**Assessment**

1. **Configuring a Basic Spring Application**

**Step 1: we can start by manually via maven server and add the following code to pom.xml:**

<dependencies>

<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>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

**Step 2:Now write a basic Spring Application Code-**

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

**Step 3: Now run the application-**

./mvnw spring-boot:run

**Step 4: Test the Endpoint**

**Open Postman or Browser-**

<http://localhost:8080/hello>

**Output for this will be-**

Hello, Spring Boot!

1. **Implementing Dependency Injection**

**Step 1- Create a Service Interface and Implementation**

package com.example.demo.service;

import org.springframework.stereotype.Service;

@Service

public class GreetingServiceImpl implements GreetingService {

@Override

public String greet() {

return "Hello from GreetingService!";

}

}

**Step 2- Inject Service into Controller**

package com.example.demo.controller;

import com.example.demo.service.GreetingService;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class GreetingController {

private final GreetingService greetingService;

// Constructor Injection

public GreetingController(GreetingService greetingService) {

this.greetingService = greetingService;

}

@GetMapping("/greet")

public String greet() {

return greetingService.greet();

}

}

**After running this code you get Output like-**

Hello from GreetingService!

1. **Creating and Configuring a Maven Project-**

**Step 1- Create Maven Project Using Terminal**

mvn archetype:generate -DgroupId=com.example.app -DartifactId=my-maven-project -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

**Step 2- Understand the Folder Structure**

my-maven-project/

├── pom.xml

├── src

│ ├── main

│ │ └── java

│ │ └── com

│ │ └── example

│ │ └── app

│ │ └── App.java

│ └── test

│ └── java

│ └── com

│ └── example

│ └── app

│ └── AppTest.java

**Step 3- Edit 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.app</groupId>

<artifactId>my-maven-project</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- Example: JUnit 5 -->

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

<version>3.10.1</version>

<configuration>

<source>17</source> <!-- Java version -->

<target>17</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Step 4- Compile and Run**

mvn clean install

mvn exec:java -Dexec.mainClass="com.example.app.App"

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

**Step 1- Prerequisites**

1. **Java 17+**
2. **Spring Boot**
3. **MySQL,H2 or PostgreSQL database**

**Step 2-Maven Dependencies(pom.xml)**

**<dependencies>**

**<!-- Spring Web -->**

**<dependency>**

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

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

**</dependency>**

**<!-- Spring Data JPA -->**

**<dependency>**

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

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

**</dependency>**

**<!-- H2 Database (for quick demo) -->**

**<dependency>**

**<groupId>com.h2database</groupId>**

**<artifactId>h2</artifactId>**

**<scope>runtime</scope>**

**</dependency>**

**</dependencies>**

**Step 3-Configuration in application.properties**

**spring.datasource.url=jdbc:h2:mem:testdb**

**spring.datasource.driverClassName=org.h2.Driver**

**spring.datasource.username=sa**

**spring.datasource.password=**

**spring.jpa.database-platform=org.hibernate.dialect.H2Dialect**

**spring.jpa.hibernate.ddl-auto=update**

**Step 4- Define Entity Class**

**import jakarta.persistence.Entity;**

**import jakarta.persistence.GeneratedValue;**

**import jakarta.persistence.GenerationType;**

**import jakarta.persistence.Id;**

**@Entity**

**public class Product {**

**@Id**

**@GeneratedValue(strategy = GenerationType.IDENTITY)**

**private Long id;**

**private String name;**

**private double price;**

**// Getters and Setters**

**}**

**Step 5- Create Repository Interface**

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

**public interface ProductRepository extends JpaRepository<Product, Long> {**

**}**

**Step 6-Write Controller or Service Layer**

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

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

**import java.util.List;**

**@RestController**

**@RequestMapping("/products")**

**public class ProductController {**

**@Autowired**

**private ProductRepository productRepository;**

**@GetMapping**

**public List<Product> getAllProducts() {**

**return productRepository.findAll();**

**}**

**@PostMapping**

**public Product addProduct(@RequestBody Product product) {**

**return productRepository.save(product);**

**}**

**}**

**Step 7- Run Application**

**@SpringBootApplication**

**public class DemoApplication {**

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

**SpringApplication.run(DemoApplication.class, args);**

**}**

**}**

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

| **Feature** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification/Standard | Implementation (Framework) | Framework built on top of JPA + Spring |
| **Main Purpose** | Defines ORM rules | Implements ORM and JPA | Simplifies data access with repositories |
| **Provided By** | Java EE (Jakarta EE) | Hibernate ORM project | Spring Framework |
| **Who Writes It?** | Oracle (originally), Jakarta EE | Red Hat (Hibernate team) | Spring Team |
| **Query Language** | JPQL | HQL (Hibernate Query Language) | Derived queries, JPQL, Native SQL |
| **Boilerplate Code** | Requires manual setup | Requires configuration | Reduces boilerplate using interfaces |
| **Setup Complexity** | Medium | Medium | Easiest |
| **Example Annotation** | @Entity, @Id | @Entity, @Id | Same as JPA + @Repository |