Book Inventory Management System - Project Documentation

Author: Carly Hampson A

# 1. Project Overview

The Book Inventory Management System is a Spring Boot–based web application designed to manage and streamline book inventory operations including book categorization, author management, publishing details, user management, shopping cart handling, and purchase logging. It provides a modular architecture separating entities, repositories, services, controllers, and exception handling, ensuring scalability, readability, and maintainability.

# 2. Application Architecture

The application follows a three-layer architecture pattern:

• Controller Layer – Handles API endpoints and HTTP requests.

• Service Layer – Contains business logic with multiple ServiceImpl classes for modularity.

• Repository Layer – Uses Spring Data JPA for database access and CRUD operations.

# 3. Modifications and Configuration Updates

During development, the following updates were made for smoother functionality and simplified deployment:

• Removed the Security Configuration temporarily to avoid authentication-related conflicts.

• Removed the explicit Swagger Configuration class and instead enabled Swagger via Maven dependency for easier integration.

• Enabled automatic schema creation using 'spring.jpa.hibernate.ddl-auto=create' initially and later updated it to 'update' for persistence.

• Improved Entity relationships (OneToMany, ManyToOne) and fixed multiple cascade and foreign key issues.

• Successfully resolved exception handling for DuplicateResourceException and ResourceNotFoundException using a GlobalExceptionHandler.

# 4. Unit Testing (JUnit 5)

Comprehensive JUnit 5 tests were implemented across 10 independent Service Implementation classes. Mockito was used to mock repositories to ensure that service logic works correctly without relying on the database. Each test follows best practices by verifying save, fetch, and delete operations, ensuring system reliability and maintainability.

Key Highlights:

• 10+ ServiceImpl classes tested independently.

• Used Mockito for mocking and dependency injection.

• Asserted correctness using assertEquals, assertNotNull, and verify methods.

• Ensured all tests pass successfully (Green status in Eclipse JUnit Runner).

# 5. API Documentation (Swagger Integration)

Swagger UI was enabled using the SpringDoc dependency, allowing real-time API documentation and endpoint testing. The Swagger interface can be accessed at: http://localhost:8080/swagger-ui/index.html.

# 6. Key Concepts Learned

Through this project, I gained hands-on experience with essential Spring Boot and backend development concepts, including:

• Spring Boot Core (Dependency Injection, Beans, Configuration, Annotations)

• Spring Data JPA and Hibernate ORM mapping

• Exception Handling using @ControllerAdvice and Custom Exceptions

• REST API Development with CRUD Operations

• Entity Relationships (One-to-Many, Many-to-One, Composite Keys)

• Swagger API Documentation (SpringDoc)

• JUnit 5 Testing and Mockito Mocking

• Application Configuration Management (application.properties)

• Maven Build Lifecycle and Dependency Management

# 7. Suggested Future Improvements

To enhance the system’s robustness and scalability, the following improvements are suggested:

• Reintroduce Spring Security with JWT Authentication for secured access control.

• Implement Role-based Authorization (Admin, Customer, Publisher).

• Integrate Email or OTP verification for new user registrations.

• Add Pagination and Sorting for large data retrieval operations.

• Include Caching (Redis) for optimizing repeated read operations.

• Deploy the application on a cloud platform (AWS EC2 or Heroku) for production readiness.

• Introduce Docker containerization for environment consistency.

• Add CI/CD pipelines (GitHub Actions) for automated builds and testing.

# 8. Personal Learning Outcome

This project strengthened my understanding of enterprise-level application development using Spring Boot. I learned how to design modular systems, handle relational data with JPA, build and test APIs, and debug integration issues. Working with Swagger and JUnit testing helped me understand the importance of documentation and validation in professional software development.

# 9. Conclusion

The Book Inventory Management System is a scalable, modular, and testable backend application. It follows modern Spring Boot practices, with clean separation of concerns, global exception handling, and integrated Swagger API documentation. The successful addition of unit tests ensures reliable functionality and demonstrates professional coding standards suitable for client or production environments.