**ShopEZ** **Application Documentation**

Contents

[1. Introduction 2](#_Toc145501093)

[2. Project Information 2](#_Toc145501094)

[*2.1* Installation 3](#_Toc145501095)

[3.Project Structure 3](#_Toc145501096)

[4. User Roles: 4](#_Toc145501097)

[4.1 Core Functionalities: 5](#_Toc145501098)

[4.2 Additional Features: 5](#_Toc145501099)

[5. User Registration and Login 6](#_Toc145501100)

[5.1 Sign In 6](#_Toc145501101)

[5.2 Register 7](#_Toc145501102)

[6 Coupon Management 8](#_Toc145501103)

[6.1 Coupon Form 8](#_Toc145501104)

[6.2 Generate Coupon 8](#_Toc145501105)

[7. Shipment Management 9](#_Toc145501106)

[7.1 Create Shipment 9](#_Toc145501107)

[7.2 Update Shipment Status 9](#_Toc145501108)

[8. Order Processing 10](#_Toc145501109)

[8.1 Approve Order Return 10](#_Toc145501110)

[8.2 Reject Order Return 10](#_Toc145501111)

[9. Product Management 11](#_Toc145501112)

[9.1 Save Product 11](#_Toc145501113)

[10. User Dashboard 12](#_Toc145501114)

[10.1 Checkout and Payment 12](#_Toc145501115)

[10.1.1 Submit Checkout and Payment 12](#_Toc145501116)

[10.2 Cart Management 12](#_Toc145501117)

[10.2.1 Manage Cart 12](#_Toc145501118)

[10.2.2 Update Cart Item Quantity 12](#_Toc145501119)

[10.4 Product Browsing 13](#_Toc145501120)

[10.4.1 Browse and Shop for Products 13](#_Toc145501121)

[10.4.2 Search for Specific Products 13](#_Toc145501122)

[11. Security Configuration 14](#_Toc145501123)

[11.1 BCrypt Password Encoder 14](#_Toc145501124)

[11.2 User Details Service 14](#_Toc145501125)

[11.3 DAO Authentication Provider 14](#_Toc145501126)

[11.4 Security Filter Chain 15](#_Toc145501127)

[11.5 Custom Authentication Success Handler 15](#_Toc145501128)

[11.6 Http Security Configuration 16](#_Toc145501129)

[12 Connecting to Local Database view MongoDB Compass 16](#_Toc145501130)

[12.1 Connecting it to Spring Boot 17](#_Toc145501131)

[13 Implementing Email Service 18](#_Toc145501132)

# 1. Introduction

Welcome to the documentation for the E-Commerce application. This documentation provides detailed information about the application's functionality, endpoints, and usage and many more features related to this project and spring boot.

# 2. Project Information

* **Project Name**: E-Commerce
* **Artifact ID**: Shop
* **Version**: 0.0.1-SNAPSHOT
* **Description**: E-Commerce

**Environment**

* **Java Version**: 17

**Spring Boot Configuration**

* **Spring Boot Version**: 2.7.2
* **Spring Boot Starter Parent**: **spring-boot-starter-parent**

**Dependencies**

* **Core Dependencies**
* **spring-boot-starter-data-mongodb**: Integration with MongoDB.
* **spring-boot-starter-mail**: Spring Boot's email support.
* **spring-boot-starter-thymeleaf**: Integration with the Thymeleaf template engine.
* **spring-boot-starter-web**: Starter for building web applications.
* **spring-boot-starter-test** (Scope: Test): Starter for Spring Boot testing.
* **JSON and Data Processing Dependencies**
* **gson** (Version: 2.6.1): Google Gson library for JSON processing.
* **stripe-java** (Version: 22.30.0): Stripe Java library for payment processing.
* **json-path**: Library for querying JSON data.
* **Development and Testing Dependencies**
* **spring-boot-devtools** (Scope: Runtime, Optional): Development-time tools, including automatic restarts.
* **lombok** (Optional): Library for reducing boilerplate code in Java.
* **spring-test** (Version: 6.0.10): Part of the Spring Framework for testing.
* **annotations** (Version: 13.0, Scope: Compile): Library for Java annotations.
* **dropwizard-core** (Version: 1.0.2): Java libraries for building production-ready web services.
* **dropwizard-views** (Version: 1.0.2): Extension for views in Dropwizard.
* **dropwizard-views-freemarker** (Version: 1.0.2): Freemarker template support.
* **razorpay-java** (Version: 1.3.1): Razorpay Java library for payment integration.
* **hibernate-validator** (Version: 5.2.4.Final): Library for implementing validation rules.
* **derby** (Version: 10.12.1.1): Apache Derby, an embedded relational database.
* **spring-security-crypto**: Spring Security library for cryptographic operations.
* **spring-security-config**: Spring Security configuration library.
* **hibernate-core** (Version: 6.2.2.Final): Hibernate ORM framework.
* **rest-api-sdk** (Version: 1.4.2): PayPal REST API SDK.
* **spring-boot-starter-security**: Starter for Spring Security.

### 

# 2.1 Installation

### Clone the Repository:

* + Download or clone the project from the source repository.
  + Repository Link: <https://github.com/moxhadeel571/ShopEZOne-Stop-Shop-for-Online-Purchases.git>
  + You can use the following Git command to clone the repository to your local machine:

git clone https://github.com/moxhadeel571/ShopEZOne-Stop-Shop-for-Online-Purchases.git

**Configure the Database:**

* + Set up the database connection details in the **application.properties** file. This file can usually be found in the **src/main/resources** directory of your project.
  + Configure properties like database URL, username, and password to establish a connection with your database server.

**Build and Run:**

* + Open a command prompt or terminal and navigate to the project directory.
  + Execute the following Maven commands to build and run the project:
    - To clean and build the project:

mvn clean install

* + - To run the application:

mvn spring-boot:run

**Access the Application:**

* + Once the application is running, open your web browser.
  + In the browser's address bar, enter the following URL: **http://localhost:8080/**
  + This will take you to the registration page of the ShopEZ application.

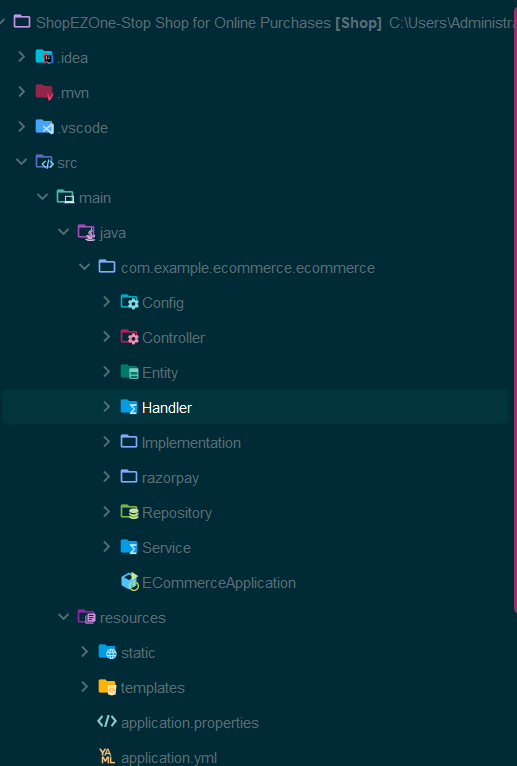
**Registration Page:**

* + On the registration page, you can create a new account to access the ShopEZ platform.
  + Fill out the required registration information, including your name, email address, password, and any other relevant details.
  + Click the "Register" or "Sign Up" button to create your account.
  + You should receive a confirmation message upon successful registration

# 3.Project Structure

**Project Structure Overview:**

The project structure follows best practices for organizing a Spring Boot application. It is designed to maintain a clean and modular codebase, making it easier to develop, test, and maintain your E-Commerce application.



# 4. User Roles:

1. **Seller:**
   * Can post products with detailed information, including name, brand, and other relevant details.
   * Manages product listings on the platform.
   * Handles return requests from customers.
   * Approves or rejects return requests based on provided proof and form details.
2. **Customer:**
   * Can view products listed on the platform.
   * Can purchase products and add them to the cart.
   * Has the ability to initiate return requests for purchased items.
   * Can track the status of their return requests.

## 4.1 Core Functionalities:

Seller-Specific Functionalities:

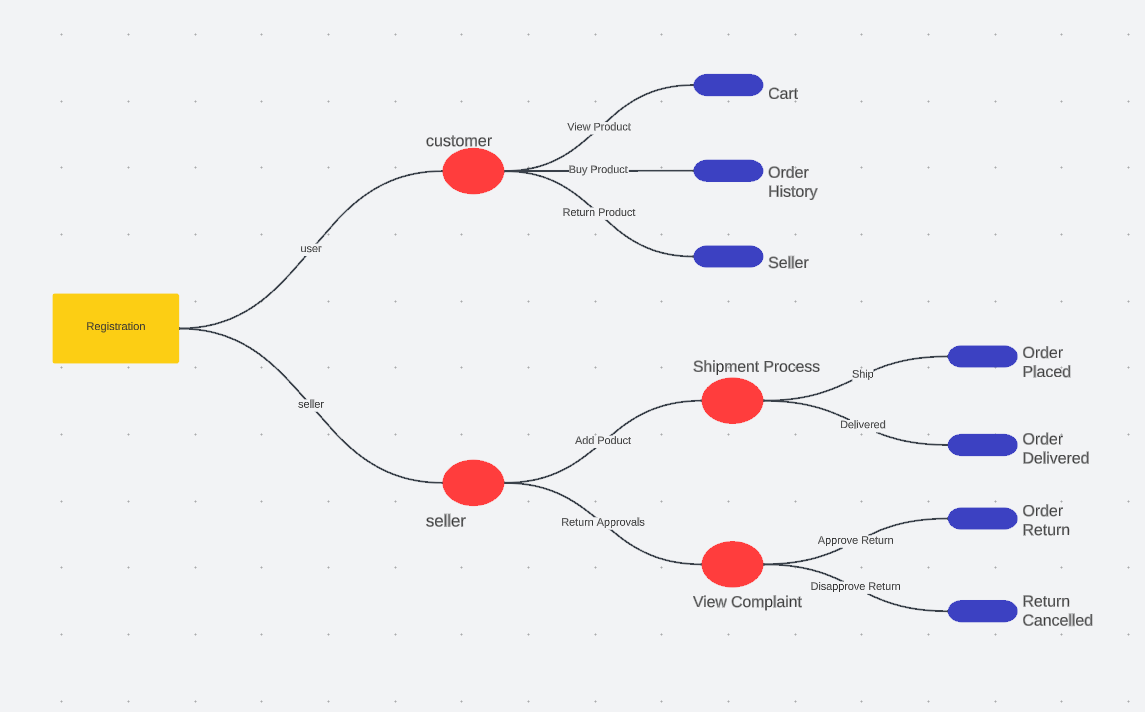
* **Product Posting:**
  + Sellers can add new products to their store, including detailed product information.
* **Return Request Handling:**
  + Sellers can view return requests initiated by customers.
  + They can review proof and form details provided by the customer.
  + Sellers can approve or reject return requests.

Customer-Specific Functionalities:

* **Product Browsing:**
  + Customers can browse products listed on the platform.
  + They can view product details, such as name, brand, and description.
* **Cart Management:**
  + Customers can add products to their shopping cart.
  + They can proceed to checkout and make purchases.
* **Return Request Tracking:**
  + Customers can initiate return requests for purchased items.
  + They can track the status of their return requests.

## 4.2 Additional Features:

* **Order Processing:**
  + Orders made by customers can be automatically processed by the seller.
  + This might include order confirmation, payment processing, and shipping arrangements.
* **Individual Dashboards:**
  + Sellers and customers each have their own dashboards.
  + Sellers can manage product listings and view return requests.
  + Customers can track their orders and return requests, as well as manage their account details.
* **Authentication and Authorization:**
  + Implement user authentication and authorization to ensure that only authorized users can access specific functionalities.
* **Database Integration:**
  + Utilize a database to store product information, user details, order history, and return request data.
* **User Interface (UI):**
  + Design user-friendly web pages for product listings, cart management, return request forms, and dashboards.
* **Payment Integration:**
  + Integrate payment gateways to facilitate secure and seamless transactions between customers and sellers.



# 5. User Registration and Login

## 5.1 Sign In

* Endpoint: /signin (GET)
* Description: Display the login page for users to sign in.
* Usage: Access this page to log in to the application.

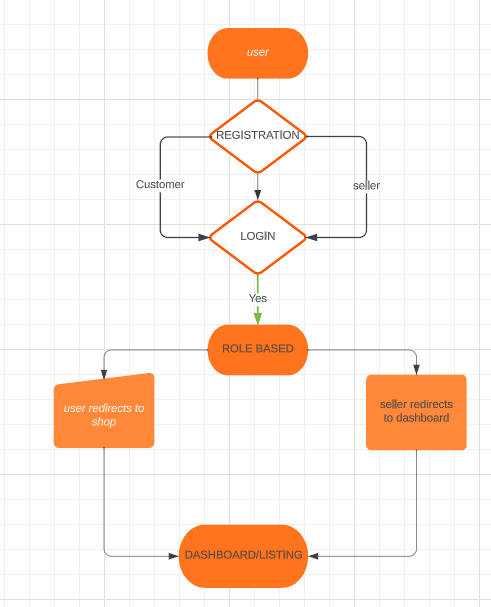
Explanation: The /signin endpoint provides a user-friendly web page where existing users can enter their credentials and sign in to the application. This page typically includes input fields for the username and password, as well as a "Sign In" button. Upon successful authentication, users gain access to their accounts and can use the application's features.

## 5.2 Register

* Endpoint: /register (GET)
* Description: Display the registration page for new users.
* Usage: Access this page to create a new account.

Explanation: The /register endpoint offers a web page where individuals who are not yet registered as users can sign up for the application. This page typically includes fields for users to provide necessary registration information, such as their name, email address, desired username, and password. After completing the registration form and submitting it, the user's account is created, allowing them to log in with their newly established credentials.

These registration and login endpoints are essential components of user authentication and account management in a web application. They facilitate user access control and personalization of content and services.



# 6 Coupon Management

## 6.1 Coupon Form

Endpoint: /seller/coupon-form (GET)

Description: Display a form for creating coupons.

Usage: Access this page to create new coupons.

Explanation: The /seller/coupon-form endpoint provides sellers with a user-friendly web page that contains a form. This form allows sellers to input details for creating new coupons, such as coupon codes, discounts, expiration dates, and any other relevant information. Sellers can submit this form to generate new coupons.

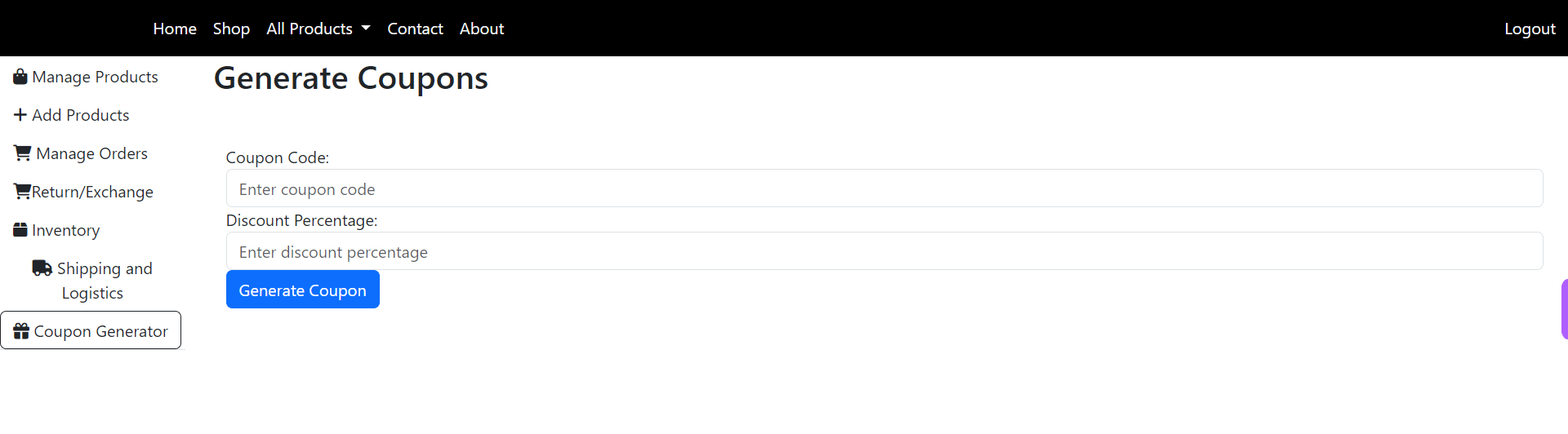
## 6.2 Generate Coupon

Endpoint: /seller/generate-coupon (POST)

Description: Generate a new coupon.

Usage: Submit the form from the /seller/coupon-form page to create a new coupon.

Explanation: The /seller/generate-coupon endpoint processes the data submitted via the coupon creation form. When sellers complete the form and submit it, this endpoint generates a new coupon based on the provided information. The newly created coupon is added to the system, making it available for use by customers.



# 7. Shipment Management

## 7.1 Create Shipment

* Endpoint: /seller/create-shipment (POST)
* Description: Create a new shipment.
* Usage: Submit shipment details to create a new shipment.

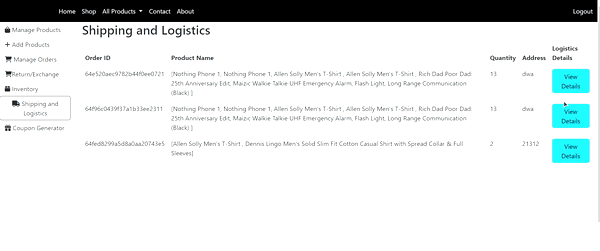
Explanation: The /seller/create-shipment endpoint allows sellers to initiate the creation of a new shipment. Sellers can submit the necessary shipment details, such as the recipient's address, package contents, and shipping method. Upon submission, the system generates a new shipment record, and the shipment process is initiated.

## 7.2 Update Shipment Status

* Endpoint: /seller/update-shipment-status/{shipmentId} (POST)
* Description: Update the status of a shipment.
* Usage: Update the shipment status using this endpoint.

Explanation: The /seller/update-shipment-status/{shipmentId} endpoint enables sellers to update the status of a specific shipment. Sellers provide the shipmentId as a parameter to specify which shipment they want to update. They can then submit the new status for the shipment, such as "Shipped," "In Transit," or "Delivered." This endpoint facilitates real-time tracking and management of shipments, allowing sellers to keep customers informed about the progress of their orders.

These shipment management endpoints empower sellers to efficiently manage the logistics and tracking of orders, ensuring that products are delivered to customers in a timely and organized manner.



# 8. Order Processing

## 8.1 Approve Order Return

* Endpoint: /seller/order-returns/approve/{id} (GET)
* Description: Approve an order return.
* Usage: Access this page to approve order returns.

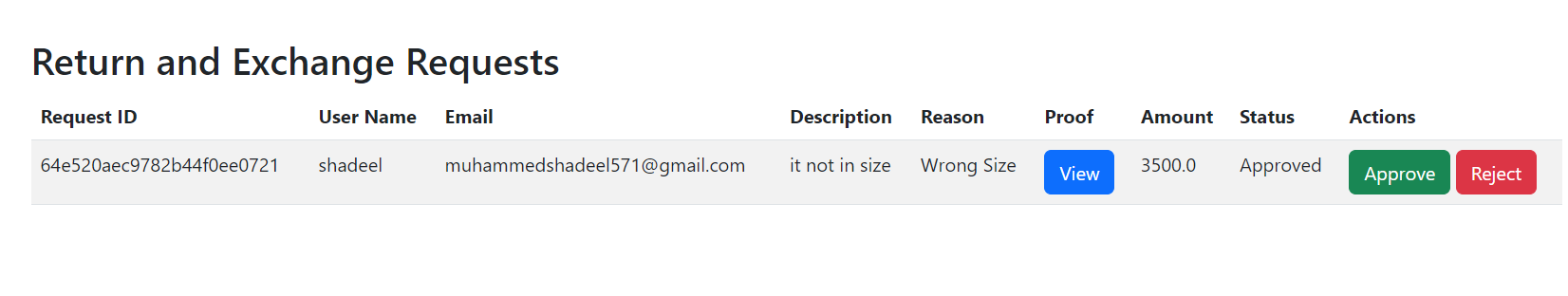
Explanation: The /seller/order-returns/approve/{id} endpoint provides sellers with a web page where they can review and approve order return requests from customers. Sellers can access this page to view the details of a specific return request identified by {id} and then choose to approve the return. This action confirms that the return is accepted, and further processing can be initiated.

## 8.2 Reject Order Return

* Endpoint: /seller/order-returns/reject/{id} (GET)
* Description: Reject an order return.
* Usage: Access this page to reject order returns.

Explanation: The /seller/order-returns/reject/{id} endpoint offers sellers a web page where they can review and reject order return requests from customers. Sellers can access this page to view the details of a specific return request identified by {id} and then choose to reject the return. Rejecting a return indicates that the seller does not approve the return, and the customer will be informed of the decision.

These order processing endpoints enable sellers to efficiently manage return requests, either by approving them for further action or rejecting them based on their assessment of the request's validity and conditions.

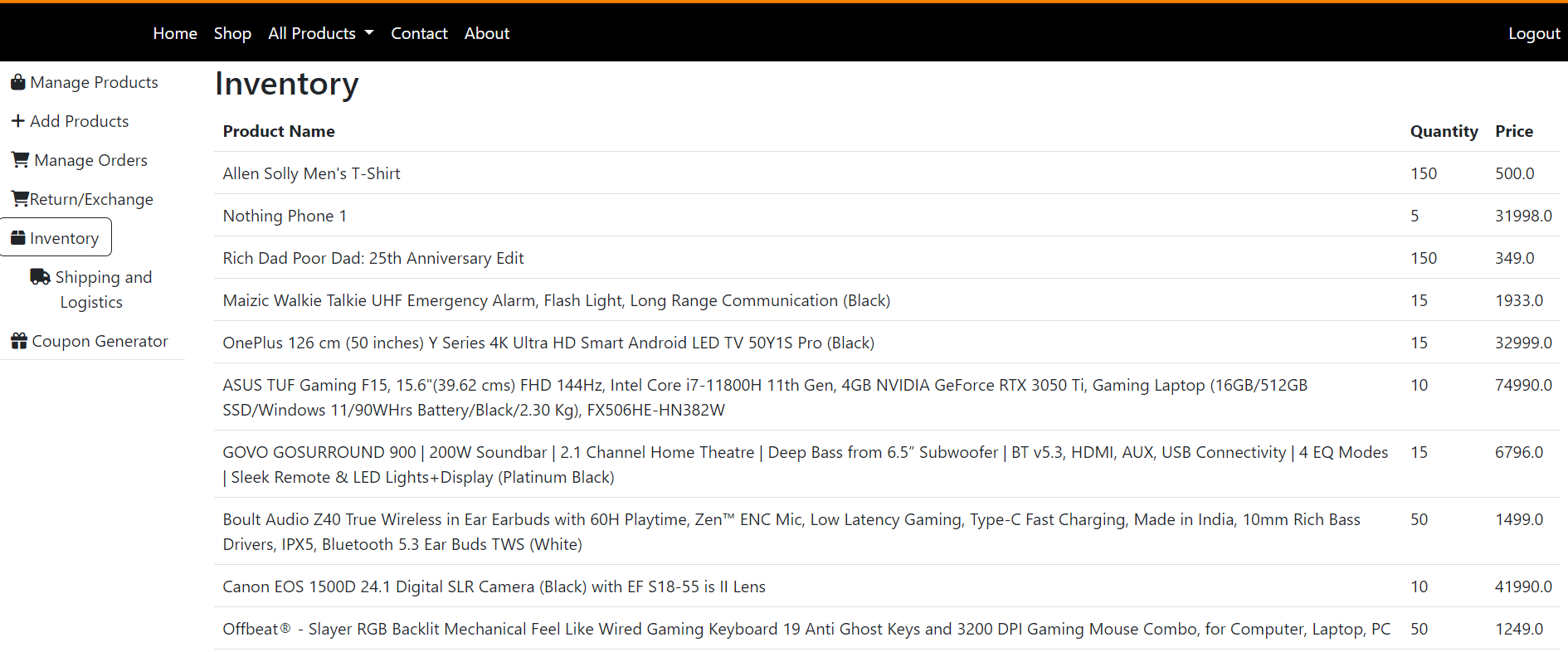


# 9. Product Management

## 9.1 Save Product

* Endpoint: /seller/saveForm (POST)
* Description: Save product information with images.
* Usage: Submit product details and images to add a new product.

Explanation: The /seller/saveForm endpoint allows sellers to add new products to their store. Sellers can submit product details, including images, through a form. Upon submission, the system saves the product information along with associated images, making the product available for customers to view and purchase.



# 10. User Dashboard

## 10.1 Checkout and Payment

### 10.1.1 Submit Checkout and Payment

* Endpoint: /users/checkoutPayment/{id} (POST)
* Description: Submit a checkout form for payment.
* Usage: Fill out the checkout form and make a payment.

Explanation: The /users/checkoutPayment/{id} endpoint allows users to complete their purchase by submitting a checkout form and making a payment. Users provide the necessary information, such as shipping address and payment details, to finalize their order and complete the payment process.

## 10.2 Cart Management

### 10.2.1 Manage Cart

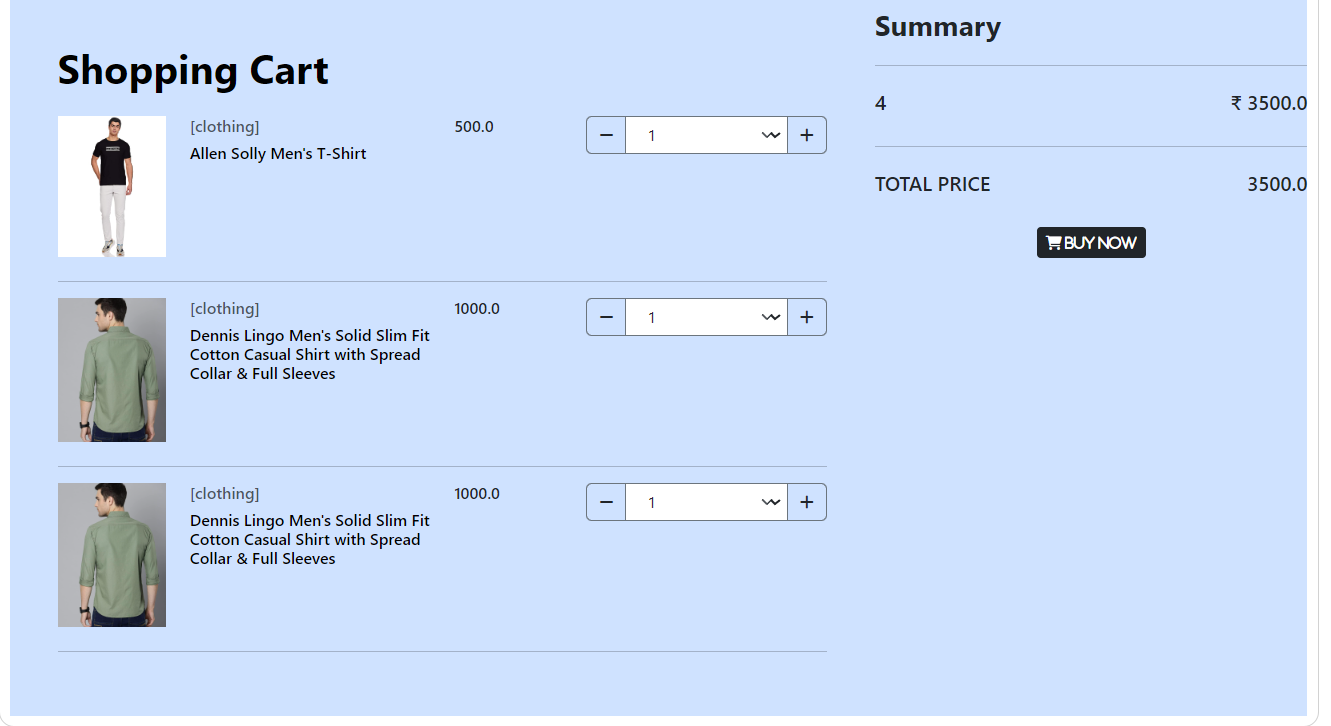
* Endpoint: /users/cart/{id} (GET)
* Description: Manage items in the shopping cart.
* Usage: Access this page to view and manage items in the cart.

Explanation: The /users/cart/{id} endpoint provides users with access to their shopping cart, where they can view the items they've added for purchase. Users can use this page to manage their cart, including adding, removing, or adjusting the quantities of items in preparation for checkout.

### 10.2.2 Update Cart Item Quantity

* Endpoint: /users/updateQuantity (POST)
* Description: Update the quantity of a product in the cart.
* Usage: Submit a request to update the quantity of a product in the cart.

Explanation: The /users/updateQuantity endpoint allows users to dynamically update the quantity of a specific product in their shopping cart. Users can adjust the quantity of items they wish to purchase, and the cart is updated accordingly.

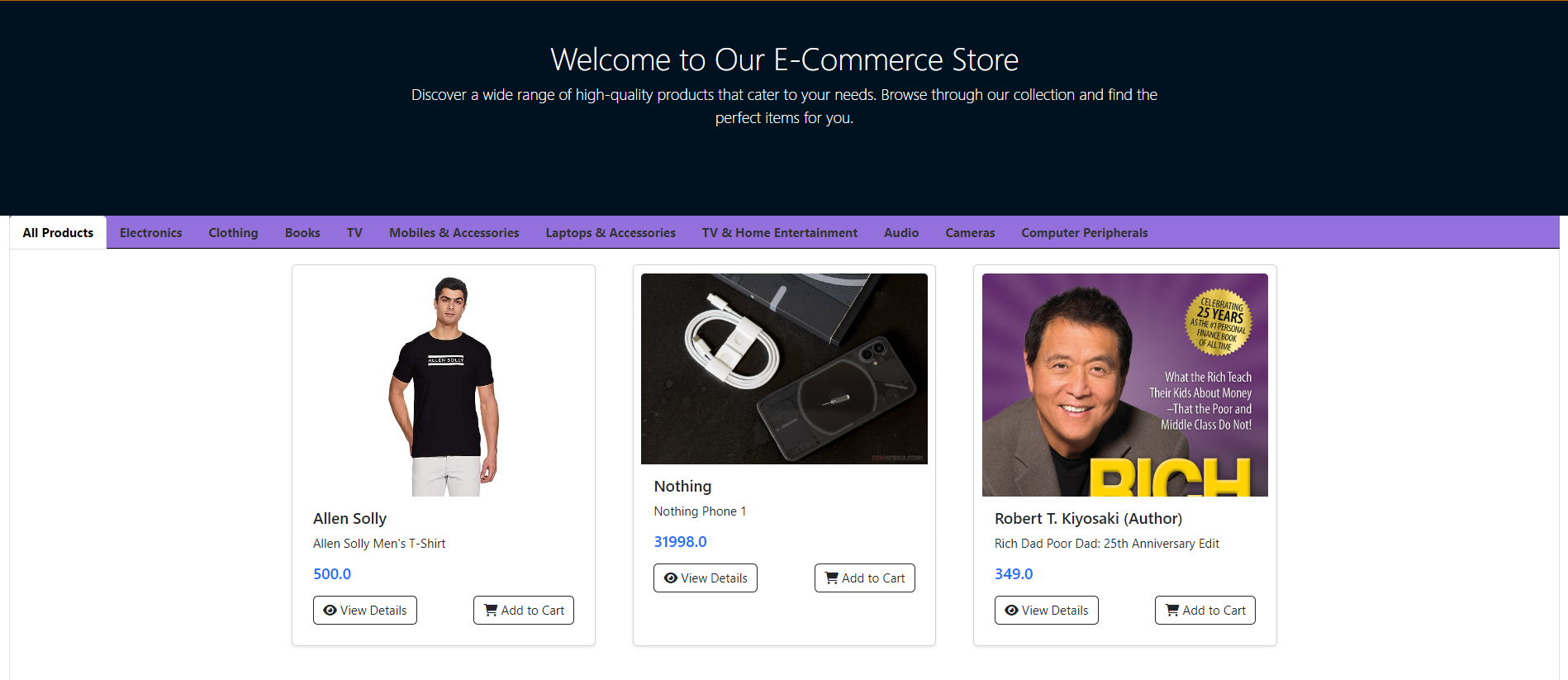


## 10.4 Product Browsing

### 10.4.1 Browse and Shop for Products

* Endpoint: /users/shop (GET)
* Description: Browse and display products for purchase.
* Usage: Access this page to browse and shop for products.

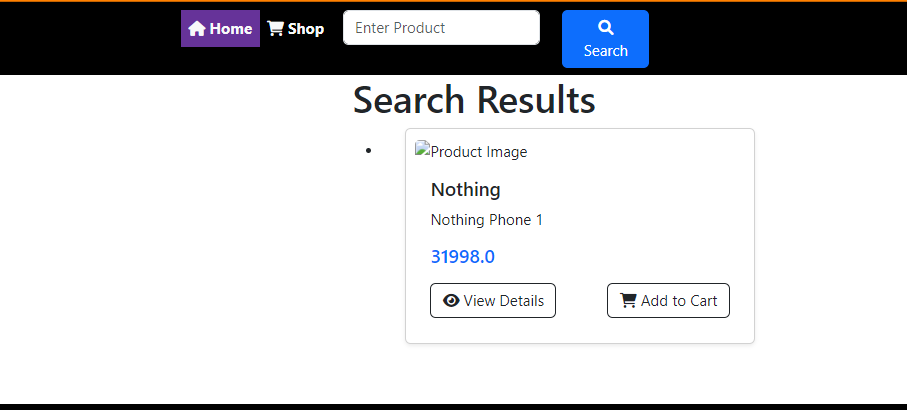
Explanation: The /users/shop endpoint allows users to explore and shop for products available on the platform. Users can view product listings, including details such as product name, brand, and description, and add desired items to their shopping cart for purchase.



### 10.4.2 Search for Specific Products

* Endpoint: /users/search (GET)
* Description: Search for specific products.
* Usage: Access this page to search for products by keyword.

Explanation: The /users/search endpoint enables users to perform product searches based on specific keywords or criteria. Users can access this page to initiate product searches and receive relevant product listings that match their search terms.



# 11. Security Configuration

The security configuration of the Food Ordering System is responsible for managing user authentication and authorization. It defines the access rules, authentication providers, and login processing settings. Below, you'll find an overview of the key elements and functionalities within the security configuration.

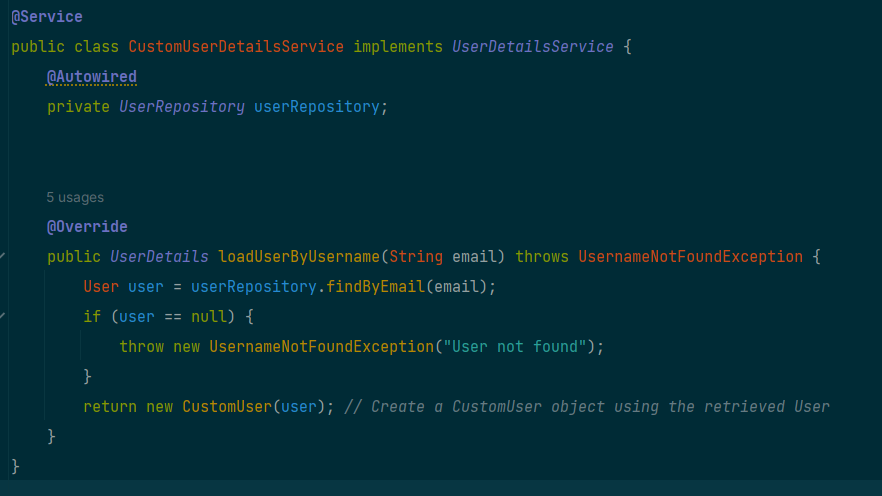
## 11.1 BCrypt Password Encoder

* **Description**: The **BCryptPasswordEncoder** bean is responsible for securely hashing and encoding user passwords.



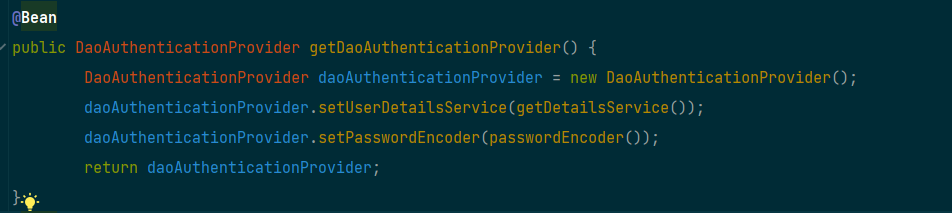
## 11.2 User Details Service

* **Description**: The **UserDetailsService** bean is used to retrieve user details from the database during authentication.



## 11.3 DAO Authentication Provider

* **Description**: The **DaoAuthenticationProvider** bean configures the authentication provider, specifying the user details service and password encoder.



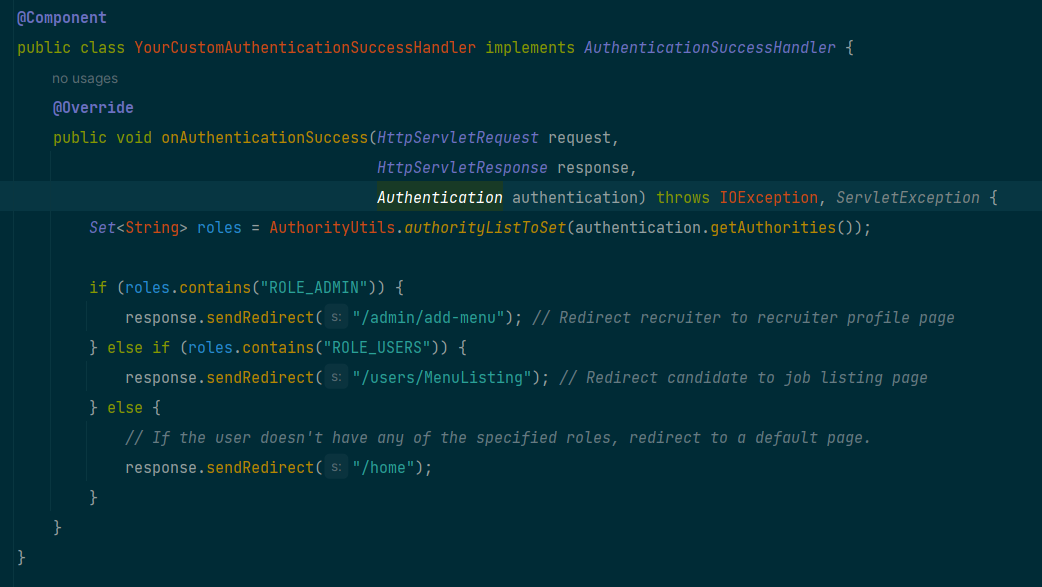
## 11.4 Security Filter Chain

* **Description**: The **SecurityFilterChain** bean defines the security filter chain, specifying various security configurations and access rules.

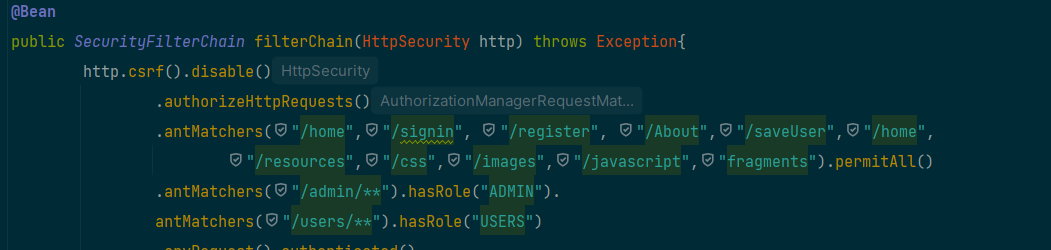


## 11.5 Custom Authentication Success Handler

* **Description**: A custom authentication success handler (**YourCustomAuthenticationSuccessHandler**) is configured to handle successful login attempts.



## 11.6 Http Security Configuration



* **Description**: The **HttpSecurity** configuration defines the security rules and access restrictions for different URL patterns. It specifies which URLs are accessible without authentication (**permitAll**), which require specific roles (**hasRole**), and configures the login process.
  + **/home**, **/signin**, **/register**, **/About**, **/saveUser**, **/home**, **/resources**, **/css**, **/images**, **/javascript**, **fragments**: These URLs are permitted for access without authentication.
  + **/admin/\*\***: Requires users with the "ADMIN" role to access URLs under this pattern.
  + **/users/\*\***: Requires users with the "USERS" role to access URLs under this pattern.
  + Any other URL requires authentication.
  + **Form Login Configuration**: Configures form-based login with a custom login page (**/signin**), login processing URL (**/userLogin**), and a custom authentication success handler.

The security configuration ensures that users have appropriate access rights based on their roles and authentication status. It also incorporates secure password hashing and customizable login handling.

# 12 Connecting to Local Database view MongoDB Compass

To connect to your local MongoDB database using MongoDB Compass, you can follow these steps:

1. **Install MongoDB Compass**:
   * If you haven't already installed MongoDB Compass, you can download it from the official MongoDB website: [MongoDB Compass Download](https://www.mongodb.com/try/download/compass)
2. **Launch MongoDB Compass**:
   * After installing, launch MongoDB Compass on your computer.
3. **Connect to Your Local MongoDB**:
   * When you first open MongoDB Compass, you'll be presented with a "Connect to Host" screen.
   * In the "Hostname" field, enter the address of your local MongoDB server. By default, MongoDB runs on **localhost** and port **27017**, so you can leave these values as is.
   * You can also specify a connection name for reference.
4. **Advanced Options**:
   * If you have additional configurations or need to connect to a specific database, you can click on "More Options" to expand the settings.
   * Here, you can specify authentication credentials, replica set, SSL options, and other advanced settings. For a local development environment, you may not need these.
5. **Test Connection**:
   * Before connecting, you can click the "Test Connection" button to ensure that MongoDB Compass can connect to your local MongoDB server without issues.
6. **Connect**:
   * Once you are satisfied with the settings, click the "Connect" button to establish a connection.
7. **Explore Your Local Database**:
   * After successfully connecting, you will see a list of available databases on your local MongoDB server.
   * Click on a database to explore its collections and documents.
8. **Perform Database Operations**:
   * You can perform various database operations using MongoDB Compass, such as querying data, inserting documents, updating records, and more. Simply navigate to the relevant collection and use the provided GUI tools.

## 12.1 Connecting it to Spring Boot

To connect your Spring Boot application to a local MongoDB database using MongoDB Compass, you need to configure your Spring Boot application's **application.properties** or **application.yml** file with the MongoDB connection details. Here's how you can do it:

**1. Add MongoDB Dependency:**

First, make sure you have the MongoDB dependency added to your Spring Boot project. You can add it to your **pom.xml** file if you're using Maven.

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-mongodb</artifactId>

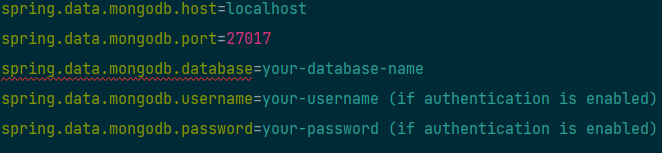
</dependency>

I have provided MongoConfig too so use it to properly connect database to your compass

9.2. Configure MongoDB Connection:

In your **application.properties** or **application.yml** file, specify the MongoDB connection details. Make sure to adjust the values to match your local MongoDB setup.

**Using application.properties (Recommended):**



# 13 Implementing Email Service

**Overview**

The **Email** class in your Spring Boot application is responsible for sending various types of emails, including promotional emails and order confirmation emails. This documentation provides an in-depth explanation of the class, its methods, and how to configure and use it effectively.

The **EmailServiceImpl** class encapsulates email-sending functionality in your application. It offers two key methods: **sendEmail** for sending general promotional or informational emails and **order\_email** for sending order confirmation emails. This documentation provides comprehensive guidance on setting up and using the **EmailServiceImpl** class effectively.

**Dependencies**

As for dependency I have already provided you the one in pom.xml file

10.1 Configurating Email Service for spring boot

Adding this to pom.xml file will configure the service with smtp service choose which smtp service you want below I have added the guidance for SMTP service from google

