**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.
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**

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0</version>

<name>LibraryManagement</name>

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

</dependencies>

</project>

**BookRepository.java**

package com.library.repository;

import java.util.List;

import java.util.Arrays;

public class BookRepository {

public List<String> getAllBooks() {

return Arrays.*asList*(

"Clean Code by Robert C. Martin",

"Effective Java by Joshua Bloch",

"Design Patterns by Erich Gamma"

);

}

}

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

import java.util.List;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayBooks() {

List<String> books = bookRepository.getAllBooks();

System.***out***.println("Available Books in the Library:");

int count = 1;

for (String book : books) {

System.***out***.println(count++ + ". " + book);

}

}

}

**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>

**MainApp.java**

package com.library.service;

import com.library.repository.BookRepository;

import java.util.List;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayBooks() {

List<String> books = bookRepository.getAllBooks();

System.***out***.println("Available Books in the Library:");

int count = 1;

for (String book : books) {

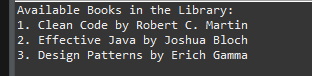
System.***out***.println(count++ + ". " + book);

}

}

}

**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:**
   * 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:**

**BookRepository.java**

package com.library.repository;

import java.util.List;

import java.util.Arrays;

public class BookRepository {

public List<String> getAllBooks() {

return Arrays.*asList*(

"Clean Code by Robert C. Martin",

"Effective Java by Joshua Bloch",

"Design Patterns by Erich Gamma"

);

}

}

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

import java.util.List;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayBooks() {

List<String> books = bookRepository.getAllBooks();

System.***out***.println("Available Books in the Library:");

int count = 1;

for (String book : books) {

System.***out***.println(count++ + ". " + book);

}

}

}

**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>

**LibraryManagementApplication.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

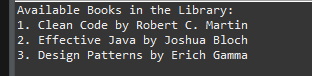
BookService bookService = (BookService) context.getBean("bookService");

bookService.displayBooks(); // Confirming DI works

}

}

**Output:**

****

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

**Scenario:**

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

**Steps:**

1. **Add Spring AOP Dependency:**
   * Update **pom.xml** to include Spring AOP dependency.
2. **Create an Aspect for Logging:**
   * Create a package **com.library.aspect** and add a class **LoggingAspect** with a method to log execution times.
3. **Enable AspectJ Support:**
   * Update **applicationContext.xml** to enable **AspectJ** support and register the aspect.
4. **Test the Aspect:**
   * Run the **LibraryManagementApplication** main class and observe the console for log messages indicating method execution times.

**Code:**

**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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0</version>

<name>LibraryManagement</name>

<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>

</project>

**LoggingAspect.java**

package com.library.aspect;

import org.aspectj.lang.ProceedingJoinPoint;

public class LoggingAspect {

public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {

long start = System.*currentTimeMillis*();

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

long end = System.*currentTimeMillis*();

System.***out***.println("[LOG] Execution time of " + joinPoint.getSignature().getName() +

"(): " + (end - start) + " ms");

return result;

}

}

**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">

<!-- AOP Support -->

<aop:config>

<aop:aspect id="loggingAspect" ref="aspectBean">

<aop:around method="logExecutionTime"

pointcut="execution(\* com.library.service.\*.\*(..))"/>

</aop:aspect>

</aop:config>

<!-- Aspect Bean -->

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

<!-- Regular Beans -->

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

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

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

</bean>

</beans>

**LibraryManagementApplication.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

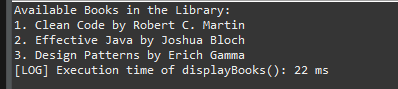
BookService bookService = (BookService) context.getBean("bookService");

bookService.displayBooks();

}

}

**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**

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0</version>

<name>LibraryManagement</name>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<!-- Spring Core + 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-aspects</artifactId>

<version>5.3.34</version>

</dependency>

<!-- Spring Web MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.34</version>

</dependency>

<!-- Optional: Servlet API for WebMVC -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>4.0.1</version>

<scope>provided</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Maven Compiler 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>

</build>

</project>

**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**

<?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 DI -->

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

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

</bean>

</beans>

**BookRepository.java**

package com.library.repository;

import java.util.Arrays;

import java.util.List;

public class BookRepository {

public List<String> getBooks() {

return Arrays.*asList*(

"Clean Code by Robert C. Martin",

"Effective Java by Joshua Bloch",

"Design Patterns by Erich Gamma",

"Refactoring by Martin Fowler",

"Java Concurrency in Practice by Brian Goetz"

);

}

}

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

import java.util.List;

public class BookService {

private BookRepository bookRepository;

// Setter for Dependency Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void showBooks() {

List<String> books = bookRepository.getBooks();

System.***out***.println("📚 Available Books in the Library:");

int i = 1;

for (String book : books) {

System.***out***.println(i++ + ". " + book);

}

}

}

**LibraryManagementApplication.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class LibraryManagementApplication{

public static void main(String[] args) {

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

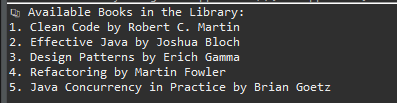
BookService service = (BookService) context.getBean("bookService");

service.showBooks();

}

}

**Output:**

****

**Exercise 6: Configuring Beans with Annotations**

**Scenario:**

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

**Steps:**

1. **Enable Component Scanning:**
   * Update **applicationContext.xml** to include component scanning for the **com.library** package.
2. **Annotate Classes:**
   * Use **@Service** annotation for the **BookService** class.
   * Use **@Repository** annotation for the **BookRepository** class.
3. **Test the Configuration:**
   * Run the **LibraryManagementApplication** main class to verify the annotation-based configuration.

**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"

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 component scanning -->

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

</beans>

**BookRepository.java**

package com.library.repository;

import java.util.Arrays;

import java.util.List;

import org.springframework.stereotype.Repository;

*@Repository* // Marks as a Spring-managed Repository bean

public class BookRepository {

public List<String> getBooks() {

return Arrays.*asList*(

"Clean Code by Robert C. Martin",

"Effective Java by Joshua Bloch",

"Design Patterns by Erich Gamma",

"Refactoring by Martin Fowler",

"Java Concurrency in Practice by Brian Goetz"

);

}

}

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

*@Service* // Marks as a Spring-managed Service bean

public class BookService {

private BookRepository bookRepository;

*@Autowired* // Automatically injects BookRepository bean

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void showBooks() {

List<String> books = bookRepository.getBooks();

System.***out***.println("📚 Available Books in the Library:");

int i = 1;

for (String book : books) {

System.***out***.println(i++ + ". " + book);

}

}

}

**LibraryManagementApplication.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class LibraryManagementApplication{

public static void main(String[] args) {

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

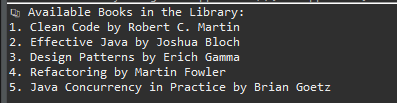
BookService service = (BookService) context.getBean("bookService");

service.showBooks();

}

}

**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 and setter injection -->

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

<!-- Constructor injection -->

<constructor-arg value="Library Book Listing Service" />

<!-- Setter injection -->

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

</bean>

</beans>

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

import java.util.List;

public class BookService {

private String serviceName; // Constructor injection

private BookRepository bookRepository; // Setter injection

// Constructor for constructor injection

public BookService(String serviceName) {

this.serviceName = serviceName;

}

// Setter for setter injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void showBooks() {

System.***out***.println("📘 " + serviceName);

List<String> books = bookRepository.getBooks();

int i = 1;

for (String book : books) {

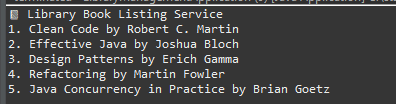
System.***out***.println(i++ + ". " + book);

}

}

}

**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.

**Steps:**

1. **Define an Aspect:**
   * Create a package **com.library.aspect** and add a class **LoggingAspect**.
2. **Create Advice Methods:**
   * Define advice methods in **LoggingAspect** for logging before and after method execution.
3. **Configure the Aspect:**
   * Update **applicationContext.xml** to register the aspect and enable **AspectJ** auto-proxying.
4. **Test the Aspect:**
   * Run the **LibraryManagementApplication** main class to verify the AOP functionality.

**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"

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 auto-proxy -->

<aop:config>

<aop:aspect id="loggingAspect" ref="loggingAspectBean">

<aop:pointcut id="bookServiceMethods"

expression="execution(\* com.library.service.BookService.\*(..))"/>

<aop:before pointcut-ref="bookServiceMethods" method="logBefore"/>

<aop:after pointcut-ref="bookServiceMethods" method="logAfter"/>

</aop:aspect>

</aop:config>

<!-- Aspect Bean -->

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

<!-- Repository Bean -->

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

<!-- Service Bean with constructor and setter injection -->

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

<constructor-arg value="Library Book Listing Service"/>

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

</bean>

</beans>

**LoggingAspect.java**

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

public class LoggingAspect {

public void logBefore(JoinPoint joinPoint) {

System.***out***.println("[AOP] Before method: " + joinPoint.getSignature().getName());

}

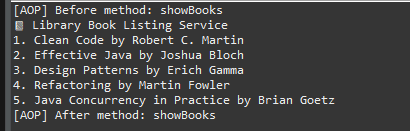
public void logAfter(JoinPoint joinPoint) {

System.***out***.println("[AOP] After method: " + joinPoint.getSignature().getName());

}

}

**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.

**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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>3.5.3</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.library</groupId>

<artifactId>LibraryManagementSystem</artifactId>

<version>1.0</version>

<name>LibraryManagementSystem</name>

<description>Demo project for Spring Boot</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>2.2.224</version> <!-- Or latest version -->

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>mysql</groupId>

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

<version>8.3.0</version> <!-- Or latest -->

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**application.properties**

spring.application.name=LibraryManagementSystem

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

spring.datasource.username=root

spring.datasource.password=your\_password

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

spring.jpa.hibernate.ddl-auto=update

spring.jpa.database-platform=org.hibernate.dialect.MySQL5Dialect

**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 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.java**

package com.library.repository;

import com.library.model.Book;

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

public interface BookRepository extends JpaRepository<Book, Long> {

}

**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;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@GetMapping("/{id}")

public Optional<Book> getBookById(@PathVariable Long id) {

return bookRepository.findById(id);

}

@PostMapping

public Book addBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@PutMapping("/{id}")

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

return bookRepository.findById(id).map(book -> {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

return bookRepository.save(book);

}).orElseGet(() -> {

updatedBook.setId(id);

return bookRepository.save(updatedBook);

});

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

LibraryManagementSystemApplication.java

package com.library;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LibraryManagementSystemApplication {

public static void main(String[] args) {

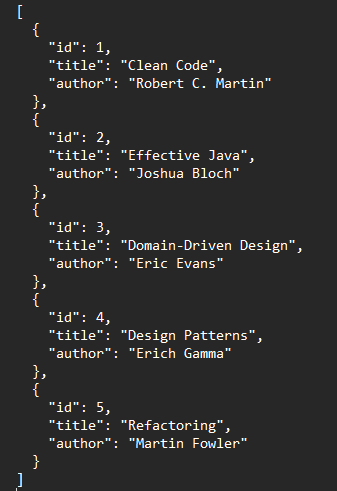
SpringApplication.run(LibraryManagementSystemApplication.class, args);

SpringApplication.run(LibraryManagementSystemApplication.class, new String[]{"--debug"});

}

}

**Output:**

****