



# THỰC HÀNH SPRING BOOT CRUD API QUẢN LÝ SINH VIÊN



# Yêu cầu về ứng dụng

- Xây dựng ứng dụng Spring Boot, cung cấp các API để quản lý sinh viên.



## Các API cần có:

- Xem danh sách sinh viên:
  - Request: **GET /api/students**
  - Response: Danh sách sinh viên hiện có trong cơ sở dữ liệu.



## Các API cần có:

- Xem thông tin chi tiết của một sinh viên:
  - Request: **GET** `/api/students/{id}`
  - Response: Thông tin chi tiết của sinh viên có ID tương ứng.



## Các API cần có:

- Tạo một sinh viên mới:
  - Request: **POST** **/api/students**
  - Body: Thông tin sinh viên (email, họ đệm, tên, ... ) trong định dạng JSON.
  - Response: Sinh viên được tạo thành công.



## Các API cần có:

- Cập nhật thông tin của một sinh viên:
  - Request: **PUT** `/api/students/{id}`
  - Body: Thông tin sinh viên cần cập nhật (email, họ đệm, tên, ...) trong định dạng JSON.
  - Response: Sinh viên được cập nhật thành công.



## Các API cần có:

- Xóa một sinh viên:
  - Request: **DELETE** `/api/students/{id}`
  - Response: Sinh viên có ID tương ứng được xóa thành công.



Phương thức	Điểm cuối (Endpoint)	Ý nghĩa
GET	<u>/api/students</u>	Lấy danh sách tất cả sinh viên
GET	/api/students/{id}	Lấy thông tin chi tiết của một sinh viên dựa trên ID
POST	/api/students	Thêm một sinh viên mới
PUT	/api/students/{id}	Cập nhật thông tin của một sinh viên dựa trên ID
DELETE	/api/students/{id}	Xóa một sinh viên dựa trên ID



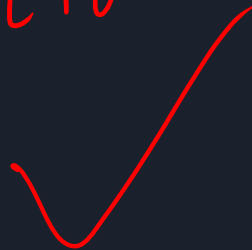
# Lưu ý về việc tạo REST API

GET

POST

PUT

DELETE




/api / getStudent

/api / addStudent

/api / deleteStudent






```
CREATE TABLE `students` (  
  `id` INT(11) NOT NULL AUTO_INCREMENT,  
  `email` VARCHAR(45) NULL DEFAULT NULL COLLATE 'utf8mb4_general_ci',  
  `first_name` VARCHAR(45) NULL DEFAULT NULL COLLATE 'utf8mb4_general_ci',  
  `last_name` VARCHAR(45) NULL DEFAULT NULL COLLATE 'utf8mb4_general_ci',  
  PRIMARY KEY (`id`) USING BTREE  
)  
COLLATE='utf8mb4_general_ci'  
ENGINE=InnoDB;
```




```
INSERT INTO students (email, last_name, first_name)
VALUES
```

```
  ('email1@example.com', 'Nguyễn', 'Văn A'),
  ('email2@example.com', 'Trần', 'Thị B'),
  ('email3@example.com', 'Lê', 'Đức C'),
  ('email4@example.com', 'Phạm', 'Minh D'),
  ('email5@example.com', 'Võ', 'Hoàng E'),
  ('email6@example.com', 'Nguyễn', 'Thị F'),
  ('email7@example.com', 'Trần', 'Văn G'),
  ('email8@example.com', 'Lê', 'Thanh H'),
  ('email9@example.com', 'Phạm', 'Quang I'),
  ('email10@example.com', 'Võ', 'Tuấn J');
```



## Step 1. Sử dụng Spring Initializr để tạo dự án Spring Boot và cấu hình các phụ thuộc cần thiết

- Spring Web
- Spring Data JPA
- Spring Boot Dev Tools
- MySQL Driver
- ...



## Step 2. Sử dụng Spring Data JPA để tương tác với cơ sở dữ liệu

- Cấu hình kết nối với cơ sở dữ liệu
- Tạo Entity
- Tạo Interface: StudentDAO với các phương thức: find, findAll, add, update, delete
- Tạo Repository: StudentDAOImpl và hiện thực các phương thức
- Tạo RestController sử dụng DAO



# Cấu hình kết nối với cơ sở dữ liệu

# Config Datasource

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

spring.datasource.username=spring

spring.datasource.password=spring

# Tạo Entity

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String email;

    private String lastName;

    private String firstName;
```

```
// Constructors, getters, and setters
```

```
public Student() {
    // Default constructor
}
```

```
public Student(String email, String lastName, String firstName) {
    this.email = email;
    this.lastName = lastName;
    this.firstName = firstName;
}
```

```
public Long getId() {
    return id;
}
```

```
public String getEmail() {
    return email;
}
```

```
public void setEmail(String email) {
    this.email = email;
}
```

```
public String getLastName() {
    return lastName;
}
```

```
public void setLastName(String lastName) {
    this.lastName = lastName;
}
```

```
public String getFirstName() {
    return firstName;
}
```

```
public void setFirstName(String firstName) {
    this.firstName = firstName;
}
```

Có thể quy định các ràng buộc, tên cột, ... ; toString, equals ...

# Tạo Interface StudentDAO

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

public interface StudentDAO extends JpaRepository<Student, Long> {
    // Find all students
    List<Student> findAll();

    // Find student by ID
    Student findById(Long id);

    // Add a new student
    Student save(Student student);

    // Update an existing student
    Student saveAndFlush(Student student);

    // Delete a student
    void delete(Student student);
}
```



# Tạo Repository

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
```

```
import javax.persistence.EntityManager;
import java.util.List;
```

```
@Repository
```

```
public class StudentDAOImpl implements StudentDAO {
    private EntityManager entityManager;
```

```
    @Autowired
```

```
    public StudentDAOImpl(EntityManager entityManager) {
        this.entityManager = entityManager;
    }
```

```
    // Rest of the methods...
```

```
}
```

@Transactional ?



```
@Override
```

```
public List<Student> findAll() {
```

```
    return entityManager.createQuery("SELECT s FROM Student s", Student.  
        .getResultList());
```

```
}
```



```
@Override
```

```
public Student findById(Long id) {  
    return entityManager.find(Student.class, id);  
}
```



```
@Override
```

```
public Student save(Student student) {  
    entityManager.persist(student);  
    return student;  
}
```



```
@Override
```

```
public Student saveAndFlush(Student student) {  
    student = entityManager.merge(student);  
    entityManager.flush();  
    return student;  
}
```

```
@Override  
public void delete(Student student) {  
    entityManager.remove(student);  
}
```

```
@Override  
public void delete(Student student) {  
    // Kiểm tra xem đối tượng sinh viên có được quản lý bởi EntityManager  
    if (!entityManager.contains(student)) {  
        // Nếu đối tượng không được quản lý, sử dụng merge để gắn kết lại  
        student = entityManager.merge(student);  
    }  
    // Xóa đối tượng sinh viên  
    entityManager.remove(student);  
}
```

# Tạo REST API & Test với Postman

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/students")
public class StudentController {
    private final StudentDAO studentDAO;

    @Autowired
    public StudentController(StudentDAO studentDAO) {
        this.studentDAO = studentDAO;
    }

    @GetMapping
    public List<Student> getAllStudents() {
        return studentDAO.findAll();
    }
}
```

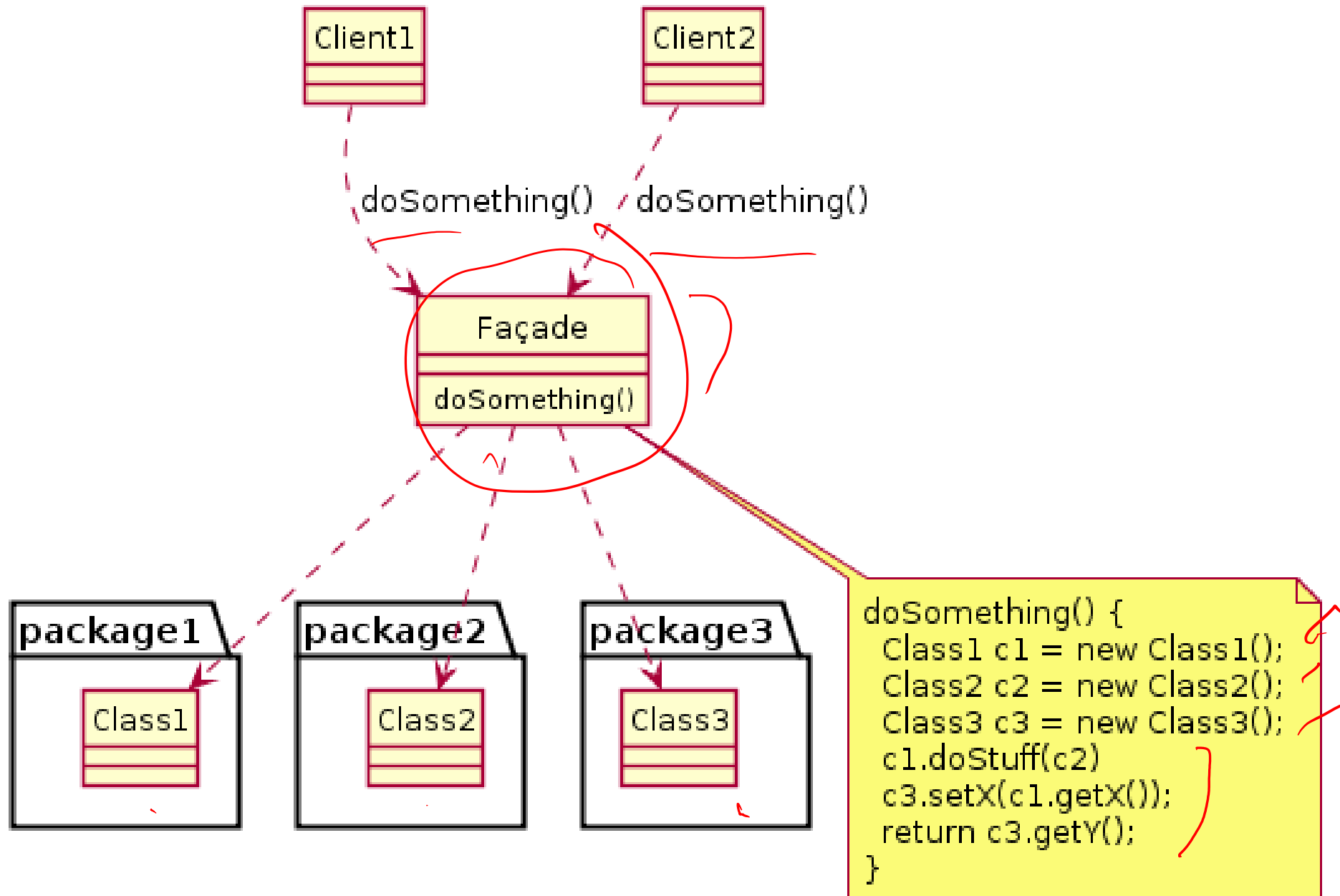
# Service Facade Design Pattern



Service Facade là một mẫu thiết kế (design pattern) cung cấp một giao diện đơn giản hoặc một API cấp cao hơn cho một hệ thống con phức tạp hoặc tập hợp các dịch vụ. Nó hoạt động như một điểm vào duy nhất, che giấu sự phức tạp của hệ thống con nằm phía dưới và cung cấp một giao diện thống nhất để truy cập vào nhiều dịch vụ.

Mục đích chính của mẫu Service Facade là cải thiện khả năng sử dụng và bảo trì của hệ thống bằng cách đóng gói các tương tác với nhiều dịch vụ vào một giao diện đơn giản, được định nghĩa rõ ràng. Điều này cho phép khách hàng tương tác với hệ thống theo cách đơn giản hơn, mà không cần biết chi tiết và sự phức tạp của từng dịch vụ cụ thể.

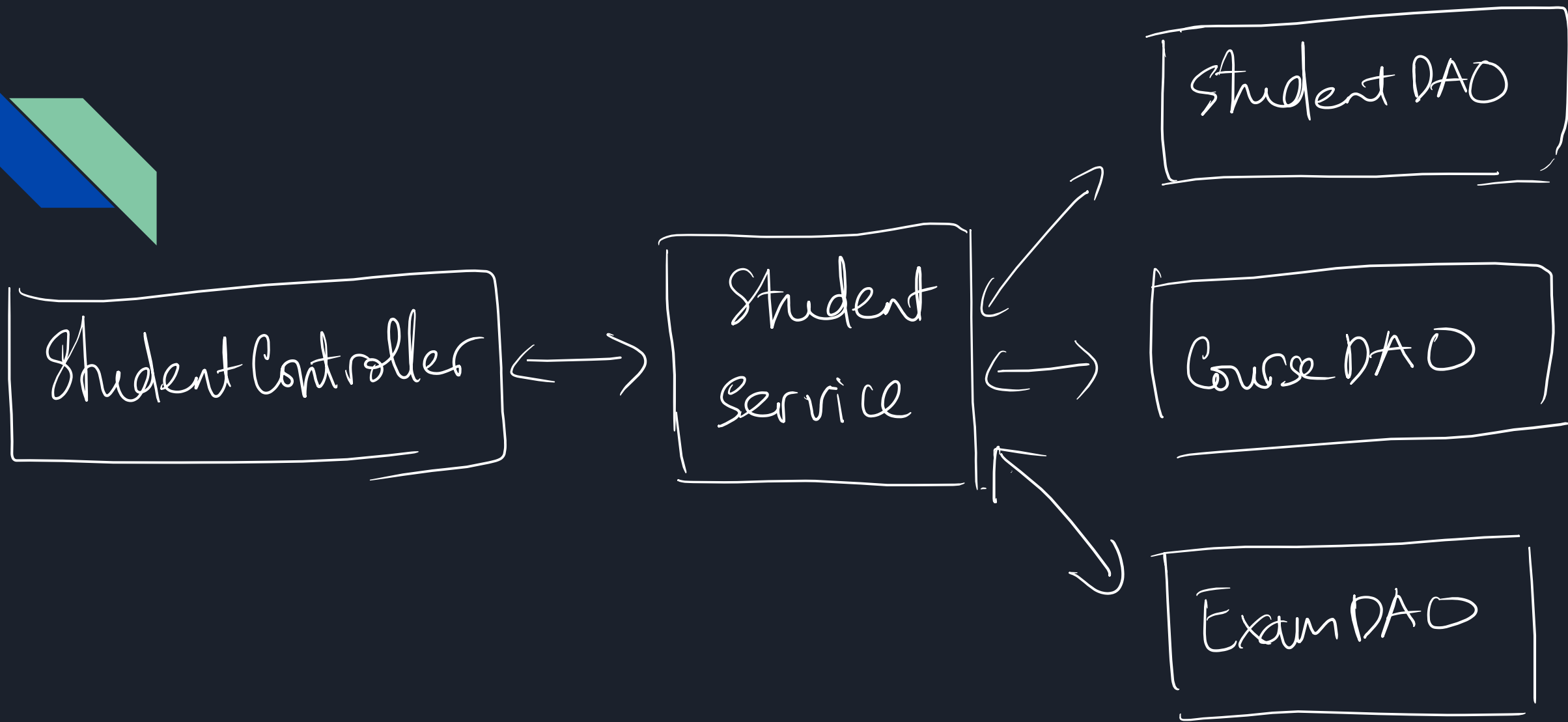




## Có nên áp dụng Service Facade?

- Có, mẫu thiết kế Service Facade có thể được áp dụng để tạo một StudentService. StudentService có thể là một lớp trung gian giữa lớp StudentController và StudentDAO để cung cấp một giao diện đơn giản và thuận tiện cho các hoạt động liên quan đến sinh viên.






...



# @Service

- @Service trong Spring được sử dụng để chỉ định rằng một lớp là một thành phần dịch vụ



```
public interface StudentService {  
    List<Student> getAllStudents();  
    Student getStudentById(Long id);  
    void addStudent(Student student);  
    void updateStudent(Student student);  
    void deleteStudent(Student student);  
}
```

```
@Service
public class StudentServiceImpl implements StudentService {
    private final StudentDAO studentDAO;

    @Autowired
    public StudentServiceImpl(StudentDAO studentDAO) {
        this.studentDAO = studentDAO;
    }

    @Override
    public List<Student> getAllStudents() {
        return studentDAO.findAll();
    }

    @Override
    public Student getStudentById(Long id) {
        return studentDAO.findById(id);
    }
}
```

```
@Override
public void addStudent(Student student) {
    studentDAO.save(student);
}

@Override
public void updateStudent(Student student) {
    studentDAO.update(student);
}

@Override
public void deleteStudent(Student student) {
    studentDAO.delete(student);
}
```



# Chuyển @Transactional

- Chúng ta nên chuyển @Transactional từ lớp DAO sang lớp Service

# Viết lại REST API

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/students")
public class StudentController {
    private StudentService studentService;

    @Autowired
    public StudentController(StudentService studentService) {
        this.studentService = studentService;
    }

    @GetMapping
    public List<Student> getAllStudents() {
        return studentService.getAllStudents();
    }
}
```



# Tóm tắt lại dự án

Phương thức	Điểm cuối (Endpoint)	Ý nghĩa
GET	/api/students	Lấy danh sách tất cả sinh viên
GET	/api/students/{id}	Lấy thông tin chi tiết của một sinh viên dựa trên ID
POST	/api/students	Thêm một sinh viên mới
PUT	/api/students/{id}	Cập nhật thông tin của một sinh viên dựa trên ID
DELETE	/api/students/{id}	Xóa một sinh viên dựa trên ID

```
@GetMapping("/{id}")
public ResponseEntity<Student> getStudentById(@PathVariable Long id) {
    Student student = studentService.getStudentById(id);
    if (student != null) {
        return ResponseEntity.ok(student);
    } else {
        return ResponseEntity.notFound().build();
    }
}
```



```
@PostMapping
```

```
public ResponseEntity<Void> addStudent(@RequestBody Student student) {  
    studentService.addStudent(student);  
    return ResponseEntity.status(HttpStatus.CREATED).build();  
}
```

Lưu ý: `student.setId(0)` → Vì tự tạo rd khi thêm mới ./



```
@PutMapping("/{id}")
```

```
public ResponseEntity<Void> updateStudent(@PathVariable Long id, @Requ
```

```
    Student existingStudent = studentService.getStudentById(id);
```

```
    if (existingStudent != null) {
```

```
        existingStudent.setEmail(student.getEmail());
```

```
        existingStudent.setHoDem(student.getHoDem());
```

```
        existingStudent.setTen(student.getTen());
```

```
        studentService.updateStudent(existingStudent);
```

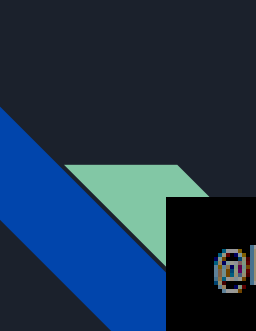
```
        return ResponseEntity.ok().build();
```

```
    } else {
```

```
        return ResponseEntity.notFound().build();
```

```
    }
```

```
}
```



```
@DeleteMapping("/{id}")
```

```
public ResponseEntity<Void> deleteStudent(@PathVariable Long id) {
```

```
    Student student = studentService.getStudentById(id);
```

```
    if (student != null) {
```

```
        studentService.deleteStudent(student);
```

```
        return ResponseEntity.ok().build();
```

```
    } else {
```

```
        return ResponseEntity.notFound().build();
```

```
    }
```

```
}
```

# Thay đổi cách viết sang dạng bắt Exception

```
@GetMapping("/{id}")
public ResponseEntity<Student> getStudentById(@PathVariable Long id) {
    try {
        Student student = studentService.getStudentById(id);
        return ResponseEntity.ok(student);
    } catch (EntityNotFoundException e) {
        throw new RuntimeException("Sinh viên không tồn tại"); // Hoặc
    }
}
```

```
@PutMapping("/{id}")
public ResponseEntity<Void> updateStudent(@PathVariable Long id, @RequestBody Student student)
    try {
        Student existingStudent = studentService.getStudentById(id);
        existingStudent.setEmail(student.getEmail());
        existingStudent.setHoDem(student.getHoDem());
        existingStudent.setTen(student.getTen());
        studentService.updateStudent(existingStudent);
        return ResponseEntity.ok().build();
    } catch (EntityNotFoundException e) {
        throw new RuntimeException("Sinh viên không tồn tại");
    }
}
```

```
@DeleteMapping("/{id}")
```

```
public ResponseEntity<Void> deleteStudent(@PathVariable Long id) {
```

```
    try {
```

```
        Student student = studentService.getStudentById(id);
```

```
        studentService.deleteStudent(student);
```

```
        return ResponseEntity.ok().build();
```

```
    } catch (EntityNotFoundException e) {
```

```
        throw new RuntimeException("Sinh viên không tồn tại");
```

```
    }
```

```
}
```





Bắt lỗi

. @ControllerAdvice