**Difference Between JPA, Hibernate, and Spring Data JPA**

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification/Standard (javax.persistence) | Implementation (ORM framework) | Abstraction Layer (Spring project) |
| Purpose | Defines API for object-relational mapping | Provides actual implementation of JPA and advanced ORM features | Simplifies JPA usage with less boilerplate code |
| Developer | Oracle (Java EE / Jakarta EE) | Red Hat | Spring Team (Pivotal) |
| Usage | Interface layer to ORM | Concrete ORM logic and extensions | Auto-implements Repository interfaces |
| Boilerplate | High (you write a lot of DAO code) | Medium (some helper features, but still manual) | Low (Spring auto-generates DAO methods) |
| Annotation Source | javax.persistence.\* | Uses JPA + some org.hibernate.\* | Uses JPA under the hood |
| Integration | Can use any JPA provider | Mostly Hibernate is used as provider | Uses JPA (usually backed by Hibernate) |

**Spring Data JPA – Quick Working Example**

**pom.xml (Spring Boot + JPA + H2)**

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

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

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

</dependency>

</dependencies>

**User.java (Entity class)**

package com.example.demo.entity;

import jakarta.persistence.\*;

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

public User() {}

public User(String name, String email) {

this.name = name;

this.email = email;

}

}

**UserRepository.java**

package com.example.demo.repository;

import com.example.demo.entity.User;

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

public interface UserRepository extends JpaRepository<User, Long> {

// Spring auto-generates CRUD methods

}

**UserController.java**

package com.example.demo.controller;

import com.example.demo.entity.User;

import com.example.demo.repository.UserRepository;

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

import java.util.List;

@RestController

@RequestMapping("/users")

public class UserController {

private final UserRepository repository;

public UserController(UserRepository repository) {

this.repository = repository;

}

@PostMapping

public User saveUser(@RequestBody User user) {

return repository.save(user);

}

@GetMapping

public List<User> getAllUsers() {

return repository.findAll();

}

}

**application.properties**

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

**DemoApplication.java**

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);

}

}

**Test the API:**

**POST**: http://localhost:8080/users  
Body:

{

"name": "Jaisurya",

"email": "jaisurya@example.com"

}

**GET**: http://localhost:8080/users  
Response:

[

{

"id": 1,

"name": "Jaisurya",

"email": "jaisurya@example.com"

}

]