**Spring Data JPA**

Spring Data JPA is a powerful part of the Spring ecosystem that simplifies working with databases in Java applications. It builds on top of the Java Persistence API (JPA) and provides a high-level abstraction for managing relational data using repositories.

**What It Does**

* **Reduces boilerplate code**: You don’t need to write complex DAO implementations.
* **Auto-generates queries**: Just define method names like findByUsername, and Spring Data JPA figures out the SQL.
* **Supports JPQL and native SQL**: You can still write custom queries when needed.
* **Integrates with Spring Boot**: Makes configuration and setup seamless.

**Key Components**

| **Component** | **Description** |
| --- | --- |
| @Entity | Marks a class as a JPA entity (mapped to a database table) |
| @Repository | Indicates a Spring Data repository interface |
| JpaRepository | Interface that provides CRUD operations and more |
| @Query | Allows defining custom JPQL or native SQL queries |

**Example**

**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);

}

}

**Student.java**

package com.example.demo.entity;

import jakarta.persistence.\*;

@Entity

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

// Getters and setters

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 String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

}

**StudentRepository.java**

package com.example.demo.repository;

import com.example.demo.entity.Student;

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

public interface StudentRepository extends JpaRepository<Student, Long> {

}

**StudentController.java**

package com.example.demo.controller;

import com.example.demo.entity.Student;

import com.example.demo.repository.StudentRepository;

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

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

import java.util.List;

@RestController

@RequestMapping("/students")

public class StudentController {

@Autowired

private StudentRepository studentRepository;

@GetMapping

public List<Student> getAllStudents() {

return studentRepository.findAll();

}

@PostMapping

public Student createStudent(@RequestBody Student student) {

return studentRepository.save(student);

}

@GetMapping("/{id}")

public Student getStudentById(@PathVariable Long id) {

return studentRepository.findById(id).orElse(null);

}

@PutMapping("/{id}")

public Student updateStudent(@PathVariable Long id, @RequestBody Student updatedStudent) {

return studentRepository.findById(id).map(student -> {

student.setName(updatedStudent.getName());

student.setEmail(updatedStudent.getEmail());

return studentRepository.save(student);

}).orElse(null);

}

@DeleteMapping("/{id}")

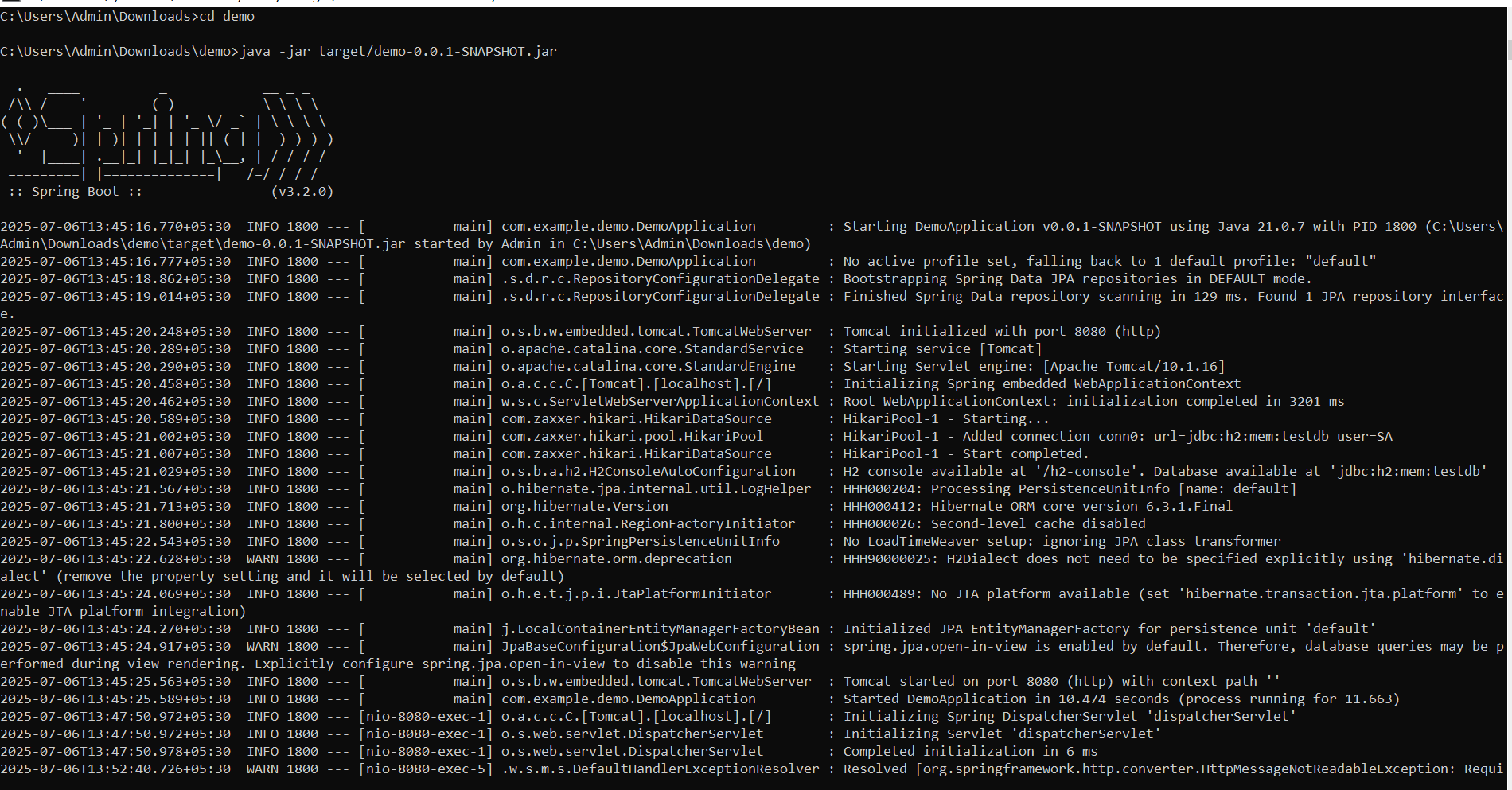
public void deleteStudent(@PathVariable Long id) {

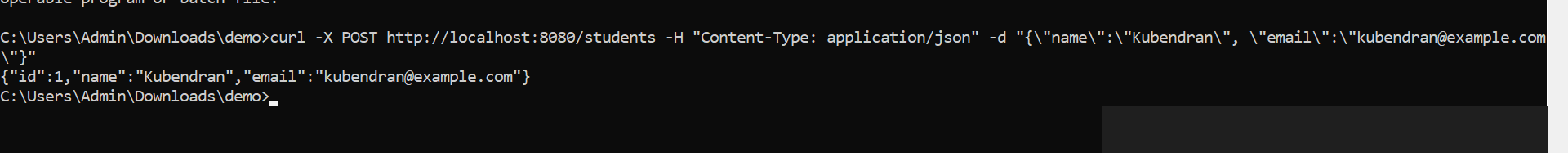
studentRepository.deleteById(id);

}

}

**Output:**

****



A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

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

**1. JPA (Java Persistence API)**

* **It is a specification**, not a tool.
* Provides a standard way to manage relational data in Java apps.
* Defines annotations like @Entity, @Id, @OneToMany, etc.
* Needs a **provider** (like Hibernate) to work.

**Example:**

java

CopyEdit

@Entity

public class Student {

@Id

private Long id;

private String name;

}

**2. Hibernate**

* **It is the most popular JPA implementation**.
* Adds extra features on top of JPA.
* Converts Java objects to database tables and vice versa.
* Handles SQL generation, caching, transactions, etc.

Example (Hibernate-specific config):

properties

CopyEdit

hibernate.show\_sql=true

hibernate.format\_sql=true

**3. Spring Data JPA**

* A **Spring framework module**.
* Makes working with JPA (and Hibernate) even easier.
* You only write **interfaces**, no need to write SQL or implementation.

Example:

java

CopyEdit

public interface StudentRepository extends JpaRepository<Student, Long> {

List<Student> findByName(String name);

}

**How They Work Together**

scss

CopyEdit

Spring Data JPA

↓

JPA (Java Persistence API)

↓

Hibernate (Implementation)

↓

Database (MySQL, H2, etc.)