**Exercise 2: Online Bookstore - Setting Up RESTful Services**

**1. Introduction**

In this exercise, we implemented RESTful endpoints to manage books for an online bookstore. The goal was to create a BookController class that handles HTTP requests, such as GET, POST, PUT, and DELETE, and ensures that responses are returned in JSON format.

**2. Key Components**

* **BookController**: The REST controller responsible for handling HTTP requests related to books.
* **BookService**: A service layer that contains the business logic for handling books.
* **BookRepository**: A repository interface that extends JpaRepository to perform CRUD operations on the Book entity.
* **Book Entity**: Represents the book model with attributes like id, title, author, price, and isbn.

**3. Implementation**

**3.1 Book Entity**

The Book entity is a simple Java class annotated with @Entity to represent the books table in the database. It includes attributes like id, title, author, price, and isbn, each mapped to a column in the database using @Column annotations.

**3.2 BookController**

The BookController is annotated with @RestController and @RequestMapping("/api/books"). It exposes the following endpoints:

* **GET /api/books/all**: Fetches all books.
* **GET /api/books/search**: Filters books based on title and author.
* **POST /api/books/add**: Adds a new book.
* **PUT /api/books/edit/{id}**: Updates an existing book.
* **DELETE /api/books/delete/{id}**: Deletes a book by ID.

**3.3 Handling HTTP Methods**

* **GET**: The getAllBooks(), getBooks(), and filterBooks() methods handle GET requests, fetching and filtering book records.
* **POST**: The addBook() method handles POST requests, allowing new books to be added to the system.
* **PUT**: The updateBook() method handles PUT requests for updating existing book records.
* **DELETE**: The deleteBook() method handles DELETE requests for removing books by ID.

**3.4 Returning JSON Responses**

Each method in the BookController ensures that the responses are returned in JSON format. This is handled by the Spring framework's automatic conversion of Java objects to JSON when annotated with @RestController.

**4. Database Configuration**

The application uses an in-memory H2 database for testing purposes, as specified in the application.properties file. The database configuration includes:

* **JDBC URL**: jdbc:h2:mem:librarydb
* **Driver Class Name**: org.h2.Driver
* **Hibernate Dialect**: org.hibernate.dialect.H2Dialect
* **JPA Configuration**: Hibernate auto-generates and updates the schema with spring.jpa.hibernate.ddl-auto=update.

**5. Testing and Validation**

* **Postman**: Postman was used to test the RESTful endpoints, ensuring that all operations (GET, POST, PUT, DELETE) were functioning correctly and returning the expected JSON responses.
* **H2 Console**: The H2 database console was used to visually verify the records in the books table after performing CRUD operations.

**6. Conclusion**

The BookController is now fully functional, capable of managing books through RESTful endpoints. The use of H2 for the database and JSON for data exchange ensures that the service is both lightweight and efficient, ready for further development, such as adding features for managing authors and customers.