**Project Documentation: Online Bookstore Management System (Spring Boot)**

Overview

This project is an Online Bookstore Management System developed using Spring Boot, Spring Data JPA, Thymeleaf, and MySQL. It allows users to view available books, manage their personal book lists, and perform CRUD (Create, Read, Update, Delete) operations on book records. The application features a simple user interface, developed using Thymeleaf, and manages data persistence using Spring Data JPA and MySQL.

Features

1. Home Page: The landing page provides navigation to other functionalities.

2. View Books: Displays a list of available books fetched from the database.

3. Add Books to My List: Users can add books to their personal book list.

4. Register New Books: Admins can register new books by providing book details (name, author, price).

5. Edit Books: Users can edit existing book details.

6. Delete Books: Users can delete books from the list.

Technologies Used

- Backend: Spring Boot, Spring Data JPA

- Frontend: Thymeleaf, Bootstrap

- Database: MySQL

- Build Tool: Maven

- IDE: Spring Tool Suite (STS)

- Other Tools: DevTools for automatic restarts

Project Structure

- Controller Layer: Handles client requests and directs them to the appropriate service.

- Service Layer: Contains the business logic and interacts with the repository layer.

- Repository Layer: Interacts with the database using Spring Data JPA.

- Entity Layer: Represents the data model (Book, MyBookList).

Installation

1. Clone the Repository:

git clone <repository-url>

2. Import the Project:

- Open Spring Tool Suite (STS) or any other IDE.

- Import the project as an Existing Maven Project.

3. Set Up the Database:

- Configure the database settings in application.properties:

properties

spring.datasource.url=jdbc:mysql://localhost:3306/bookstore

spring.datasource.username=<your-username>

spring.datasource.password=<your-password>

spring.jpa.hibernate.ddl-auto=update

4. Build and Run:

- Use Maven to build and run the project:

bash

mvn spring-boot:run

Dependencies

Below are the key dependencies used:

- Spring Web: For building the web application.

- Thymeleaf: For server-side HTML rendering.

- Spring Data JPA: For database interactions.

- MySQL Connector: To connect to MySQL.

- Spring Boot DevTools: To enable automatic restarts during development.

API Endpoints

- / (GET): Home page.

- /books (GET): Fetches all available books.

- /mylist (POST): Adds a book to the user's personal list.

- /register (POST): Registers a new book.

- /editBook/{id} (PUT): Edits book details.

- /deleteBook/{id} (DELETE): Deletes a book from the list.

Troubleshooting

1. 404 Not Found: Ensure correct URL mappings in the @Controller classes and validate Thymeleaf template names.

2. Database Connection Issues: Verify MySQL configurations in application.properties and check the MySQL service status.

3. Whitelabel Error Page: Verify that the correct templates are placed inside the src/main/resources/templates/ folder.

Future Enhancements

1. User Authentication: Implement login functionality for admin and users.

2. Search Books: Add search functionality to find books based on various criteria.

3. Pagination: Implement pagination for large book lists.

Conclusion

The Online Bookstore Management System is a basic but functional project that demonstrates how to integrate various Spring Boot components, including JPA, Thymeleaf, and MySQL, to create a complete web application. The application can be further enhanced with additional features like user authentication and search functionalities.

Debugging steps:

1. Understand the Error: Start by analyzing error messages and logs. For instance, 404 Not Found may indicate a missing endpoint or misconfigured URL.

2. Check Dependencies: Ensure proper Maven/Gradle dependencies are declared, especially for Spring Data JPA, MySQL, etc.

3. Validate Configurations: Verify application.properties or YAML configurations for correct database and server settings.

4. Entity Mapping: Check entity annotations like @Entity, @Id, @GeneratedValue for correctness, ensuring proper ORM mapping with the database.

5. Verify URLs and Controllers: Ensure correct mappings in controllers using @GetMapping, @PostMapping, and test with tools like Postman.

6. Debugging IDE Tools: Use breakpoints and debug mode in your IDE (e.g., STS) to inspect variable states and flow.

7. Database Integrity: Test SQL queries directly in MySQL to verify table structure and data.

Thought Process:

- Start with error logs and work outward: focus on the stack trace to identify where the issue is.

- Check configurations early in development because errors often stem from misconfigurations.

- Test in small pieces—validate individual layers like the repository, service, and controller to narrow down the issue.

This approach ensures systematic troubleshooting while isolating potential issues one step at a time.