Lab 2: Xây Dựng Backend REST API

Ở Lab 1, chúng ta đã khởi tạo thành công project Spring Boot, cấu hình kết nối tới cơ sở dữ liệu SQLite và hiểu rõ kiến trúc Layered Architecture. Trong bài Lab 2 này, chúng ta sẽ bắt tay vào viết code chi tiết cho từng tầng để hoàn thiện các chức năng quản lý sinh viên.

Mục Tiêu

- Hiện thực hóa các layer: Entity, Repository, Service, Controller.
- Xây dựng API truy vấn dữ liệu (Read).
- *Lưu ý: Các API Create, Update, Delete sẽ được thực hiện trong phần bài tập về nhà ở Lab 4.*

> [!IMPORTANT] **Tại sao lại cần làm API (Lab 2) nếu Lab 3/4 làm Web SSR?**
>
> Trong thực tế, Backend thường được thiết kế để phục vụ đa nền tảng (Web, Mobile App, 3rd Party Partners). Việc xây dựng các REST API trả về JSON (ở Lab 2) giúp chúng ta:
>
> 1. **Kiểm thử độc lập**: Đảm bảo logic nghiệp vụ (Service/Repository) chạy đúng mà không cần phụ thuộc vào giao diện.

> 2. **Mở rộng**: Sau này nếu muốn làm App Mobile (React Native/Flutter), bạn chỉ cần gọi lại chính các API này.
>
> Vì vậy, ở Lab 2 chúng ta sẽ tập trung xây dựng "lõi" xử lý dữ liệu chuẩn xác trước. Đến Lab 3 & 4, chúng ta sẽ xây dựng giao diện ("gương mặt") cho người dùng cuối.

1. Thiết Kế API (API Specification)

Trước khi viết code, chúng ta cần thống nhất các Endpoint sẽ cung cấp:

Chức Năng	Method	Endpoint	Request Body	Response
Lấy danh sách	`GET`	`/api/students`	-	`List<Student>`
Lấy chi tiết	`GET`	`/api/students/{id}`	-	`Student`

(Các API POST, PUT, DELETE sẽ được hướng dẫn và yêu cầu thực hiện trong Lab 4)

> [!NOTE] Lab 3 sẽ hướng dẫn cách xây dựng giao diện Web (SSR) để hiển thị dữ liệu này.

2. Kiến Thức Nền Tảng (Lý Thuyết)

Trước khi code, hãy làm quen với các khái niệm cốt lõi của Spring Boot.

2.1. Dependency Injection (DI) & Inversion of Control (IoC)

Trong lập trình truyền thống, khi Class A cần dùng Class B, Class A sẽ tự khởi tạo B (`B b = new B()`).

- **Vấn đề**: Code bị phụ thuộc chặt (Tightly Coupled). Nếu B thay đổi cách khởi tạo, A cũng phải sửa theo. Khó viết Unit Test.

- **Giải pháp (IoC)**: Thay vì tự tạo, Class A sẽ "nhờ" một người quản lý (Container) cung cấp instance của B cho mình. Việc cung cấp này gọi là **Dependency Injection (DI)**.

2.2. Annotation `@Autowired`

Đây là cách chúng ta yêu cầu Spring thực hiện DI.

```
@Autowired
private StudentRepository repository;
```

- **Ý nghĩa**: "Spring ơi, tôi cần dùng `StudentRepository`. Hãy tìm trong kho chứa (Application Context) xem đã có instance nào của nó chưa, nếu có thì gán vào biến `repository` giúp tôi."
- **Kết quả**: Bạn không cần viết `new StudentRepository()`, biến `repository` đã sẵn sàng để sử dụng.

3. Data Access Layer (Entity & Repository)

- **Entity**: Tạo class `Student` map với bảng database.
  - **Lưu ý**: `id` sẽ không tự động tăng (để làm quen với việc handle ID manual hoặc UUID về sau).

```
@Entity
@Table(name = "students")
public class Student {
    @Id
    private String id; // Sử dụng String (ví dụ MSSV hoặc UUID) // Không
dùng @GeneratedValue

    private String name;
    private String email;
    private int age;
    // ... Getters/Setters/Constructors
}
```

- **Repository**: Tạo interface `StudentRepository` kế thừa `JpaRepository<Student, String>`.
    - **Thắc mắc**: "Ủa đây là Interface, vậy class implement nó ở đâu?"
    - **Trả lời**: Đây là "ma thuật" của Spring Data JPA. Bạn **KHÔNG CẦN** tạo class implement. Spring sẽ tự động tạo một instance ẩn (Proxy) lúc chạy (Runtime) và thực thi các câu lệnh SQL tương ứng cho bạn.

4. Business Layer (Service)

Tạo class `StudentService` để xử lý logic:

```
@Service
public class StudentService {
    @Autowired
    private StudentRepository repository;

    public List<Student> getAll() {
        return repository.findAll();
    }

    public Student getById(String id) {
        return repository.findById(id).orElse(null);
    }
}
```

5. Controller Layer (API)

Tạo class `StudentController` để tiếp nhận request từ người dùng và gọi xuống Service.

```
@RestController
@RequestMapping("/api/students")
public class StudentController {
    @Autowired
    private StudentService service;

    // 1. API Lấy danh sách: GET http://localhost:8080/api/students
    @GetMapping
    public List<Student> getAllStudents() {
        return service.getAll();
    }
```

```
    // 2. API Lấy chi tiết: GET http://localhost:8080/api/students/{id}
    @GetMapping("/{id}")
    public Student getStudentById(@PathVariable String id) {
        // Lưu ý: Cần thêm method getById trong Service trước
        return service.getById(id);
    }
}
```

Giải thích Annotation:

- `@RestController`: Báo cho Spring biết đây là nơi xử lý API, kết quả trả về sẽ là JSON (thay vì giao diện HTML).
- `@RequestMapping("/api/students")`: Đường dẫn gốc cho tất cả API trong class này.
- `@GetMapping`: Xử lý request `GET`.
- `@PathVariable`: Lấy giá trị `{id}` trên URL bỏ vào biến `id` của hàm.

6. Kiểm Thử Backend (Testing)

Sau khi đã viết code xong (Entity -> Repository -> Service -> Controller), chúng ta cần chạy lại ứng dụng để Spring cập nhật các thay đổi.

6.1. Chạy Ứng Dụng

- **Stop**: Nếu ứng dụng đang chạy ở terminal, nhấn `Ctrl + C` để dừng.
- **Start**: Chạy lại lệnh sau tại thư mục gốc:

```
./mvnw spring-boot:run
```

- **Check Log**: Đảm bảo không có lỗi và thấy dòng `Started StudentManagementApplication`.

6.2. Kiểm Tra API bằng Trình Duyệt (Browser)

Vì phương thức là `GET`, bạn có thể kiểm tra trực tiếp trên trình duyệt (Chrome/Edge/Firefox). Truy cập các đường dẫn sau:

Test Case 1: Lấy danh sách sinh viên (Dữ liệu này đã được tạo ở Lab 1)

- **Method**: `GET`
- **URL**: `http://localhost:8080/api/students`
- **Kết quả mong đợi**:


```
[
    {
        "id": "1",
        "name": "Nguyen Van A",
        "email": "vana@example.com",
        "age": 20
    },
    {
        "id": "2",
        "name": "Tran Thi B",
        "email": "thib@example.com",
        "age": 21
    }
]
```


Test Case 2: Lấy chi tiết sinh viên

- **Method**: `GET`
- **URL**: `http://localhost:8080/api/students/1`
- **Kết quả mong đợi**: Trả về đúng thông tin của sinh viên có id=1.

Test Case 3: Lấy sinh viên không tồn tại

- **Method**: `GET`
- **URL**: `http://localhost:8080/api/students/999`
- **Kết quả mong đợi**: Trả về rỗng (hoặc null tùy logic), nhưng Status Code phải là `200 OK` (chúng ta chưa xử lý Exception handling ở bài này).