Online-Shopping-Platform

An online shopping system enables customers to browse, select, and purchase products or services through a website or app. Customers can add items to a virtual shopping cart, review their selections, and proceed to checkout, where they enter payment and shipping information. It also manages inventory, provides customer support, and offers analytics for businesses to track sales and customer behaviour. Overall, online shopping systems provide a convenient and efficient way for customers to shop and for businesses to sell products or services online.

Installation & Setup

Create a Spring Boot Project: Use your IDE or the Spring Initializer to create a new Spring Boot project with dependencies for Spring Security, JWT, and your chosen database.

Configure Spring Security: Configure authentication and authorization rules based on your requirements. Configure JWT authentication filter and add it to the Spring Security filter chain.

Architecture Overview

Server Side (Backend):

Spring Boot Application: Handles business logic, data access, and serves as the RESTful API endpoint for the frontend.

Spring Security: Provides authentication and authorization features.

MySQL Database: Stores product information, user details, orders, and other relevant data.

Spring Data JPA: Facilitates interaction with the MySQL database through JPA (Java Persistence API) entities and repositories.

Spring MVC: Handles incoming HTTP requests and routes them to the appropriate controller methods.

Swagger UI: A web interface that visualizes and interacts with the API documentation.

Swagger Core: A library that generates the API documentation based on annotations in the Java code.

Authentication and Authorization:

- Uses Spring Security to secure the API endpoints and authenticate users.
- Generates and validates JWT (JSON Web Tokens) for stateless authentication.

Architecture Overview Diagram

In this detailed diagram:

- The backend, implemented using Spring Boot, includes security features (Spring Security with JWT authentication), business logic, controllers, data access (Spring Data JPA with repositories), and integration with external APIs (e.g., payment gateways, shipping APIs).
- The MySQL database stores data related to products, users, orders, and other relevant information.
- Other components, such as microservices for additional functionality, can also interact with the backend and the database.

Platform Owner:

Manage Users: Ability to view and manage user accounts (Product Owners and Customers).

Manage Products: Add, edit, and remove products from the platform.

Manage Product Categories: Create and manage product categories.

Product Owner:

Product Management: Add, edit, and remove products from their catalog.

Inventory Management: Manage product inventory and availability.

Order Management: View and manage orders placed for their products.

Customer:

User Account: Create and manage their account, including personal information and order history.

Browse Products: Search and browse products based on categories, filters, and keywords.

Add to Cart: Select and add products to their shopping cart for later purchase.

Common Features:

Authentication and Authorization: Secure login and access control based on roles using JWT.

Search and Filters: Search functionality and filters to easily find products.

Order Management: Ability to track and manage orders for both customers and product owners.

DETAILS:

Platform Owner:

The Platform Owner of an online shopping platform is responsible for the strategic direction and overall success of the business. They oversee key areas such as strategic planning, business development, financial management, user acquisition and retention, partnership management, technology and innovation, and legal and compliance. The Platform Owner works to expand the platform's reach and market presence, manage relationships with partners and stakeholders, and ensure compliance with relevant laws and regulations. They also have access to features such as a comprehensive dashboard for monitoring performance, user and product management capabilities, analytics and reporting tools, promotions and discounts management, content management, order management, customer support, integration management, and ensuring security and compliance.

Product Owner:

The Product Owner in an online shopping platform is responsible for managing a specific set of products within the platform's catalogue. Their primary role involves product management, including adding new products, updating existing ones, and removing outdated or discontinued items. Product Owners also manage inventory levels, ensuring that products

are available for purchase and coordinating with suppliers to restock as needed. They analyse sales data and customer feedback to make informed decisions about pricing, promotions, and product improvements. Product Owners work closely with the Platform Owner and other stakeholders to align product strategies with the platform's overall goals. They also handle order management for their products, overseeing the fulfilment process and ensuring timely delivery to customers. Additionally, Product Owners may be involved in marketing and sales efforts, collaborating with the marketing team to promote their products and increase sales.

Customer:

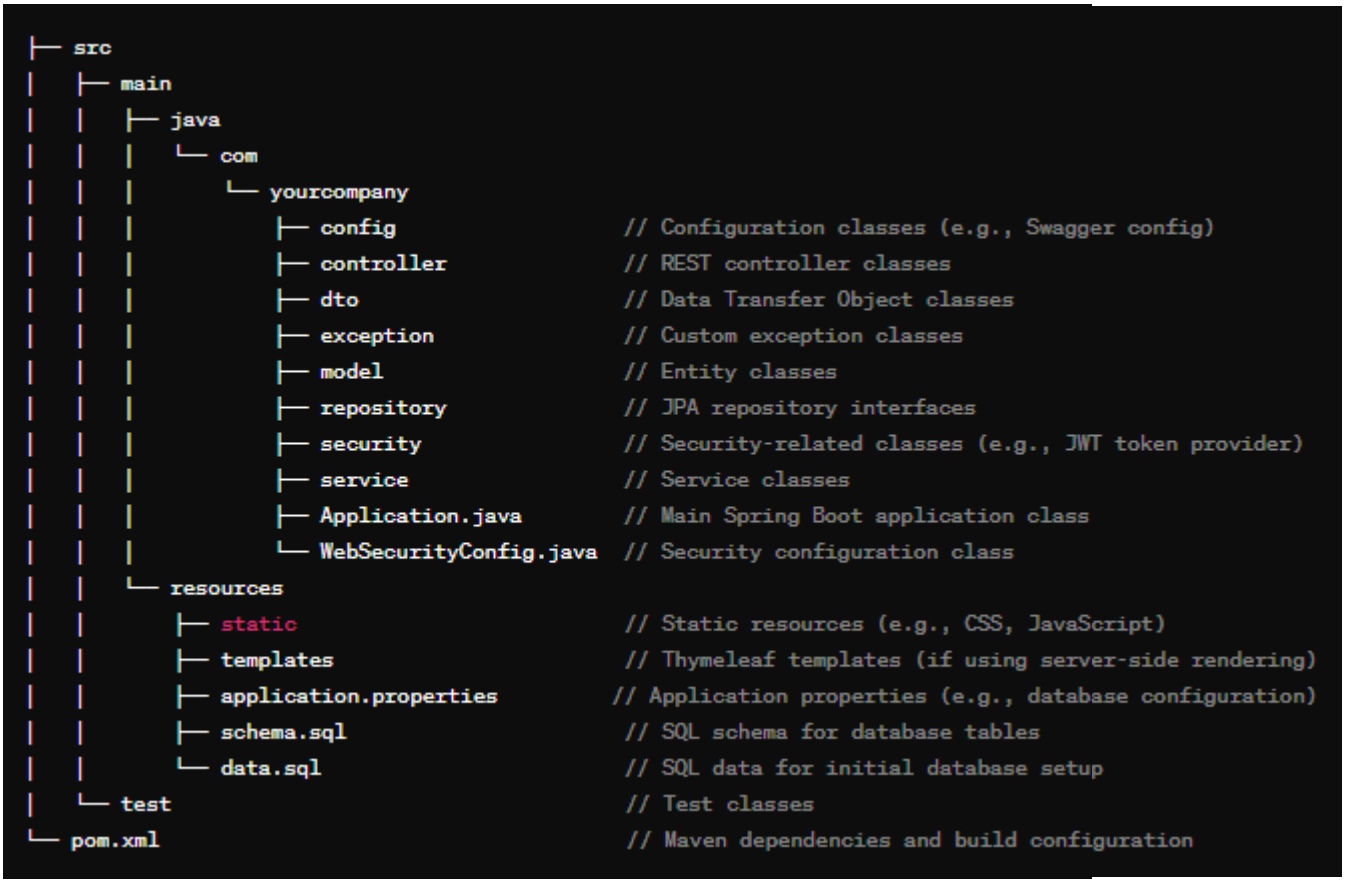
Customers are the backbone of any online shopping platform, playing a crucial role in its success. As the end-users of the platform, customers browse products, make purchases, and provide feedback, shaping the platform's offerings and user experience. Customers create accounts to manage their personal information, view order history, and track deliveries. They interact with the platform's user interface to search for products, add items to their shopping carts, and proceed to checkout. Customers value a seamless and secure shopping experience, expecting features like product reviews, ratings, and recommendations to aid their decision-making process. Customer support is essential, and customers rely on timely assistance for inquiries, issues, and returns. Ensuring a positive customer experience encourages repeat purchases and builds brand loyalty, highlighting the importance of customer-centric features and services on the platform.

User Management:

User management for an online shopping platform involves the comprehensive administration of user accounts and interactions to ensure a seamless and secure experience. This process begins with user registration, where individuals create accounts by providing essential information such as name, email, and password. Authentication mechanisms are then employed to verify user identity and grant access to the platform. Role-based access control (RBAC) is implemented to assign different roles to users, such as customer, product owner, or platform owner, each with specific permissions and access levels. Users can manage their profiles, update personal information, and select preferences.

Additionally, features like order history, tracking, and wish list management are provided in a personalized user dashboard. Security measures, including password hashing and encryption, are enforced to protect user accounts and data. Communication channels, such as notifications for order updates and promotions, are integrated, along with feedback and review systems for users to share their experiences. User management also includes compliance with data protection regulations, such as GDPR, and analytics to understand user behaviour and enhance the overall shopping experience.

Project Structure:



COMPONENTS:

Model:

Defines the structure of entities such as Task, Employee, Employer, etc. Includes JPA annotations for mapping entities to database tables.

View:

Frontend interface for interacting with the application. Implemented using HTML, CSS, and JavaScript frameworks (not covered in this document).

Controller:

Handles HTTP requests and orchestrates the flow of data between the model and view. Implements RESTful APIs for task management operations.

Service:

Contains business logic and interacts with the repository layer. Manages tasks, users, and other application functionalities.

Repository:

Interface for interacting with the database. Implements CRUD operations and data access logic using Spring Data JPA.

API DEVELOPMENT :

1. JWT Authentication for Role-Based Access:

Implement JWT token generation upon successful login. Use roles (Employer and Employee) to restrict access to certain endpoints.

2. Auditing for Task Changes:

Utilize Spring Data JPA's auditing features to automatically track changes to entities. Implement custom auditing for specific entities like tasks.

3. Documenting API Endpoints:

Create detailed documentation specifying each API endpoint, HTTP method, request/response format, and functionality. Include examples for request/response payloads. Use tools like Swagger or Spring Rest Docs for automated API documentation.

4. User Authentication APIs:

• Login Endpoint: Authenticates users and generates JWT tokens.

• Logout Endpoint: Clears the user's authentication token.

• User Profile Endpoint: Retrieves user details based on the authenticated token.

5. Task CRUD Operations:

• Create Task Endpoint: Allows authenticated users to create tasks.

• Read Task Endpoint: Retrieves task details based on task ID.

• Update Task Endpoint: Allows authorized users to update task details.

• Delete Task Endpoint: Allows authorized users to delete tasks

Api Documentation:

Add Swagger Dependencies:

Update your Maven '**pom.xml**' file to include the necessary Swagger dependencies. Look for dependencies related to Swagger or OpenAPI.

Configure Swagger:

Configure Swagger in your project by creating a configuration class. This class should specify the base package where your controller classes are located and any additional settings, such as API versioning and documentation details.

Annotate Controller Methods:

Annotate your controller methods with Swagger annotations to provide descriptions, examples, and other metadata. Use annotations like '**@ApiOperation**', '**@ApiParam**', and '**@ApiResponse**'.

Access Swagger UI:

Run your Spring Boot application and navigate to the Swagger UI endpoint in your web browser. By default, the Swagger UI is often accessible at a URL like "**http://localhost:8080/swagger-ui/**"

View Documentation:

Once you access the Swagger UI, you should see your API documentation, including clear descriptions and examples for your endpoints.

Authentication Mechanisms:

Basic Authentication:

In this method, clients include a base64-encoded username and password in the Authorization header of their HTTP requests. Spring Security provides easy integration for basic authentication.

JWT (JSON Web Tokens):

JWT is a popular method for token-based authentication. When a user logs in, the server generates a JWT containing user information and sends it back to the client. The client includes this token in subsequent requests to authenticate itself.

OAuth 2.0:

OAuth 2.0 is an authorization framework that allows third-party applications to obtain limited access to an HTTP service. It provides different grant types like Authorization Code, Implicit, Resource Owner Password Credentials, and Client Credentials.

API Keys:

API keys are unique identifiers passed in the request headers to authenticate the client. They are often used for rate limiting and access control.

Custom Authentication:

You can also implement custom authentication mechanisms based on your specific requirements, such as using a third-party authentication service.

Conclusion:

In conclusion, implementing an online shopping application with Spring Boot, MySQL, role-based authorization, and JWT token authentication offers a robust and secure solution for managing an e-commerce platform. By following best practices in project structuring, authentication, and API documentation with Swagger, you can create a scalable and maintainable application.

Using Spring Boot simplifies the development process by providing a comprehensive framework for building RESTful APIs, while MySQL offers a reliable database solution for storing product, user, and order information. Role-based authorization ensures that users have the appropriate permissions to access different parts of the application, enhancing security.

JWT token authentication adds an extra layer of security by securely transmitting user information between the client and server. Integrating Swagger for API documentation helps developers and users understand and interact with the API more effectively, thanks to clear descriptions and examples.

Overall, this project demonstrates how to leverage modern technologies to create a feature-rich online shopping application that meets the demands of both developers and users in terms of functionality, security, and usability.