**SPRING CORE AND MAVEN**

**Exercise 1: Configuring a Basic Spring Application**

**Scenario:**

Your company is developing a web application for managing a library. You need to use the Spring Framework to handle the backend operations.

**Steps:**

1. **Set Up a Spring Project:**
   * Create a Maven project named **LibraryManagement**.
   * Add Spring Core dependencies in the **pom.xml** file.

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.29</version>

</dependency>

1. **Configure the Application Context:**
   * 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">

<!-- BookRepository Bean -->

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

<!-- BookService Bean, inject bookRepository -->

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

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

</bean>

</beans>

1. **Define Service and Repository Classes:**
   * Create a package **com.library.service** and add a class **BookService**.

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookrepo;

public void setBookRepository(BookRepository bookrepo) {

this.bookrepo=bookrepo;

}

public void addBook(String title) {

System.***out***.println("Adding Book...");

bookrepo.Book(title);

}

* + Create a package **com.library.repository** and add a class **BookRepository**.

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void Book(String title) {

System.***out***.println("Book : "+title);

}

}

1. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

**Main.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

// Load context from XML

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

// Get BookService bean

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

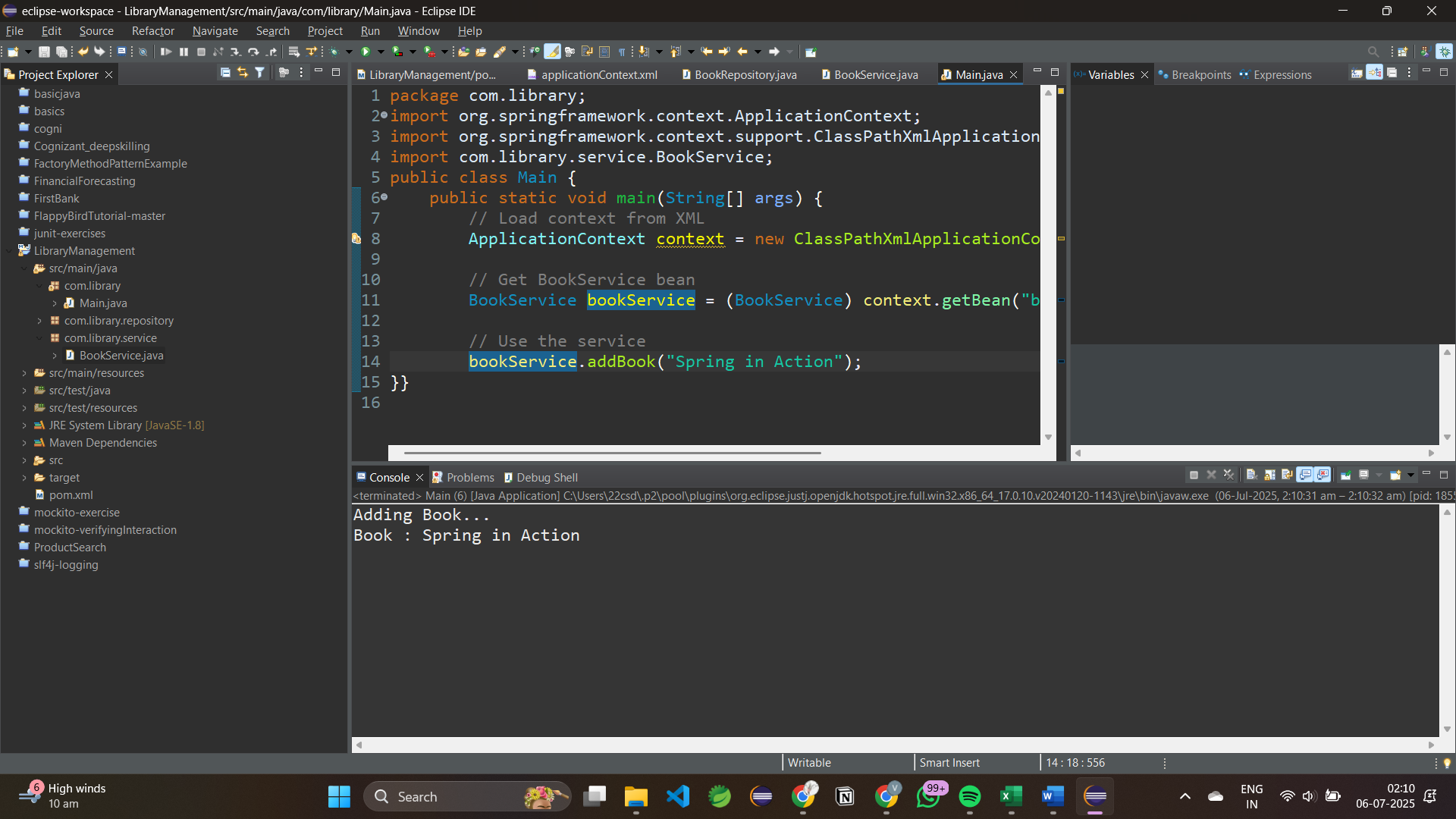
// Use the service

bookService.addBook("Spring in Action");

}

}

**OUTPUT:**



**Exercise 2: Implementing Dependency Injection**

**Scenario:**

In the library management application, you need to manage the dependencies between the BookService and BookRepository classes using Spring's IoC and DI.

**Steps:**

1. **Modify the XML Context:**
   * Update **applicationContext.xml** to wire **BookRepository** into **BookService**.

<?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">

<!-- BookRepository Bean -->

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

<!-- BookService Bean, inject bookRepository -->

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

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

</bean>

</beans>

1. **Update the BookService Class:**
   * Ensure that **BookService** class has a setter method for **BookRepository**.

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookrepo;

//setter

public void setBookRepository(BookRepository bookrepo) {

this.bookrepo=bookrepo;

}

public void addBook(String title) {

System.***out***.println("Injecting Book...");

bookrepo.Book(title);

}

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void Book(String title) {

System.***out***.println("Book saved into library : "+title);

}

}

1. **Test the Configuration:**
   * Run the **LibraryManagementApplication** main class to verify the dependency injection.

**Main.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

// Load context from XML

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

// Get BookService bean

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

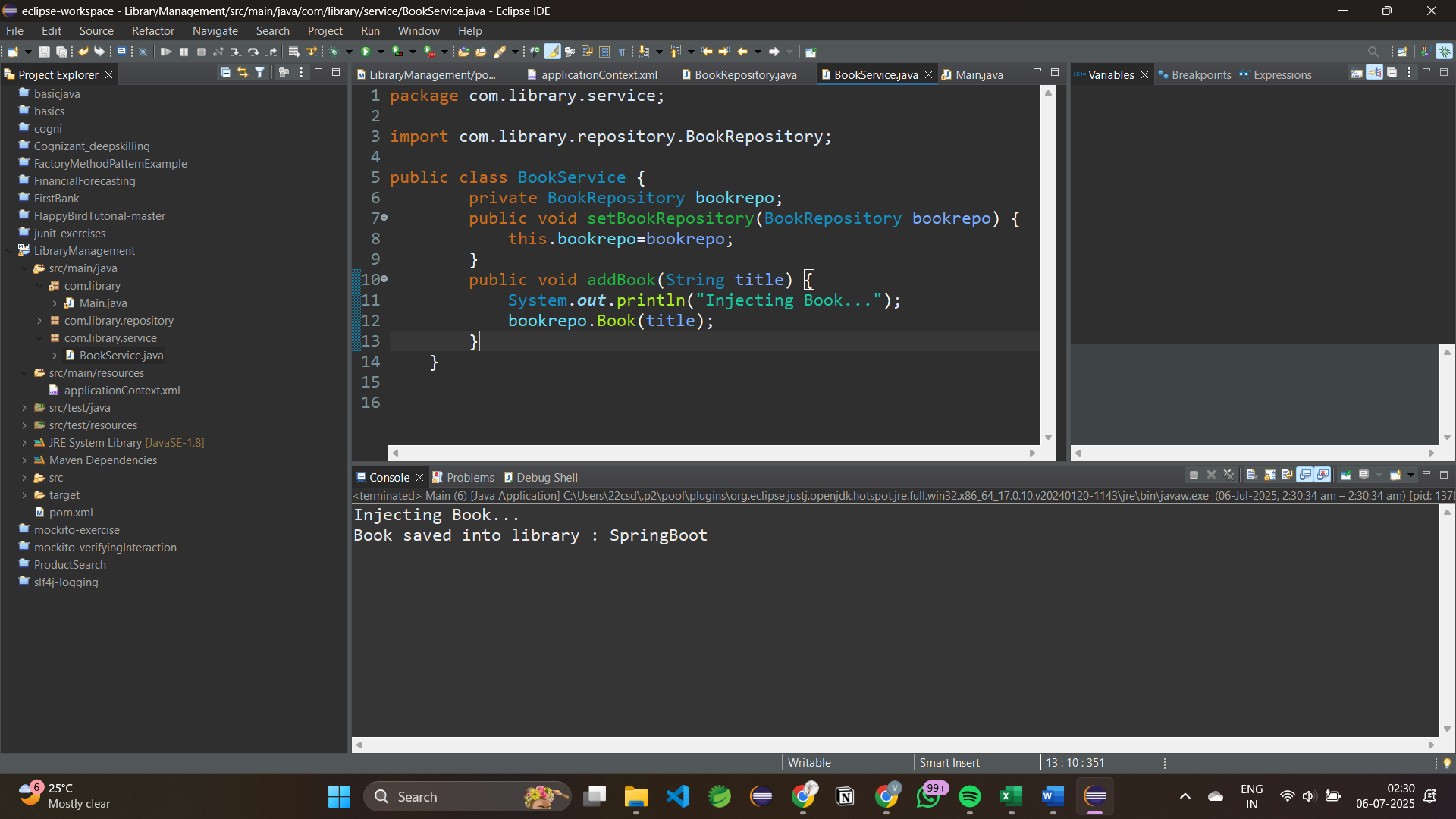
// Use the service

bookService.addBook("SpringBoot");

}

}

**OUTPUT:**



**Exercise 4: Creating and Configuring a Maven Project**

**Scenario:**

You need to set up a new Maven project for the library management application and add Spring dependencies.

**Steps:**

1. **Create a New Maven Project:**
   * Create a new Maven project named **LibraryManagement.**
2. **Add Spring Dependencies in pom.xml:**
   * Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.

**pom.xml**

<dependencies>

<!-- Spring Core Container -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.29</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.29</version>

</dependency>

<!-- Spring Web MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.29</version>

</dependency>

</dependencies>

1. **Configure Maven Plugins:**
   * Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

**pom.xml**

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

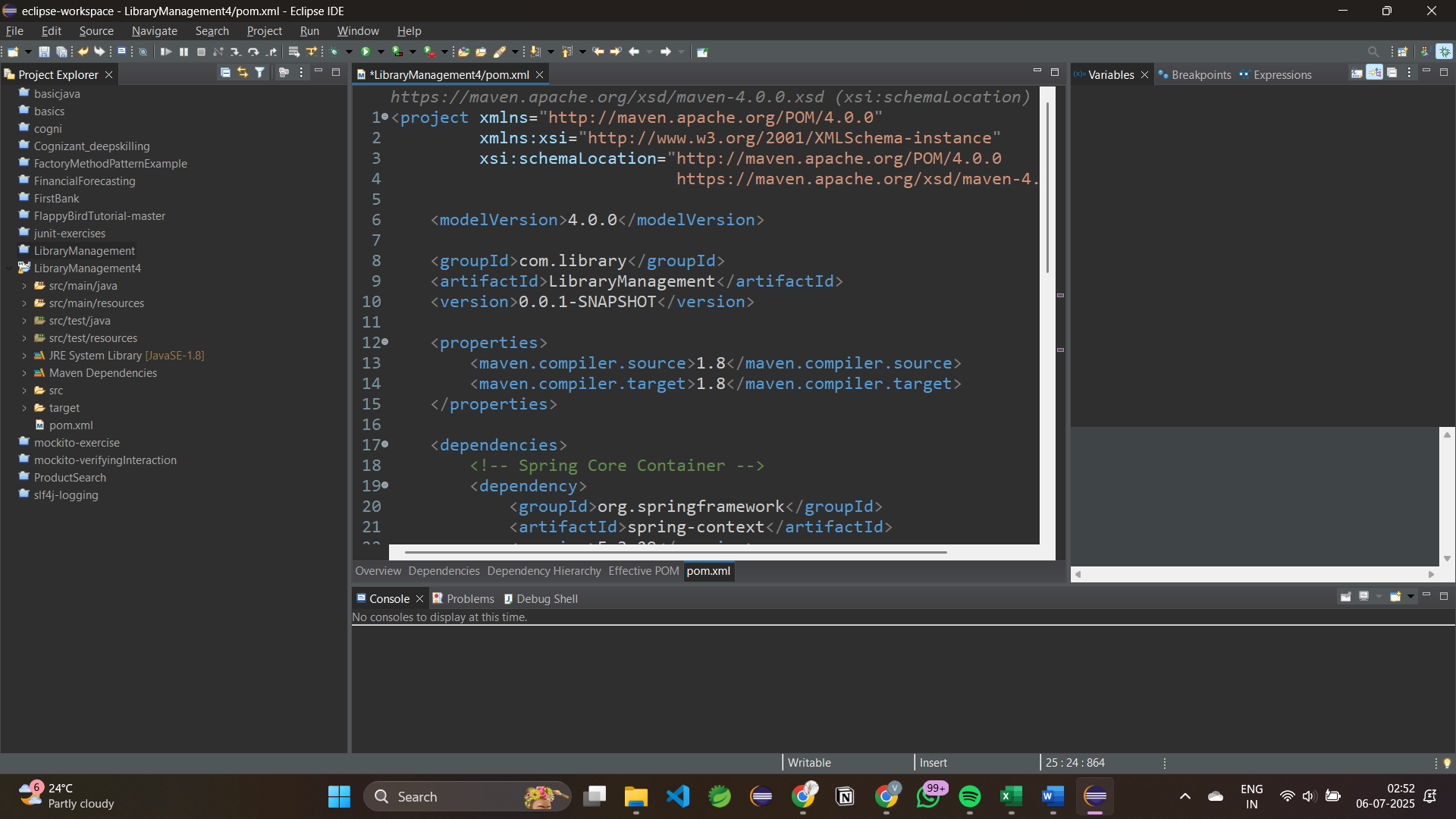
<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

**OUTPUT:**



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

**Scenario:**

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

**Steps:**

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">

<!-- BookRepository Bean -->

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

<!-- BookService Bean with dependency injection -->

<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.

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookrepo;

//setter

public void setBookRepository(BookRepository bookrepo) {

this.bookrepo=bookrepo;

}

public void addBook(String title) {

System.***out***.println("Adding Book Repository...");

bookrepo.Book(title);

}

}

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void Book(String title) {

System.***out***.println("Book: "+title);

}

}

1. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

**Main.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

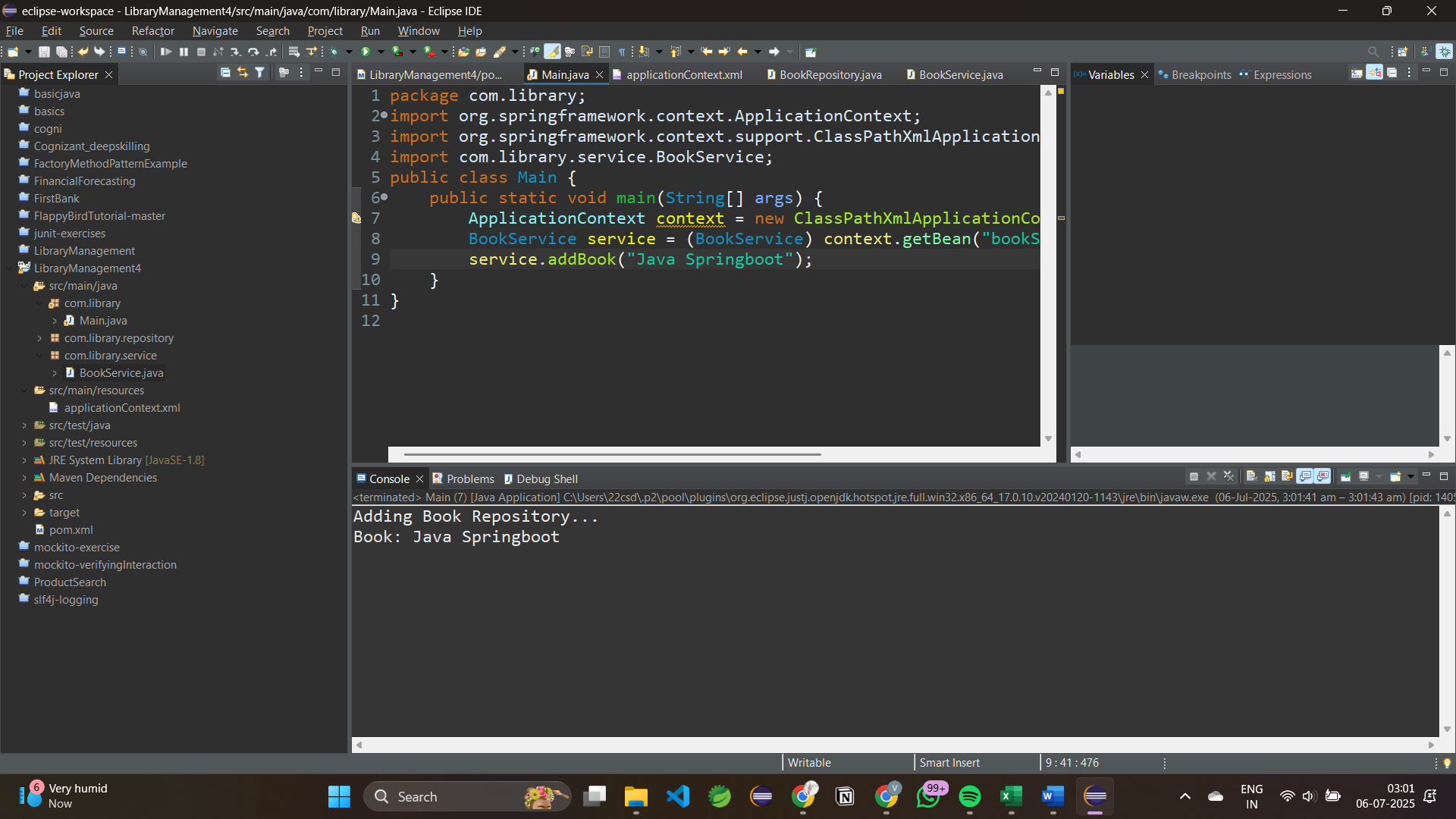
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

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

service.addBook("Java Springboot");

}}

**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.

**Steps:**

1. **Configure Constructor Injection:**
   * Update applicationContext.**xml** to configure constructor injection for **BookService**.

<?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

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

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

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

<!-- Enable component scanning -->

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

</beans>

1. **Configure Setter Injection:**
   * Ensure that the **BookService** class has a setter method for **BookRepository** and configure it in **applicationContext.xml**.

**BookService.java**

package com.library.service;

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

import org.springframework.stereotype.Service;

import com.library.repository.BookRepository;

*@Service*

public class BookService {

private BookRepository bookRepository;

*@Autowired*

public BookService(BookRepository bookRepository) {

System.out.println("Constructor Injection..");

this.bookRepository = bookRepository;

}

*@Autowired*

public void setBookRepository(BookRepository bookRepository) {

System.out.println("Setter Injection");

this.bookRepository = bookRepository;

}

public void displayBook() {

System.*out*.println("Book Service");

bookRepository.get();

}}

**BookRepository.java**

package com.library.repository;

import org.springframework.stereotype.Repository;

*@Repository*

public class BookRepository {

public void get() {

System.*out*.println("Book saved successfully");}}

1. **Test the Injection:**
   * Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.

**Main.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

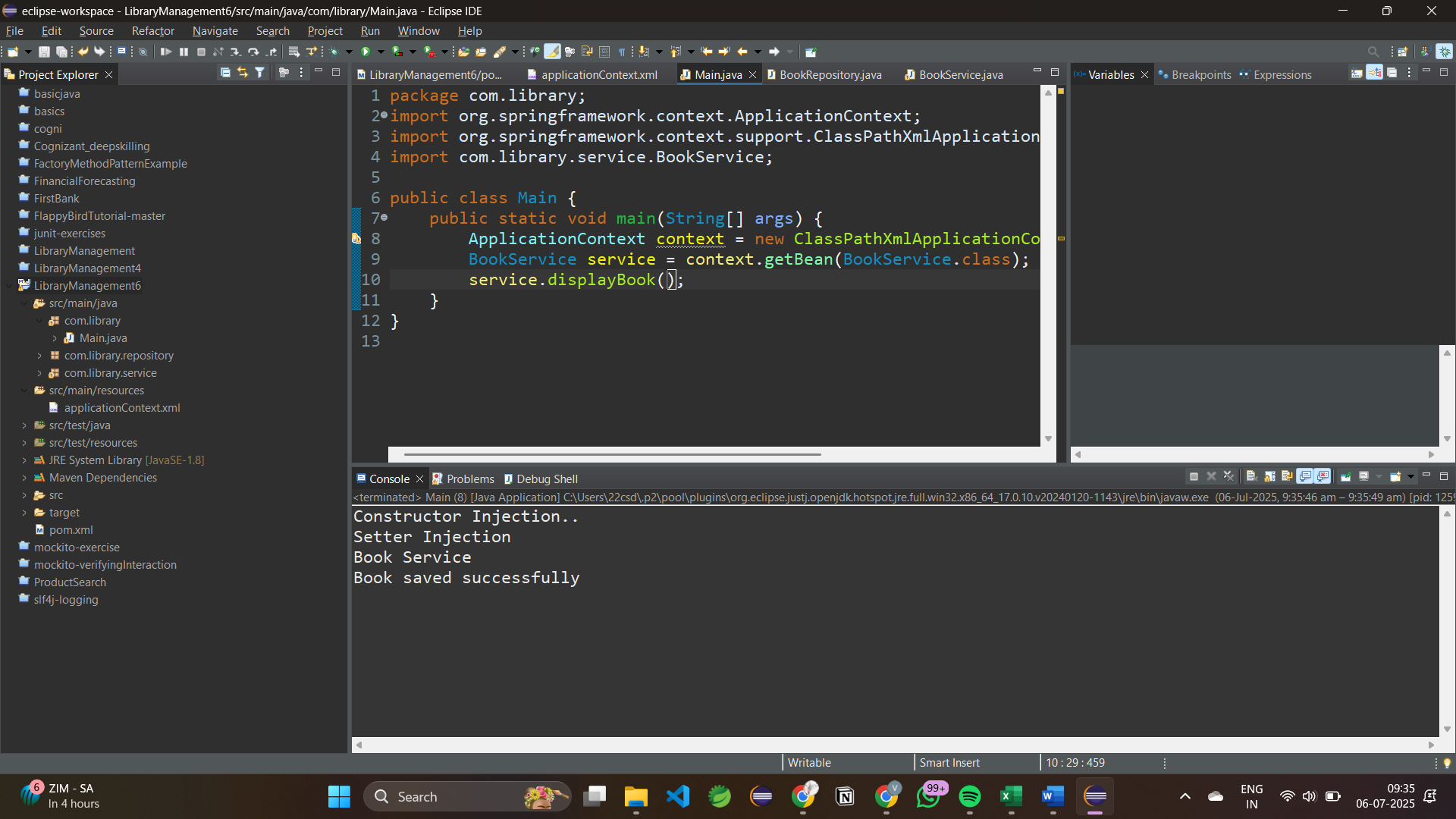
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

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

service.displayBook();

}}

**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.

**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**.

<dependencies>

<!-- Spring Boot Web Starter -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Boot JPA Starter -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- MySQL JDBC Driver -->

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

1. **Create Application Properties:**
   * Configure database connection properties in **application.properties.**

spring.application.name=LibraryManagement

spring.datasource.url=jdbc:mysql://localhost:3306/librarydb

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

1. **Define Entities and Repositories:**
   * Create **Book** entity and **BookRepository** interface.

**BookRepository.java**

package com.example.library.repository;

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

import org.springframework.stereotype.Repository;

import com.example.library.entity.BookEntity;

*@Repository*

public interface BookRepository extends JpaRepository<BookEntity,Long> {

}

**BookEntity.java**

package com.example.library.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@Entity

public class BookEntity {

@Id

@GeneratedValue(strategy=GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

private String book;

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 getBook() {

return book;

}

public void setBook(String book) {

this.book = book;

}

}

1. **Create a REST Controller:**
   * Create a BookController class to handle CRUD operations.

package com.example.library.controller;

import java.util.List;

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

import org.springframework.web.bind.annotation.DeleteMapping;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.PutMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.example.library.entity.BookEntity;

import com.example.library.repository.BookRepository;

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookRepository bookrepo;

@GetMapping

public List<BookEntity> getAllBooks(){

return bookrepo.findAll();

}

@PostMapping

public BookEntity createBook(@RequestBody BookEntity book) {

return bookrepo.save(book);

}

@GetMapping("/{id}")

public BookEntity getBookbyId(@PathVariable Long id) {

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

}

@PutMapping("/{id}")

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

BookEntity book = bookrepo.findById(id).orElse(null);

if (book != null) {

book.setTitle(bookDetails.getTitle());

book.setAuthor(bookDetails.getAuthor());

book.setBook(bookDetails.getBook());

return bookrepo.save(book);

}

return null;

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

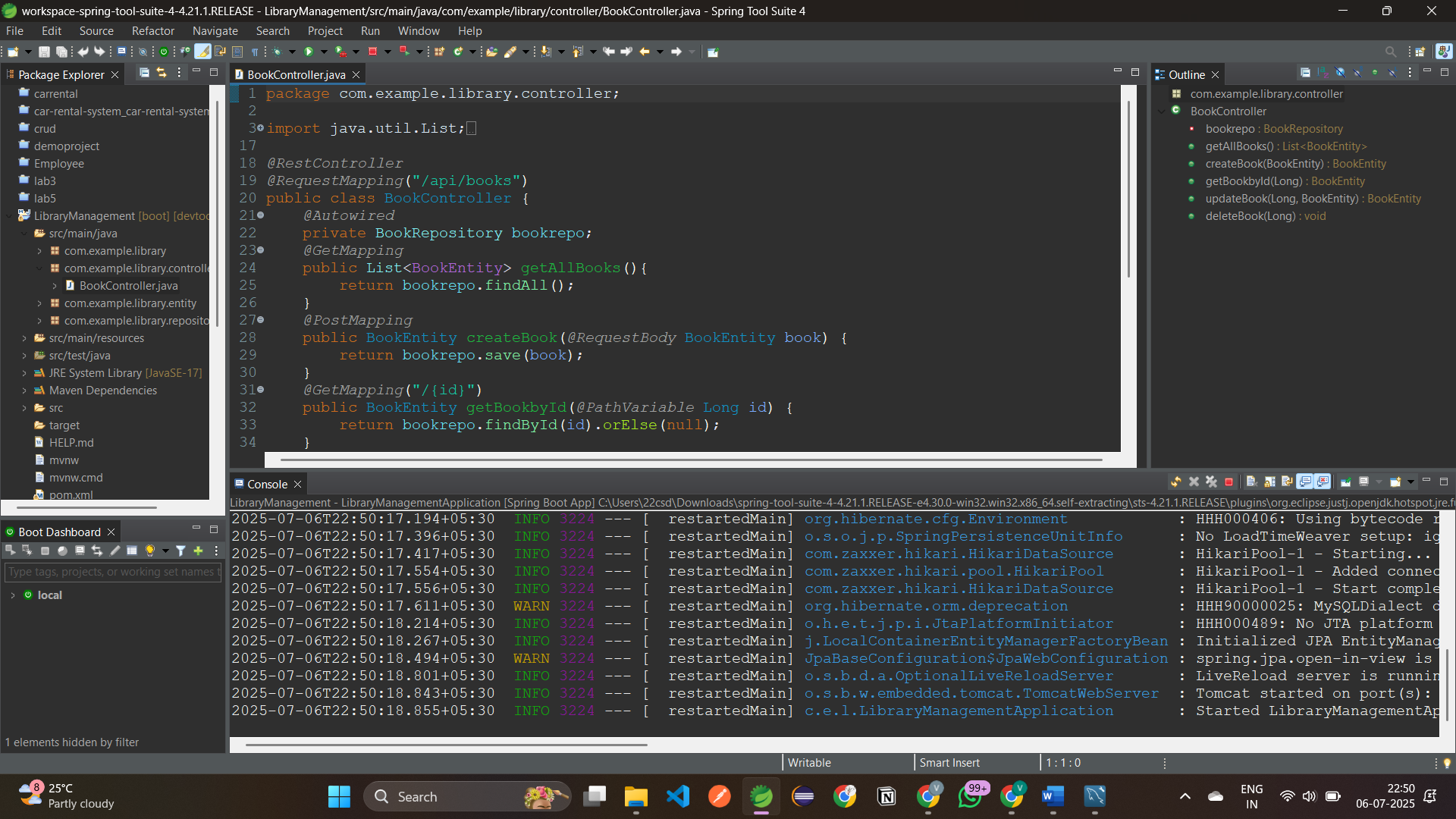
bookrepo.deleteById(id);

}

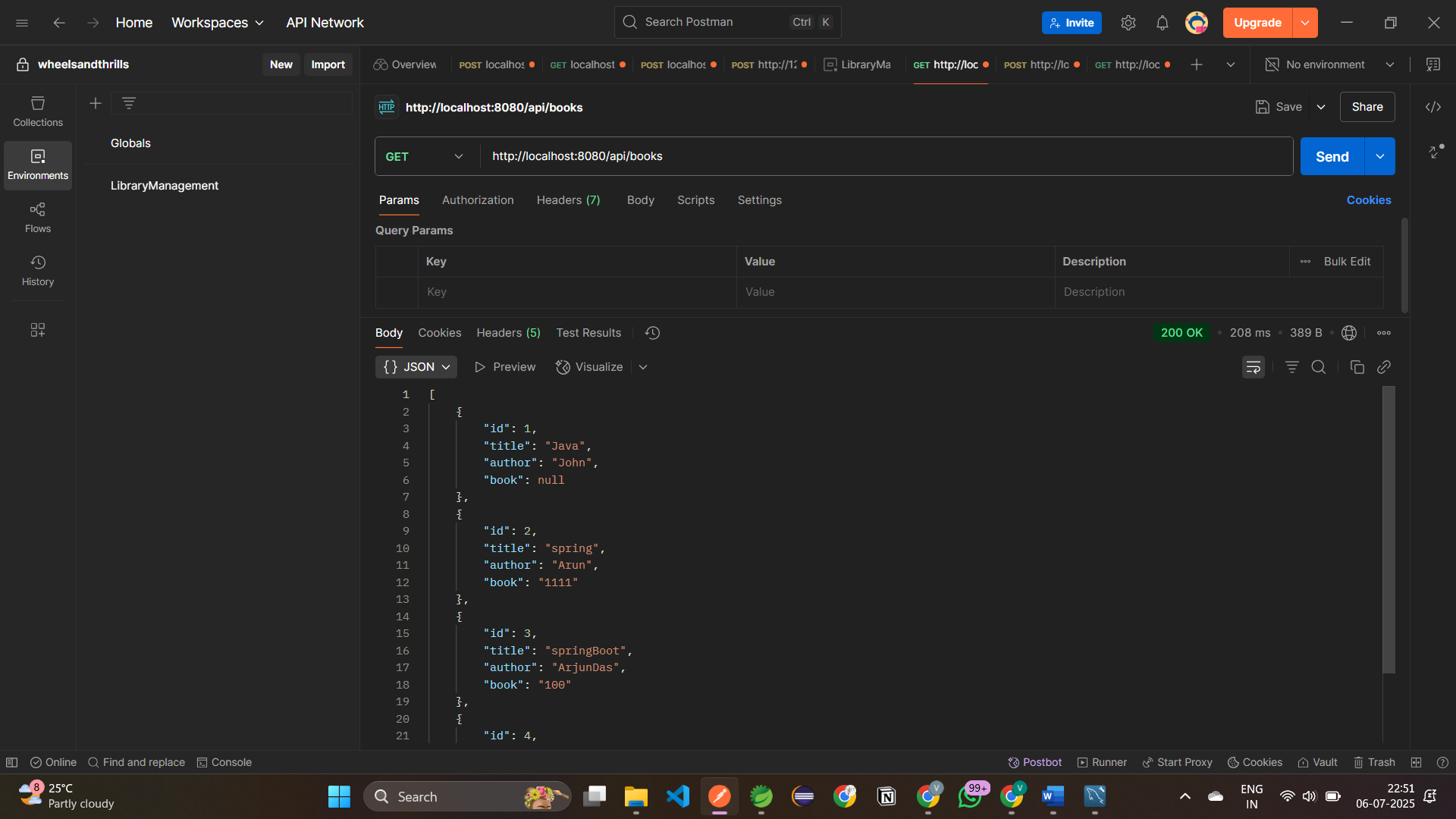
}

1. **Run the Application:**
   * Run the Spring Boot application and test the REST endpoints.

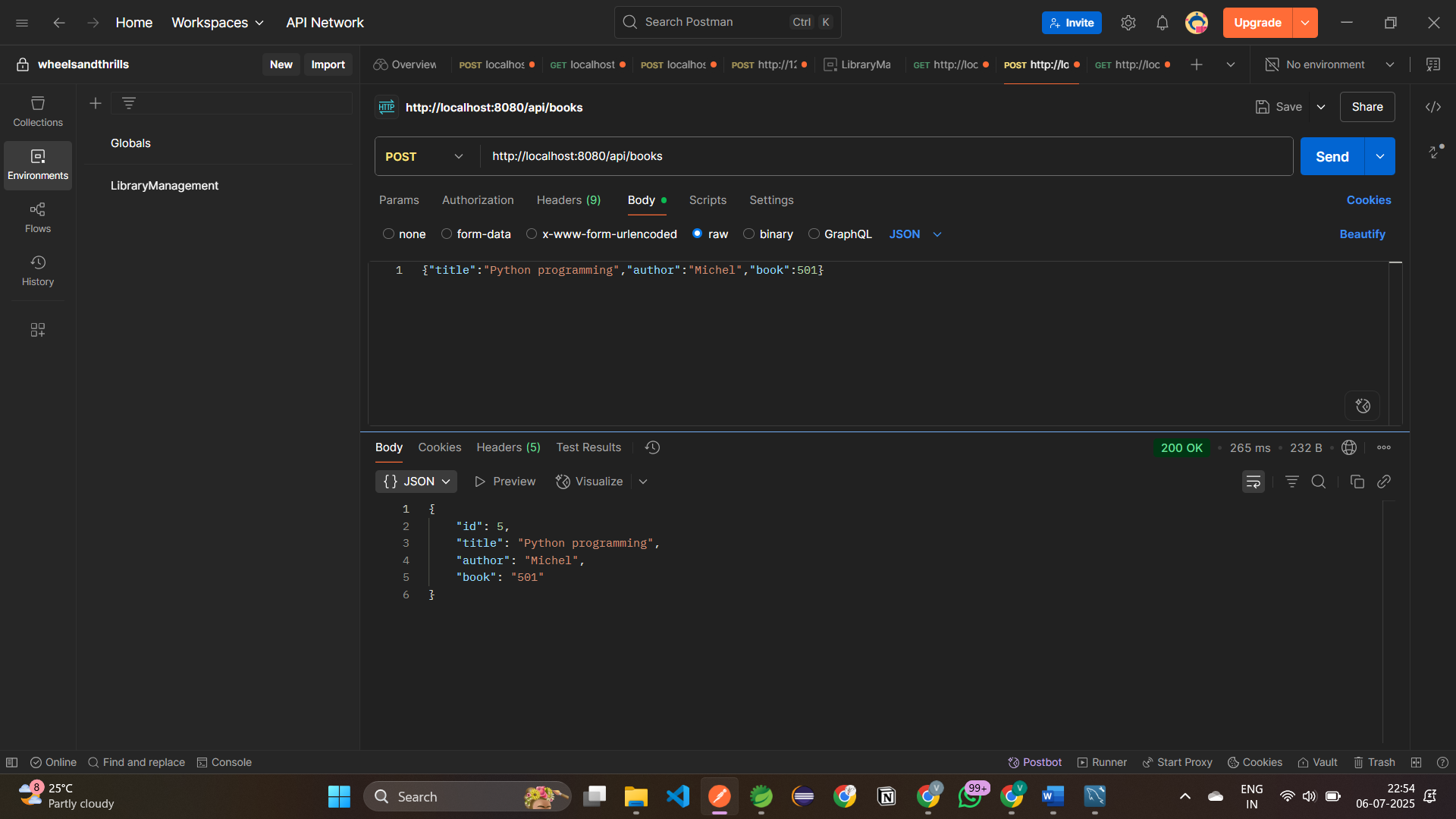
**OUTPUT:**



**Postman testing:** GetMapping(all data)



PostMapping:



MySQL DATABASE:

