**DN 4.0 WEEK 3 MANDATORY PROBLEMS**

**MODULE 6: 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.

**PROGRAM:**

**LIBRARYAPP.JAVA**

//DEPS org.springframework:spring-context:5.3.27

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.GenericXmlApplicationContext;

public class LibraryApp {

public static void main(String[] args) {

ApplicationContext context = new GenericXmlApplicationContext("file:applicationContext.xml");

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

bookService.addBook("Spring app in one file");

}

}

class BookService {

private BookRepository bookRepository;

// Spring will inject this dependency

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

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

bookRepository.saveBook(bookName);

}

}

class BookRepository {

public void saveBook(String bookName) {

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

}

}

**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="BookRepository" />

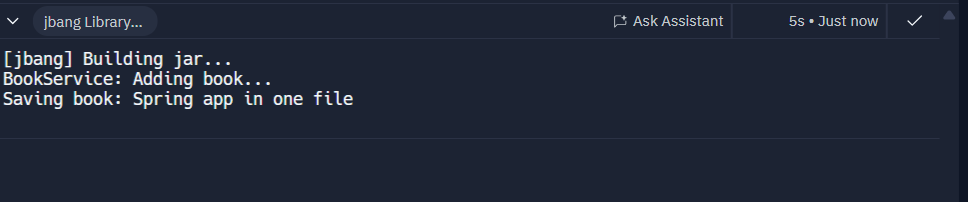
<bean id="bookService" class="BookService">

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

</bean>

</beans>

**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.

**PROGRAM:**

**LIBRARYMANAGEMENTAPPLICATION.JAVA**

//DEPS org.springframework:spring-context:5.3.27

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.GenericXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

ApplicationContext context = new GenericXmlApplicationContext("file:applicationContext.xml");

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

bookService.addBook("Dependency Injection Example");

}

}

class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

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

bookRepository.saveBook(bookName);

}

}

class BookRepository {

public void saveBook(String bookName) {

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

}

}

**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">

<!-- Define the repository bean -->

<bean id="bookRepository" class="BookRepository" />

<!-- Define the service bean and inject bookRepository -->

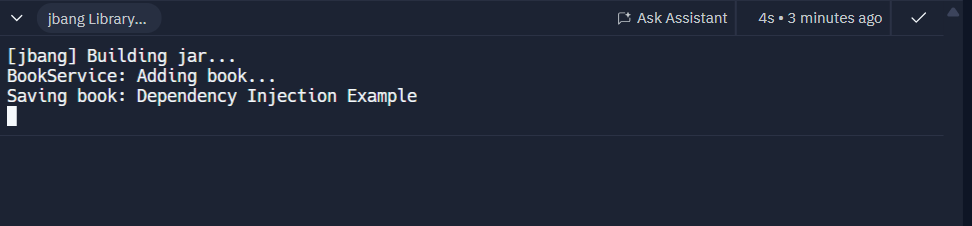
<bean id="bookService" class="BookService">

<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.

**PROGRAM:**

**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>library</groupId>

<artifactId>LibraryApp</artifactId>

<version>1.0-SNAPSHOT</version>

<build>

<!-- Tell Maven to use current directory for source files -->

<sourceDirectory>.</sourceDirectory>

<plugins>

<!-- Compiler Plugin for Java 8 -->

<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>

<!-- Exec Plugin to run Main class -->

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>3.1.0</version>

<configuration>

<mainClass>Main</mainClass>

</configuration>

</plugin>

</plugins>

</build>

<dependencies>

<!-- Spring Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.27</version>

</dependency>

</dependencies>

</project>

**BOOKREPOSITORY.JAVA**

public class BookRepository {

public void saveBook(String bookName) {

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

}

}

**BOOKSERVICE.JAVA**

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

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

bookRepository.saveBook(bookName);

}

}

**MAIN.JAVA**

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.GenericXmlApplicationContext;

public class Main {

public static void main(String[] args) {

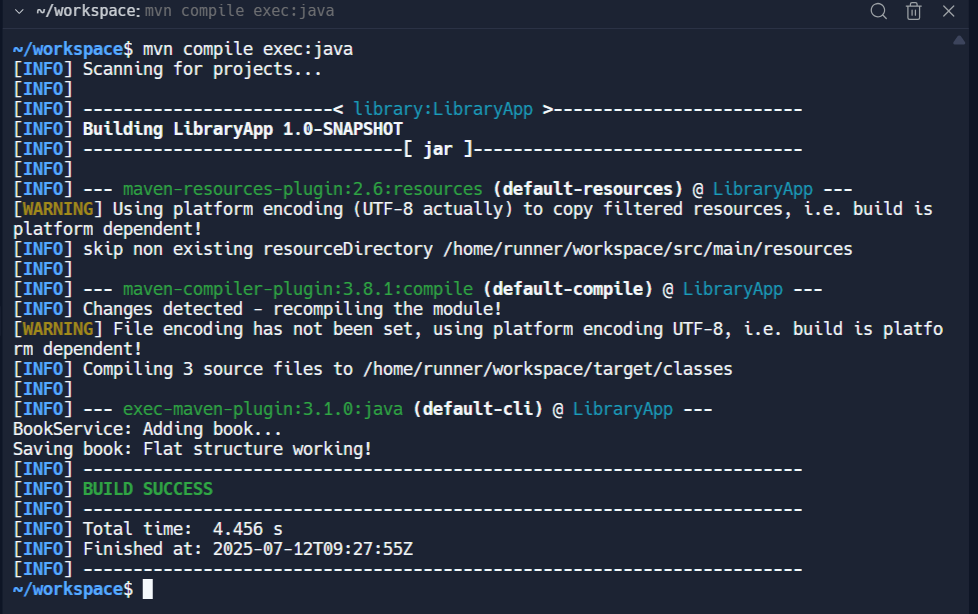
ApplicationContext context = new GenericXmlApplicationContext("file:applicationContext.xml");

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

service.addBook("Flat structure working!");

}

}



Here I created a src folder java folder and main where the files are located in the path

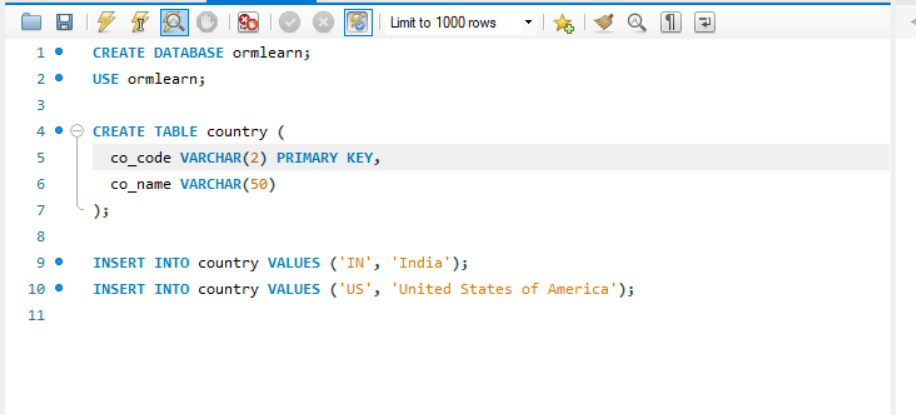
Src/main/java.

**MODULE 7: SPRING CORE AND MAVEN**

1. **Spring Data JPA - Quick Example**

* **Group:** com.cognizant
* **Artifact:** orm-learn
* **Description:** Demo project for Spring Data JPA and Hibernate
* **Packaging:** Jar
* **Java Version:** 17 (recommended)
* **Dependencies:**
  + Spring Boot DevTools
  + Spring Data JPA
  + MySQL Driver

MySQL Schema in Workbench:



**PROGRAM**

**Application.properties:**

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

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=suriya@345

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

**Country.java in model package:**

package com.cognizant.ormlearn.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;

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 inside repository package:**

package com.cognizant.ormlearn.repository;

import com.cognizant.ormlearn.model.Country;

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

import org.springframework.stereotype.Repository;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**CountryService.java inside package:**

package com.cognizant.ormlearn.service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

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

import org.springframework.stereotype.Service;

import jakarta.transaction.Transactional;

import java.util.List;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

**}**

**OrmLearnApplication.java:**

package com.cognizant.ormlearn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class OrmLearnApplication {

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

}

}

**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.cognizant</groupId>

<artifactId>orm-learn</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<name>orm-learn</name>

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

<parent>

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

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

<version>2.5.5</version>

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

</parent>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<!-- Spring Boot -->

<dependency>

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

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

</dependency>

<!-- Spring Data JPA -->

<dependency>

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

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

</dependency>

<!-- MySQL Driver -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Logging (Optional) -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

</dependency>

<!-- Spring Boot DevTools (optional for hot reload) -->

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Testing (optional) -->

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Spring Boot Maven Plugin -->

<plugin>

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

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

<configuration>

<mainClass>com.cognizant.ormlearn.OrmLearnApplication</mainClass>

</configuration>

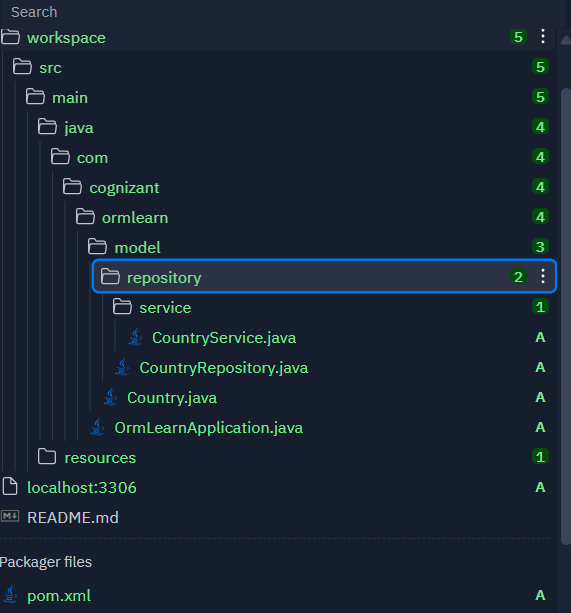
</plugin>

</plugins>

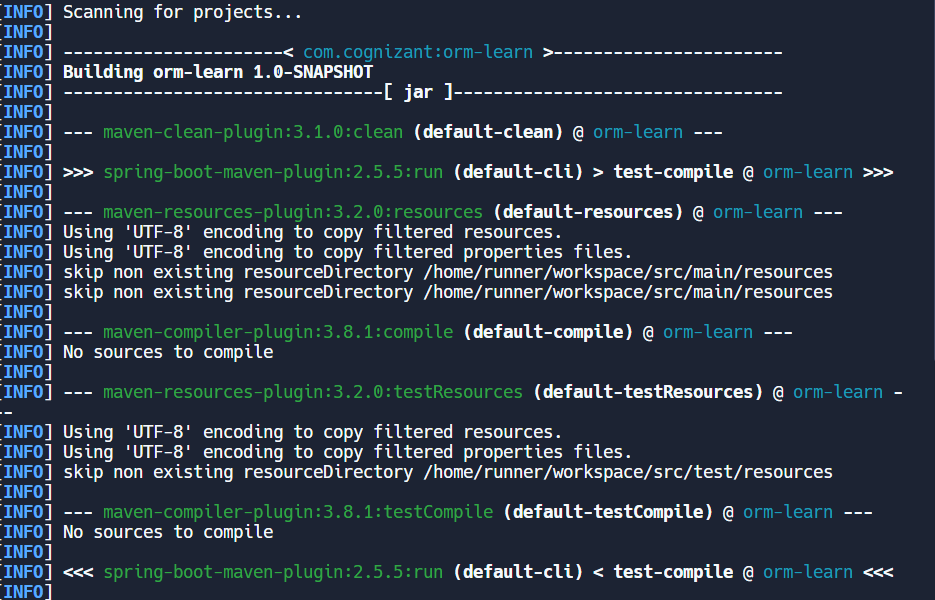
</build>

</project>

**FILE STRUCTURE**

****

**OUTPUT:**

****

**Difference between JPA, Hibernate and Spring Data JPA**   
  
Java Persistence API (JPA)

* JSR 338 Specification for persisting, reading and managing data from Java objects
* Does not contain concrete implementation of the specification
* Hibernate is one of the implementation of JPA

**JPA (Java Persistence API)**

* It is a specification, not an implementation.
* Provides standard annotations and interfaces for ORM
* Used to persist, read, and manage data between Java objects and a relational database.
* Requires an implementation like Hibernate or EclipseLink.
* Part of Java EE / Jakarta EE.
* Example: You use EntityManager to manage persistence operations.

**Hibernate**

* It is an ORM framework and a JPA implementation.
* Provides the actual working code behind the JPA interfaces.
* Can be used with or without JPA.
* Adds extra features not present in JPA (like caching, dirty checking, batch fetching).
* Uses its own API if used directly.
* Most commonly used JPA provider.

**Spring Data JPA**

* It is a Spring project that simplifies JPA usage.
* Provides a higher-level abstraction over JPA (usually with Hibernate underneath).
* Eliminates boilerplate code for DAO/Repository layers.
* You just define an interface, and Spring generates implementations automatically.
* Supports method name-based query generation (findByName, findByAgeGreaterThan).
* Requires a JPA provider like Hibernate under the hood.

**PROGRAM**

**EMPLOYEE.JAVA**

package com.examPle.demo.entity;

import javax.persistence.\*;

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private String name;

// Constructors

public Employee() {}

public Employee(String name) {

this.name = name;

}

// Getters and Setters

public Integer getId() { return id; }

public void setId(Integer id) { this.id = id; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

}

**EMPLOYEEREPOSITORY.JAVA**

package com.example.demo.repository;

import com.example.demo.entity.Employee;

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

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EMPLOYEESERVICE.JAVA**

package com.example.demo.service;

import com.example.demo.entity.Employee;

import com.example.demo.repository.EmployeeRepository;

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

import org.springframework.stereotype.Service;

import javax.transaction.Transactional;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

**SPRINGDATAJPADEMOAPPLICATION**

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringDataJpaDemoApplication {

public static void main(String[] args) {

SpringApplication.run(SpringDataJpaDemoApplication.class, args);

}

}

**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>

<!-- ✅ Spring Boot Parent to manage dependency versions -->

<parent>

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

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

<version>2.7.5</version>

<relativePath/> <!-- Required for Replit -->

</parent>

<groupId>com.example</groupId>

<artifactId>spring-data-jpa-demo</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<name>Spring Data JPA Demo</name>

<properties>

<java.version>11</java.version>

</properties>

<dependencies>

<!-- Core Spring Boot -->

<dependency>

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

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

</dependency>

<!-- Spring Data JPA -->

<dependency>

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

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

</dependency>

<!-- H2 in-memory DB -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Optional: Testing -->

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Compiler Plugin -->

<plugin>

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

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

<version>3.8.1</version>

<configuration>

<source>${java.version}</source>

<target>${java.version}</target>

</configuration>

</plugin>

<!-- Spring Boot Plugin -->

<plugin>

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

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

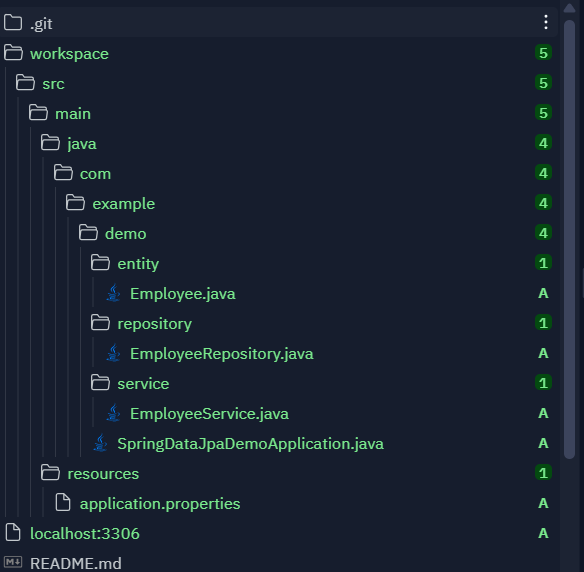
</plugin>

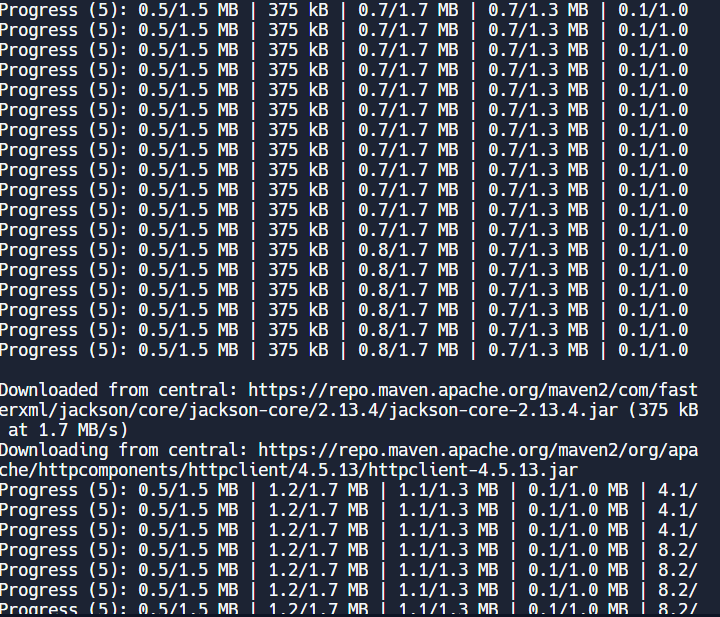
</plugins>

</build>

</project>

**FOLDER STRUCTURE:**

****

****

****