**Spring Data JPA vs Hibernate**

**1. Overview**

This document compares **Spring Data JPA** and **Hibernate** using practical code examples for each. Both technologies are used for data persistence in Java applications, but they serve slightly different purposes.

**2. Spring Data JPA Example**

**Configuration:**

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

spring.datasource.username=root

spring.datasource.password=yourpassword

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

**Entity Class:**

package com.example.jpademo.entity;

import jakarta.persistence.\*;

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private String name;

private String department;

}

**Repository Interface:**

package com.example.jpademo.repository;

import com.example.jpademo.entity.Employee;

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

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {}

**Service Layer:**

package com.example.jpademo.service;

import com.example.jpademo.entity.Employee;

import com.example.jpademo.repository.EmployeeRepository;

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

import org.springframework.stereotype.Service;

import jakarta.transaction.Transactional;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public Employee addEmployee(Employee employee) {

return employeeRepository.save(employee);

}

}

**Controller:**

package com.example.jpademo.controller;

import com.example.jpademo.entity.Employee;

import com.example.jpademo.service.EmployeeService;

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

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

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

@PostMapping

public Employee createEmployee(@RequestBody Employee employee) {

return employeeService.addEmployee(employee);

}

}

**3. Hibernate Example**

**pom.xml Dependencies (summary):**

<dependencies>

<!-- Hibernate Core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.6.15.Final</version>

</dependency>

<!-- MySQL Connector -->

<dependency>

<groupId>mysql</groupId>

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

<version>8.0.33</version>

</dependency>

<!-- JPA API -->

<dependency>

<groupId>jakarta.persistence</groupId>

<artifactId>jakarta.persistence-api</artifactId>

<version>2.2.3</version>

</dependency>

<!-- Logging -->

<dependency>

<groupId>org.jboss.logging</groupId>

<artifactId>jboss-logging</artifactId>

<version>3.4.3.Final</version>

</dependency>

</dependencies>

**Entity Class:**

package com.example.entity;

import jakarta.persistence.\*;

@Entity

@Table(name = "employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private String name;

private String department;

public Employee() {}

public Employee(String name, String department) {

this.name = name;

this.department = department;

}

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; }

public String getDepartment() { return department; }

public void setDepartment(String department) { this.department = department; }

}

**hibernate.cfg.xml:**

<hibernate-configuration>

<session-factory>

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/jpadb</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">yourpassword</property>

<property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>

<property name="hibernate.show\_sql">true</property>

<property name="hibernate.hbm2ddl.auto">update</property>

<mapping class="com.example.entity.Employee"/>

</session-factory>

</hibernate-configuration>

**HibernateUtil.java:**

package com.example.util;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

public class HibernateUtil {

private static final SessionFactory sessionFactory;

static {

try {

sessionFactory = new Configuration()

.configure("hibernate.cfg.xml")

.buildSessionFactory();

} catch (Throwable ex) {

throw new ExceptionInInitializerError("Initial SessionFactory creation failed " + ex);

}

}

public static SessionFactory getSessionFactory() {

return sessionFactory;

}

}

**App.java:**

package com.example;

import com.example.entity.Employee;

import com.example.util.HibernateUtil;

import org.hibernate.Session;

import org.hibernate.Transaction;

public class App {

public static void main(String[] args) {

Employee emp = new Employee("Sriram", "ECE");

Session session = HibernateUtil.getSessionFactory().openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

session.save(emp);

tx.commit();

System.out.println("Employee saved with ID: " + emp.getId());

} catch (Exception e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

}

}

**4. Comparison**

* **Hibernate** is an Object Relational Mapping (ORM) tool that implements the **JPA** specification.
* **Spring Data JPA** is a framework built on top of JPA and Hibernate that simplifies data access by reducing boilerplate code.
* With **Spring Data JPA**, you don’t have to write SQL or HQL for basic operations. You just define repository interfaces.
* **Hibernate** provides more flexibility and control but requires more code for session and transaction management.
* **Spring Data JPA** uses Hibernate internally by default, so using JPA doesn’t mean Hibernate is excluded.