Spring Data JPA with Spring Boot, Hibernate

**Exercise 1. Spring Data JPA - Quick Example  
  
Step 1) : Generate the Project from Spring Initializer  
  
1. Go to** [**https://start.spring.io**](https://start.spring.io) **2.** Fill the fields:

**Group:** com.cognizant

**Artifact:** orm-learn

**Description:** Demo project for Spring Data JPA and Hibernate

Click **Add Dependencies**, select:

1. Spring Boot DevTools
2. Spring Data JPA
3. MySQL Driver

**Step 2) : Extract and Import the Project**

1. Extract the ZIP file to a **folder inside your Eclipse Workspace**.

2. In Eclipse:

* File > Import > Maven > Existing Maven Projects
* Browse to the **extracted folder**.
* Click **Finish**.

**Step 3) : Create MySQL Schema**

In MySQL Workbench or terminal:

CREATE SCHEMA ormlearn;

**Step 4) : Configure 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=root

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

**Step 5) : Build the Project via Maven**mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456

**Step 6) : Create “country” Table in MySql**

CREATE TABLE country (

co\_code VARCHAR(2) PRIMARY KEY,

co\_name VARCHAR(50)

);

INSERT INTO country VALUES ('IN', 'India');

INSERT INTO country VALUES ('US', 'United States of America');

**Step 7) : Create necessary Classes  
  
Country.java**

**Code :**

package com.cognizant.ormlearn.model;

import javax.persistence.\*;

@Entity

@Table(name="country")

public class Country {

@Id

@Column(name="co\_code")

private String code;

@Column(name="co\_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**

**Code :**

package com.cognizant.ormlearn.repository;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**CountryService.java**

**Code :**

package com.cognizant.ormlearn.service;

import java.util.List;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**OrmLearnApplication.java(Update)**

**Code :**

package com.cognizant.ormlearn;

import java.util.List;

@SpringBootApplication

public class OrmLearnApplication {

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

private static CountryService countryService;

public static void main(String[] args) {

ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);

countryService = context.getBean(CountryService.class);

LOGGER.info("Inside main");

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

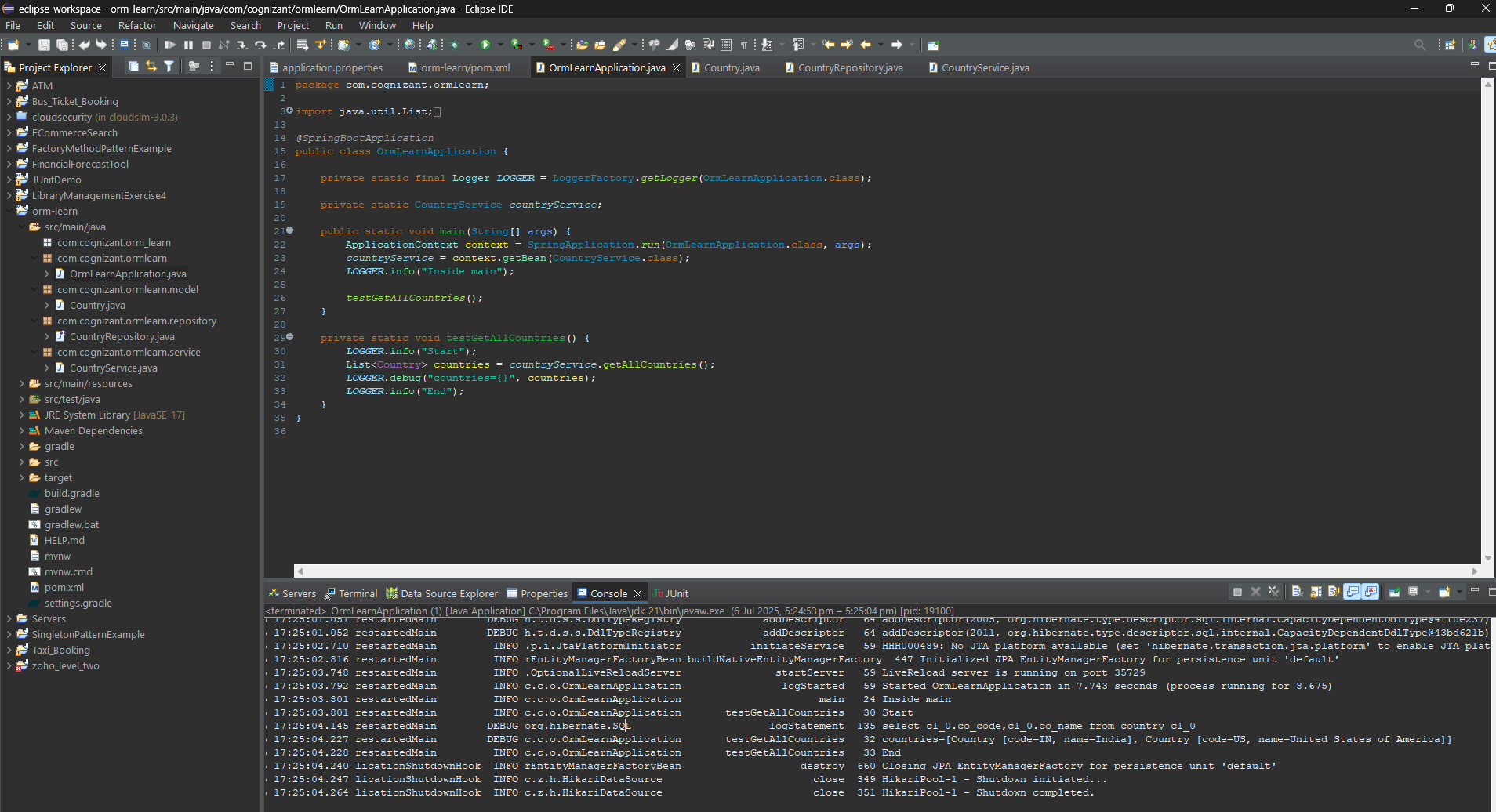
List<Country> countries = countryService.getAllCountries();

LOGGER.debug("countries={}", countries);

LOGGER.info("End");

}

}

**Output :  
  
  
**

**Exercise 1. Difference between JPA, Hibernate and Spring Data JPA  
  
  
Java Persistence API (JPA)**

* **What it is**:  
  A **specification** (JSR 338) that defines how to map Java objects to relational database tables.
* **Key point**:  
  It **does not** provide any implementation itself.
* **Analogy**:  
  Think of JPA like an **interface or a contract** describing what should be done but not how.
* **Example**:  
  You write code using @Entity, EntityManager, @Query annotations, but you still need an implementation like Hibernate underneath.

**Hibernate**

* **What it is**:  
  A **popular ORM framework** that implements the JPA specification.
* **Key point**:  
  Hibernate can be used **with or without JPA**.
* **Example (manual code)**:

java

CopyEdit

Session session = factory.openSession();

Transaction tx = session.beginTransaction();

session.save(employee);

tx.commit();

session.close();

* **Features beyond JPA**:
* Caching
* Dirty checking
* Custom query language (HQL)

## ****Spring Data JPA****

* **What it is**:  
  A **Spring project that builds on JPA**, making it much easier to work with.
* **Key point**:
  + **Does not implement JPA itself**, but **uses Hibernate or another JPA provider under the hood**.
  + Adds **powerful abstractions** to reduce boilerplate.
* **Benefits**:
  + You don’t write DAOs manually.
  + CRUD operations are auto-implemented.
  + Paging, sorting, query methods (findByName) are generated.
  + Integration with Spring’s @Transactional management.

**Quick Comparison Table**

| **Aspect** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification | Implementation of JPA | Abstraction over JPA implementations |
| **Provides Implementation?** | ❌ No | ✅ Yes | ❌ No |
| **Needs separate provider?** | ✅ Yes (e.g., Hibernate) | ❌ No (self-contained) | ✅ Yes (uses Hibernate or EclipseLink) |
| **Boilerplate Code** | Medium | High | Very Low |
| **Extra Features** | No extra features beyond spec | Rich features like HQL, caching | Auto repositories, query derivation |

**A. Running the Hibernate Example (Plain Hibernate)**

**Step 1) Create Maven Project :**

**File > New > Maven Project**

**Step 2) Add Dependencies :**

<dependencies>

<!-- Hibernate Core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>6.6.18.Final</version>

</dependency>

<!-- MySQL Connector -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>8.3.0</version>

</dependency>

<!-- Logging (SLF4J) -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-simple</artifactId>

<version>2.0.9</version>

</dependency>

</dependencies>

**Step 3) Create Hibernate Configuration File**

Create hibernate.cfg.xml

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<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/your\_database</property>

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

<property name="hibernate.connection.password">your\_password</property>

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

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

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

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

</session-factory>

</hibernate-configuration>

**Step 2) Cretae necessary Classes :**

**Employeee.java**

**Code :**

package com.example;

import javax.persistence.\*;

@Entity

@Table(name = "employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

@Column

private String name;

public Employee() {}

public Employee(String name) {

this.name = name;

}

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; }

}

**HibernateUtil.java  
Code :**

package com.example;

public class HibernateUtil {

private static final SessionFactory sessionFactory = buildSessionFactory();

private static SessionFactory buildSessionFactory() {

try {

return new Configuration().configure().buildSessionFactory();

} catch (Throwable ex) {

throw new ExceptionInInitializerError(ex);

}

}

public static SessionFactory getSessionFactory() {

return sessionFactory;

}

}

**App.java  
Code :**

package com.example;

public class App {

public static void main(String[] args) {

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

Transaction tx = null;

try {

tx = session.beginTransaction();

Employee emp = new Employee("John Doe");

Integer empId = (Integer) session.save(emp);

tx.commit();

System.out.println("Employee created with ID: " + empId);

} catch (Exception e) {

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

e.printStackTrace();

} finally {

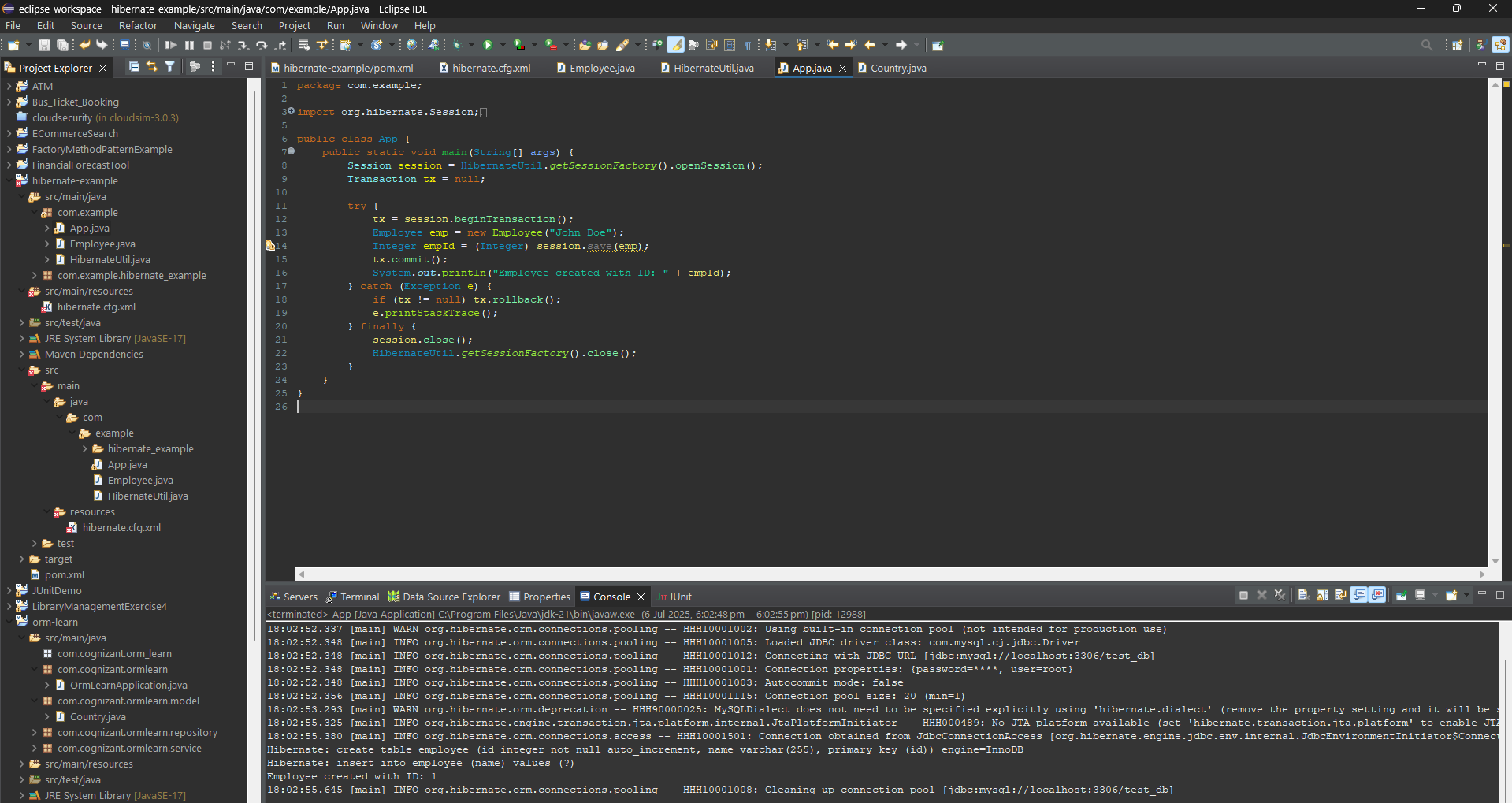
session.close();

HibernateUtil.getSessionFactory().close();

}

}

}

**RESULT :  
  
Output :**  
  


**Hibernate Example**

This example manually opens a session, starts a transaction, saves the entity, commits the transaction, and closes the session. All persistence logic is handled manually.

**B. Running the Spring Data JPA Example**

## Step 1) : ****Create a Spring Boot Project****

Use [Spring Initializr](https://start.spring.io):

* Project: Maven
* Spring Boot: 3.5.x
* Dependencies:
  + Spring Data JPA
  + MySQL Driver

Download and unzip the project.

**Step 2) Configure application.properties :**

spring.datasource.url=jdbc:mysql://localhost:3306/your\_database

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

**Step 3) Create necessary classes :**

**Employee.java  
Code :**package com.example.demo;

import jakarta.persistence.\*;

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private String name;

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  
Code :**package com.example.demo;

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

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**DemoApplication.java  
Code :**

package com.example.demo;

@SpringBootApplication

public class DemoApplication implements CommandLineRunner {

@Autowired

private EmployeeRepository employeeRepository;

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@Override

public void run(String... args) throws Exception {

Employee e = new Employee();

e.setName("Jane Doe");

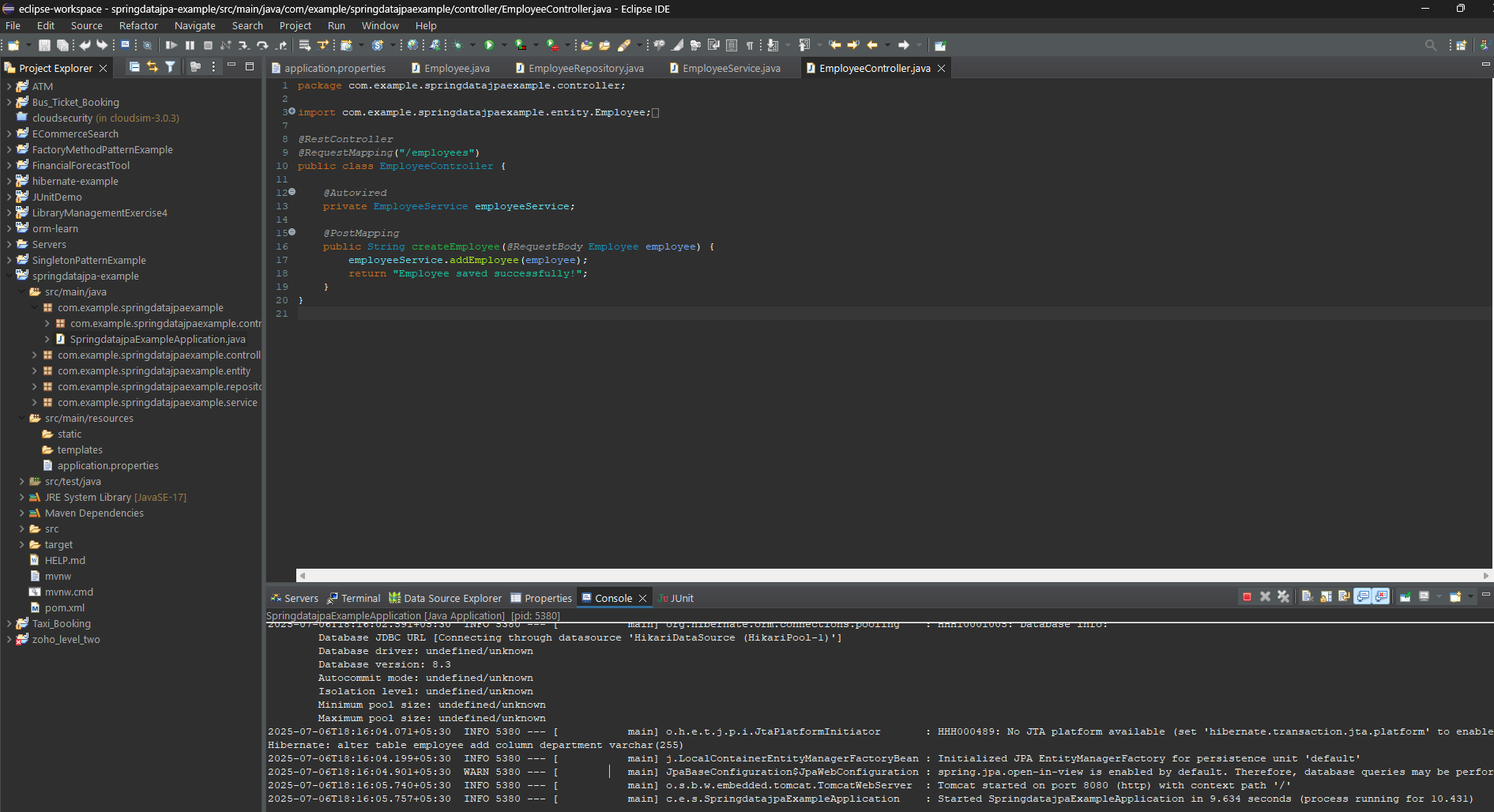
employeeRepository.save(e);

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

}

}

**Output :**



**Spring Data JPA Example**

This example uses a JpaRepository, which automatically provides CRUD methods. The save() method persists the entity, and transaction management is handled by Spring.