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

Spring Data JPA - Quick Example

**Source Code:**

pom.xml

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

<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>shop</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<parent>

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

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

<version>3.2.5</version>

<relativePath/>

</parent>

<properties>

<java.version>17</java.version> <!-- or 21 if your JDK supports -->

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<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-devtools</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

<configuration>

<source>${java.version}</source>

<target>${java.version}</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

Category.java

package com.example.shop.entity;

import jakarta.persistence.\*;

import java.util.List;

@Entity

@Table(name = "categories")

public class Category {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@OneToMany(mappedBy = "category")

private List<Product> products;

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 List<Product> getProducts() { return products; }

public void setProducts(List<Product> products) { this.products = products; }

}

Product.java

package com.example.shop.entity;

import jakarta.persistence.\*;

@Entity

@Table(name = "products")

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private double price;

@ManyToOne

@JoinColumn(name = "category\_id")

private Category category;

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 double getPrice() { return price; }

public void setPrice(double price) { this.price = price; }

public Category getCategory() { return category; }

public void setCategory(Category category) { this.category = category; }

}

CategoryRepository.java

package com.example.shop.repository;

import com.example.shop.entity.Category;

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

public interface CategoryRepository extends JpaRepository<Category, Long> {

}

ProductRepository.java

package com.example.shop.repository;

import com.example.shop.entity.Product;

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

import java.util.List;

public interface ProductRepository extends JpaRepository<Product, Long> {

List<Product> findByCategoryName(String categoryName);

}

ShopApplication.java

package com.example.shop;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

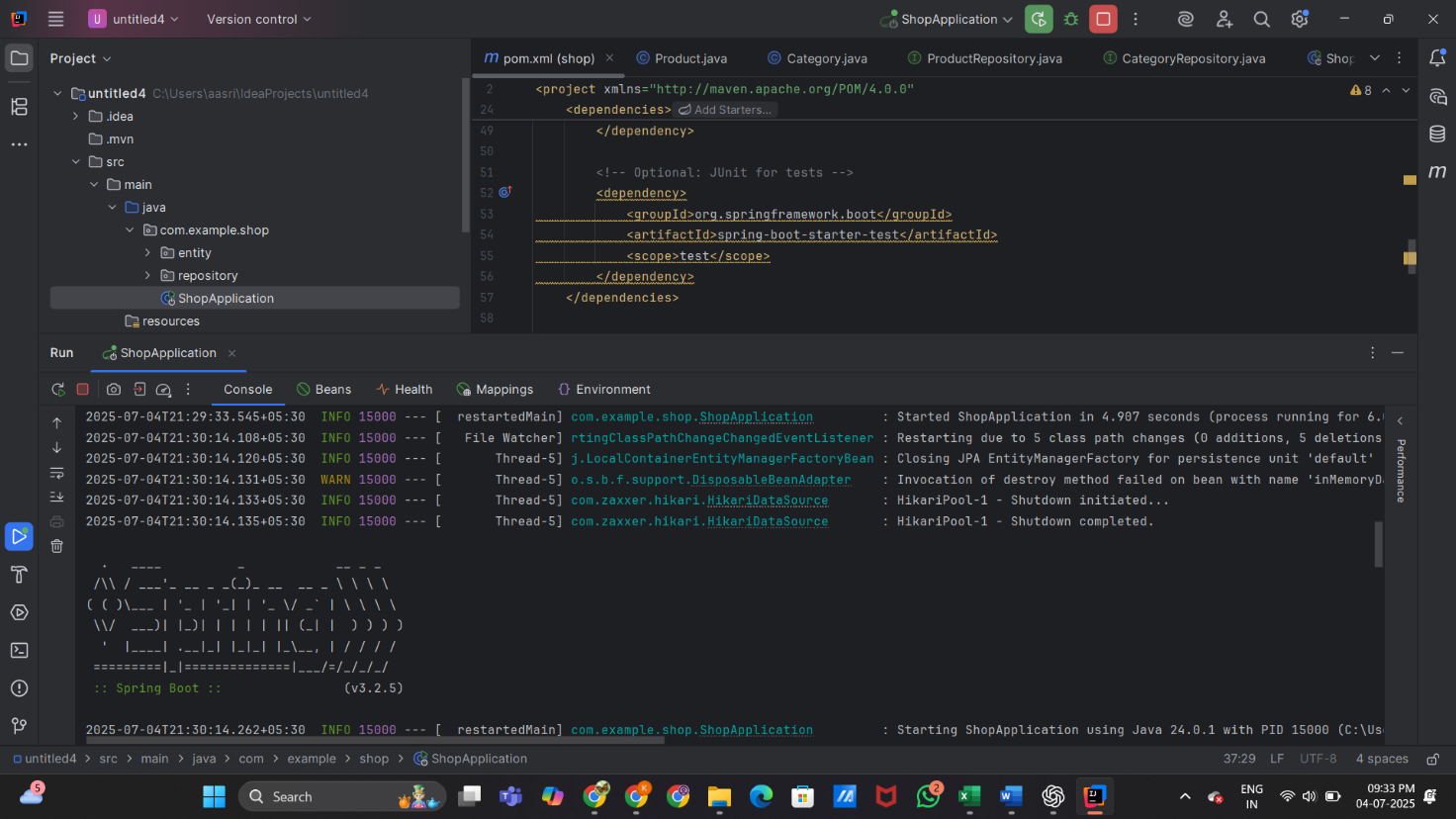
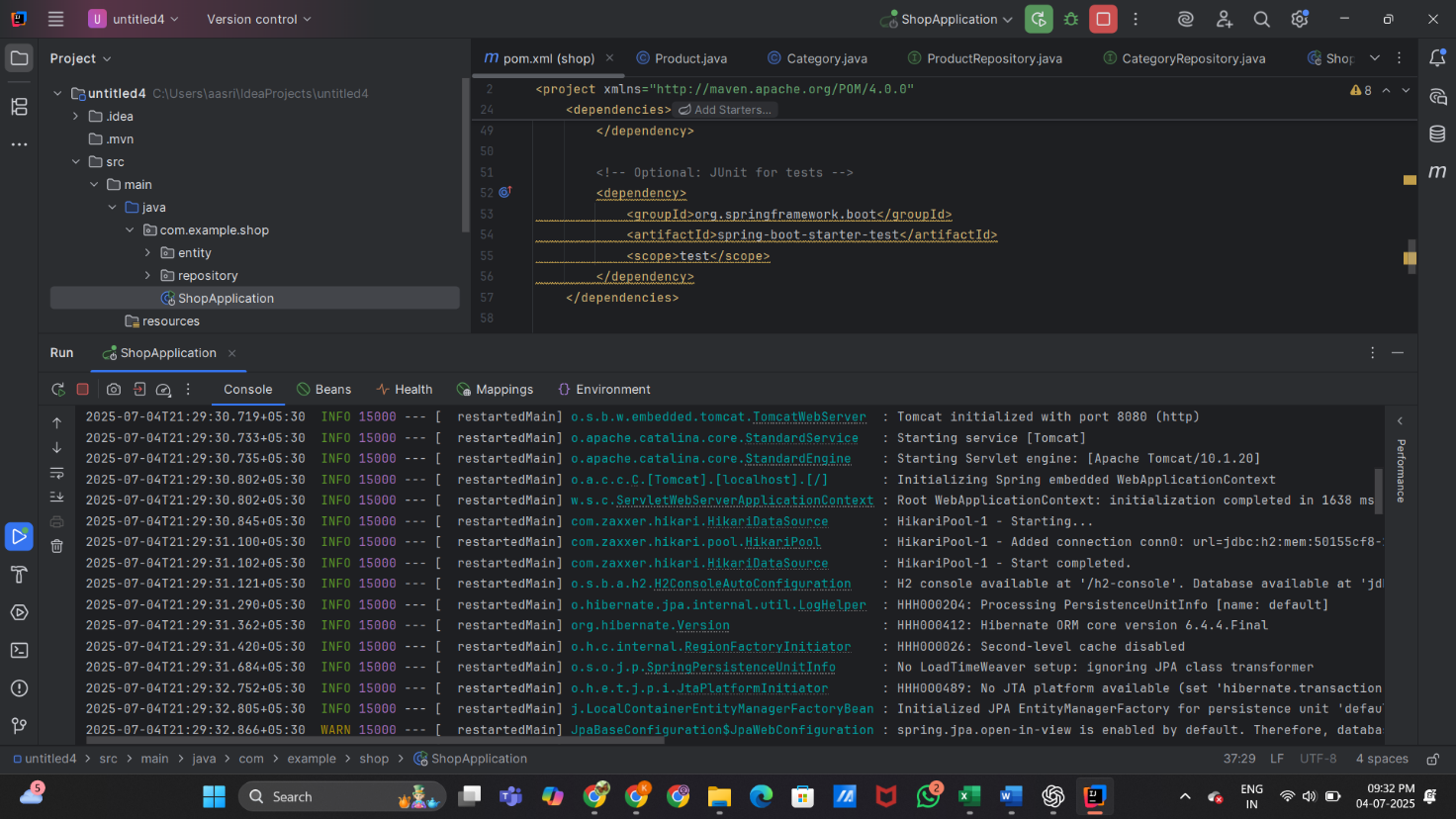
public class ShopApplication {

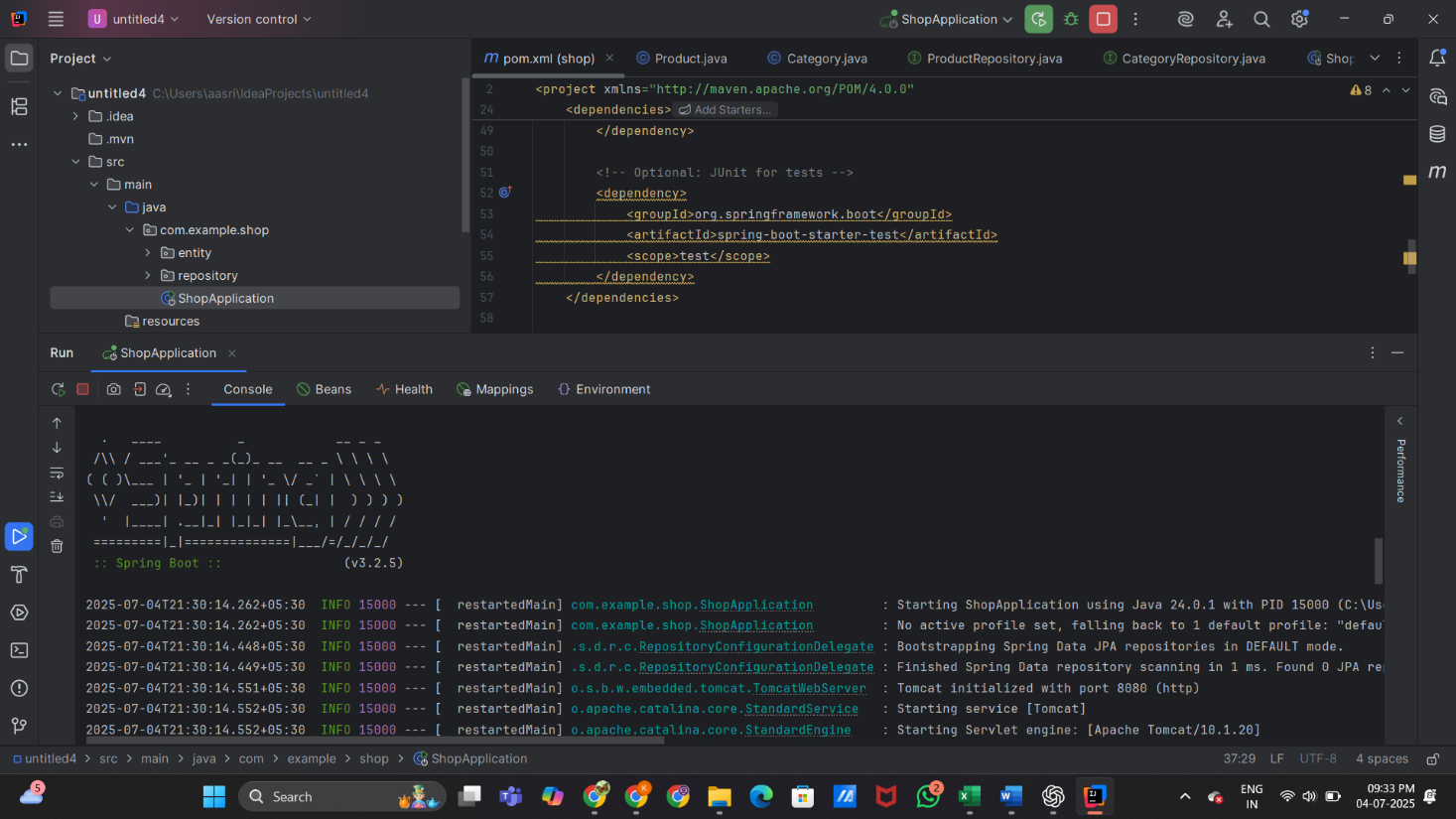
public static void main (String [] args) {

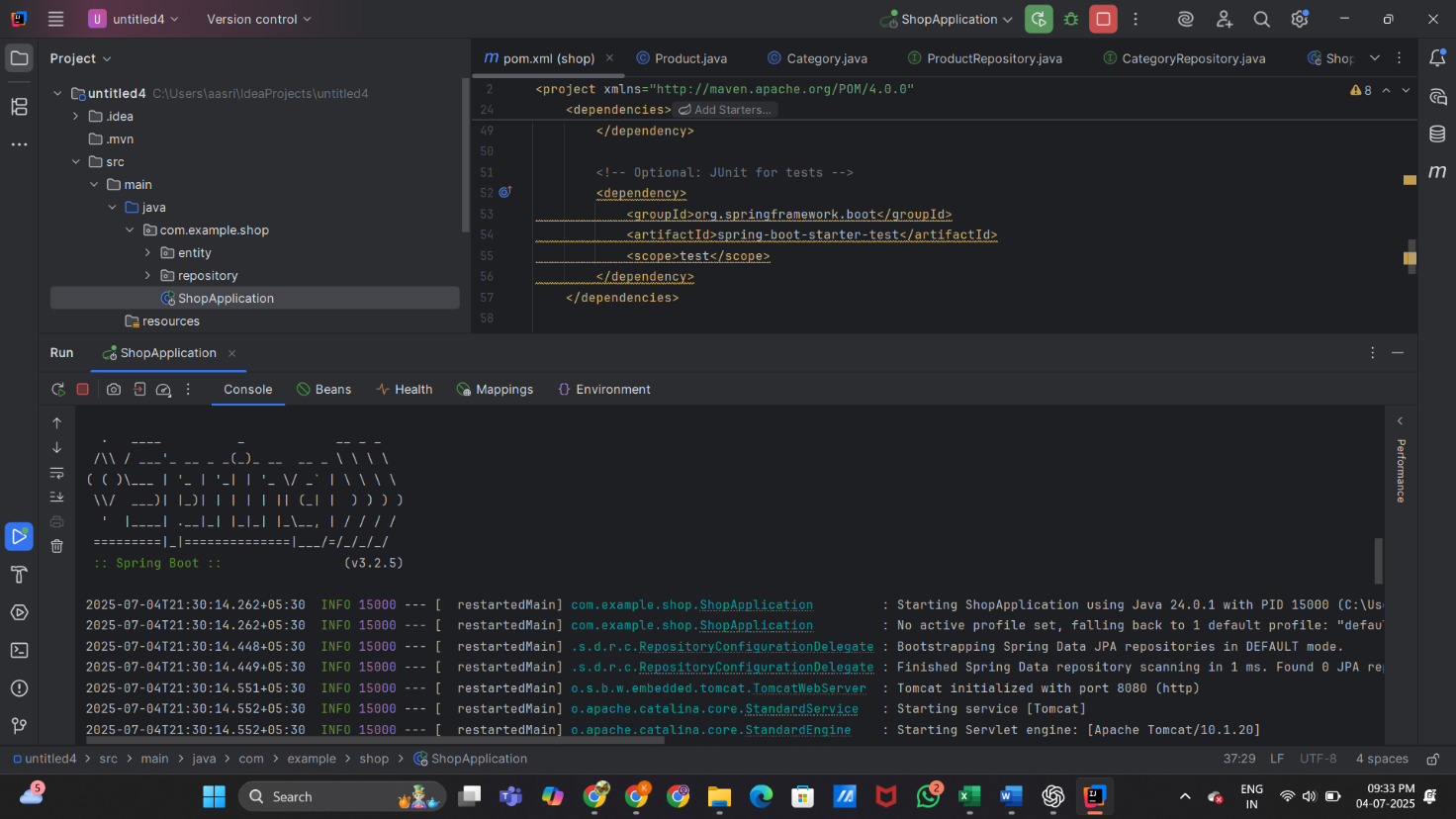
SpringApplication.run(ShopApplication.class, args);

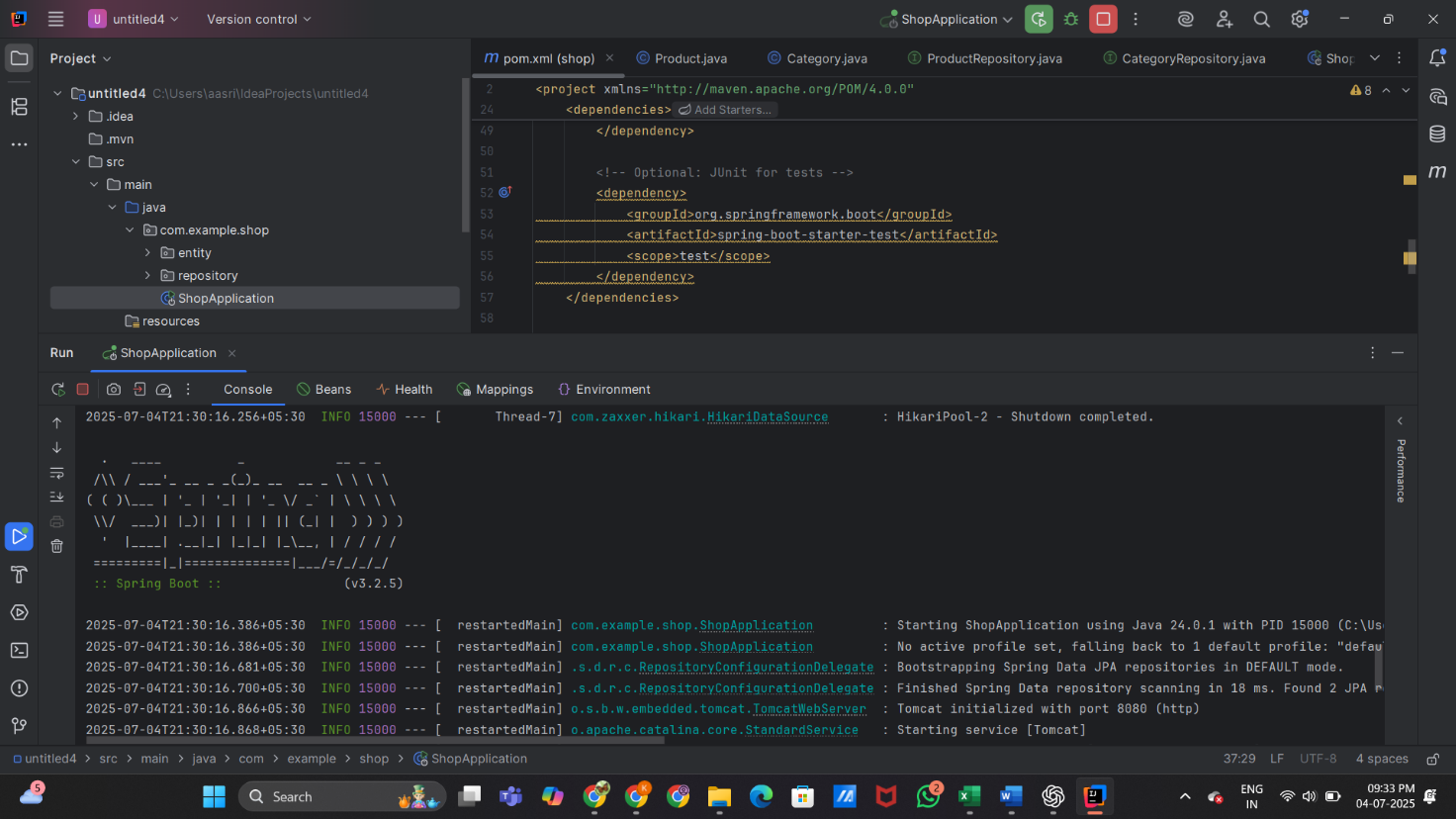
}

}

**Output:**

****





Difference between JPA, Hibernate and Spring Data JPA

**JPA (Java Persistence API)**

**What it is:**

Think of it as a standard API for ORM (Object Relational Mapping) in Java. It lays out how Java objects connect with relational database tables.

**Key characteristics:**

* Vendor-independent: It simply defines interfaces, annotations, and contracts like @Entity, @Id, and Entity Manager.
* No implementation: JPA doesn’t handle the actual work of saving objects to a database. You’ll need a JPA provider, such as Hibernate, Eclipse Link, or Open JPA, to get things rolling.
* Portability: Since it’s a standard, switching JPA providers requires only minor code tweaks.

**Example:**

@Entity

public class User {

@Id

private Long id;

private String name;

}

**Hibernate**

**What it is:**

Hibernate is a well-known implementation of JPA and also a standalone ORM framework that predates JPA.

**Key characteristics:**

JPA provider: It covers all the JPA specifications, allowing you to use it with standard JPA APIs.

Standalone ORM: You can also work with Hibernate directly through its native APIs (Session, Criteria, HQL) without relying on JPA.

Beyond JPA: It offers a bunch of advanced features like:

* Second-level caching
* Lazy/eager fetching optimizations
* Batch processing
* A richer Hibernate Query Language (HQL) compared to JPQL

Examples:

Session session = sessionFactory.openSession();

session.save(user);

**Spring Data JPA**

**What it is:**

This is a Spring project (part of Spring Data) that enhances JPA to make writing data access layers easier.

**Key characteristics:**

* Abstraction layer: It doesn’t implement JPA itself; it relies on a JPA provider underneath, often Hibernate in Spring Boot applications.
* Repository pattern: It introduces interfaces like JpaRepository that automatically generate CRUD operations and support query methods just by declaring method names like findByLastName.
* Boilerplate elimination: Removes the need to write manual DAO classes or EntityManager code.

**Example:**

public interface UserRepository extends JpaRepository<User, Long> {

List<User>findByEmail (String email);

}