# Spring Core and Maven

# Additional Hands-On

**Exercise 3: Implementing Logging with Spring AOP**

**Scenario:**

The library management application requires logging capabilities to track method execution times

1. **Add Spring AOP Dependency:**

Update **pom.xml** to include Spring AOP dependency.

.

<dependencies>

<!-- Spring Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.34</version>

</dependency>

<!-- AspectJ Weaver -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.21</version>

</dependency>

</dependencies>

1. **Create an Aspect for Logging:**

Create a package **com.library.aspect** and add a class **LoggingAspect** with a method to log execution times.

package com.library.aspect;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

@Aspect

public class LoggingAspect {

@Around("execution(\* com.library.service.\*.\*(..))")

public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {

// BEFORE the actual method runs

System.out.println("Before method");

long start = System.currentTimeMillis();

// CALL the actual method

Object returnValue = joinPoint.proceed(); // actual method call

// AFTER the method runs

System.out.println("After method");

long duration = System.currentTimeMillis() - start;

System.out.println("[AOP] Method " + joinPoint.getSignature() + " executed in " + duration + " ms");

return returnValue;

}

}

1. **Enable AspectJ Support:**

Update **applicationContext.xml** to enable **AspectJ** support and register the aspect.

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:aop="http://www.springframework.org/schema/aop"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd">

**<!-- Enable AspectJ support -->**

<aop:aspectj-autoproxy/>

<!-- Define the repository bean -->

<bean id="bookRepository" class="com.library.repository.BookRepository"/>

<!-- Define the service bean -->

<bean id="bookService" class="com.library.service.BookService">

<property name="bookRepository" ref="bookRepository"/>

</bean>

**<!-- Register the logging aspect -->**

<bean id="loggingAspect" class="com.library.aspect.LoggingAspect"/>

</beans>

1. **Test the Aspect:**

Run the **LibraryManagementApplication** main class and observe the console for log messages indicating method execution times.

**MainApp.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp{

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

System.out.println("[Main] Spring context initialized.");

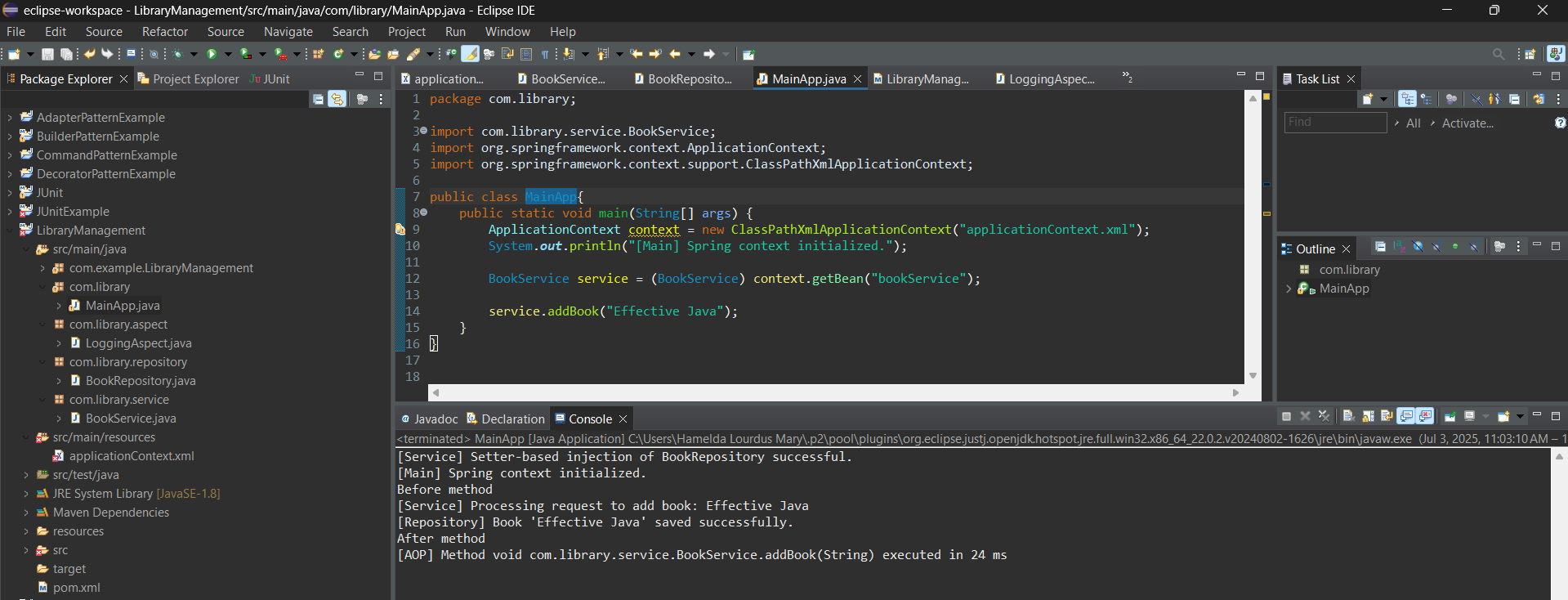
BookService service = (BookService) context.getBean("bookService");

service.addBook("Effective Java");

}

}

**Output:**



**Exercise 5: Configuring the Spring IoC Container**

**Scenario:**

The library management application requires a central configuration for beans and dependencies.

1. **Create Spring Configuration File:**

Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.

Define beans for **BookService** and **BookRepository** in the XML file.

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

***<!-- Repository Bean -->***

<bean id="bookRepository" class="com.library.repository.BookRepository" />

***<!-- Service Bean -->***

<bean id="bookService" class="com.library.service.BookService">

<property name="bookRepository" ref="bookRepository" />

</bean>

</beans>

1. **Update the BookService Class:**

Ensure that the **BookService** class has a setter method for **BookRepository**.

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for DI

public void setBookRepository(BookRepository bookRepository) {

System.out.println("[Service] BookRepository injected.");

this.bookRepository = bookRepository;

}

public void searchBook(String title) {

System.out.println("[Service] Searching for book: " + title);

String result = bookRepository.findBookByTitle(title);

System.out.println("[Result] " + result);

}

}

**BookRepository Class:**

package com.library.repository;

import java.util.HashMap;

import java.util.Map;

public class BookRepository {

private Map<String, String> bookDatabase;

public BookRepository() {

// Dummy data

bookDatabase = new HashMap<>();

bookDatabase.put("Java Fundamentals", "John Doe");

bookDatabase.put("Spring in Action", "Craig Walls");

bookDatabase.put("Clean Code", "Robert C. Martin");

}

public String findBookByTitle(String title) {

if (bookDatabase.containsKey(title)) {

return "Found: '" + title + "' by " + bookDatabase.get(title);

} else {

return "Book titled '" + title + "' not found in the library.";

}

}

}

1. **Run the Application:**

Create a main class to load the Spring context and test the configuration.

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryAppMain {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService bookService = (BookService) context.getBean("bookService");

// Test the search feature

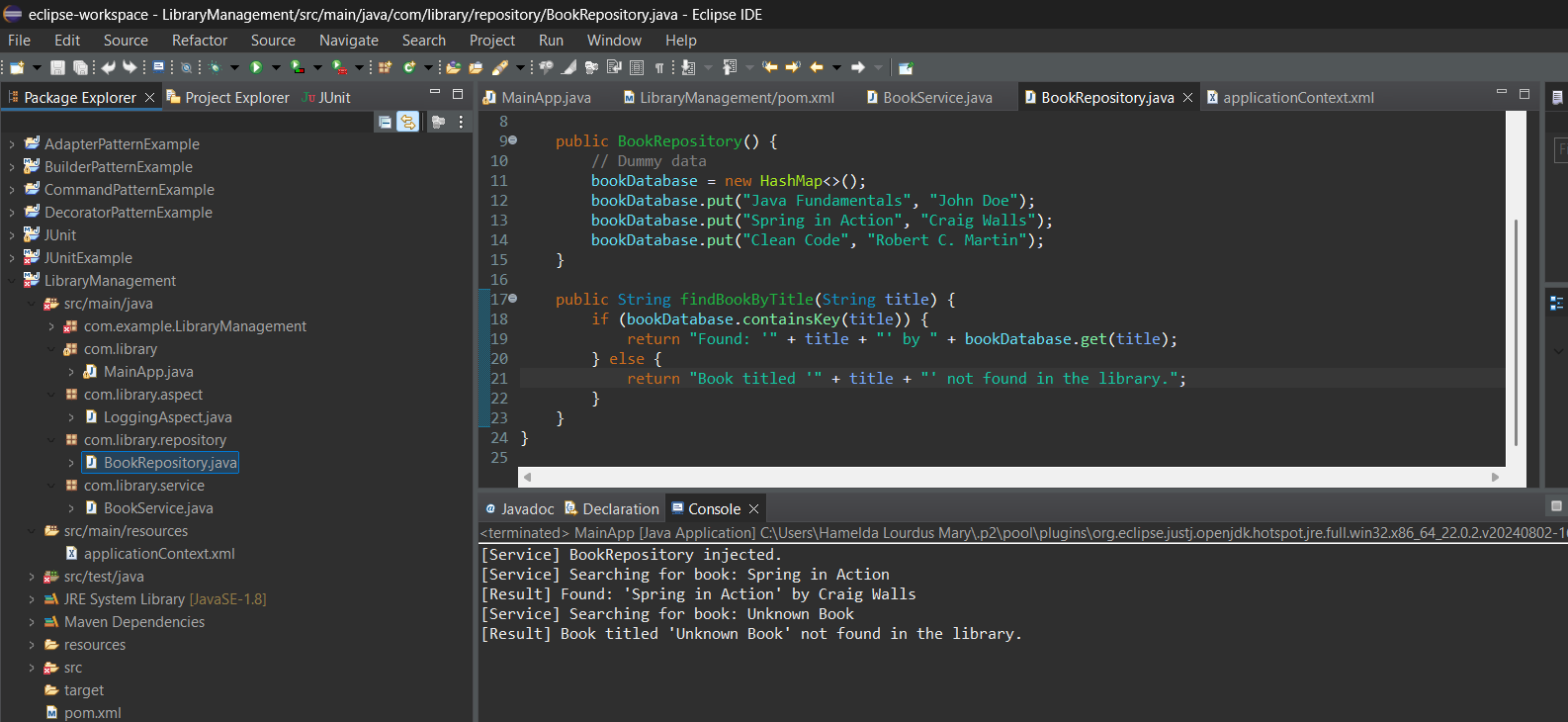
bookService.searchBook("Spring in Action");

bookService.searchBook("Unknown Book");

}

}

**Output:**



**Exercise 6: Configuring Beans with Annotations**

**Scenario:**

You need to simplify the configuration of beans in the library management application using annotations.

1. **Enable Component Scanning:**

Update **applicationContext.xml** to include component scanning for the **com.library** package.

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd">

**<!-- Enable annotation-based configuration -->**

<context:component-scan base-package="com.library" />

</beans>

1. **Annotate Classes:**

Use **@Service** annotation for the **BookService** class.

package com.library.service;

import com.library.repository.BookRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service // Marks this as a service component

public class BookService {

@Autowired // Automatically injects BookRepository

private BookRepository bookRepository;

public void search(String title) {

System.out.println("[Service] Searching for book: " + title);

String author = bookRepository.findBook(title);

if (author != null) {

System.out.println("[Result] Found: " + title + " by " + author);

} else {

System.out.println("[Result] Book not found.");

}

}

}

Use **@Repository** annotation for the **BookRepository** class.

package com.library.repository;

import org.springframework.stereotype.Repository;

import java.util.HashMap;

import java.util.Map;

@Repository // Marks this as a DAO component

public class BookRepository {

private Map<String, String> books = new HashMap<>();

public BookRepository() {

books.put("Effective Java", "Joshua Bloch");

books.put("Spring in Action", "Craig Walls");

}

public String findBook(String title) {

return books.getOrDefault(title, null);

}

}

1. **Test the Configuration:**

Run the **LibraryManagementApplication** main class to verify the annotation-based configuration.

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService service = context.getBean(BookService.class);

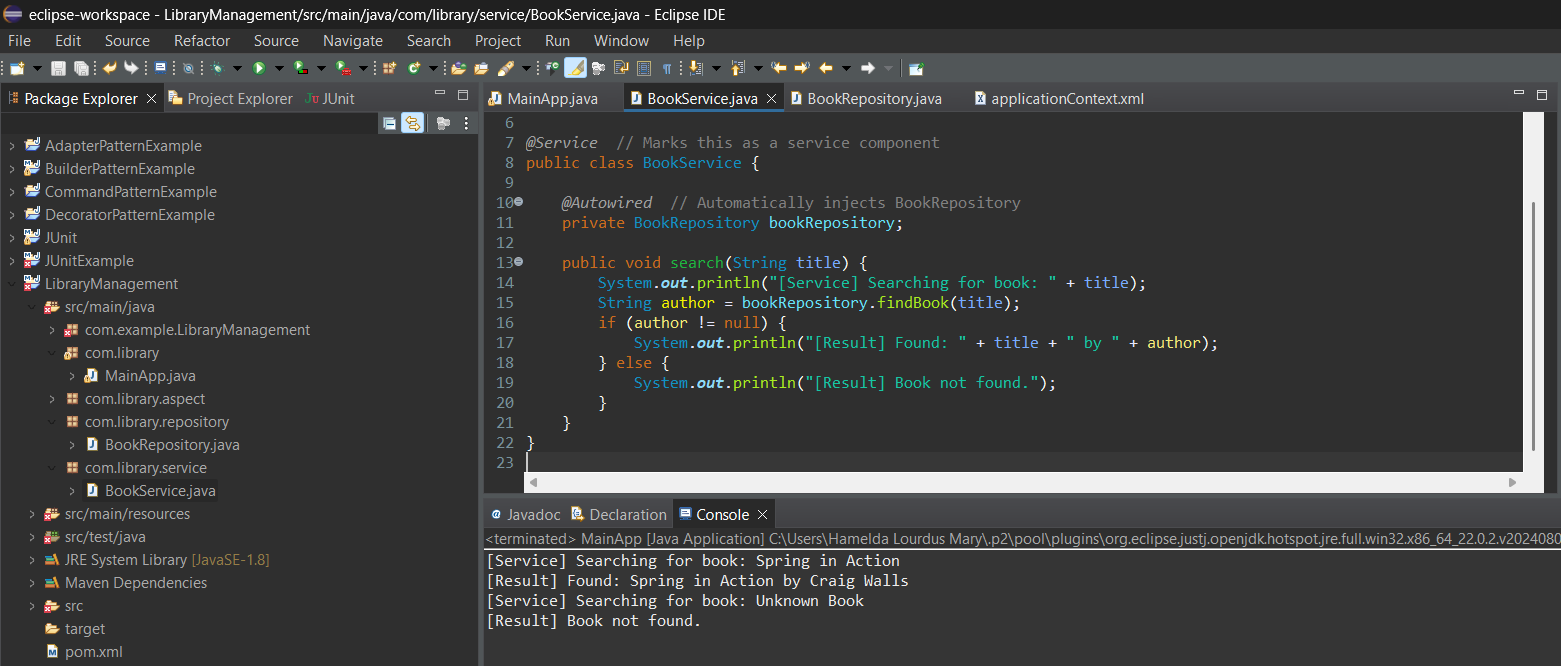
service.search("Spring in Action");

service.search("Unknown Book");

}

}

**Output:**



**Exercise 7: Implementing Constructor and Setter Injection**

**Scenario:**

The library management application requires both constructor and setter injection for better control over bean initialization.

1. **Configure Constructor Injection and Setter Injection:**

Update applicationContext.**xml** to configure constructor injection for **BookService**.

Ensure that the **BookService** class has a setter method for **BookRepository** and configure it in **applicationContext.xml**.

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

**<!-- Define BookRepository bean -->**

<bean id="bookRepository" class="com.library.repository.BookRepository" />

**<!-- Define BookService bean with both constructor and setter injection -->**

<bean id="bookService" class="com.library.service.BookService">

**<!-- Constructor injection -->**

<constructor-arg value="LibraryService" />

**<!-- Setter injection -->**

<property name="bookRepository" ref="bookRepository" />

</bean>

</beans>

1. **Create a service class**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private String serviceName;

private BookRepository bookRepository;

**// Constructor for injecting service name**

public BookService(String serviceName) {

this.serviceName = serviceName;

System.out.println("[Constructor] BookService initialized with serviceName: " + serviceName);

}

**// Setter for injecting BookRepository**

public void setBookRepository(BookRepository bookRepository) {

System.out.println("[Setter] BookRepository injected.");

this.bookRepository = bookRepository;

}

public void searchBook(String title) {

System.out.println("[" + serviceName + "] Searching for book: " + title);

String author = bookRepository.findBook(title);

if (author != null) {

System.out.println("[Result] Found: " + title + " by " + author);

} else {

System.out.println("[Result] Book not found.");

}

}

}

**Create a repository class:**

package com.library.repository;

import java.util.HashMap;

import java.util.Map;

public class BookRepository {

private Map<String, String> books = new HashMap<>();

public BookRepository() {

books.put("Spring in Action", "Craig Walls");

books.put("Clean Code", "Robert C. Martin");

}

public String findBook(String title) {

return books.getOrDefault(title, null);

}

}

1. **Test the Injection:**

Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService bookService = (BookService) context.getBean("bookService");

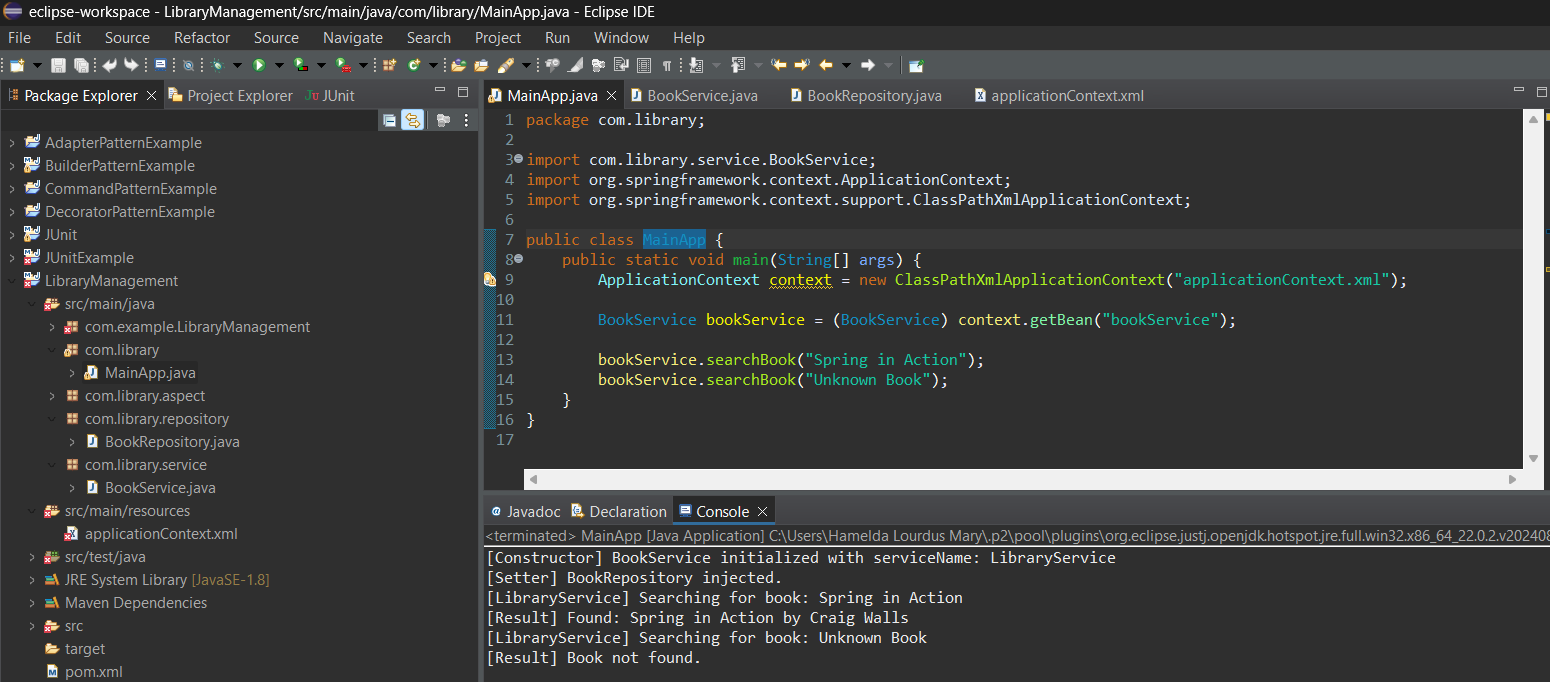
bookService.searchBook("Spring in Action");

bookService.searchBook("Unknown Book");

}

}

**Output:**



**Exercise 8: Implementing Basic AOP with Spring**

**Scenario:**

The library management application requires basic AOP functionality to separate cross-cutting concerns like logging and transaction management.

1. **Define an Aspect and Create Advice Methods:**

Create a package **com.library.aspect** and add a class **LoggingAspect**.

Define advice methods in **LoggingAspect** for logging before and after method execution.

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.After;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

**@Aspect**

public class LoggingAspect {

**@Before("execution(\* com.library.service.\*.\*(..))")**

public void logBefore(JoinPoint joinPoint) {

System.out.println("[AOP - BEFORE] Executing method: " + joinPoint.getSignature().getName());

}

**@After("execution(\* com.library.service.\*.\*(..))")**

public void logAfter(JoinPoint joinPoint) {

System.out.println("[AOP - AFTER] Completed method: " + joinPoint.getSignature().getName());

}

}

1. **Create a service class :**

package com.library.service;

import com.library.repository.BookRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

public void searchBook(String title) {

System.out.println("[Service] Searching for book: " + title);

String author = bookRepository.findBook(title);

if (author != null) {

System.out.println("[Result] Found: " + title + " by " + author);

} else {

System.out.println("[Result] Book not found.");

}

}

}

**Create a repository class:**

package com.library.repository;

import org.springframework.stereotype.Repository;

import java.util.HashMap;

import java.util.Map;

@Repository

public class BookRepository {

private Map<String, String> books = new HashMap<>();

public BookRepository() {

books.put("Spring in Action", "Craig Walls");

books.put("Effective Java", "Joshua Bloch");

}

public String findBook(String title) {

return books.getOrDefault(title, null);

}

}

1. **Configure the Aspect:**

Update **applicationContext.xml** to register the aspect and enable **AspectJ** auto-proxying.

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd">

<!-- Enable component scanning -->

<context:component-scan base-package="com.library" />

**<!-- Enable AspectJ auto-proxying -->**

<aop:aspectj-autoproxy/>

**<!-- Register Aspect as Bean -->**

<bean id="loggingAspect" class="com.library.aspect.LoggingAspect" />

</beans>

1. **Test the Aspect:**

Run the **LibraryManagementApplication** main class to verify the AOP functionality.

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService bookService = context.getBean(BookService.class);

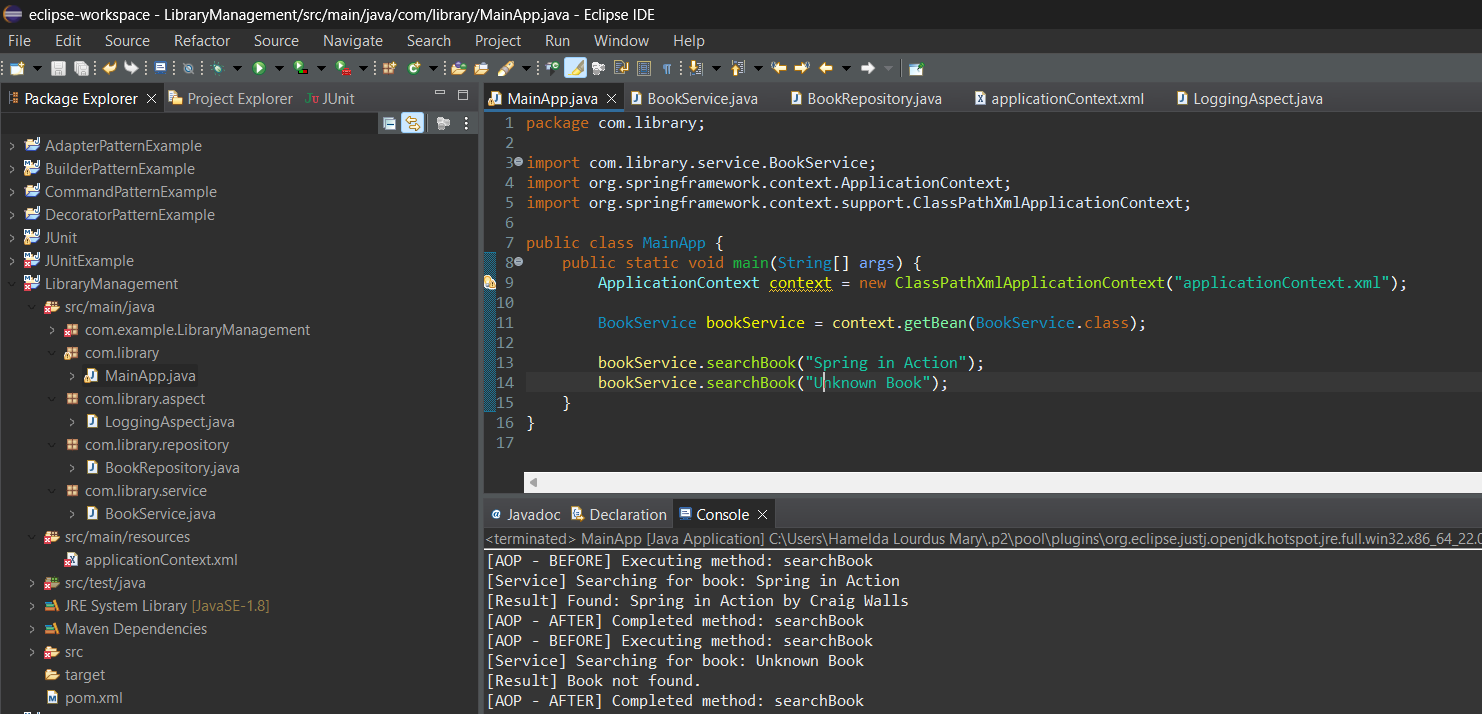
bookService.searchBook("Spring in Action");

bookService.searchBook("Unknown Book");

}

}

**Output:**



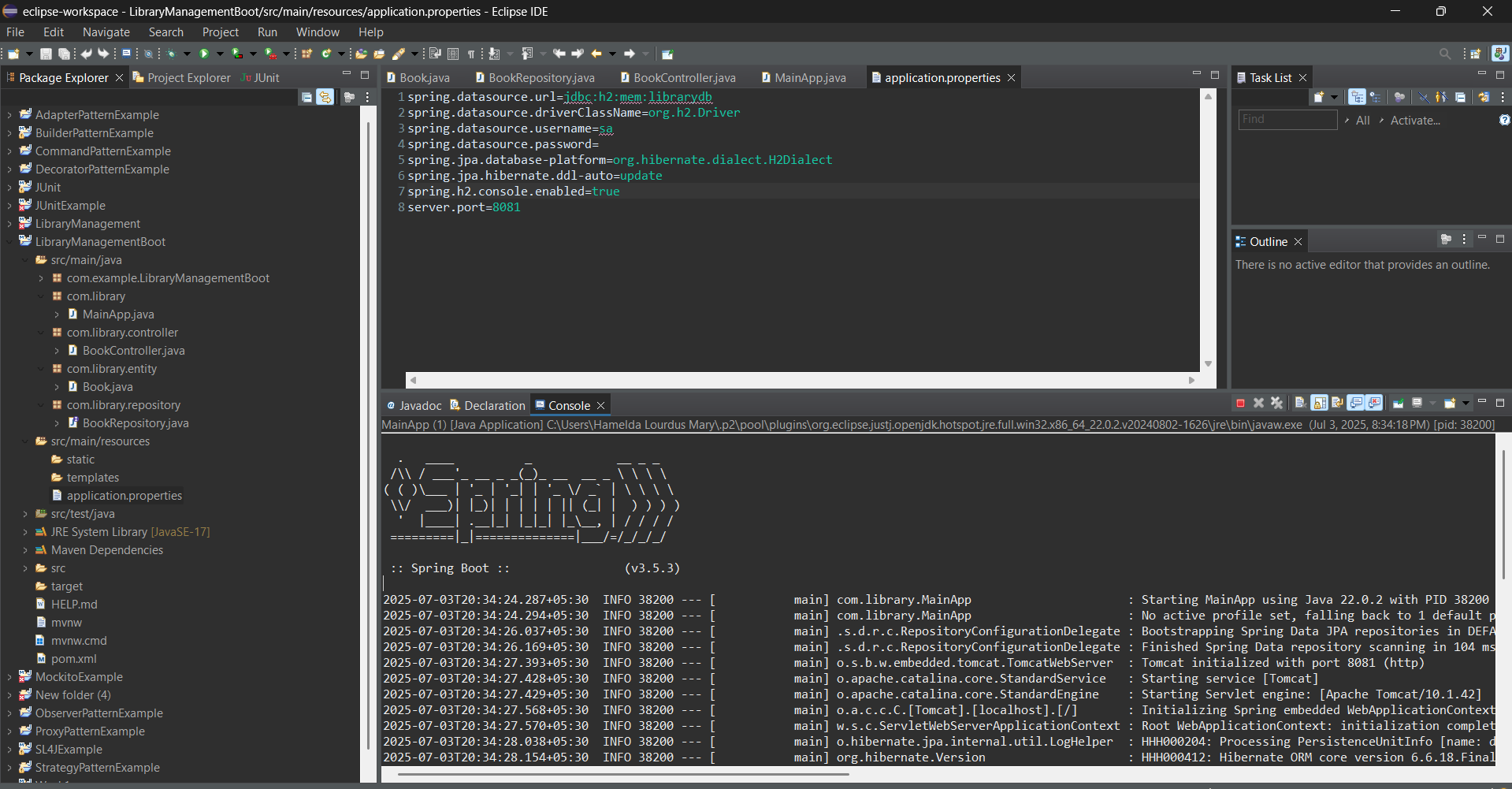
**Exercise 9: Creating a Spring Boot Application**

**Scenario:**

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

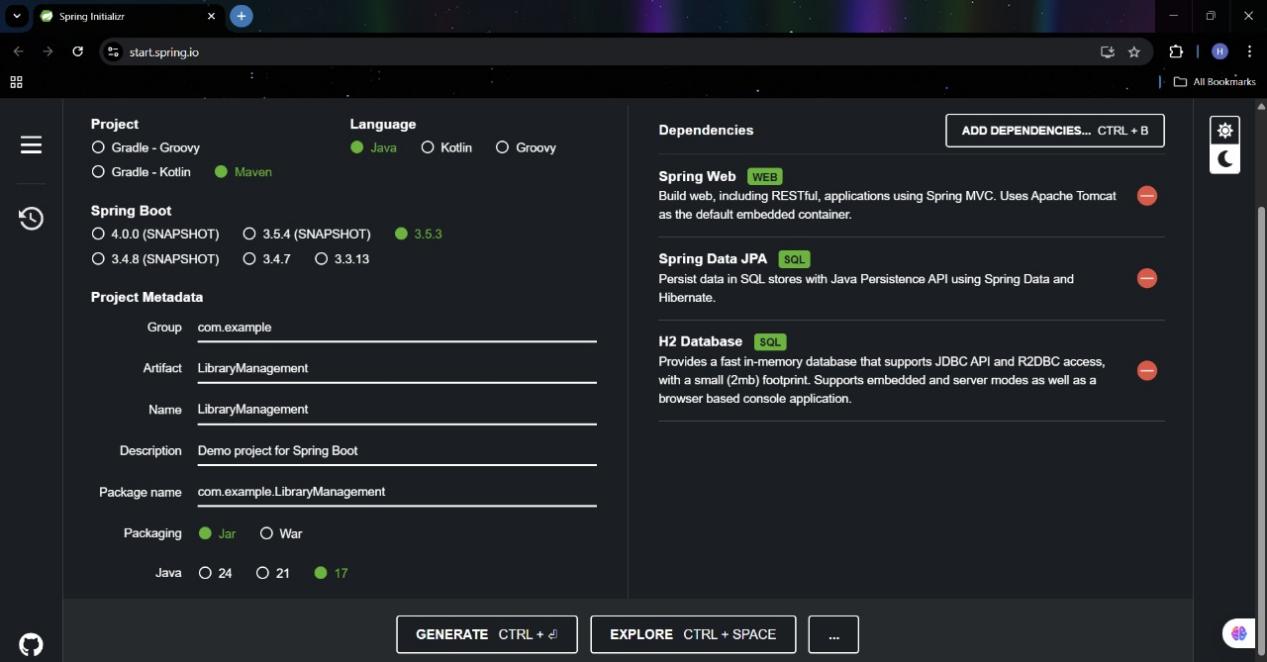
1. **Create a Spring Boot Project:**

Use **Spring Initializr** to create a new Spring Boot project named **LibraryManagement**.



1. **Add Dependencies:**

Include dependencies for **Spring Web, Spring Data JPA, and H2 Database**.



1. **Create Application Properties:**

Configure database connection properties in **application.properties**.

spring.datasource.url=jdbc:h2:mem:librarydb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

server.port=8081

1. **Define Entities and Repositories:**

**Create Book entity :**

package com.library.entity;

import jakarta.persistence.\*;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

// Constructors

public Book() {}

public Book(String title, String author) {

this.title = title;

this.author = author;

}

// Getters & Setters

public Long getId() { return 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; }

}

**Create BookRepository interface:**

package com.library.repository;

import com.library.entity.Book;

import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {

// Additional custom queries can go here

}

1. **Create a REST Controller:**

Create a **BookController** class to handle CRUD operations.

package com.library.controller;

import com.library.entity.Book;

import com.library.repository.BookRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

// Create a book

@PostMapping

public Book createBook(@RequestBody Book book) {

return bookRepository.save(book);

}

// Read all books

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

// Read a book by ID

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

return bookRepository.findById(id).orElse(null);

}

// Update a book

@PutMapping("/{id}")

public Book updateBook(@PathVariable Long id, @RequestBody Book bookDetails) {

Book book = bookRepository.findById(id).orElse(null);

if (book != null) {

book.setTitle(bookDetails.getTitle());

book.setAuthor(bookDetails.getAuthor());

return bookRepository.save(book);

}

return null;

}

// Delete a book

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

1. **Run the Application:**

Run the Spring Boot application and test the REST endpoints.

package com.library;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class MainApp {

public static void main(String[] args) {

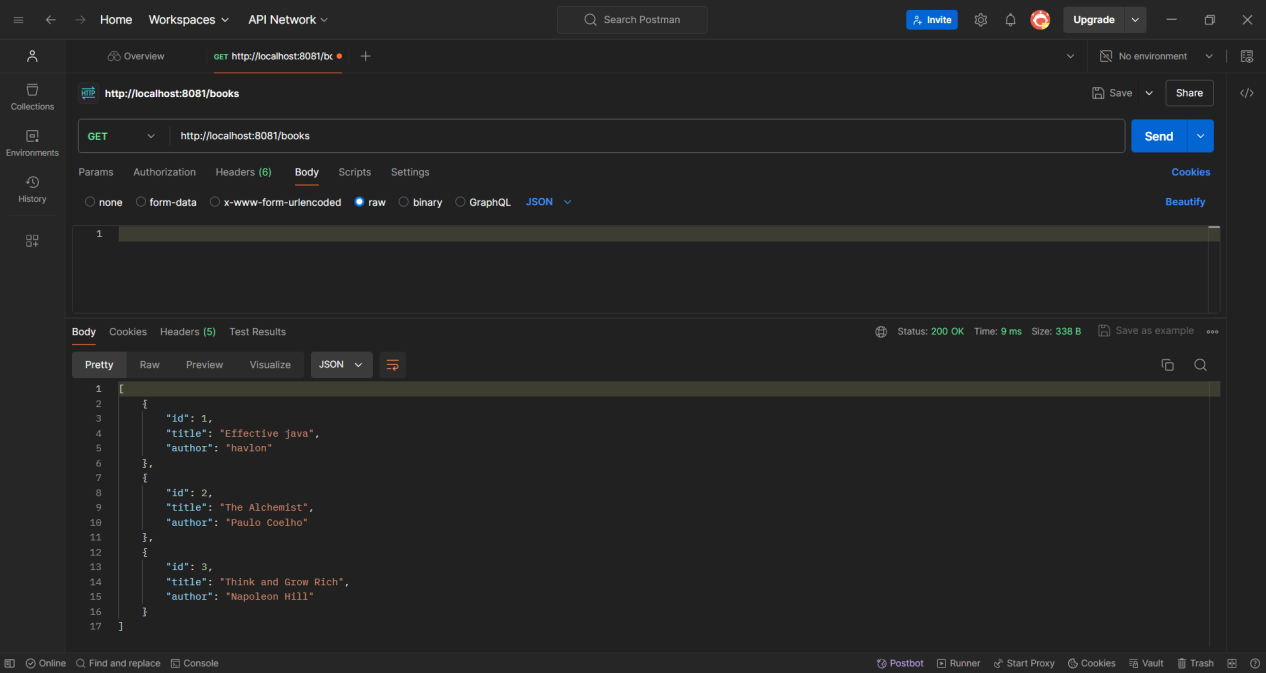
SpringApplication.run(MainApp.class, args);

}

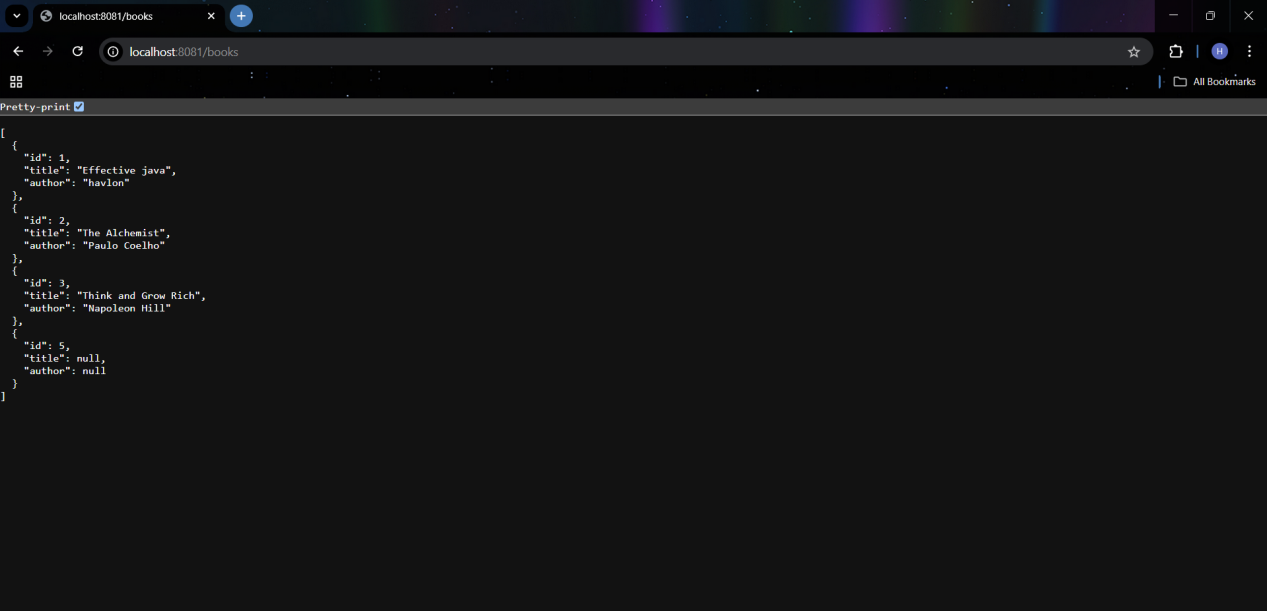
}

**Output:**

Postman API Platform:



http://localhost:8081/books



H2 Console:

