Project Specification Document By Priyanshu Ranjan

1. Product Overview

SportyShoes E-commerce Portal Prototype

SportyShoes is a web-based e-commerce platform for selling sports shoes. The prototype aims to showcase the essential features of the application, allowing users to browse and purchase products, and administrators to manage the website efficiently.

2. Product Capabilities

User Features:

- User Registration: Users can sign up for an account.
- Product Browsing: Users can view a list of products available in the store.
- Product Categorization: Products are categorized for easy navigation.
- User Authentication: Secure login mechanism for user access.

Admin Features:

- Admin Login: Administrators have a dedicated login to access the admin page.
- Password Management: Admins can change their password for security.
- Product Management: Admins can add, delete, and categorize products.
- User Management: Admins can browse and search for signed-up users.
- Purchase Reports: Admins can view purchase reports, filterable by date and category.

3. Product Appearance and User Interactions

The user interface will be designed with a clean and intuitive layout, featuring:

- Responsive design for optimal user experience on various devices.
- Clear navigation menus for users and admins.
- Interactive product listings with details and images.
- User-friendly forms for registration and login.
- Admin control panel for managing products, users, and reports.

4. Source Code Management

The source code for the project will be stored and tracked on GitHub. Each feature enhancement and bug fix will be maintained as a separate branch, and changes will be merged into the main branch upon completion.

5. Java Concepts Used

The project utilizes key Java concepts such as:

- **Spring Boot:** For building the backend application.
- **Spring Security:** For implementing authentication and authorization.
- **JPA (Java Persistence API):** For database interactions.
- **Spring MVC:** For handling web requests.
- **Lombok:** For reducing boilerplate code.
- **RESTful API Design:** For communication between the frontend and backend.

6. Generic Features Discussion

Admin Management:

- The admin login ensures secure access to administrative functionalities.
- Password change option for admin security.

Product Management:

- Admins can perform CRUD operations on products.
- Categorization allows organized product listings.

User Management:

- Admins can view a list of signed-up users.
- Search functionality for efficient user lookup.

Purchase Reports:

- Admins can view purchase reports.
- Reports are filterable by date and category.

Conclusion

The SportyShoes e-commerce prototype is designed to provide a glimpse of the application's core features. The source code is version-controlled on GitHub, and the implementation is based on well-established Java concepts. The admin panel allows for efficient management of products, users, and purchase reports, creating a seamless experience for both administrators and users.

API Documentation

Admin Controller

Get All Products

- **URL:** `/admin/products`
- **Method:** `GET`
- **Description:** Retrieve a list of all products.
- **Response:**
  - `200 OK` with a JSON array of products if successful.
  - `204 No Content` if no products are found.
  - `401 Unauthorized` if the request lacks valid authentication.
  - `500 Internal Server Error` for other server-side issues.

Get Products by Category

- **URL:** `/admin/products/categorize/{category}`
- **Method:** `GET`
- **Description:** Retrieve products based on a specified category.
- **Parameters:**
  - `{category}`: The category to filter products.
- **Response:**
  - `200 OK` with a JSON array of products if successful.
  - `204 No Content` if no products are found.
  - `401 Unauthorized` if the request lacks valid authentication.
  - `500 Internal Server Error` for other server-side issues.

Add Product

- **URL:** `/admin/products`
- **Method:** `POST`
- **Description:** Add a new product to the inventory.
- **Request Body:**
  - JSON object representing the new product.
- **Response:**
  - `200 OK` with the details of the added product if successful.
  - `400 Bad Request` if the request is malformed.
  - `401 Unauthorized` if the request lacks valid authentication.
  - `500 Internal Server Error` for other server-side issues.

Get Product by ID

- **URL:** `/admin/products/{productId}`
- **Method:** `GET`
- **Description:** Retrieve product details based on the product ID.
- **Parameters:**
  - `{productId}`: The ID of the product to retrieve.
- **Response:**
  - `200 OK` with the details of the product if found.
  - `204 No Content` if no product is found.
  - `401 Unauthorized` if the request lacks valid authentication.
  - `500 Internal Server Error` for other server-side issues.

Delete Product by ID

- **URL:** `/admin/products/{productId}`
- **Method:** `DELETE`
- **Description:** Delete a product based on the product ID.
- **Parameters:**
  - `{productId}`: The ID of the product to delete.
- **Response:**
  - `200 OK` if the product is successfully deleted.
  - `401 Unauthorized` if the request lacks valid authentication.
  - `500 Internal Server Error` for other server-side issues.

User Controller

User Signup

- **URL:** `/users/signup`
- **Method:** `POST`
- **Description:** Register a new user.
- **Request Body:**
  - JSON object representing user details.
- **Response:**
  - `200 OK` if the user is successfully registered.
  - `400 Bad Request` if the request is malformed.
  - `500 Internal Server Error` for other server-side issues.

User Purchase Product

- **URL:** `/users/{userId}/buy/{productName}`
- **Method:** `POST`
- **Description:** Allow a user to purchase a product.
- **Parameters:**

    - `{userId}`: The ID of the user making the purchase.
    - `{productName}`: The name of the product being purchased.
- **Response:**
    - `200 OK` with a success message if the purchase is successful.
    - `400 Bad Request` if the request is malformed.
    - `404 Not Found` if the user or product is not found.
    - `500 Internal Server Error` for other server-side issues.

Security Configuration

Security Configuration

- **Class:** `SprotyShoesSecurityConfiguration`
- **Description:** Configures security settings for the Sporty Shoes application.
- **Features:**
    - Restricts access to certain URLs based on roles.
    - Configures HTTP Basic authentication.
    - Disables CSRF protection.
    - Defines an in-memory user with the role "ADMIN" for testing.

Entity Classes

Product Entity

- **Class:** `Product`
- **Description:** Represents a product in the Sporty Shoes application.
- **Attributes:**
    - `productId`: ID of the product.
    - `productName`: Name of the product.
    - `productPrice`: Price of the product.
    - `category`: Category of the product.
    - `users`: List of users who purchased the product.

PurchaseReport Entity

- **Class:** `PurchaseReport`
- **Description:** Represents a purchase report in the Sporty Shoes application.
- **Attributes:**
    - `id`: ID of the purchase report.
    - `categoryOfProduct`: Category of the purchased product.
    - `productName`: Name of the purchased product.
    - `priceOfTheProduct`: Price of the purchased product.
    - `userWhoBoughtTheProduct`: User who bought the product.

- `userEmailBoughtTheProduct`: Email of the user who bought the product.
- `dateOfProductPurchase`: Date of the product purchase.

User Entity

- **Class:** `User`
- **Description:** Represents a user in the Sporty Shoes application.
- **Attributes:**
  - `userId`: ID of the user.
  - `userName`: Name of the user.
  - `userEmail`: Email of the user.
  - `userPassword`: Password of the user.
  - `products`: List of products purchased by the user.