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

**Scenario:**

Set up a basic Spring application to manage a library using Spring Core with XML-based configuration.

**Solution:**

1. **Create Maven Project: LibraryManagement**

**2. Add Spring Core Dependency in pom.xml**

<project>

<modelVersion>4.0.0</modelVersion>

<groupId>com.library</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Maven Compiler Plugin -->

<plugin>

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

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**3. Create the XML Configuration File: applicationContext.xml**

Location: src/main/resources/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">

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

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

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

</bean>

</beans>

**4. Define Java Classes**

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void save() {

System.out.println("Book saved to the repository.");

}

}

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook() {

System.out.println("BookService: Adding book...");

bookRepository.save();

}

}

**5. Main Class to Load Context**

**LibraryManagementApplication.java**

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");

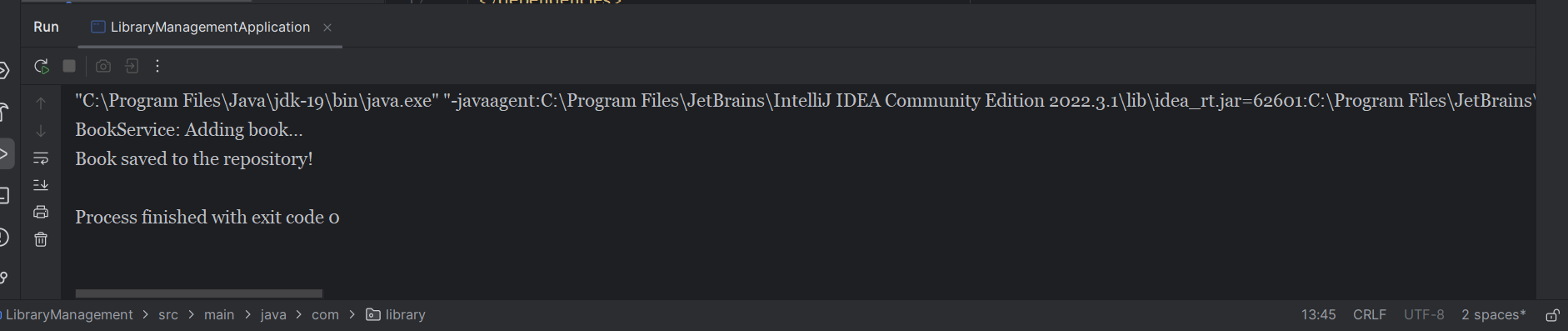
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook();

}

}

**Output**

****

**Exercise 2: Implementing Dependency Injection**

**Objective**

Use Spring's **IoC (Inversion of Control)** and **Dependency Injection (DI)** to wire BookRepository into BookService via setter injection.

**Step 1: Update BookService.java (already done in Exercise 1)**

Make sure it has a setter for BookRepository:

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

**Step 2: Update applicationContext.xml to Wire the Beans**

This is also already done in Exercise 1, but for clarity:

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

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

<!-- Setter-based DI -->

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

</bean>

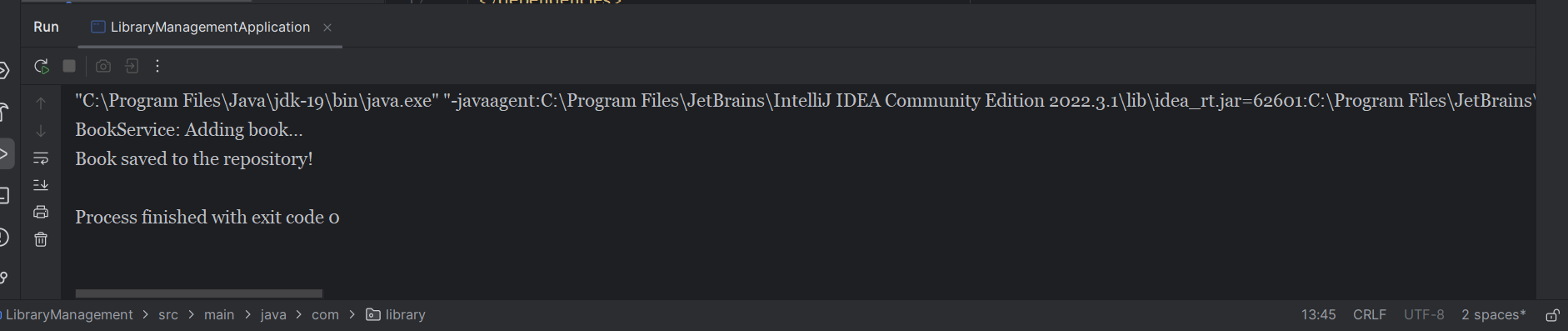
**Step 3: Run the Main Class**

**LibraryManagementApplication.java**

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

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

bookService.addBook();

**Output:**

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

**Objective**

Add logging to measure method execution times using Spring AOP (Aspect-Oriented Programming).

**Step 1: Add Spring AOP Dependency in pom.xml**

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>5.3.34</version>

</dependency>

**Step 2: Create the Aspect Class**

Create package: **com.library.aspect**

Then create class: **LoggingAspect.java**

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 {

long start = System.currentTimeMillis();

Object result = joinPoint.proceed(); // method execution

long end = System.currentTimeMillis();

System.out.println("Execution of " + joinPoint.getSignature() + " took " + (end - start) + " ms");

return result;

}

}

**Step 3: Enable AspectJ in applicationContext.xml**

Update **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"

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 AOP proxy support -->

<aop:aspectj-autoproxy/>

<!-- Aspect -->

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

<!-- Repository and Service Beans -->

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

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

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

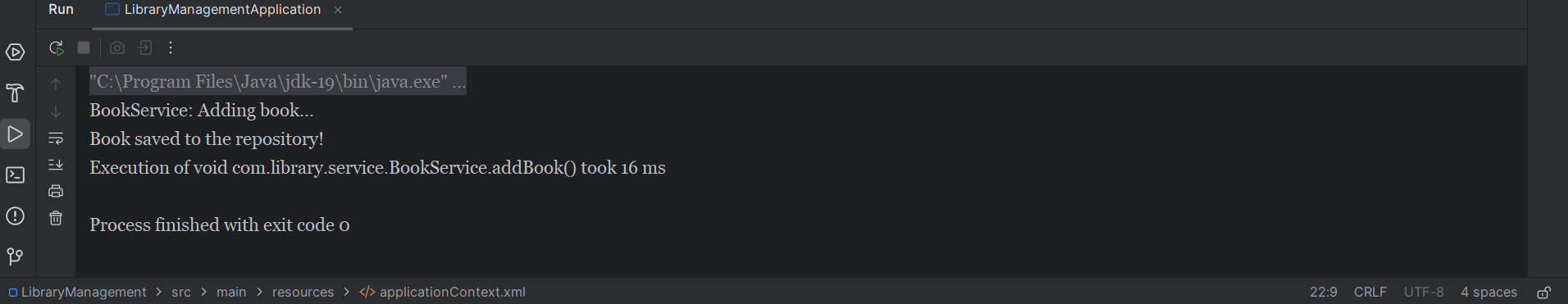
</bean>

</beans>

**Step 4: Run the Application**

Run **LibraryManagementApplication.java** again.

**Output:**

****

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

**🔹 Objective**

Set up a **new Maven project** for the library management system with required Spring dependencies and Maven plugins.

**Step 1: Create a New Maven Project (Optional if already have one)**

1. Go to **File → New → Project**
2. Select **Maven**
3. Fill in:
   * GroupId: com.library
   * ArtifactId: LibraryManagement

**Step 2: Add Spring Dependencies in pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

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

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.library</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>5.3.34</version>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Compiler Plugin -->

<plugin>

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

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Step 3: Verify Project Structure**

**Step 4: Build the Project**

To make sure Maven is set up properly:

1. Go to the **Maven window** (Right side of IntelliJ)
2. Expand your project → Lifecycle → double-click on **compile**
3. Make sure it compiles without errors

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

**Objective**

Centralize bean and dependency configuration using **Spring’s IoC (Inversion of Control) Container** with XML configuration.

**Step 1: Verify applicationContext.xml Exists**

**Step 2: Define Beans for BookService and BookRepository**

<?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 with Dependency Injection -->

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

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

</bean>

</beans>

**Step 3: Ensure BookService has a Setter**

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

**Step 4: Load the IoC Container in Main Class**

In **LibraryManagementApplication.java:**

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

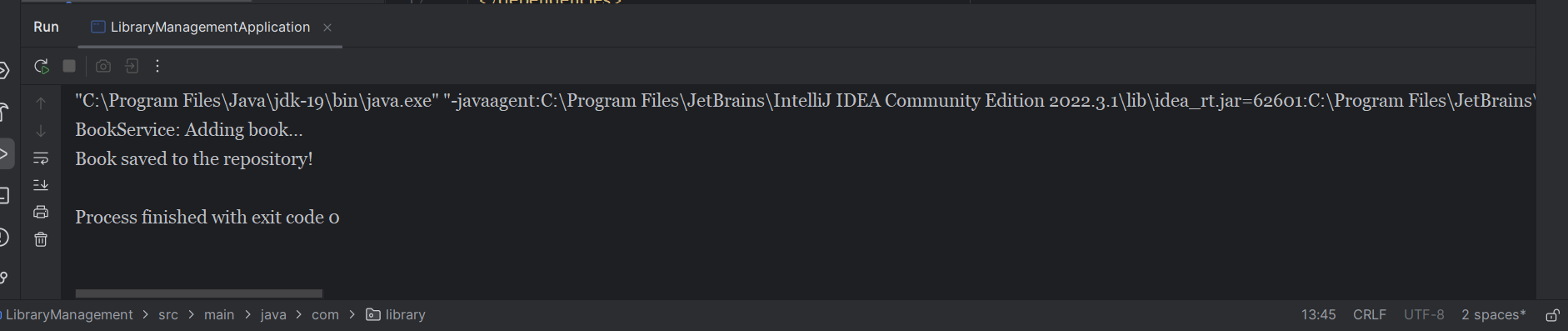
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook();

**Step 5: Run the Application**

Right-click on **LibraryManagementApplication** → **Run**

**Output:**

****

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

**Objective**

Demonstrate both **constructor injection** and **setter injection** in the Spring IoC container using XML configuration.

**Step 1: Modify BookService Class to Support Both Types of Injection**

**Add Constructor in BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Constructor Injection

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

// Setter Injection (optional)

public void setBookRepository(BookRepository bookRepository) {

System.out.println("Setter injection used.");

this.bookRepository = bookRepository;

}

public void addBook() {

System.out.println("BookService: Adding book...");

bookRepository.save();

}

}

**Step 2: Update applicationContext.xml for Constructor Injection**

Use the <constructor-arg> tag for constructor injection:

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

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

<constructor-arg ref="bookRepository"/>

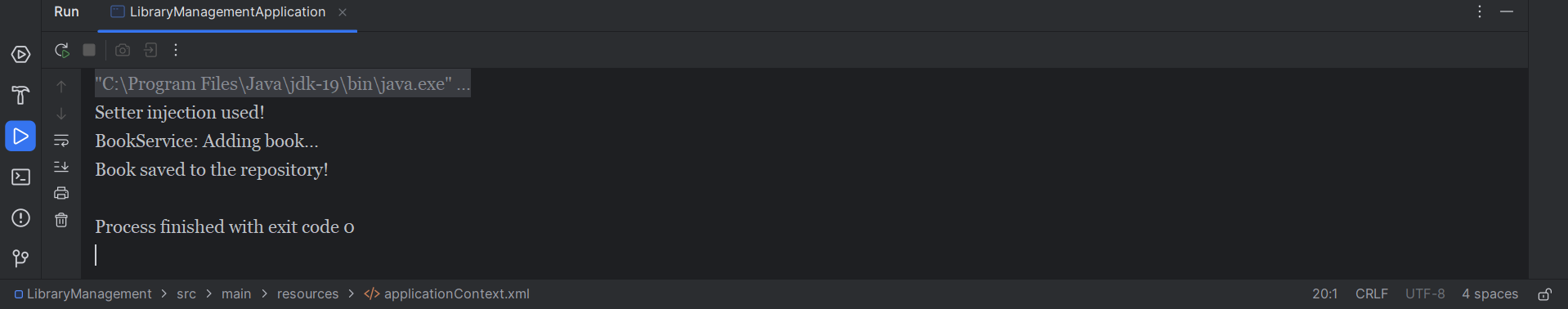
<!-- If you want to demonstrate both, add setter too -->

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

</bean>

**Step 3: Run LibraryManagementApplication.java**

**Output:**

****

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

**Objective**

Convert the library management system into a **Spring Boot application** with:

* RESTful APIs
* JPA for data access
* In-memory H2 database

**Step 1: Create the Spring Boot Project**

**Option 1: Use Spring Initializr (Recommended)**

1. Visit <https://start.spring.io>
2. Choose:
   * **Project**: Maven
   * **Language**: Java
   * **Spring Boot**: 3.2.x or latest
3. Project Metadata:
   * **Group**: com.library
   * **Artifact**: LibraryManagement
   * **Package Name**: com.library
4. Dependencies to add:
   * **Spring Web**
   * **Spring Data JPA**
   * **H2 Database**

**Step 2: Add the Book Entity**

src/main/java/com/library/model/Book.java

package com.library.model;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

// 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; }

}

**Step 3: Create the Repository**

src/main/java/com/library/repository/BookRepository.java

package com.library.repository;

import com.library.model.Book;

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

public interface BookRepository extends JpaRepository<Book, Long> {

}

**Step 4: Create the REST Controller**

src/main/java/com/library/controller/BookController.java

package com.library.controller;

import com.library.model.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("/api/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@PostMapping

public Book addBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@GetMapping("/{id}")

public Book getBook(@PathVariable Long id) {

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

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

**Step 6: Configure Database in application.properties**

src/main/resources/application.properties

# H2 In-Memory DB Config

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

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

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

# H2 Console (optional for debugging)

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

**Step 7: Main Application Class**

src/main/java/com/library/LibraryManagementApplication.java

package com.library;

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);

}

}