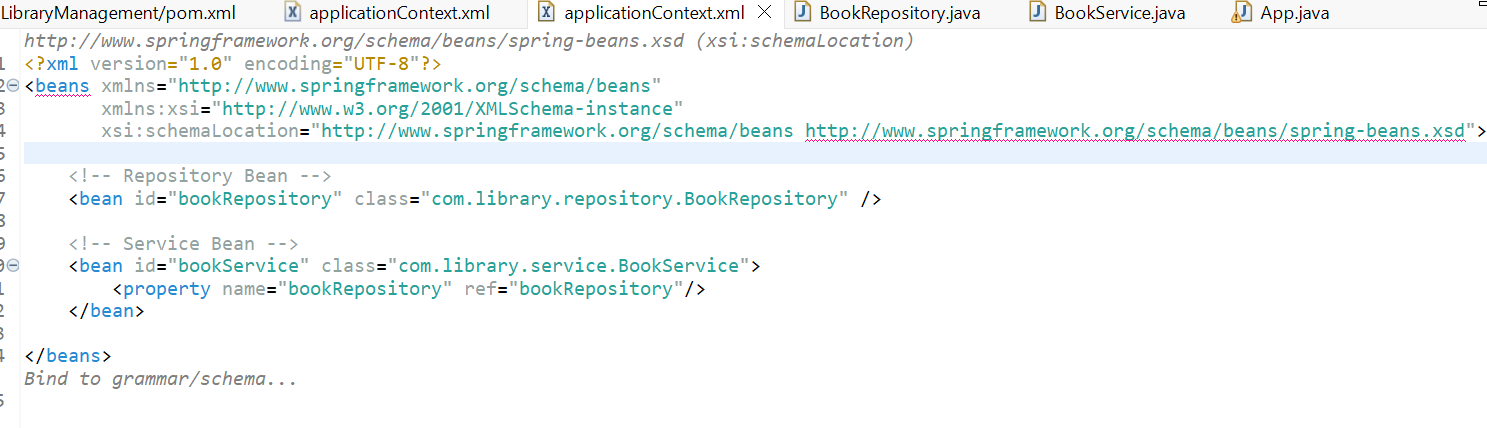
**Exercise 1: Configuring a Basic Spring Application**

**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.
2. **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.
3. **Define Service and Repository Classes:**
   * Create a package **com.library.service** and add a class **BookService**.
   * Create a package **com.library.repository** and add a class **BookRepository**.
4. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

**Code:**

**Pom.XML**

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

<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>0.0.1-SNAPSHOT</version>

<name>LibraryManagement</name>

<!-- FIXME change it to the project's website -->

<url>http://www.example.com</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.release>17</maven.compiler.release>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.junit</groupId>

<artifactId>junit-bom</artifactId>

<version>5.11.0</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<scope>test</scope>

</dependency>

<!-- Optionally: parameterized tests support -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-params</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.30</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.30</version>

</dependency>

<!-- Spring WebMVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.30</version>

</dependency>

</dependencies>

<build>

<pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->

<plugins>

<!-- clean lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#clean\_Lifecycle -->

<plugin>

<artifactId>maven-clean-plugin</artifactId>

<version>3.4.0</version>

</plugin>

<!-- default lifecycle, jar packaging: see https://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin\_bindings\_for\_jar\_packaging -->

<plugin>

<artifactId>maven-resources-plugin</artifactId>

<version>3.3.1</version>

</plugin>

<plugin>

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

<version>3.13.0</version>

</plugin>

<plugin>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.3.0</version>

</plugin>

<plugin>

<artifactId>maven-jar-plugin</artifactId>

<version>3.4.2</version>

</plugin>

<plugin>

<artifactId>maven-install-plugin</artifactId>

<version>3.1.2</version>

</plugin>

<plugin>

<artifactId>maven-deploy-plugin</artifactId>

<version>3.1.2</version>

</plugin>

<!-- site lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#site\_Lifecycle -->

<plugin>

<artifactId>maven-site-plugin</artifactId>

<version>3.12.1</version>

</plugin>

<plugin>

<artifactId>maven-project-info-reports-plugin</artifactId>

<version>3.6.1</version>

</plugin>

<plugin>

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

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

<version>3.10.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</pluginManagement>

</build>

</project>

**BookRepository.java  
package** com.library.repository;

**public** **class** BookRepository {

**public** **void** saveBook(String bookName) {

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

}

}

**BookService.java**

**package** com.library.service;

**import** com.library.repository.BookRepository;

**public** **class** BookService {

**private** BookRepository bookRepository;

// Setter for dependency injection

**public** **void** setBookRepository(BookRepository bookRepository) {

**this**.bookRepository = bookRepository;

}

**public** **void** addBook(String bookName) {

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

bookRepository.saveBook(bookName);

}

}

**App.java**

**package** com.library.LibraryManagement;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** com.library.service.BookService;

**import** com.library.repository.BookRepository;

**public** **class** App {

**public** **static** **void** main(String[] args) {

// Load the Spring container

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

// Get both beans separately

BookRepository repo = context.getBean("bookRepository", BookRepository.**class**);

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

// Call a method from each to verify they exist

System.***out***.println("Beans loaded successfully.");

repo.saveBook("Test Book from Repo"); // independent call

service.addBook("Test Book from Service"); // this will fail unless DI is done

}

}

**ApplicationContext.java**

<?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 -->

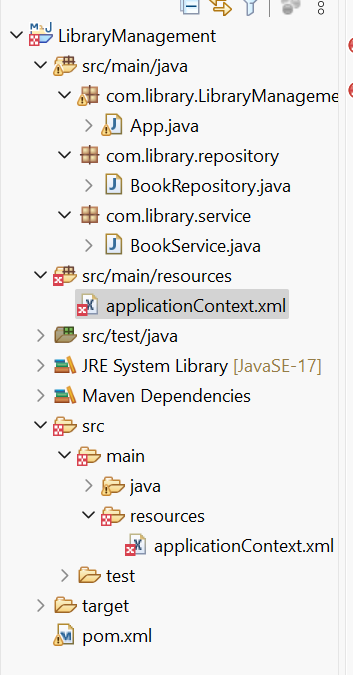
<bean id="bookService" class="com.library.service.BookService">

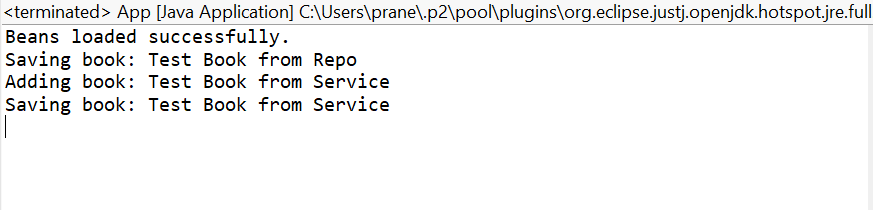
<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

**Folder Structure**



**Ouput:**

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

**Code:**

After Updating Both:  
**App.java**

**package** com.library.LibraryManagement;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** com.library.service.BookService;

**public** **class** App {

**public** **static** **void** main(String[] args) {

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

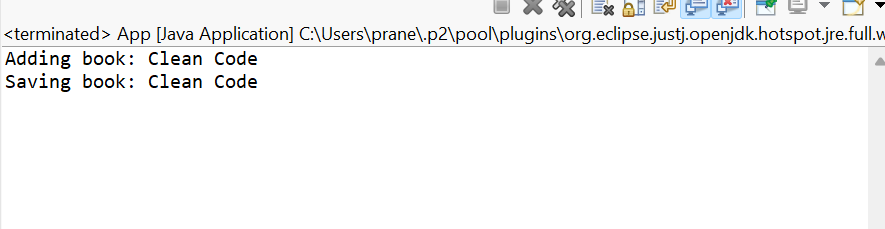
BookService bookService = context.getBean("bookService", BookService.**class**);

bookService.addBook("Clean Code");

}

}

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

**Code:  
pom.xml**

**\**<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.30</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.30</version>

</dependency>

<!-- Spring WebMVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.30</version>

</dependency>

<plugin>

<artifactId>maven-site-plugin</artifactId>

<version>3.12.1</version>

</plugin>

<plugin>

<artifactId>maven-project-info-reports-plugin</artifactId>

<version>3.6.1</version>

</plugin>

<plugin>

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

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

<version>3.10.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

**Update Project and Done  
  
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.
2. **Update the BookService Class:**
   * Ensure that the **BookService** class has a setter method for **BookRepository**.
3. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

**Code:**

**ApplicationContext.xml:  
package** com.library.LibraryManagement;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** com.library.service.BookService;

**public** **class** App {

**public** **static** **void** main(String[] args) {

// Load Spring configuration

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

// Get the service bean

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

// Test the configuration

bookService.addBook("Design Patterns");

}

}

**BookService.java  
package** com.library.service;

**import** com.library.repository.BookRepository;

**public** **class** BookService {

**private** BookRepository bookRepository;

// Setter injection

**public** **void** setBookRepository(BookRepository bookRepository) {

**this**.bookRepository = bookRepository;

}

**public** **void** addBook(String name) {

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

bookRepository.saveBook(name);

}

}

**Main.java:  
package** com.library.LibraryManagement;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** com.library.service.BookService;

**public** **class** App {

**public** **static** **void** main(String[] args) {

// Load Spring configuration

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

// Get the service bean

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

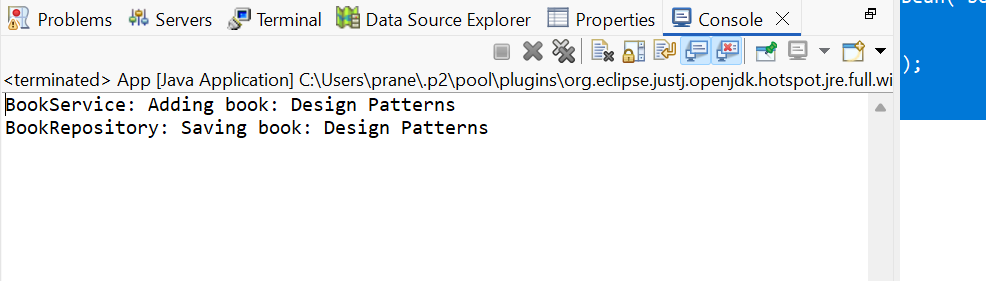
// Test the configuration

bookService.addBook("Design Patterns");

}

}

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

**Code:**

**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">

<!-- BookRepository Bean -->

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

<!-- BookService Bean with Constructor + Setter Injection -->

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

<!-- Constructor injection -->

<constructor-arg value="City Library"/>

<!-- Setter injection -->

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

</bean>

</beans>

**Main.java**

**package** com.library.LibraryManagement;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** com.library.service.BookService;

**public** **class** App {

**public** **static** **void** main(String[] args) {

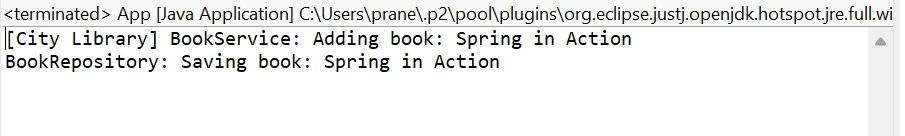
ApplicationContext context = **new** ClassPathXmlApplicationContext("applicationContext.xml");

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

service.addBook("Spring in Action");

}

}

**Output:**

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

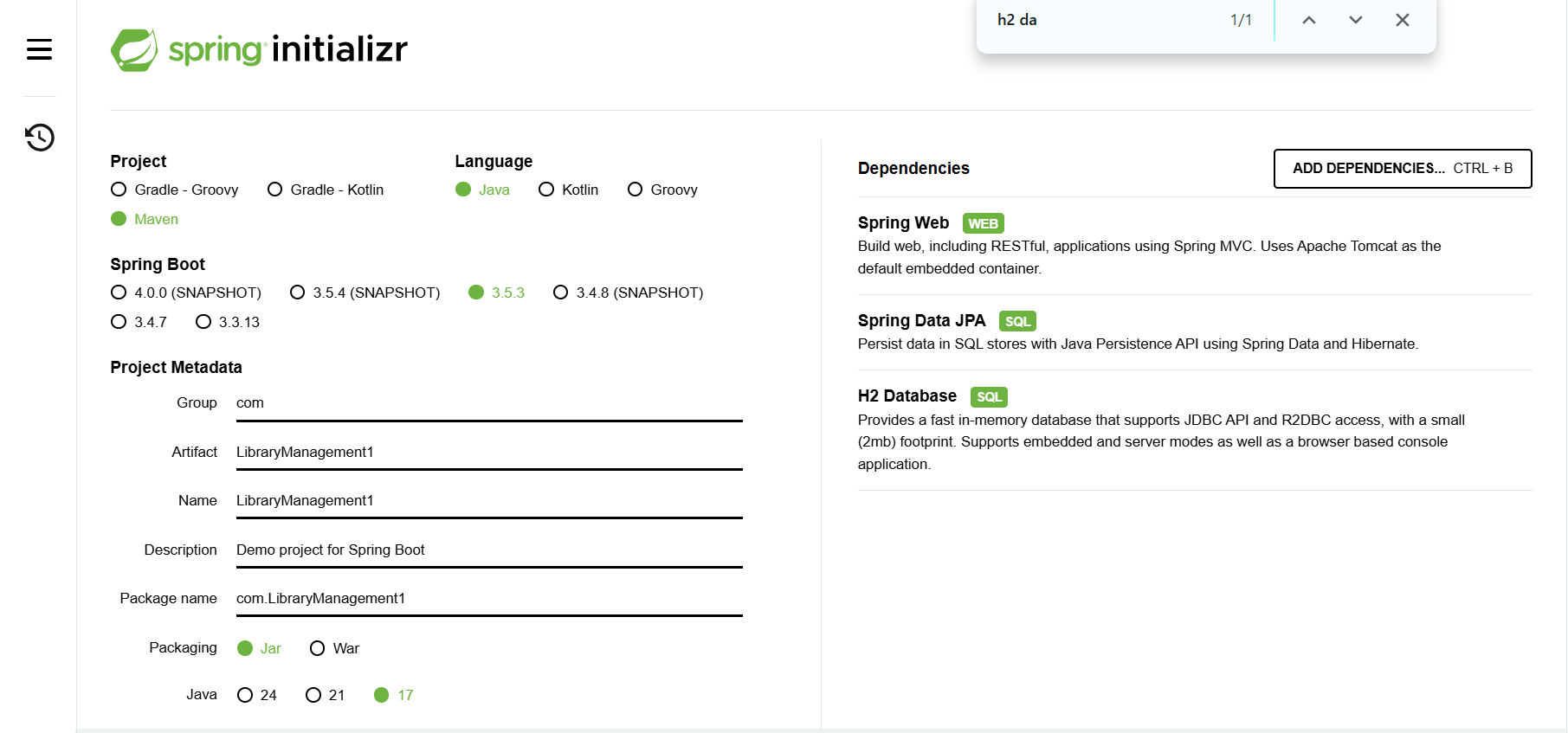
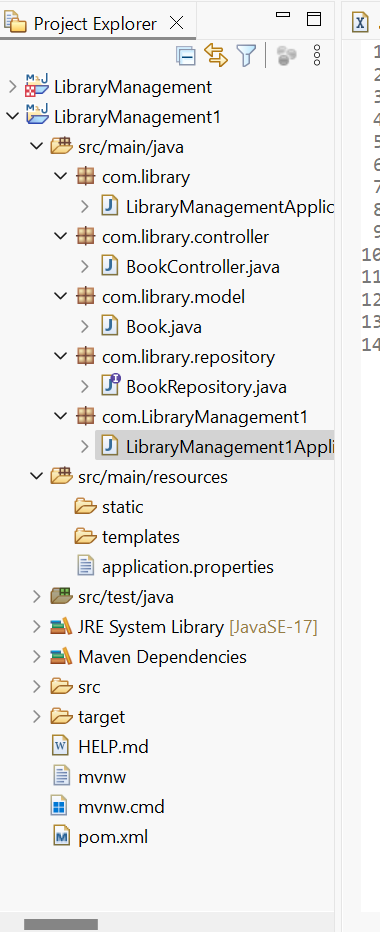
**Scenario:**

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**.
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.

**Spring instalizer:**

  
**Folder Structure:**

**LibraryManagementApplication:  
Code:  
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);

}

}

**BookController:  
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("/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**);

}

@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**;

}

@DeleteMapping("/{id}")

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

bookRepository.deleteById(id);

}

}

**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;

// Getters and 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:  
package** com.library.repository;

**import** com.library.model.Book;

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

**public** **interface** BookRepository **extends** JpaRepository<Book, Long> {

}

**Application.properties:**spring.application.name=LibraryManagement1

# H2 In-Memory DB Config

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

# Show SQL in console

spring.jpa.show-sql=true

# Auto create tables

spring.jpa.hibernate.ddl-auto=update

# H2 Console

spring.h2.console.enabled=true

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

**Output:**