**Superset ID: 6394725**

**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:**
   1. Create a Maven project named **LibraryManagement**.
   2. Add Spring Core dependencies in the **pom.xml** file.
2. **Configure the Application Context:**
   1. Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
   2. Define beans for **BookService** and **BookRepository** in the XML file.
3. **Define Service and Repository Classes:**
   1. Create a package **com.library.service** and add a class **BookService**.
   2. Create a package **com.library.repository** and add a class **BookRepository**.
4. **Run the Application:**
   1. Create a main class to load the Spring context and test the configuration.

**Solution:**

**Step-1: Set Up a Spring Project:**

1. Created a Maven project named **LibraryManagement**.

Go to **File → New → Maven Project**

Check Create a simple project (skip archetype selection) → Next

Fill in:

1. **Group Id**: com
2. **Artifact Id**: LibraryManagement
3. **Version**: 1.0-SNAPSHOT (default)
4. Finish
5. Adding the below dependencies in **pom.xml** file.

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>

**Step-2: Configure the Application Context**

1. Created an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
2. Define beans for **BookService** and **BookRepository** in the XML file.

<!-- applicationContext.xml -->

<**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)

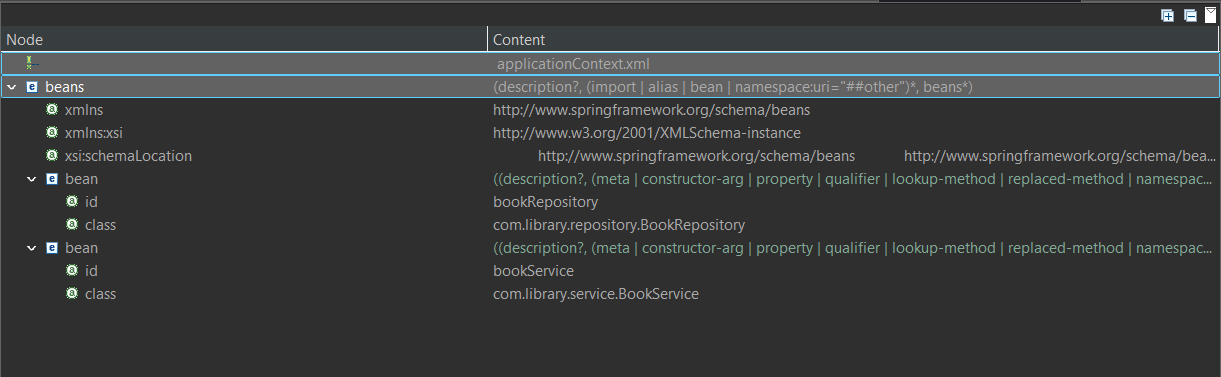
[*http://www.springframework.org/schema/beans/spring-beans.xsd*](http://www.springframework.org/schema/beans/spring-beans.xsd)*"*>

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

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

</**beans**>

XML File design:



**Step-3: Define Service and Repository Classes:**

1. Created a package **com.library.service** and add a class **BookService**.

package com.library.service;

public class BookService {

public void addBook(String bookName) {

System.***out***.println("Adding book: " + bookName);

}

}

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

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.***out***.println("Saving book: " + bookName);

}

}

**Step-4: Run the Application:**

1. 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 LibraryManagementApplication {

public static void main(String[] args) {

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

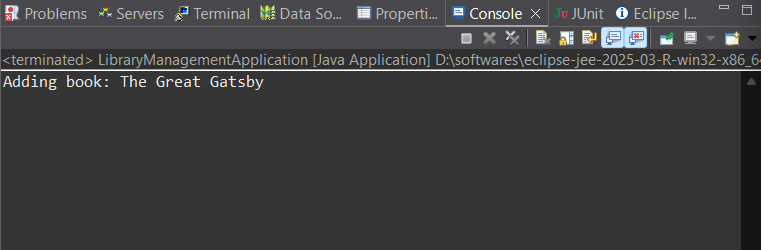
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("The Great Gatsby");

}

}

**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 Configuration:**
   1. Update **applicationContext.xml** to wire **BookRepository** into **BookService**.
2. **Update the BookService Class:**
   1. Ensure that **BookService** class has a setter method for **BookRepository**.
3. **Test the Configuration:**
   1. Run the **LibraryManagementApplication** main class to verify the dependency injection.

**Solution:**

**Step-1: Modify the XML Configuration:**

1. Update **applicationContext.xml** to wire **BookRepository** into **BookService**.

<**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)

[*http://www.springframework.org/schema/beans/spring-beans.xsd*](http://www.springframework.org/schema/beans/spring-beans.xsd)*"*>

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

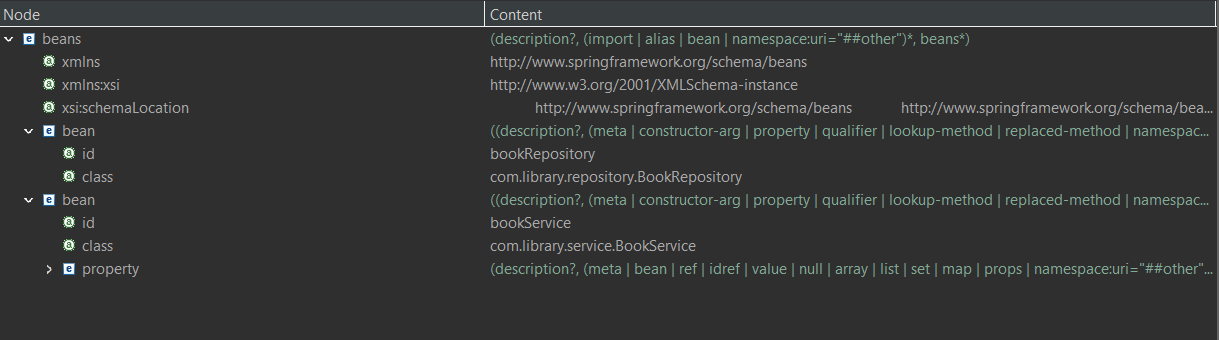
<**bean** id=*"bookService"* class=*"com.library.service.BookService"*>

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

</**bean**>

</**beans**>

XML File design:



**Step-2: Update the BookService Class:**

1. 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 bookRepository;

// Setter-based injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

System.***out***.println("Delegating to repository...");

bookRepository.saveBook(bookName);

}

}

BookRepository.java

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.***out***.println("Saving book: " + bookName);

}

}

**Step-3: Test the Configuration:**

1. Run the **LibraryManagementApplication** main class to verify the dependency 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");

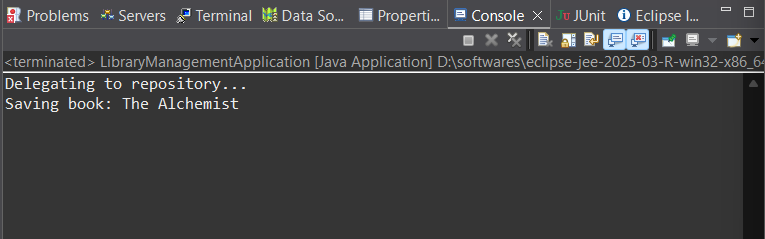
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("The Alchemist");

}

}

**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:**
   1. Create a new Maven project named **LibraryManagement**.
2. **Add Spring Dependencies in pom.xml:**
   1. Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.
3. **Configure Maven Plugins:**
   1. Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

**Solution:**

**Step-1: Created a Maven project named LibraryManagement.**

Go to **File → New → Maven Project**

Check Create a simple project (skip archetype selection) → Next

Fill in:

1. **Group Id**: com.library
2. **Artifact Id**: LibraryManagement
3. **Version**: 1.0-SNAPSHOT (default)
4. Finish

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

Add the below dependencies code to pom.xml file.

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.32</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>

**Step-3: Configure Maven Plugins:**

Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file with below code.

<build>

<plugins>

<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>

</plugins>

</build>

After editing the pom.xml, **right-click the project → Maven → Update Project** in Eclipse to download dependencies.

**Additional Exercises:**

**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:**
   1. Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
   2. Define beans for **BookService** and **BookRepository** in the XML file.
2. **Update the BookService Class:**
   1. Ensure that the **BookService** class has a setter method for **BookRepository**.
3. **Run the Application:**
   1. Create a main class to load the Spring context and test the configuration.

**Solution:**

**Step-1: Create Spring Configuration File:**

1. Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
2. Define beans for **BookService** and **BookRepository** in the XML file.

**applicationContext.xml**

<**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)

[*http://www.springframework.org/schema/beans/spring-beans.xsd*](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**>

**Step-2: Update the BookService Class:**

1. 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 bookRepository;

// Setter for DI

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

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

bookRepository.saveBook(bookName);

}

}

BookRepository.java

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.***out***.println("Saving book: " + bookName);

}

}

**Step-3: Run the Application:**

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) {

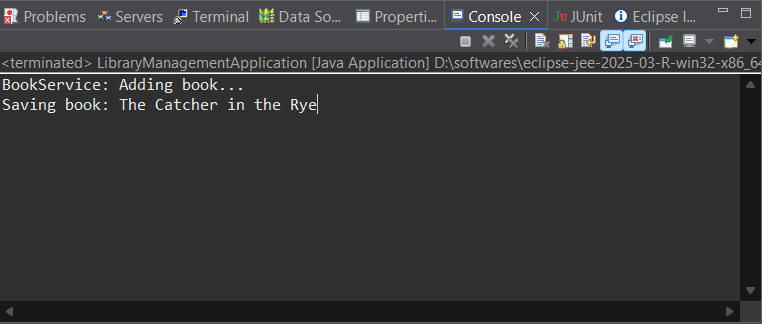
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

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

bookService.addBook("The Catcher in the Rye");

}

}

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:**
   1. Update applicationContext.**xml** to configure constructor injection for **BookService**.
2. **Configure Setter Injection:**
   1. Ensure that the **BookService** class has a setter method for **BookRepository** and configure it in **applicationContext.xml**.
3. **Test the Injection:**
   1. Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.

**Solution:**

**Step-1: Configure Constructor Injection:**

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

<!-- applicationContext.xml -->

<**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)

[*http://www.springframework.org/schema/beans/spring-beans.xsd*](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"*>

<!-- Constructor Injection -->

<**constructor-arg** value=*"Central Library"*/>

<!-- Setter Injection -->

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

</**bean**>

</**beans**>

**Step-2: Configure Setter Injection:**

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

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private String libraryName;

private BookRepository bookRepository;

// Constructor Injection

public BookService(String libraryName) {

this.libraryName = libraryName;

System.***out***.println("Constructor: Library Name set to " + libraryName);

}

// Setter Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

System.***out***.println(libraryName + " - Adding Book: " + bookName);

bookRepository.saveBook(bookName);

}

}

**Step-4: Test the 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");

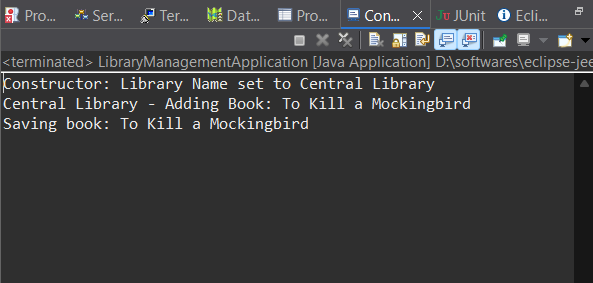
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("To Kill a Mockingbird");

}

}

**Output:**



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

**Scenario:**

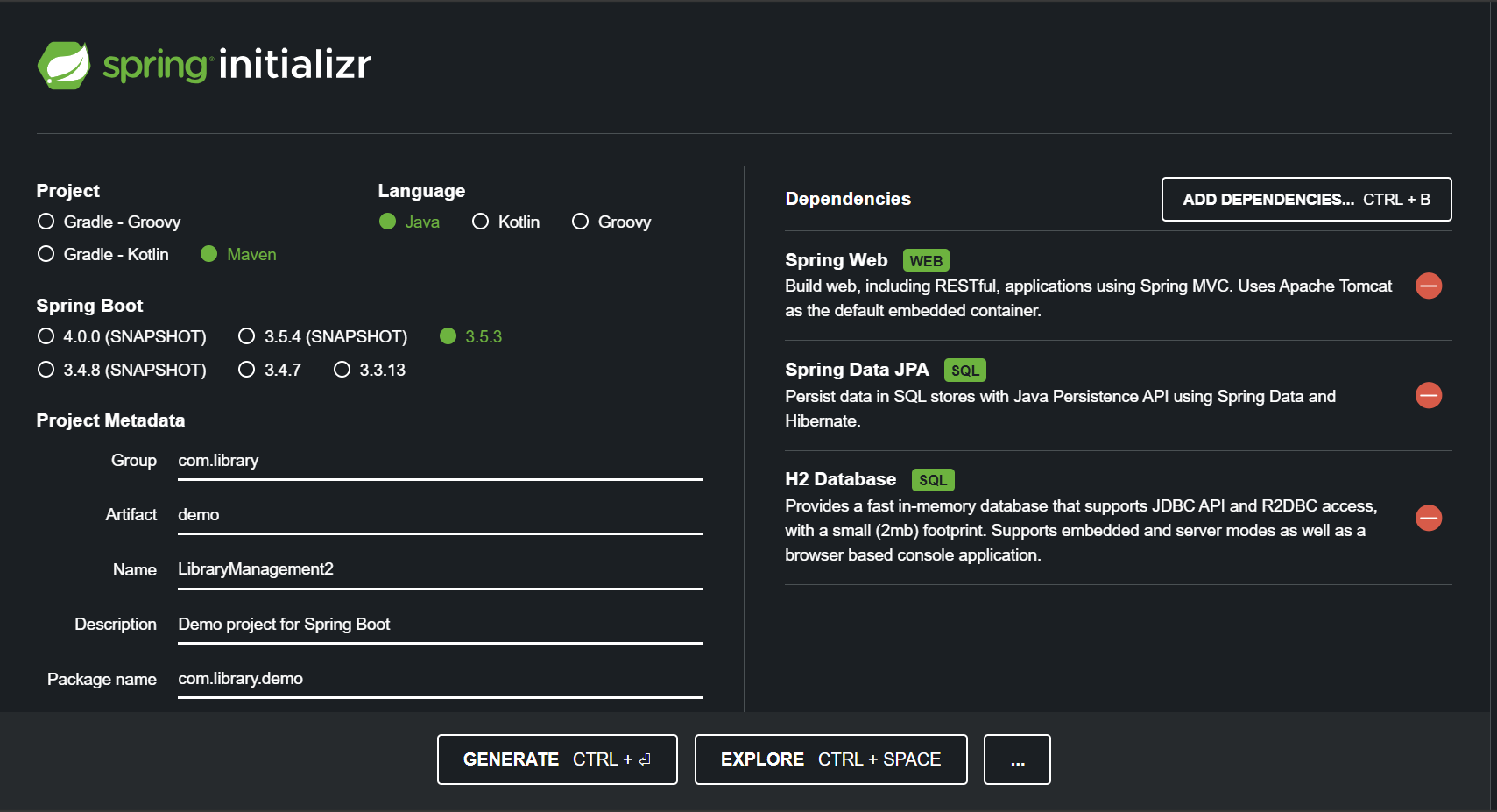
You need to create a Spring Boot application for the library management system to simplify configuration and deployment.

**Steps:**

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

**Solution:**

**Step-1&2: Create a Spring Boot Project, Add Dependencies**



**Step-3: Create Application Properties**

In (src/main/resources/application.properties) add the below code

spring.application.name=LibraryManagement2

# H2 database config

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

# JPA config

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

spring.jpa.hibernate.ddl-auto=update

# H2 console (optional)

spring.h2.console.enabled=true

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

**Step-4: Define Entities and Repositories**

***Book.java (Entity)***

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

package com.library.model;

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 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; }

}

#### **BookRepository.java**

#### 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-5: Create a REST Controller:**

#### **BookController.java**

#### 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;

import java.util.Optional;

*@RestController*

*@RequestMapping*("/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);

}

// POST - add new book

*@PostMapping*

public Book addBook(*@RequestBody* Book book) {

return bookRepository.save(book);

}

// PUT - update book

*@PutMapping*("/{id}")

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

Book book = bookRepository.findById(id).orElseThrow();

book.setTitle(bookDetails.getTitle());

book.setAuthor(bookDetails.getAuthor());

return bookRepository.save(book);

}

// DELETE - delete book

*@DeleteMapping*("/{id}")

public void deleteBook(*@PathVariable* Long id) {

bookRepository.deleteById(id);

}

}

**Step-6: Run the Application:**

Run LibraryManagement2Application.java:

package com.library.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class LibraryManagement2Application {

public static void main(String[] args) {

SpringApplication.*run*(LibraryManagement2Application.class, args);

}

}

Output:

