**Exercise 9: Creating a Spring Boot Application**

**Scenario:**

You need to create a Spring Boot application for the library management system to simplify configuration and deployment.

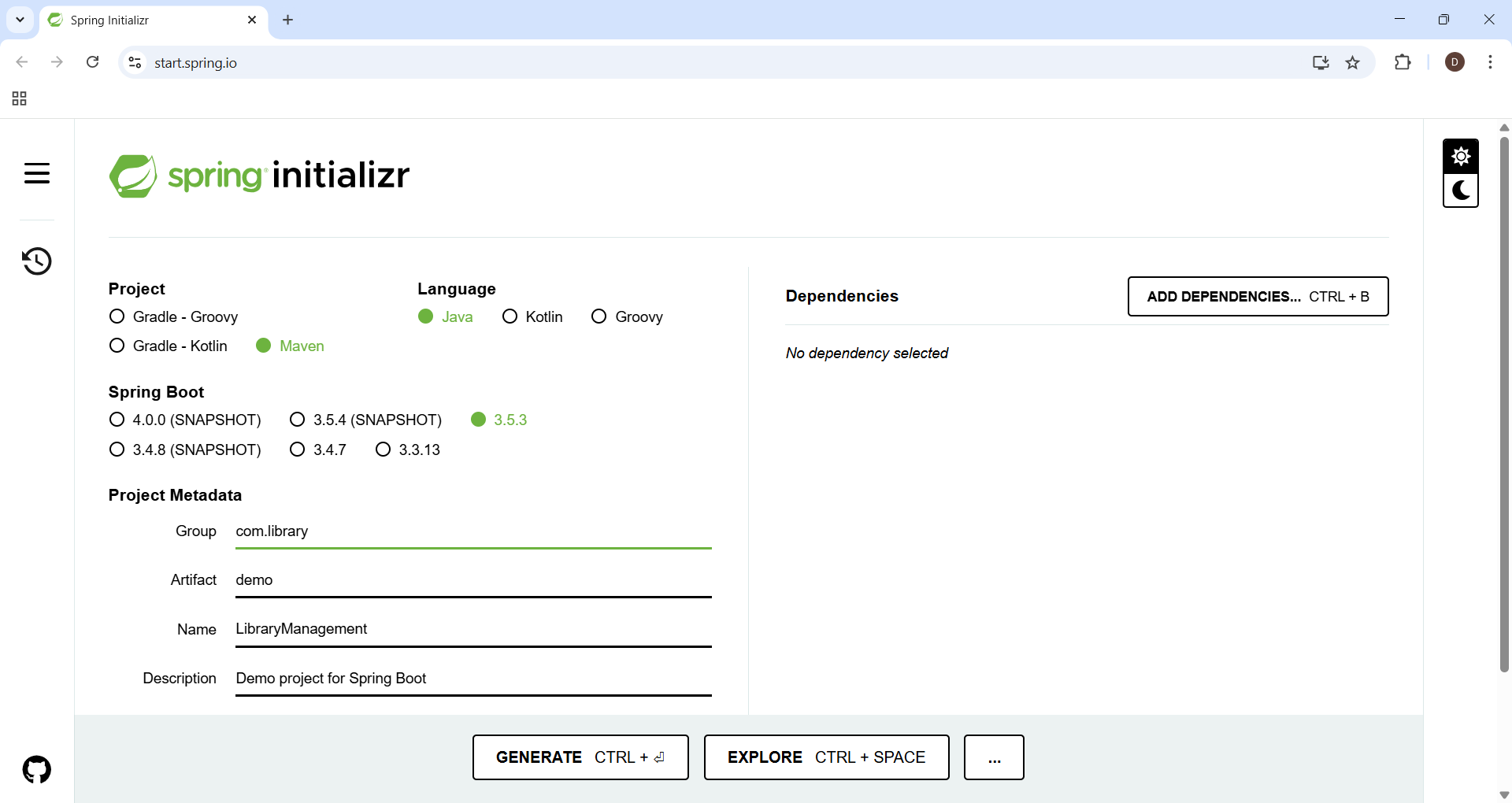
**Steps:**

1. **Create a Spring Boot Project:**
   * Use **Spring Initializr** to create a new Spring Boot project named **LibraryManagement**.
2. **Add Dependencies:**
   * Include dependencies for **Spring Web, Spring Data JPA, and H2 Database**.
3. **Create Application Properties:**
   * Configure database connection properties in **application.properties**.
4. **Define Entities and Repositories:**
   * Create **Book** entity and **BookRepository** interface.
5. **Create a REST Controller:**
   * Create a **BookController** class to handle CRUD operations.
6. **Run the Application:**

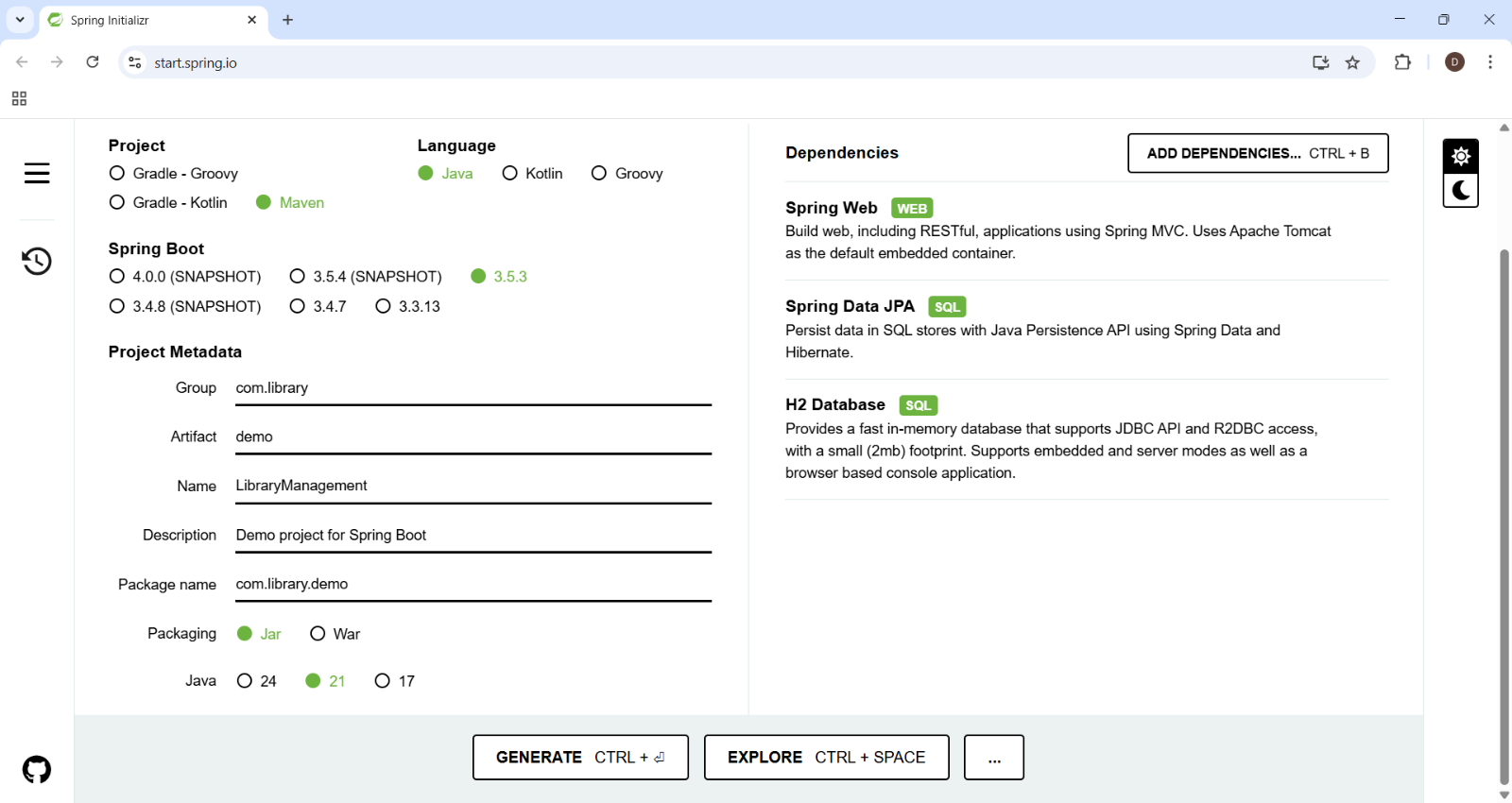
Run the Spring Boot application and test the REST endpoints.

**Solution:**

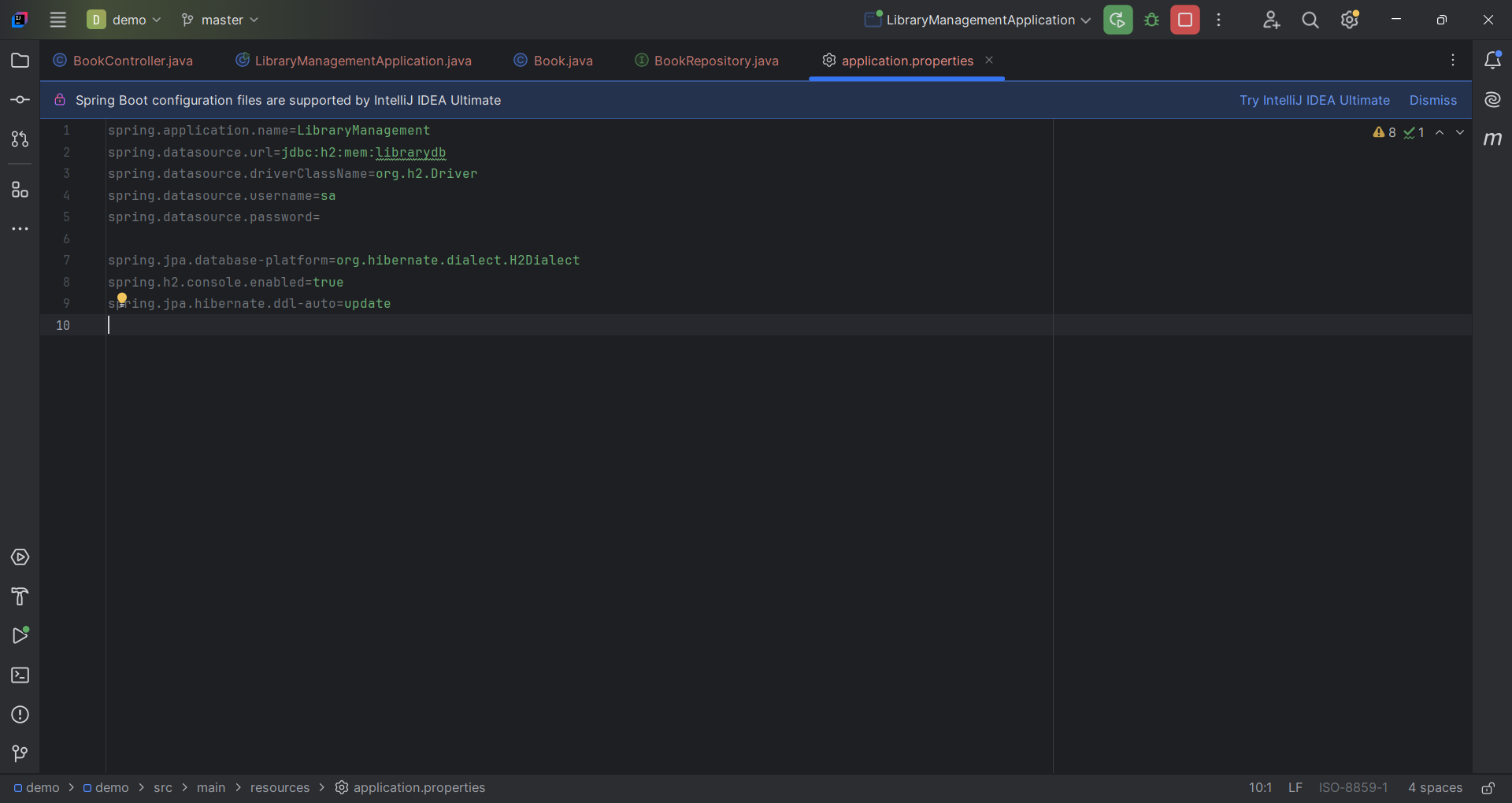
1. **Create a Spring Boot Project:**

****

1. **Add Dependencies:**



**3. Create Application Properties:**

****

**4. Define Entities and Repositories:**

**// interface BookRepository.java**

package com.library.demo.repository;  
  
import com.library.demo.entity.Book;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface BookRepository extends JpaRepository<Book, Long> {  
}

// file name Book.java for entities.

package com.library.demo.entity;  
import jakarta.persistence.\*;  
  
@Entity  
public class Book {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 private String title;  
 private String author;  
 private String genre;  
  
 // Constructors  
 public Book() {}  
  
 public Book(String title, String author, String genre) {  
 this.title = title;  
 this.author = author;  
 this.genre = genre;  
 }  
  
 // Getters & Setters  
 public Long getId() { return id; }  
 public void setId(Long id) {this.id=id;}  
 public String getTitle() { return title; }  
 public void setTitle(String title) { this.title = title; }  
  
 public String getAuthor() { return author; }  
 public void setAuthor(String author) { this.author = author; }  
  
 public String getGenre() { return genre; }  
 public void setGenre(String genre) { this.genre = genre; }  
  
  
}

//File name LibraryManagementApplication.java

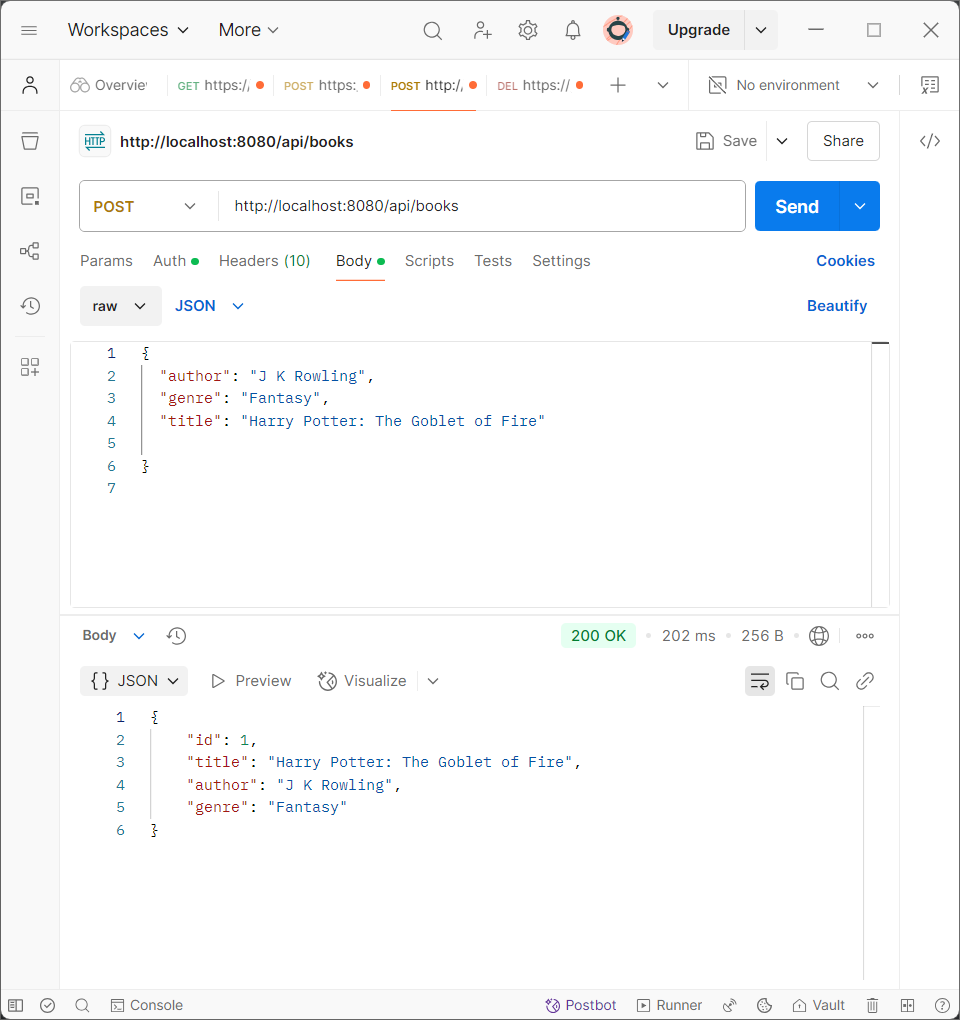
package com.library.demo;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class LibraryManagementApplication {  
  
 public static void main(String[] args) {  
  
 SpringApplication.*run*(LibraryManagementApplication.class, args);  
 }  
  
}

5. **Create a REST Controller:**

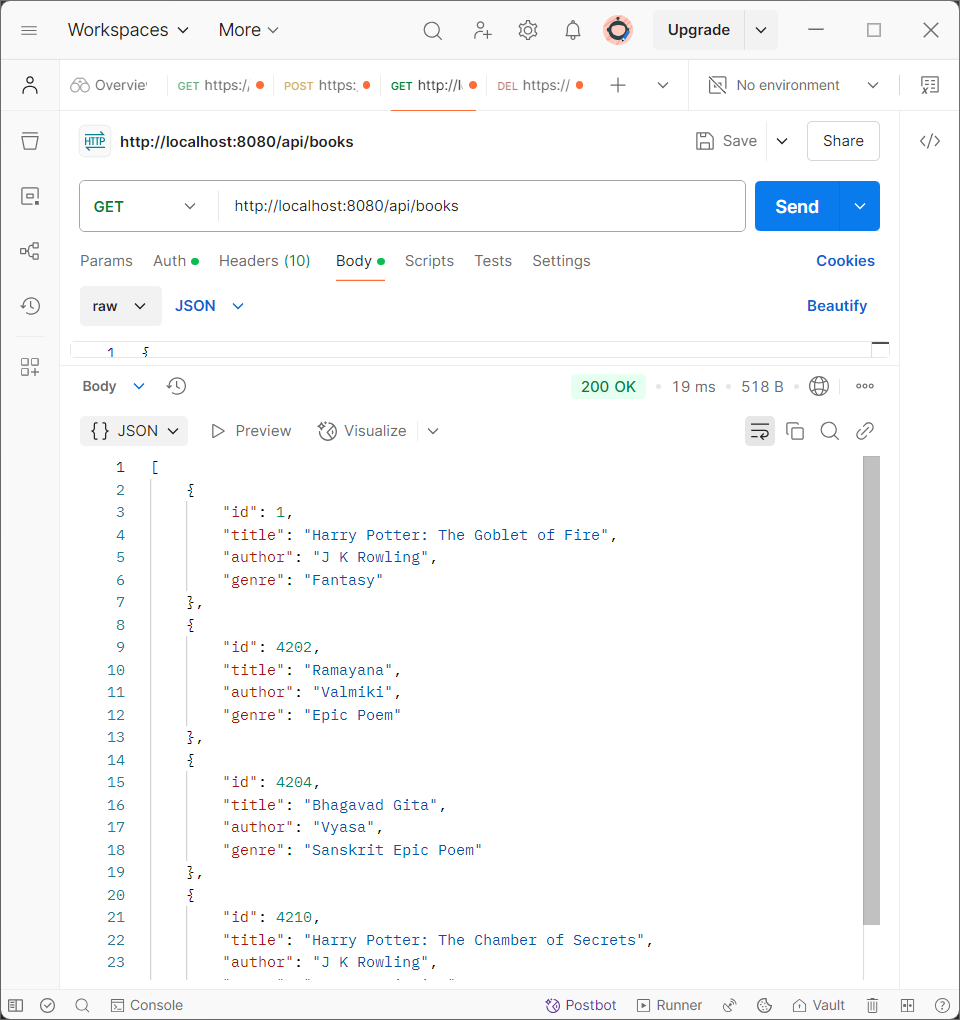
package com.library.demo.controller;  
import com.library.demo.entity.Book;  
import com.library.demo.repository.BookRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
import java.util.Optional;  
  
@RestController  
@RequestMapping("/api/books")  
public class BookController {  
  
 @Autowired  
 private BookRepository bookRepository;  
  
 // Get all books  
 @GetMapping  
 public List<Book> getAllBooks() {  
 return bookRepository.findAll();  
 }  
  
 // Get book by ID  
 @GetMapping("/{id}")  
 public Optional<Book> getBookById(@PathVariable Long id) {  
 return bookRepository.findById(id);  
 }  
  
 // Create new book  
 @PostMapping  
 public Book createBook(@RequestBody Book book) {  
 return bookRepository.save(book);  
 }  
  
 // Update book  
 @PutMapping("/{id}")  
 public Book updateBook(@PathVariable Long id, @RequestBody Book updatedBook) {  
 return bookRepository.findById(id)  
 .map(book -> {  
 book.setTitle(updatedBook.getTitle());  
 book.setAuthor(updatedBook.getAuthor());  
 book.setGenre(updatedBook.getGenre());  
 return bookRepository.save(book);  
 })  
 .orElseGet(() -> {  
 updatedBook.setId(id);  
 return bookRepository.save(updatedBook);  
 });  
 }  
  
 // Delete book  
 @DeleteMapping("/{id}")  
 public void deleteBook(@PathVariable Long id) {  
 bookRepository.deleteById(id);  
 }  
}

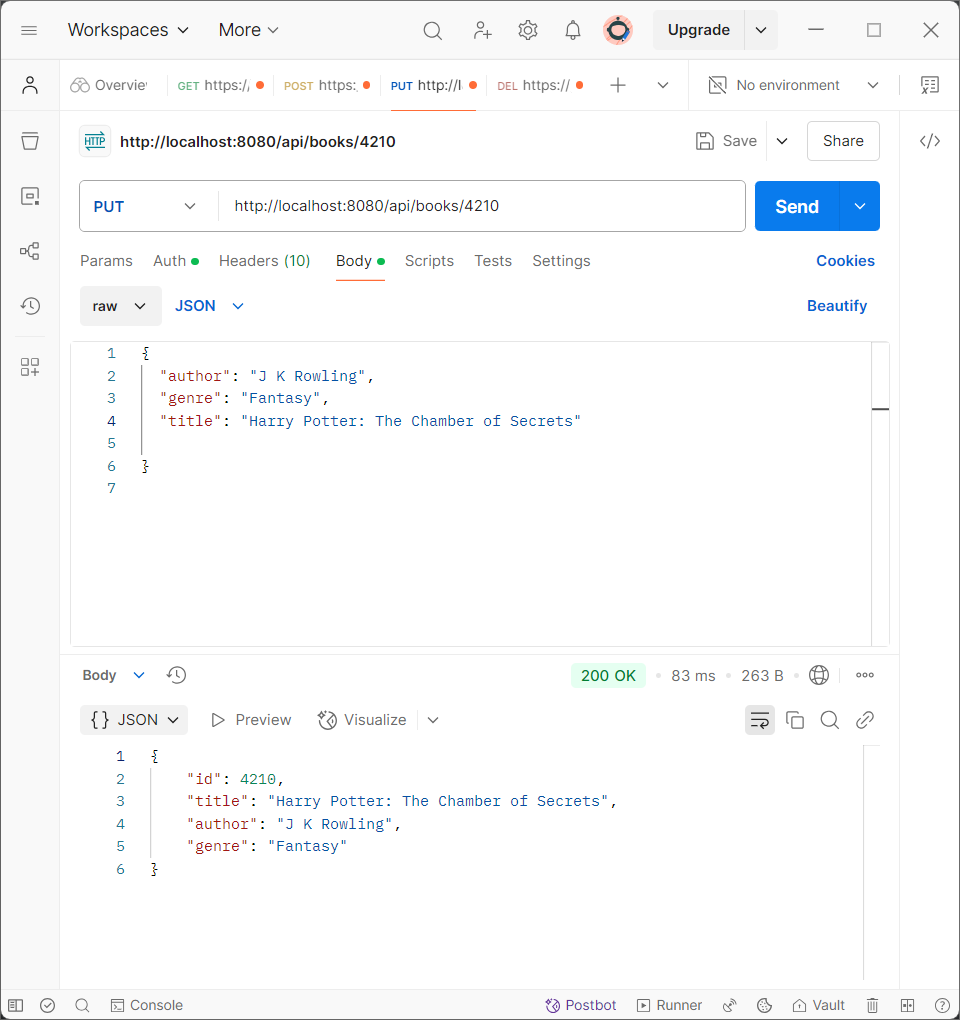
6. **Run the Application:**

**POST**

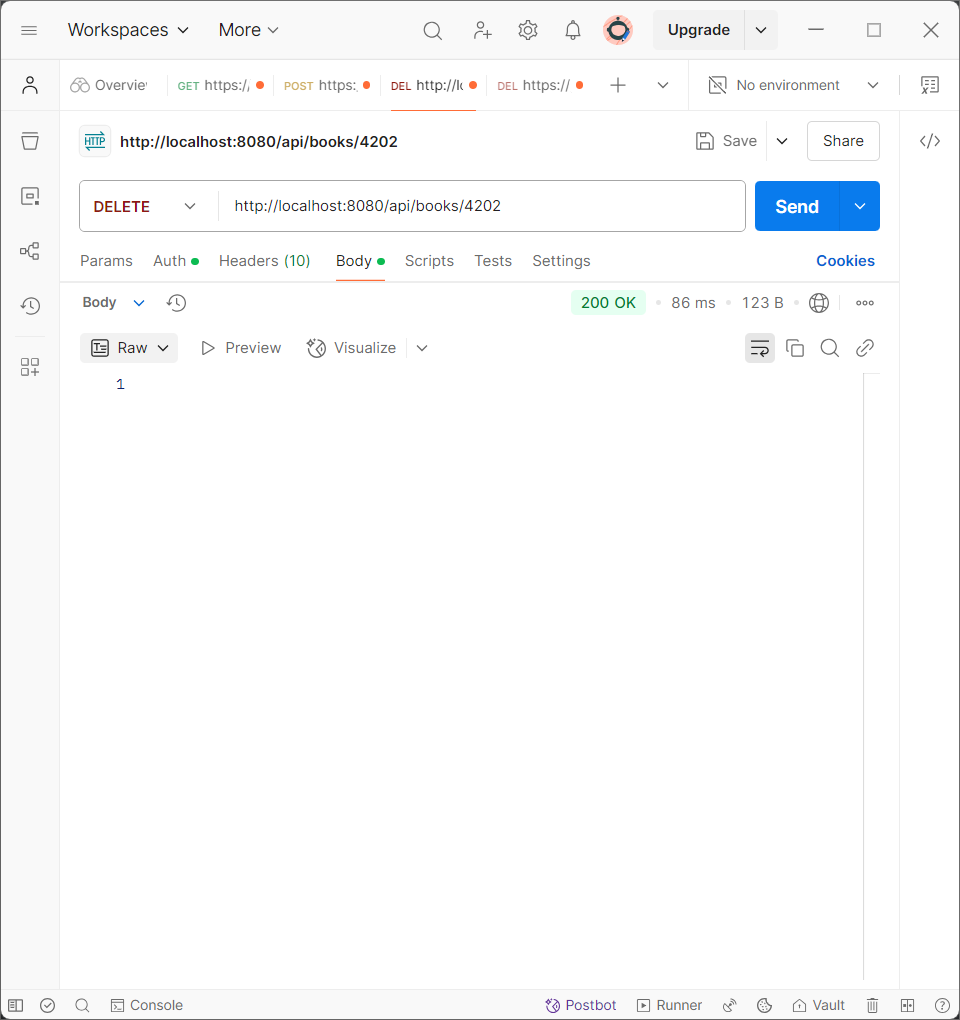
****

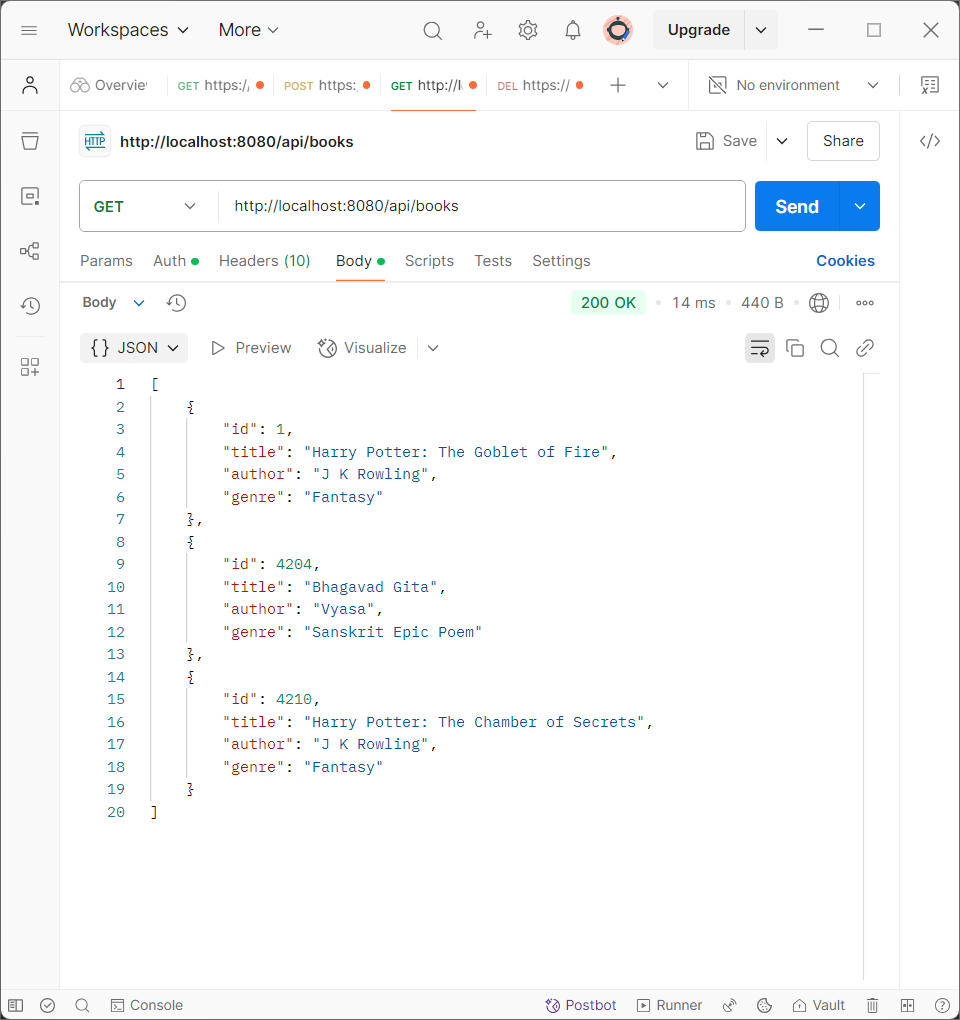
**GET**

****

**PUT**

**DELETE**



**Final GET to check the output:**

In Exercise 9, we created a Spring Boot application for a Library Management System using Spring Web, Spring Data JPA, and H2 Database. We defined a Book entity, created a BookRepository for data access, and built a BookController to handle CRUD operations through REST APIs. The app was configured using application.properties and tested via Postman and the H2 console. This exercise showed how Spring Boot simplifies backend development with minimal configuration and built-in support for REST and database integration.