SPRING BOOT 3 REST API JPA

JAVA ACADEMY - XIDERAL

SEPTEMBER 6, 2024

AUTHOR: REGINA RODRIGUEZ CAMPO GARRIDO

Spring Boot 3 REST API JPA
September 6, 2024

Introduction

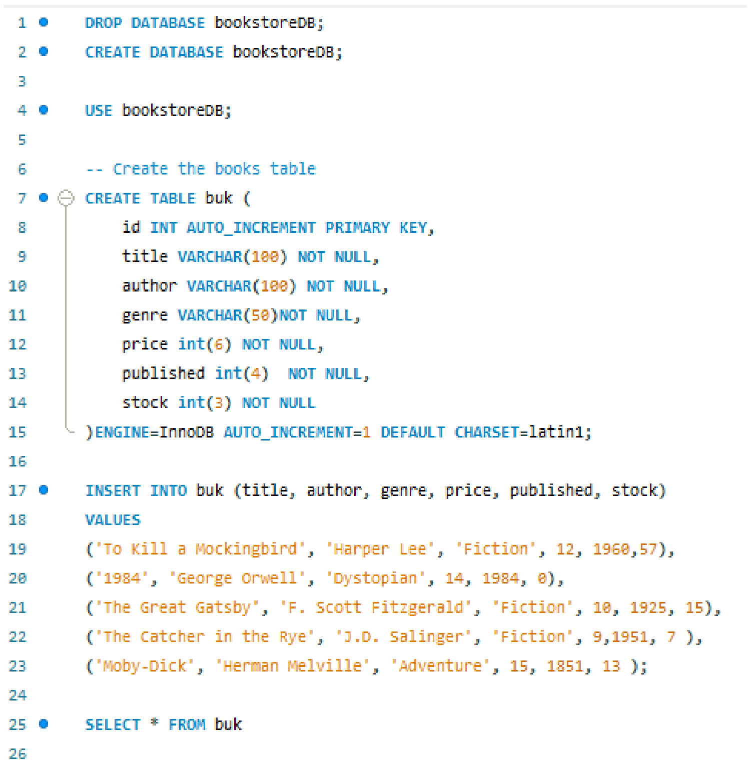
Spring Boot is a tool that makes it easier to create JAVA applications. It is responsible for configuring many things automatically so that it can be programmed faster; JPA is a way to save and obtain databases. By using them at the same time, this allows working with databases more easily, since Spring Boot configures everything necessary in JPA to be able to save, search and manage the database.

In the following project, using Spring Boot and JPA, we created a library system that will allow us to:

1. Show all the books we have.
2. Add books.
3. Delete books.
4. Search for books by ID.
5. Count the number of books we have by category.
6. Show whether the book is in stock or not

Database Schema

Cree una tabla con el nombre de “buk”.

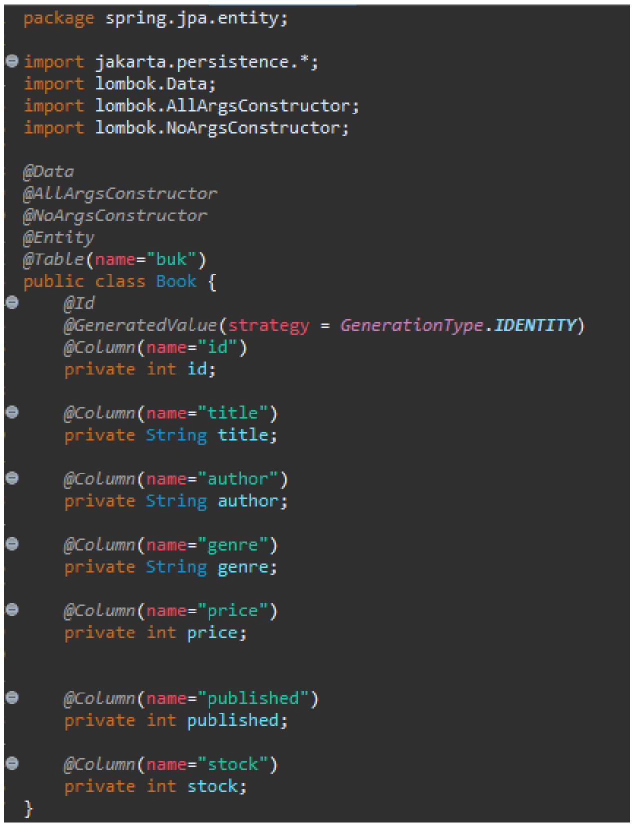


```sql
DROP DATABASE bookstoreDB;
CREATE DATABASE bookstoreDB;

USE bookstoreDB;

-- Create the books table
CREATE TABLE buk (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    author VARCHAR(100) NOT NULL,
    genre VARCHAR(50)NOT NULL,
    price int(6) NOT NULL,
    published int(4)  NOT NULL,
    stock int(3) NOT NULL
)ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;

INSERT INTO buk (title, author, genre, price, published, stock)
VALUES
('To Kill a Mockingbird', 'Harper Lee', 'Fiction', 12, 1960,57),
('1984', 'George Orwell', 'Dystopian', 14, 1984, 0),
('The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction', 10, 1925, 15),
('The Catcher in the Rye', 'J.D. Salinger', 'Fiction', 9,1951, 7 ),
('Moby-Dick', 'Herman Melville', 'Adventure', 15, 1851, 13 );

SELECT * FROM buk
```

Application

Entity

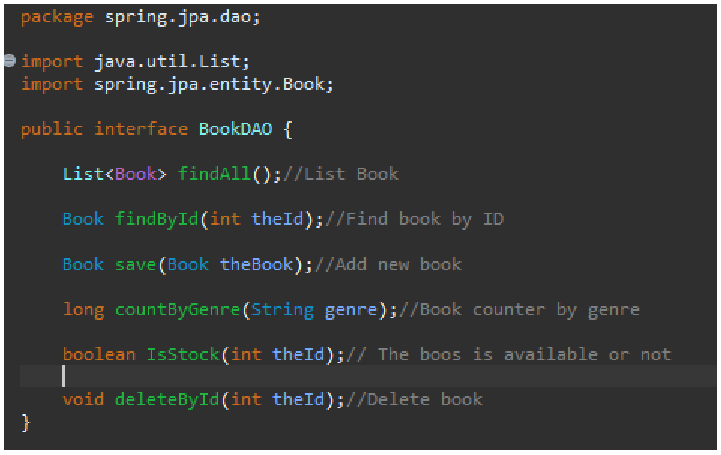
The Book class represents the Book entity. It uses JPA annotations to map the class to a table in the database and Lombok annotations to reduce code.



```java
package spring.jpa.entity;

import jakarta.persistence.*;
import lombok.Data;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name="buk")
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="title")
    private String title;

    @Column(name="author")
    private String author;

    @Column(name="genre")
    private String genre;

    @Column(name="price")
    private int price;


    @Column(name="published")
    private int published;

    @Column(name="stock")
    private int stock;
}
```

Dao Interface

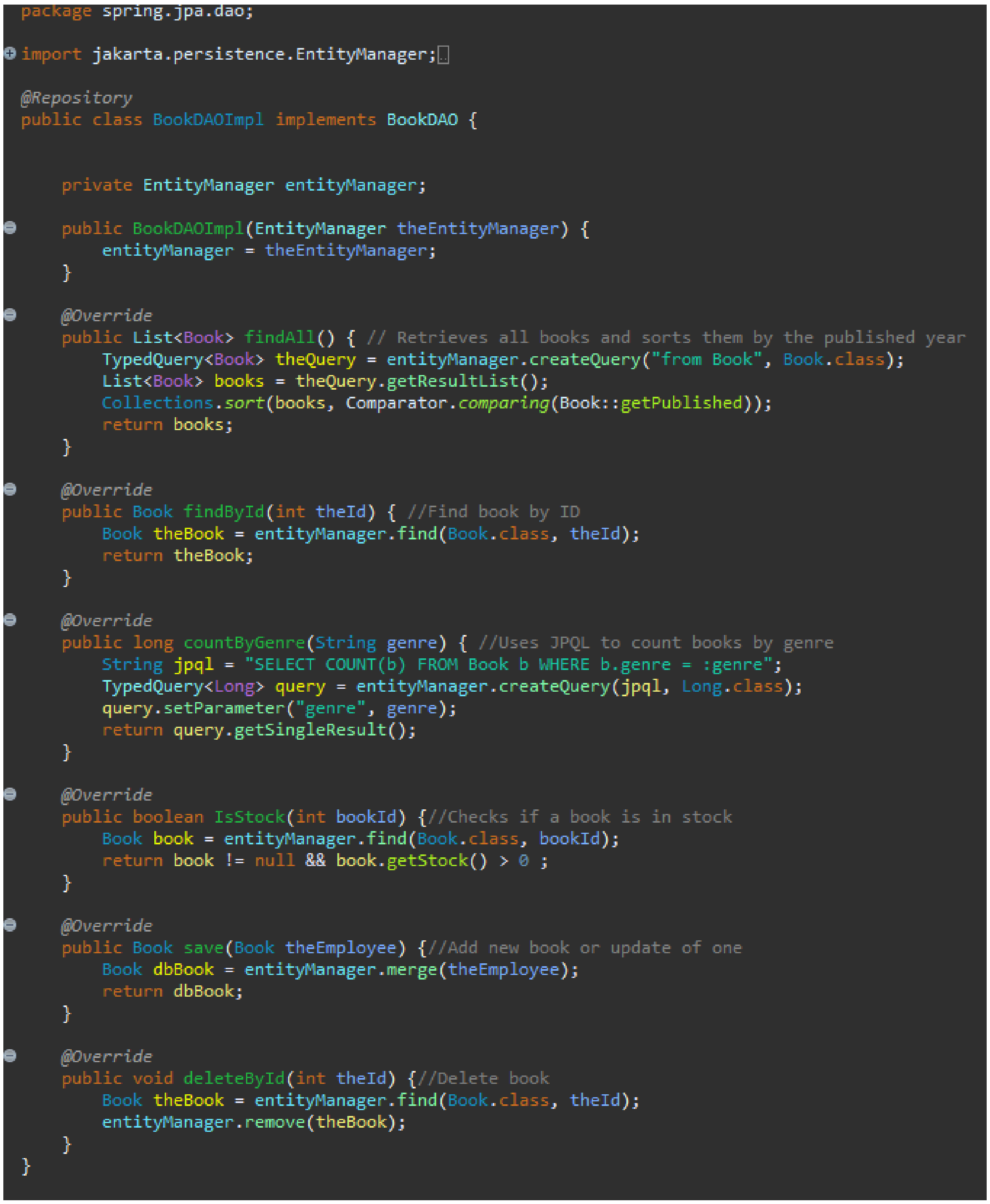
The BookDAO interface defines methods for CRUD operations and additional queries.



```java
package spring.jpa.dao;

import java.util.List;
import spring.jpa.entity.Book;

public interface BookDAO {

    List<Book> findAll();//List Book

    Book findById(int theId);//Find book by ID

    Book save(Book theBook);//Add new book

    long countByGenre(String genre);//Book counter by genre

    boolean IsStock(int theId);// The boos is available or not

    void deleteById(int theId);//Delete book
}
```

Dao Implementation

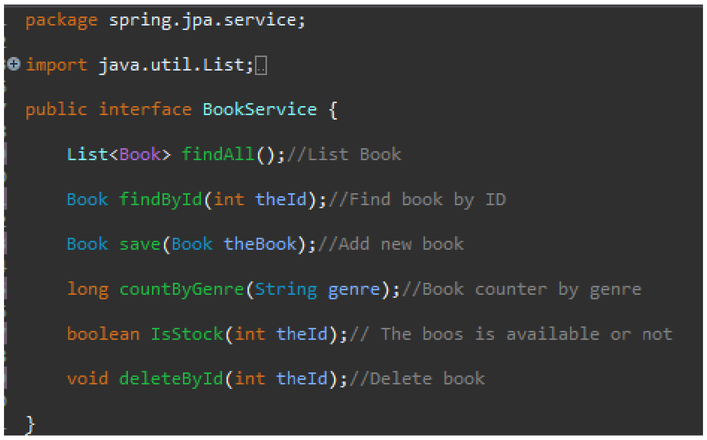
The BookDAOImpl class implements the BookDAO interface. It uses JPA's EntityManager to perform database operations.



```java
package spring.jpa.dao;

import jakarta.persistence.EntityManager;

@Repository
public class BookDAOImpl implements BookDAO {


    private EntityManager entityManager;

    public BookDAOImpl(EntityManager theEntityManager) {
        entityManager = theEntityManager;
    }

    @Override
    public List<Book> findAll() { // Retrieves all books and sorts them by the published year
        TypedQuery<Book> theQuery = entityManager.createQuery("from Book", Book.class);
        List<Book> books = theQuery.getResultList();
        Collections.sort(books, Comparator.comparing(Book::getPublished));
        return books;
    }

    @Override
    public Book findById(int theId) { //Find book by ID
        Book theBook = entityManager.find(Book.class, theId);
        return theBook;
    }

    @Override
    public long countByGenre(String genre) { //Uses JPQL to count books by genre
        String jpql = "SELECT COUNT(b) FROM Book b WHERE b.genre = :genre";
        TypedQuery<Long> query = entityManager.createQuery(jpql, Long.class);
        query.setParameter("genre", genre);
        return query.getSingleResult();
    }

    @Override
    public boolean IsStock(int bookId) {//Checks if a book is in stock
        Book book = entityManager.find(Book.class, bookId);
        return book != null && book.getStock() > 0 ;
    }

    @Override
    public Book save(Book theEmployee) {//Add new book or update of one
        Book dbBook = entityManager.merge(theEmployee);
        return dbBook;
    }

    @Override
    public void deleteById(int theId) {//Delete book
        Book theBook = entityManager.find(Book.class, theId);
        entityManager.remove(theBook);
    }
}
```

Service Interface

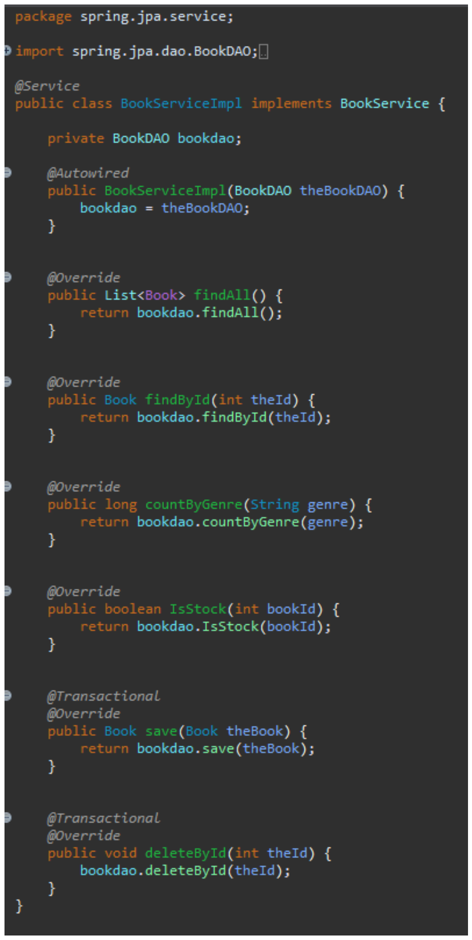
The BookService interface defines the business logic methods for managing books.



```java
package spring.jpa.service;

import java.util.List;

public interface BookService {

    List<Book> findAll();//List Book

    Book findById(int theId);//Find book by ID

    Book save(Book theBook);//Add new book

    long countByGenre(String genre);//Book counter by genre

    boolean IsStock(int theId);// The boos is available or not

    void deleteById(int theId);//Delete book

}
```

Service Implementation

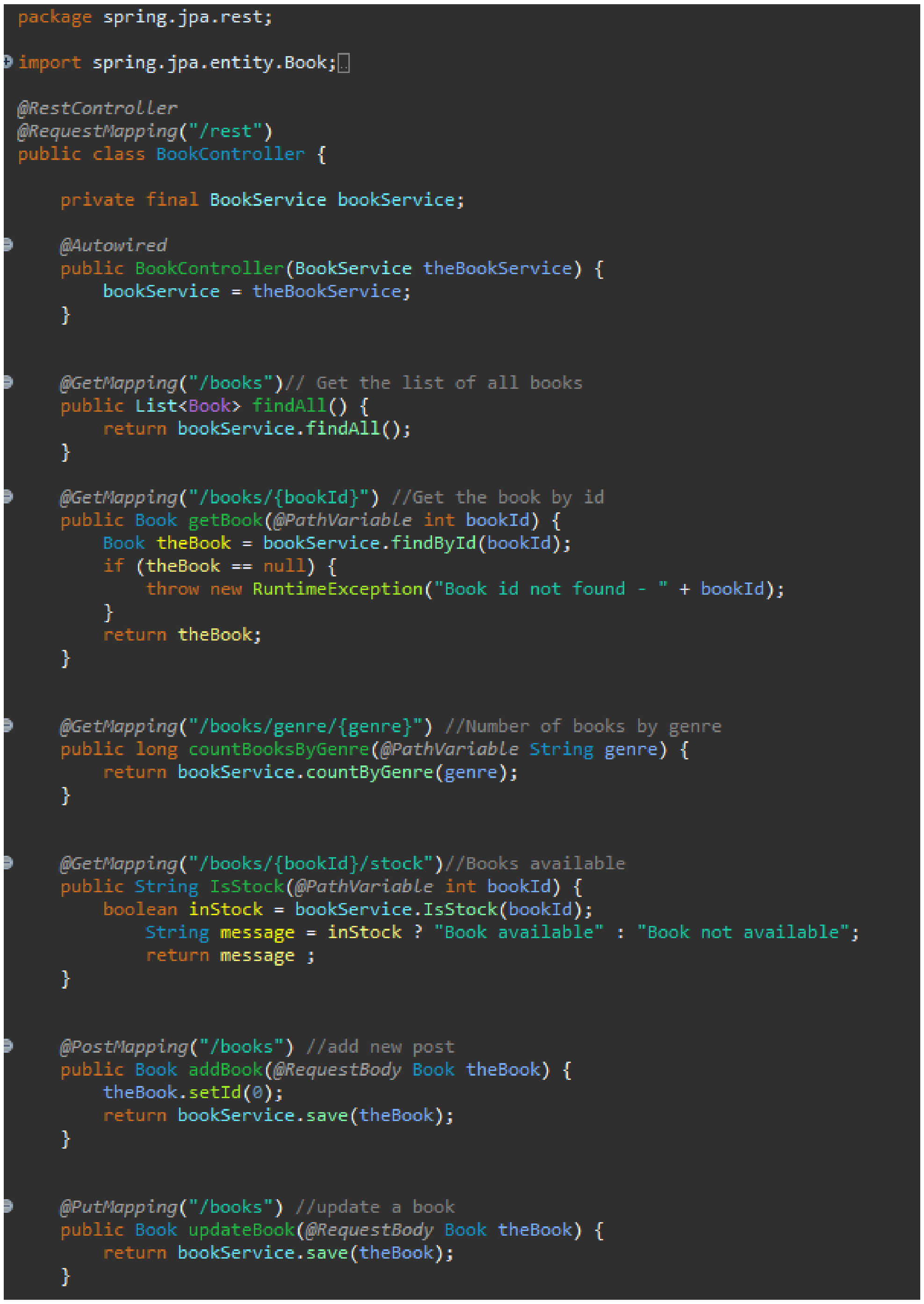
The BookServiceImpl class implements the BookService interface.



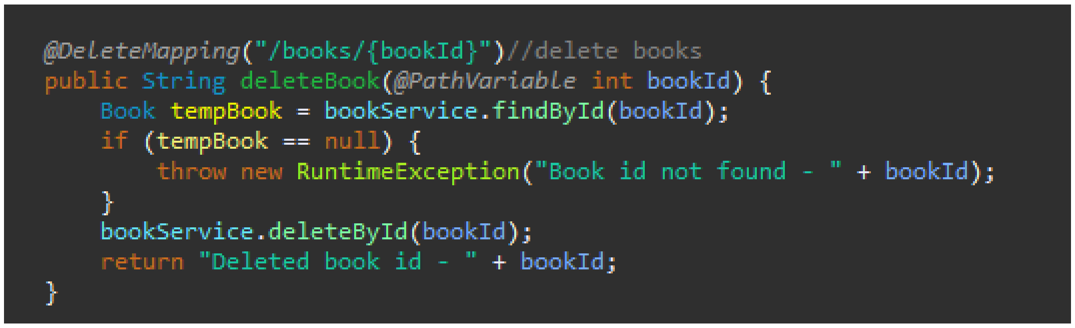
```java
package spring.jpa.service;

import spring.jpa.dao.BookDAO;

@Service
public class BookServiceImpl implements BookService {

    private BookDAO bookdao;

    @Autowired
    public BookServiceImpl(BookDAO theBookDAO) {
        bookdao = theBookDAO;
    }

    @Override
    public List<Book> findAll() {
        return bookdao.findAll();
    }

    @Override
    public Book findById(int theId) {
        return bookdao.findById(theId);
    }

    @Override
    public long countByGenre(String genre) {
        return bookdao.countByGenre(genre);
    }

    @Override
    public boolean IsStock(int bookId) {
        return bookdao.IsStock(bookId);
    }

    @Transactional
    @Override
    public Book save(Book theBook) {
        return bookdao.save(theBook);
    }

    @Transactional
    @Override
    public void deleteById(int theId) {
        bookdao.deleteById(theId);
    }
}
```

Rest Controller

The BookController serves as the REST controller, handling HTTP requests related to book management.

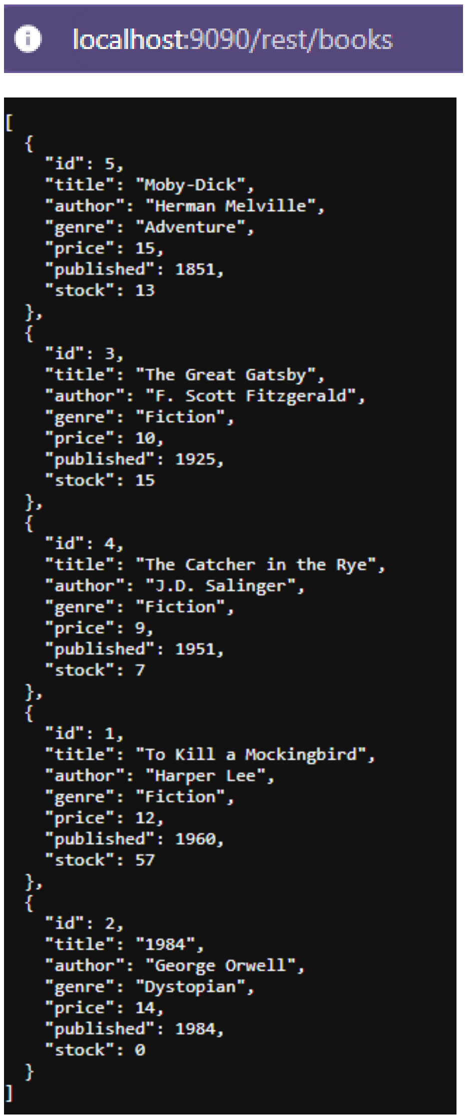


```
package spring.jpa.rest;

import spring.jpa.entity.Book;

@RestController
@RequestMapping("/rest")
public class BookController {

    private final BookService bookService;

    @Autowired
    public BookController(BookService theBookService) {
        bookService = theBookService;
    }


    @GetMapping("/books")// Get the list of all books
    public List<Book> findAll() {
        return bookService.findAll();
    }

    @GetMapping("/books/{bookId}") //Get the book by id
    public Book getBook(@PathVariable int bookId) {
        Book theBook = bookService.findById(bookId);
        if (theBook == null) {
            throw new RuntimeException("Book id not found - " + bookId);
        }
        return theBook;
    }


    @GetMapping("/books/genre/{genre}") //Number of books by genre
    public long countBooksByGenre(@PathVariable String genre) {
        return bookService.countByGenre(genre);
    }


    @GetMapping("/books/{bookId}/stock")//Books available
    public String IsStock(@PathVariable int bookId) {
        boolean inStock = bookService.IsStock(bookId);
            String message = inStock ? "Book available" : "Book not available";
            return message ;
    }


    @PostMapping("/books") //add new post
    public Book addBook(@RequestBody Book theBook) {
        theBook.setId(0);
        return bookService.save(theBook);
    }


    @PutMapping("/books") //update a book
    public Book updateBook(@RequestBody Book theBook) {
        return bookService.save(theBook);
    }
```



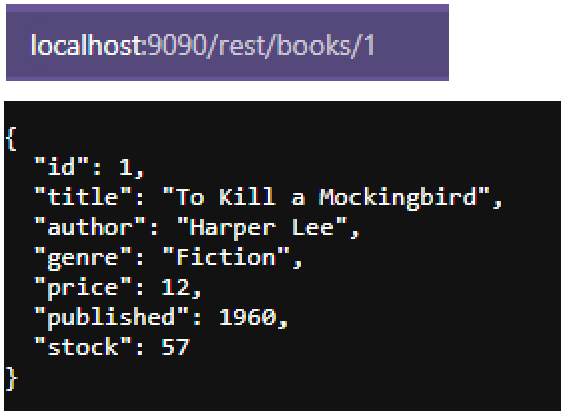
```
@DeleteMapping("/books/{bookId}")//delete books
public String deleteBook(@PathVariable int bookId) {
    Book tempBook = bookService.findById(bookId);
    if (tempBook == null) {
        throw new RuntimeException("Book id not found - " + bookId);
    }
    bookService.deleteById(bookId);
    return "Deleted book id - " + bookId;
}
```

Outputs



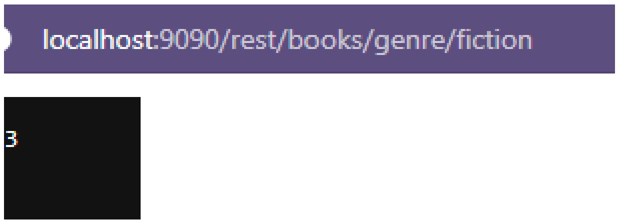
localhost:9090/rest/books

```
[
  {
    "id": 5,
    "title": "Moby-Dick",
    "author": "Herman Melville",
    "genre": "Adventure",
    "price": 15,
    "published": 1851,
    "stock": 13
  },
  {
    "id": 3,
    "title": "The Great Gatsby",
    "author": "F. Scott Fitzgerald",
    "genre": "Fiction",
    "price": 10,
    "published": 1925,
    "stock": 15
  },
  {
    "id": 4,
    "title": "The Catcher in the Rye",
    "author": "J.D. Salinger",
    "genre": "Fiction",
    "price": 9,
    "published": 1951,
    "stock": 7
  },
  {
    "id": 1,
    "title": "To Kill a Mockingbird",
    "author": "Harper Lee",
    "genre": "Fiction",
    "price": 12,
    "published": 1960,
    "stock": 57
  },
  {
    "id": 2,
    "title": "1984",
    "author": "George Orwell",
    "genre": "Dystopian",
    "price": 14,
    "published": 1984,
    "stock": 0
  }
]
```



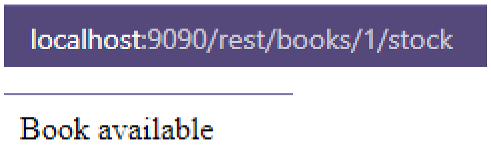
localhost:9090/rest/books/1

```
{
  "id": 1,
  "title": "To Kill a Mockingbird",
  "author": "Harper Lee",
  "genre": "Fiction",
  "price": 12,
  "published": 1960,
  "stock": 57
}
```



localhost:9090/rest/books/genre/fiction

```
3
```



localhost:9090/rest/books/1/stock

```
Book available
```