**Week-3**

**Spring Core Maven**

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

**Scenario:**

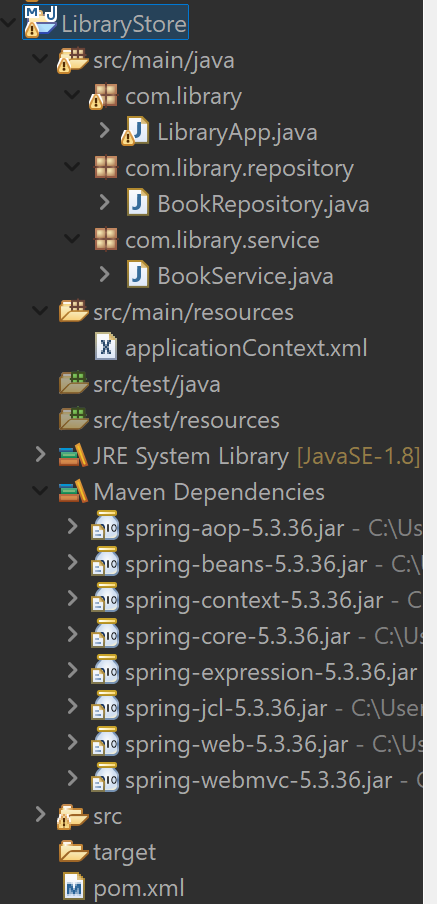
Your company is developing a web application for managing a library. You need to use the Spring Framework to handle the backend operations.

**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.

Spring is a lightweight Java framework used to build loosely coupled applications. In this exercise, you configure a basic Spring app using XML and define beans for services and repositories. It introduces the Spring IoC container and bean management.

**Folder Structure:**



**LibraryApp.java**

// LibraryApp.java

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryApp {

public static void main(String[] args) {

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

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

bookService.display();

}

}

**BookRepository.java**

// BookRepository.java

package com.library.repository;

public class BookRepository {

public void display() {

System.*out*.println("BookRepository is working.");

}

}

**BookService.java**

// BookService.java

package com.library.service;

public class BookService {

public void display() {

System.out.println("BookService is working.");

// No BookRepository used yet

}

}

**applicationContext.xml**

// 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/spring-beans.xsd">

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

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

</beans>

**pom.xml**

//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>

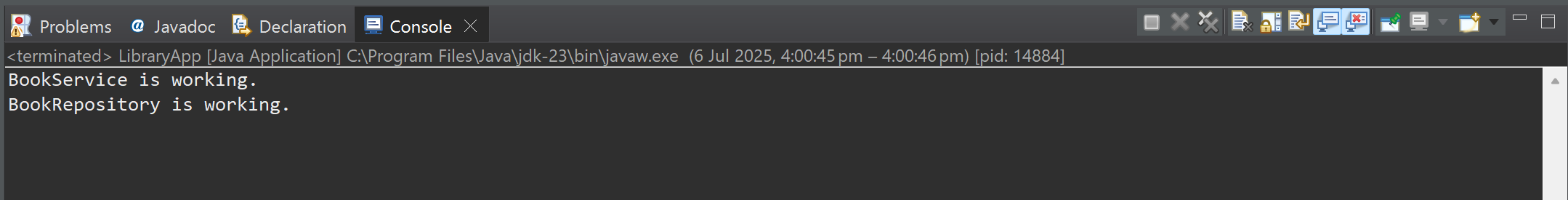
<groupId>com.library</groupId>

<artifactId>LibraryStore</artifactId>

<version>0.0.1-SNAPSHOT</version>

</project>

**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.

Dependency Injection (DI) is a design pattern where dependencies are provided externally instead of being created inside a class. Spring uses DI to wire objects together. In this task, you inject BookRepository into BookService using setter-based DI via XML.

**BookService.java**

// 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 display() {

System.out.println("BookService is working.");

bookRepository.display(); // Using the injected dependency

}

}

**applicationContext.xml**

// 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/spring-beans.xsd">

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

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

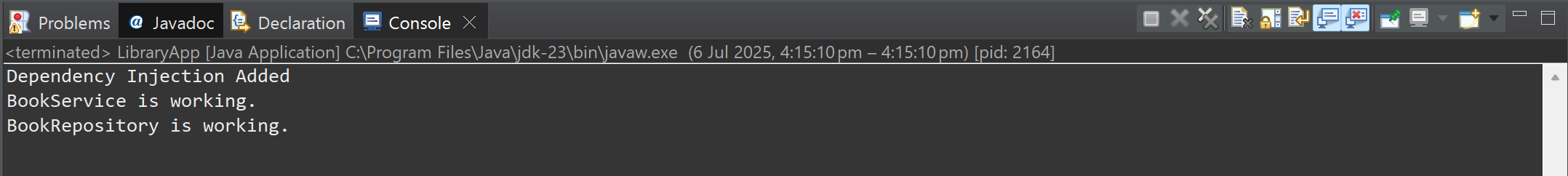
<!-- Setter injection -->

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

</bean>

</beans>

**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.

Maven is a build tool for managing project dependencies and lifecycle. Here, you create a Maven-based Spring project and add dependencies like Spring Core, AOP, and WebMVC. You also configure the compiler plugin to ensure Java 1.8 compatibility.

**pom.xml**

//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.library</groupId>

<artifactId>LibraryStore</artifactId>

<version>0.0.1-SNAPSHOT</version>

<properties>

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

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

</properties>

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.36</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.36</version>

</dependency>

<!-- Spring WebMVC (optional for future web features) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.36</version>

</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>

**Conclusion:**

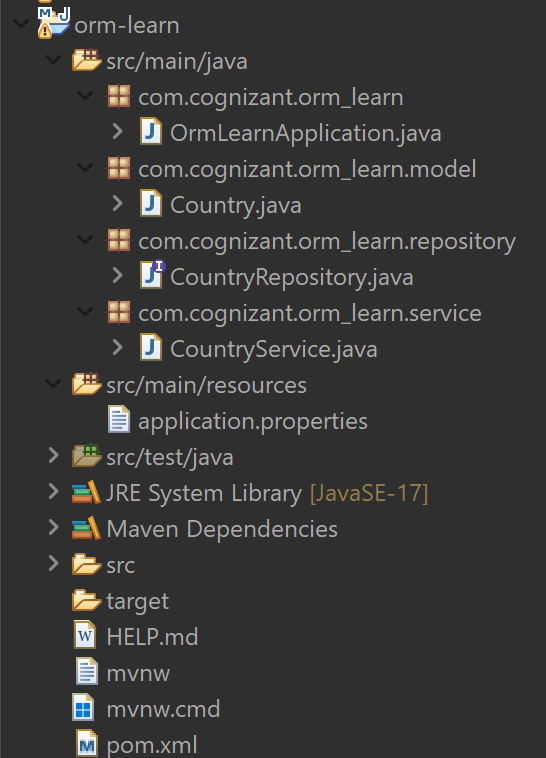
Through these exercises, we learned how to set up a Spring-based Java application using Maven. We explored how Spring's IoC container manages object creation and wiring through Dependency Injection, promoting loose coupling. By configuring beans and dependencies using XML, we gained a clear understanding of how services and repositories interact. Additionally, we prepared the project for scalability by adding essential Spring modules and Maven plugins.

**Spring Data JPA**

**Exercise 1: Spring Data JPA – Quick Example**

This exercise involves creating a Spring Boot application named orm-learn that integrates with a MySQL database using Spring Data JPA. It demonstrates how to configure application properties, map Java entities to database tables, and handle schema updates using Hibernate. The exercise also includes managing logging levels and resolving common startup issues like missing columns or port conflicts.

**Folder Structure:**



**application.properties**

//application.properties

#spring.application.name=orm-learn

# Logging configuration

logging.level.org.springframework=info

logging.level.com.cognizant=debug

logging.level.org.hibernate.SQL=trace

logging.level.org.hibernate.type.descriptor.sql=trace

logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger**{25}** %25M %4L %m%n

# Database config

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

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

spring.datasource.username=root

spring.datasource.password=Anasuya\_15

# Hibernate config

spring.jpa.hibernate.ddl-auto=update

#spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

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

server.port=8081

**Country.java**

//Country.java

package com.cognizant.orm\_learn.model;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "code")

private String code;

@Column(name = "name")

private String name;

// Getters & Setters

public String getCode() {

return code;

}

public void setCode(String code) {

this.code = code;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

**CountryRepository.java**

//CountryRepository.java

package com.cognizant.orm\_learn.repository;

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

import org.springframework.stereotype.Repository;

import com.cognizant.orm\_learn.model.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**CountryService.java**

//CountryService.java

package com.cognizant.orm\_learn.service;

import com.cognizant.orm\_learn.model.Country;

import com.cognizant.orm\_learn.repository.CountryRepository;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.List;

*@Service*

public class CountryService {

*@Autowired*

private CountryRepository countryRepository;

*@Transactional*

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**OrmLearnApplication.java**

//OrmLearnApplication.java

package com.cognizant.orm\_learn;

import com.cognizant.orm\_learn.model.Country;

import com.cognizant.orm\_learn.service.CountryService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import java.util.List;

@SpringBootApplication

public class OrmLearnApplication {

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

private static CountryService countryService;

public static void main(String[] args) {

ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);

LOGGER.info("Inside main");

countryService = context.getBean(CountryService.class);

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

List<Country> countries = countryService.getAllCountries();

LOGGER.debug("countries={}", countries);

LOGGER.info("End");

}

}

**pom.xml**

//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>4.0.0-SNAPSHOT</version>

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

</parent>

<groupId>com.cognizant</groupId>

<artifactId>orm-learn</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>orm-learn</name>

<description>Demo project for Spring Data JPA and Hibernate</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-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

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

<scope>runtime</scope>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

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

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

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

<repository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<releases>

<enabled>false</enabled>

</releases>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

<snapshots>

<enabled>false</enabled>

</snapshots>

</pluginRepository>

<pluginRepository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<releases>

<enabled>false</enabled>

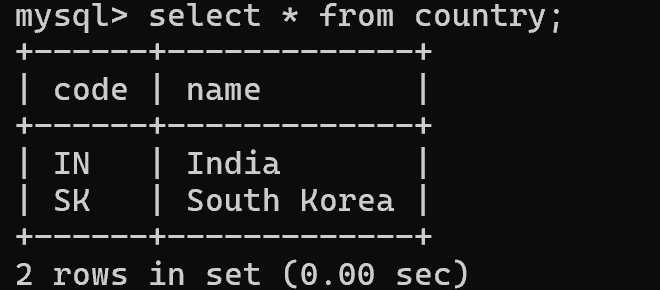
</releases>

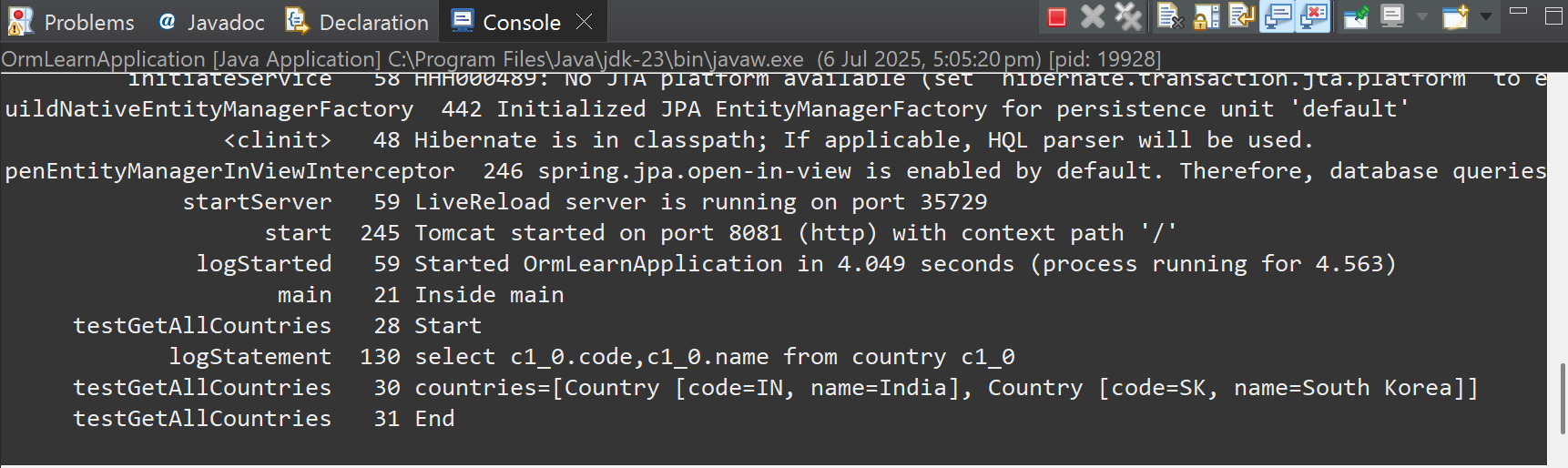
</pluginRepository>

</pluginRepositories>

</project>

**Output:**





**Exercise 2: Difference between JPA, Hibernate and Spring Data JPA**

**JPA (Java Persistence API)**

* A specification (interface) provided by Java for object-relational mapping (ORM).
* It Defines how Java objects interact with relational databases using annotations and APIs.
* It is not an implementation; it requires a provider like Hibernate to work.

**Hibernate**

* A popular implementation of JPA (also a standalone ORM framework).
* This Provides the actual functionality to map Java classes to database tables, manage sessions, handle queries, caching, etc.
* Hibernate can work with or without JPA, but it typically implements JPA standards.

**Spring Data JPA**

* A part of the Spring ecosystem that builds on top of JPA.
* This Simplifies data access using repository interfaces. Automatically provides basic CRUD operations without boilerplate code.
* It Internally uses JPA (and often Hibernate) to perform database operations.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| **Type** | Specification (Interface) | ORM Framework | Spring Module |
| **Purpose** | Define ORM standards | Implements ORM logic | Simplifies JPA usage in Spring |
| **Provided by** | Java EE / Jakarta EE | Red Hat | Spring Framework |
| **Requires implementation** | Yes | Implements JPA | Uses JPA provider (like Hibernate) |
| **Ease of Use** | Medium | Medium | High (uses repositories) |

**Conclusion:**

Through these exercises, we understood how to integrate Spring Data JPA with MySQL using Hibernate as the JPA provider. We learned how to configure the application properties, map entities to database tables, and resolve schema validation issues. The theory questions deepened our understanding of the roles and differences between JPA, Hibernate, and Spring Data JPA, which are essential for building robust and maintainable database-backed applications using the Spring ecosystem.

**-- THE END --**