**Week 3**

**Spring Data JPA with Spring Boot, Hibernate**

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

**Spring Data JPA** is a part of the larger Spring Data family that simplifies the implementation of data access layers for applications using the **Java Persistence API (JPA)**. It eliminates the need for boilerplate code by allowing developers to define repository interfaces, which Spring will automatically implement at runtime. By extending interfaces like JpaRepository, developers can perform standard CRUD operations, pagination, and sorting without writing a single query. Spring Data JPA also supports custom queries using method naming conventions or annotations like @Query. It internally uses JPA providers like **Hibernate** to interact with the database. This makes it ideal for building fast, clean, and maintainable data access layers, especially in **Spring Boot** applications.

**Code 1: LibraryManagemnetApllication**

package com.example.library;

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 service = context.getBean(BookService.class);

service.display();

}

}

**Code 2: SpringDataJpaDemoApplication .java**

package com.example.springdatajpademo;

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);

}

}

**Code 3: User.java**

package com.example.springdatajpademo;

import jakarta.persistence.\*;

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

public Long getId() { return id; }

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

public String getName() { return name; }

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

public String getEmail() { return email; }

public void setEmail(String email) { this.email = email; }

}

**Code 4: UserController.java**

package com.example.springdatajpademo;

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

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/users")

public class UserController {

@Autowired

private UserService service;

@GetMapping

public List<User> getUsers() {

return service.getAllUsers();

}

@PostMapping

public User createUser(@RequestBody User user) {

return service.addUser(user);

}

}

**Code 5: UserRepository.java**

package com.example.springdatajpademo;

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

public interface UserRepository extends JpaRepository<User, Long> {

User findByEmail(String email);

}

**Code 6: UserService.java**

package com.example.springdatajpademo;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class UserService {

@Autowired

private UserRepository repo;

public List<User> getAllUsers() {

return repo.findAll();

}

public User addUser(User user) {

return repo.save(user);

}

}

**Code 7: 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.example</groupId>

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

<version>1.0.0</version>

<parent>

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

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

<version>3.2.0</version>

</parent>

<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>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

<version>3.1.0</version>

<configuration>

<mainClass>com.example.library.LibraryManagementApplication</mainClass>

</configuration>

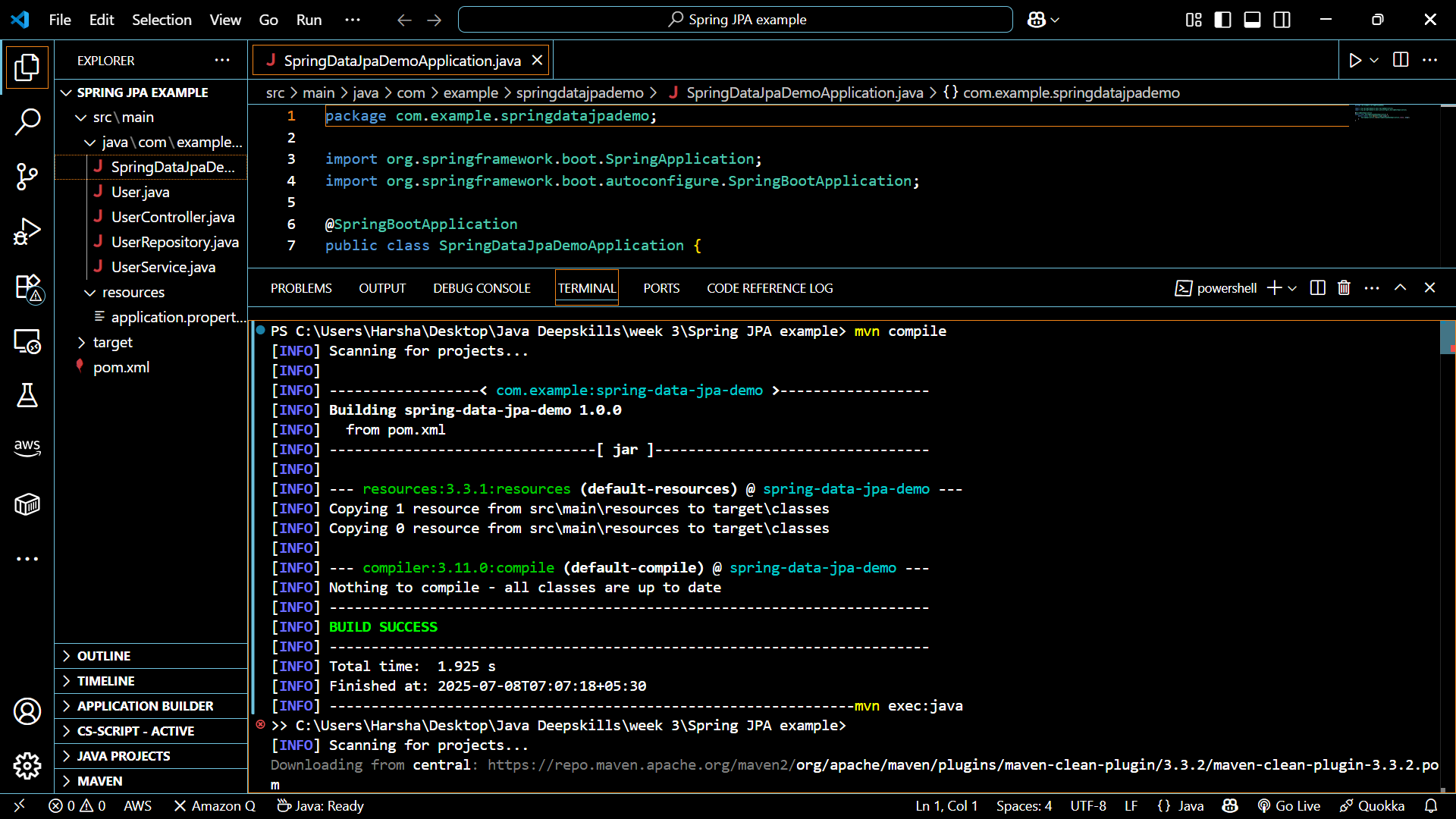
</plugin>

</plugins>

</build>

</project>

**Output:**

****

1. **Difference between JPA, Hibernate and Spring Data JPA**

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **What it is** | A **specification** (interface) | An **implementation** of JPA | A **framework** that builds on JPA and Hibernate |
| **Type** | Interface | ORM tool | Abstraction layer over JPA + Spring |
| **Query Support** | JPQL (Java Persistence Query Language) | JPQL + Native SQL + Criteria API | JPQL + Derived Queries + @Query annotations |
| **Transaction Handling** | Managed manually or via Spring | Managed via code or Spring | Integrated with Spring’s @Transactional support |
| **Ease of Use** | Moderate | Complex setup | Very easy (best suited for Spring Boot projects) |
| **Who Uses It** | Developers using JPA standard | Developers preferring fine-grained ORM control | Spring Boot/Spring Framework developers needing rapid dev |