**Spring Data JPA - Quick Example**

**Student.java :**

package com.example.studentapp.model;  
import jakarta.persistence.\*;  
  
@Entity  
public class Student {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 private String name;  
 private String email;  
  
 public Student() {}  
  
 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.studentapp.repository;  
import com.example.studentapp.model.Student;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface StudentRepository extends JpaRepository<Student, Long> {}

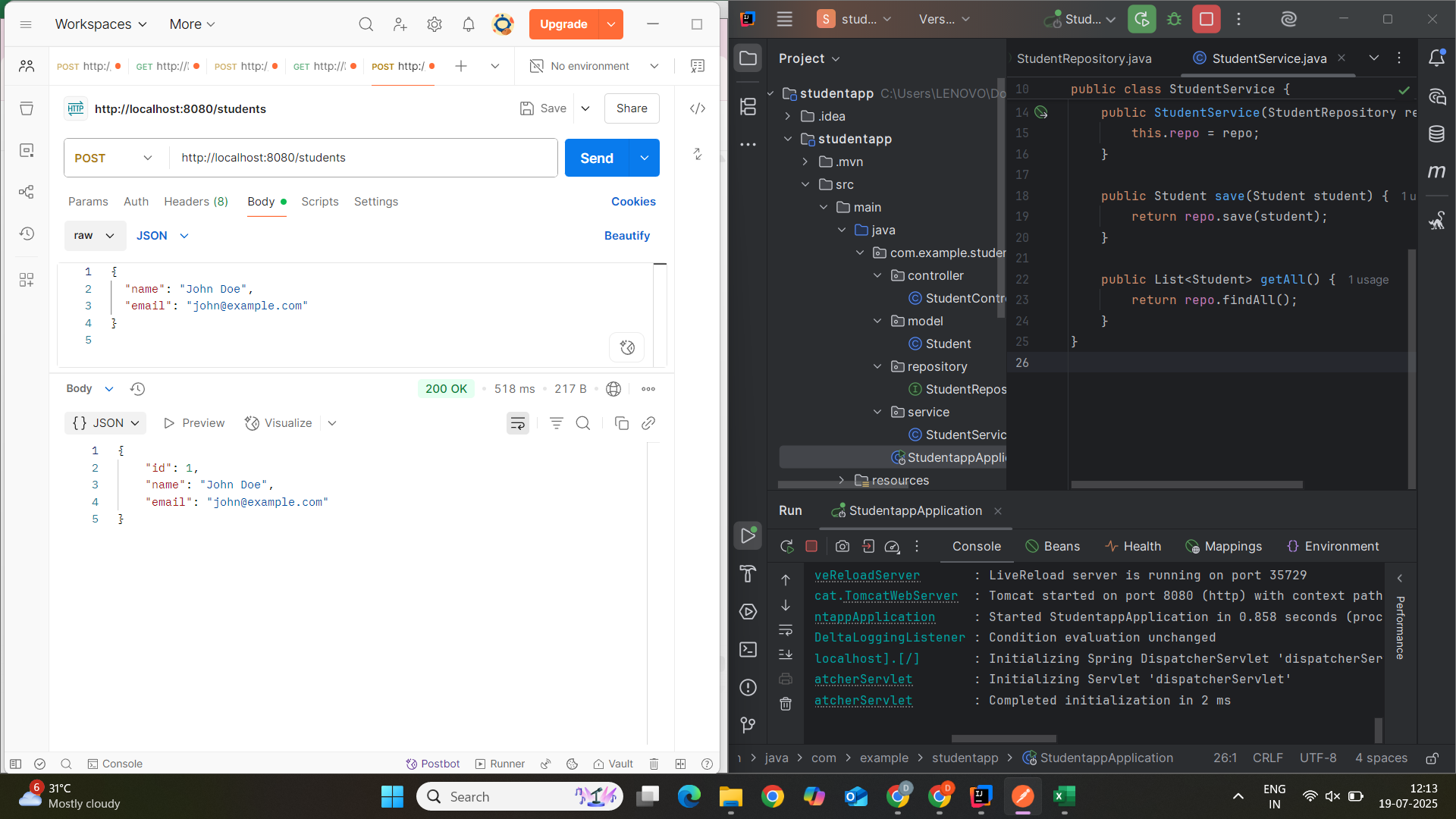
**StudentService.java :**

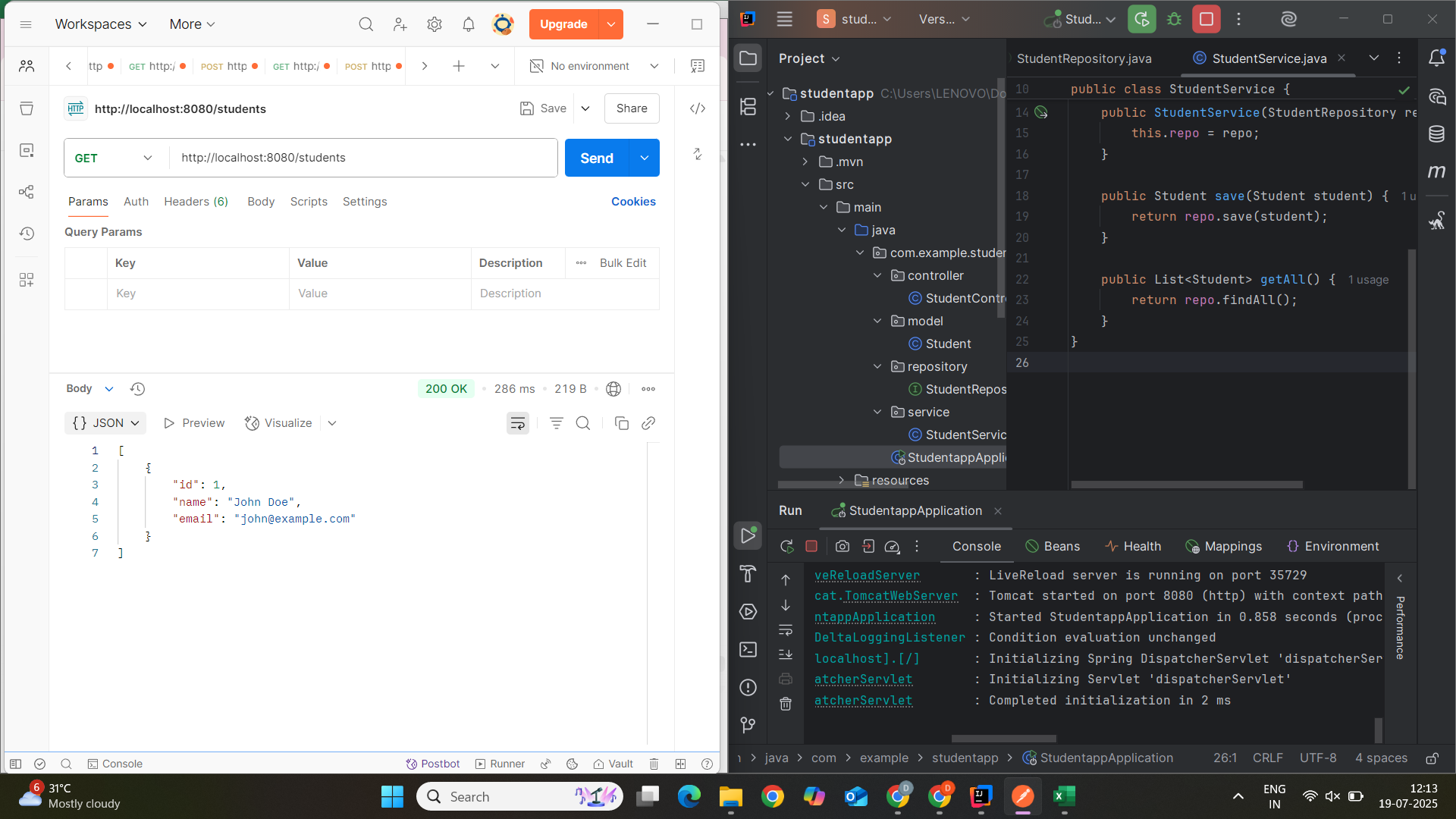
package com.example.studentapp.service;  
import com.example.studentapp.model.Student;  
import com.example.studentapp.repository.StudentRepository;  
import org.springframework.stereotype.Service;  
import java.util.List;  
  
@Service  
public class StudentService {  
  
 private final StudentRepository repo;  
  
 public StudentService(StudentRepository repo) {  
 this.repo = repo;  
 }  
  
 public Student save(Student student) {  
 return repo.save(student);  
 }  
  
 public List<Student> getAll() {  
 return repo.findAll();  
 }  
}

**StudentController.java :**

package com.example.studentapp.controller;  
import com.example.studentapp.model.Student;  
import com.example.studentapp.service.StudentService;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/students")  
public class StudentController {  
  
 private final StudentService service;  
  
 public StudentController(StudentService service) {  
 this.service = service;  
 }  
  
 @PostMapping  
 public Student addStudent(@RequestBody Student student) {  
 return service.save(student);  
 }  
  
 @GetMapping  
 public List<Student> getStudents() {  
 return service.getAll();  
 }  
}

**OUTPUT :**

****

****

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

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

JPA is a Java specification for accessing, persisting, and managing data between Java objects and relational databases.

* Introduced as part of Java EE (now Jakarta EE).
* Provides annotations like @Entity, @Table, @Id, @OneToMany, etc.
* Works as an abstraction layer over ORM tools (like Hibernate).
* Requires a JPA provider (e.g., Hibernate, EclipseLink) for implementation.

**Example**:

@Entity

public class Student {

@Id

private Long id;

private String name;

}

**2. Hibernate**

Hibernate is a powerful ORM (Object-RelationalMapping) tool for Java. It is also the most widely used JPA implementation.

* Provides its own rich set of features beyond JPA.
* Can be used with or without JPA.
* Manages database operations like CRUD, transactions, caching, lazy loading, etc.
* Supports both XML-based and annotation-based configurations.

**Example :**

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

session.save(new Student(1L, "John"));

tx.commit();

**3. Spring Data JPA**

Spring Data JPA is a Spring Framework project that provides easy integration with JPA, reducing boilerplate code using repositories and method-name-based queries.

* Built on top of JPA.
* Works well with Spring Boot.
* Reduces boilerplate code significantly using interfaces like JpaRepository.
* Automatically implements query methods based on method names.

**Example :**

public interface StudentRepository extends JpaRepository<Student, Long> {

List<Student> findByName(String name);

}

| **Aspect** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| 1. Type | Specification (interface-based) | ORM Framework and JPA implementation | Abstraction layer built on top of JPA and Hibernate |
| 2. Provided By | Oracle / Jakarta EE | Red Hat | Spring Framework |
| 3. Purpose | Defines a standard for ORM in Java | Provides ORM features and implements JPA | Simplifies data access with auto-generated queries |
| 4. Implementation | Requires a provider (e.g., Hibernate, EclipseLink) | Concrete implementation of JPA and ORM engine | Uses JPA provider like Hibernate internally |
| 5. Ease of Use | Moderate (requires boilerplate code and manual setup) | Easier than JDBC but still requires configurations | Very easy; reduces boilerplate with repositories |
| 6. Query Support | JPQL (Java Persistence Query Language) | JPQL + HQL + Criteria API | Derived query methods, JPQL, native queries |
|  |  |  |  |