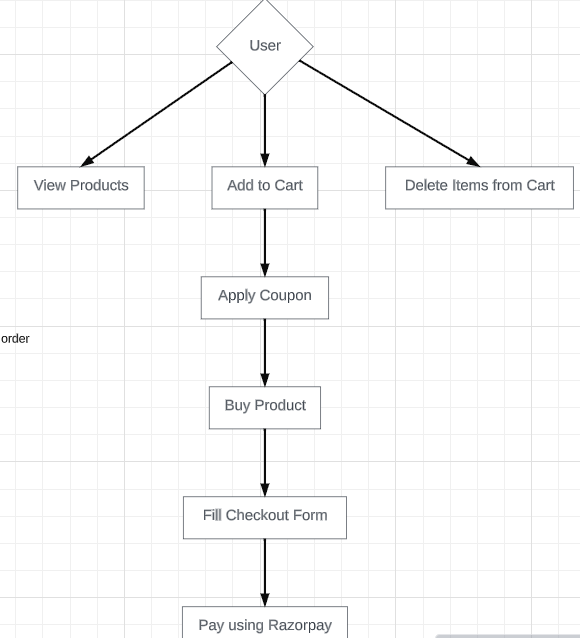
* **Fields:**
  + id (String): Unique identifier for the shipment.
  + orderId (String): Identifier of the associated order.
  + trackingNumber (String): Tracking number for the shipment.
  + productNames (List<String>): List of product names in the shipment.
  + carrier (String): Shipping carrier.
  + estimatedDelivery (Date): Estimated delivery date.

## Database Schema Relation Diagram



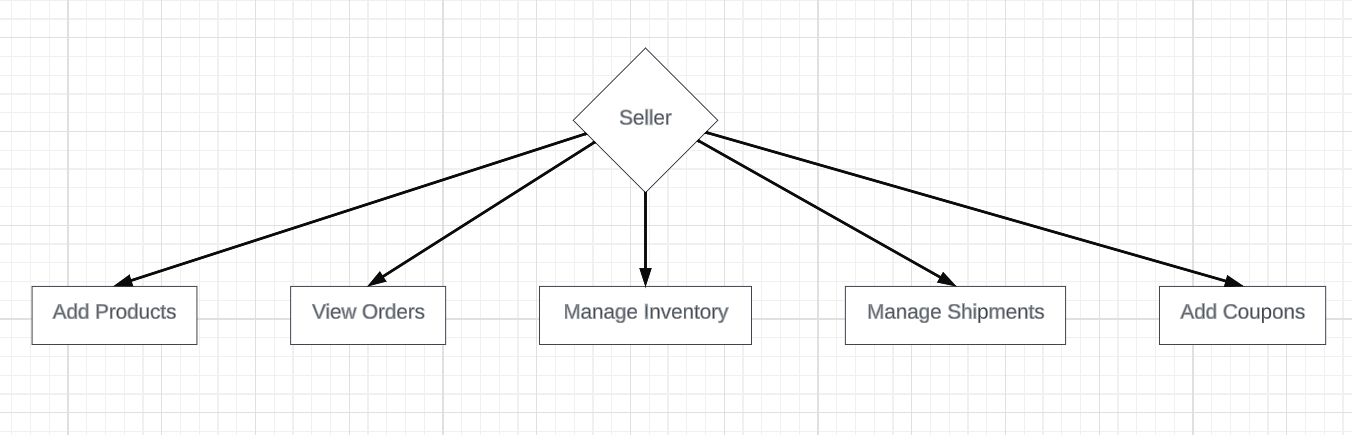
# User Roles and Permissions

## **1.** User Role: Customer

* **Permissions:**
  + View Products: Customers can browse and view the list of available products.
  + Add to Cart: Customers can add products to their shopping cart.
  + Apply Coupon: Customers can apply coupons during the checkout process to reduce the total amount.
  + Checkout: Customers can proceed to checkout, providing necessary details and selecting a payment method.
  + Pay using Razorpay: Customers can make payments for their orders using the Razorpay payment gateway.
  + Delete Items from Cart: Customers can remove items from their shopping cart. 

## 2. User Role: Seller

* **Permissions:**
  + Add Products: Sellers can add new products to the system for customers to view and purchase.
  + View Orders: Sellers have access to view the orders placed by customers.
  + Manage Inventory: Sellers can manage the inventory of products, including updating quantities and availability.
  + Manage Shipments: Sellers can oversee and manage the shipment process for orders they are responsible for.
  + Add Coupons: Sellers can create and add promotional coupons to attract customers and offer discounts.



# Authentication and Authorization Summary:

## 1. Authentication Configuration (SecurityConfig class):

* **Password Encoding:**
  + Configured BCryptPasswordEncoder as the password encoder bean.
* **User Details Service:**
  + Defined a UserDetailsService bean with a custom implementation (CustomUserDetailsService).
* **Authentication Provider:**
  + Configured a DaoAuthenticationProvider bean with the custom user details service and password encoder.
* **Security Filter Chain:**
  + Configured a SecurityFilterChain bean to define security settings.
  + Disabled CSRF protection.
  + Specified white-label paths that are accessible to everyone.
  + Defined authorization rules for specific paths based on user roles.
  + Configured a custom login page, processing URL, and success handler.

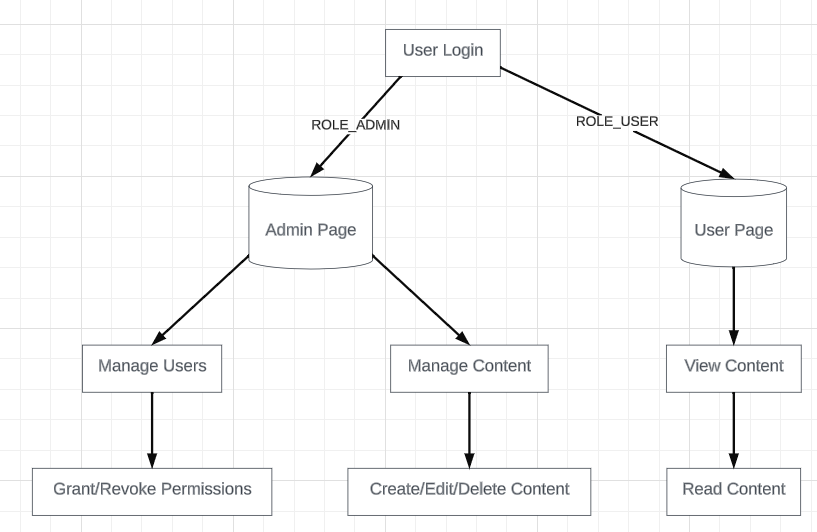
2. Custom Authentication Success Handler **(YourCustomAuthenticationSuccessHandler class):**

* **Role-Based Redirects:**
  + Implemented a custom AuthenticationSuccessHandler.
  + Redirects users based on their roles after successful authentication.
  + Redirects users with the "ROLE\_ADMIN" role to "/seller/dashboard".
  + Redirects users with the "ROLE\_USERS" role to "/users/home".
  + Redirects users with other roles to "/error".

**Additional Suggestions:**

* **Role Prefix Convention:**
  + Ensure that role names in the database follow a convention (e.g., "ROLE\_ADMIN", "ROLE\_USERS").
* **Error Handling:**
  + Consider adding a failure handler to handle authentication failures gracefully.
* **Logout Configuration:**
  + If applicable, consider adding logout configuration to handle user logout.
* **Session Management:**
  + Configure session management settings if needed (e.g., session fixation protection, maximum sessions).

## Diagram Role Based Login And Operation



# **Endpoints and API Documentation**

**1. Show Coupon Form**

* **Endpoint:**
  + GET /seller/coupon-form
* **Description:**
  + Displays the coupon form, allowing sellers to create new discount coupons for products. This form includes fields for specifying coupon details such as discount percentage, expiration date, and applicable products.

**2. Create Shipment**

* **Endpoint:**
  + POST /seller/create-shipment
* **Description:**
  + Creates a shipment for an order. Sellers can use this endpoint to initiate the shipping process for customer orders. The request should include details such as the order ID, shipping address, and other relevant information.
* **Request:**
  + Method: POST
  + Body: Shipment details including order ID, shipping address, etc.
* **Response:**
  + Status: 200 OK
  + Redirects to /seller/dashboard

**3. Process Order**

* **Endpoint:**
  + POST /users/process/{id}
* **Description:**
  + Processes a customer order, updating the order status and triggering subsequent actions. Sellers utilize this endpoint to manage the fulfillment of orders.
* **Request:**
  + Method: POST
  + Path Variable: id (Order ID)
  + Body: None
* **Response:**
  + Status: 302 Found
  + Redirects to /seller/dashboard

**4. Generate Coupon**

* **Endpoint:**
  + POST /seller/generate-coupon
* **Description:**
  + Generates a new coupon with specified details. Sellers can use this endpoint to create discounts for marketing campaigns or promotions.
* **Request:**
  + Method: POST
  + Body: Coupon details (e.g., discount amount, usage limits)
* **Response:**
  + Status: 302 Found
  + Redirects to /seller/dashboard

**5. List Coupons**

* **Endpoint:**
  + GET /seller/coupon-list
* **Description:**
  + Retrieves a list of all active coupons for a seller. The response includes details about each coupon, such as discount amount, expiration date, and applicable products.
* **Response:**
  + Status: 302 Found
  + Redirects to /seller/dashboard

**6. Get Dashboard**

* **Endpoint:**
  + GET /seller/dashboard
* **Description:**
  + Retrieves and displays the seller dashboard, providing an overview of key metrics and activities. This includes information on sales, order status, and product inventory.
* **Request:**
  + Method: GET
* **Response:**
  + Status: 200 OK
  + HTML template: sellerDashboard

**7. Approve Order Return**

* **Endpoint:**
  + GET /seller/order-returns/approve/{id}
* **Description:**
  + Approves a customer's request to return a product. This endpoint is used by sellers to manage product returns and issue refunds or replacements.
* **Request:**
  + Method: GET
  + Path Variable: id (Return request ID)
* **Response:**
  + Status: 302 Found
  + Redirects to /seller/dashboard

**8. Reject Order Return**

* **Endpoint:**
  + GET /seller/order-returns/reject/{id}
* **Description:**
  + Rejects a customer's request to return a product. Sellers can provide reasons for rejection, and the order return status is updated accordingly.
* **Request:**
  + Method: GET
  + Path Variable: id (Return request ID)
* **Response:**
  + Status: 302 Found
  + Redirects to /seller/dashboard

**9. Update Quantity**

* **Endpoint:**
  + POST /seller/updateQuantity
* **Description:**
  + Updates the quantity of a specific product in the inventory. Sellers use this endpoint to manage stock levels, such as restocking products or adjusting inventory after sales.
* **Request:**
  + Method: POST
  + Parameters: productId (Product ID), newQuantity (Integer)
* **Response:**
  + Status: 200 OK
  + Body: Response message

**10. Save Form**

* **Endpoint:**
  + POST /seller/saveForm
* **Description:**
  + Saves a form submitted by sellers, which may include images and other product-related details. This endpoint is typically used for updating product information.
* **Request:**
  + Method: POST
  + Form Data: Files and product details
* **Response:**
  + Status: 302 Found
  + Redirects to /seller/dashboard

**11. Increase Quantity**

* **Endpoint:**
  + GET /seller/dashboard/plus/{product-id}
* **Description:**
  + Increases the quantity of a specific product in the inventory. Sellers can use this endpoint for quick adjustments to stock levels.
* **Request:**
  + Method: GET
  + Path Variable: product-id (Product ID)
* **Response:**
  + Status: 200 OK
  + Body: New quantity (Integer)

**12. Decrease Quantity**

* **Endpoint:**
  + GET /seller/dashboard/minus/{product-id}
* **Description:**
  + Decreases the quantity of a specific product in the inventory. Sellers can use this endpoint for quick adjustments to stock levels.
* **Request:**
  + Method: GET
  + Path Variable: product-id (Product ID)
* **Response:**
  + Status: 200 OK
  + Body: Response message

**13. Delete User**

* **Endpoint:**
  + GET /seller/dashboard/delete/{id}
* **Description:**
  + Deletes a user account associated with a seller. This action may be used in scenarios where a user account needs to be removed from the system.
* **Request:**
  + Method: GET
  + Path Variable: id (User ID)
* **Response:**
  + Status: 302 Found
  + Redirects to /seller/dashboard

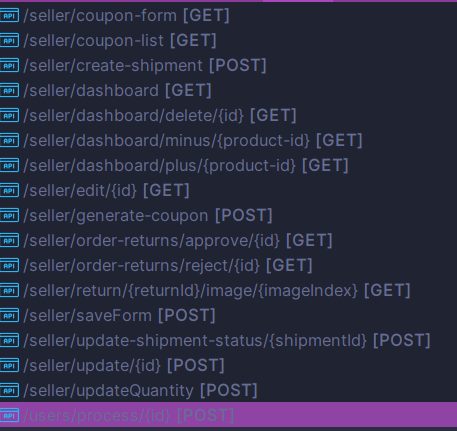
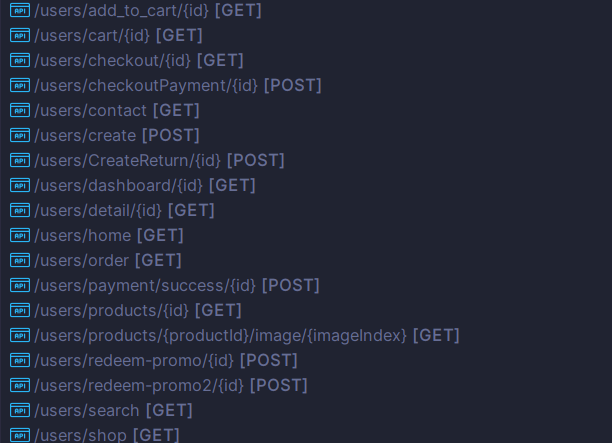
**14. Edit Product**

* **Endpoint:**
  + GET /seller/edit/{id}
* **Description:**
  + Retrieves and displays a form for editing product details. Sellers can use this form to update information such as product name, description, and images.
* **Request:**
  + Method: GET
  + Path Variable: id (Product ID)
* **Response:**
  + Status: 200 OK
  + HTML template: editProduct

**15. Update Product**

* **Endpoint:**
  + POST /seller/update/{id}
* **Description:**
  + Updates the details of a specific product. Sellers submit this form with new product information, including images and descriptions.
* **Request:**
  + Method: POST
  + Path Variable: id (Product ID)
  + Form Data: Files and updated product details
* **Response:**
  + Status: 302 Found
  + Redirects to /seller/dashboard

## RestAPIS

# Conclusion:

Thank you for exploring our Seller API documentation. This set of endpoints empowers sellers to efficiently manage their storefront, process orders, and engage in various activities crucial to a successful online business. As you integrate these endpoints into your application, please ensure adherence to the provided request and response formats.