RESTful Bookstore API Project Report

Introduction

This report outlines the development of a RESTful Bookstore API, a backend application designed to manage books and authors. The primary objective was to create a robust and scalable service that supports all standard CRUD (Create, Read, Update, Delete) operations and incorporates advanced features such as filtering, pagination, and sorting. The project serves as a practical demonstration of modern backend development principles using the Java ecosystem.

Abstract

The RESTful Bookstore API is a single-module, self-contained backend service built with Java and the Spring Boot framework. The application uses Spring Data JPA for data persistence and an in-memory H2 database for a lightweight, development-friendly data store. The API endpoints are fully documented using Swagger/OpenAPI, providing an interactive interface for developers. All endpoints were rigorously tested and validated using Postman to ensure functionality and data integrity. This project showcases the ability to design, implement, and document a production-ready RESTful service.

Tools Used

- **Java 17:** The core programming language.
- **Spring Boot 3.x:** A framework for building stand-alone, production-grade Spring applications with minimal configuration.
- **Spring Data JPA:** Simplifies data access and persistence operations by automatically providing repository implementations.
- **H2 Database:** An in-memory, open-source relational database used for development and testing.
- **Maven:** The build automation tool used to manage project dependencies.
- **Postman:** An API platform for building and using APIs. It was used to test each endpoint's functionality.
- **SpringDoc OpenAPI:** A library that automatically generates API documentation from the code, including a Swagger UI.

Steps Involved in Building the Project

1. **Project Setup:** The project was initialized using the Spring Initializr tool, with `Spring Web`, `Spring Data JPA`, `H2 Database`, and `Lombok` dependencies. The `application.properties` file was configured to enable the H2 console for database inspection.
2. **Entity Modeling:** Two core entity classes, `Book` and `Author`, were defined. The `Book` entity contained fields such as `title`, `genre`, `publicationDate`, and a

`ManyToOne` relationship with the `Author` entity. This approach correctly mapped the entities to the database tables using JPA annotations.

3. **Repository Layer Implementation:** Custom repository interfaces were created for both `Book` and `Author` by extending `JpaRepository`. This allowed for inherited CRUD methods and the ability to define custom query methods for filtering and searching without writing any boilerplate code.
4. **Controller and Service Logic:** A `@RestController` class was implemented to handle incoming HTTP requests. Each method in the controller was mapped to a specific endpoint and HTTP verb (`GET`, `POST`, `PUT`, `DELETE`). The controller used a service layer to interact with the repositories, ensuring a clean separation of concerns.
5. **Advanced Features:**
    - **Filtering:** The `GET /api/books` endpoint was enhanced to accept query parameters for filtering by author, title, or genre.
    - **Pagination and Sorting:** The `GET` method was updated to accept `page`, `size`, and `sort` parameters, leveraging Spring Data JPA's `Pageable` functionality to return results in a paginated and sorted manner.
6. **API Documentation:** The `SpringDoc OpenAPI` dependency was added to the project, which automatically generated comprehensive API documentation. The Swagger UI was made available at `http://localhost:8080/swagger-ui.html`, allowing for easy visualization and interaction with all API endpoints.
7. **API Testing:** A Postman collection was created to test every implemented endpoint. Requests for creating a new book, retrieving all books, updating a book's details, and deleting a book were configured and validated. This collection serves as a test suite for the API's functionality.

Conclusion

The completion of the RESTful Bookstore API project successfully demonstrates proficiency in building a professional-grade backend service. The project leveraged key technologies like Spring Boot and Spring Data JPA to create a robust application with minimal effort. By implementing advanced features such as pagination, sorting, and comprehensive API documentation, the project is a strong testament to a solid understanding of RESTful architectural principles. This project provides a confident foundation for discussing backend development skills in a professional setting.

Deliverables

- **Source Code:** The complete source code for the Spring Boot application.
- **Postman Collection:** A JSON file containing all the API requests used for testing.
- **Swagger UI Link:** The URL to access the API documentation when the application is running, typically `http://localhost:8080/swagger-ui.html`.