Lab Series: Xây Dựng Web App Căn Bản
Lab 2: Xây Dựng Backend REST API

Ở bài thực hành Lab 1, sinh viên đã hoàn tất việc khởi tạo dự án Spring Boot, cấu hình kết nối cơ sở dữ liệu SQLite và tìm hiểu kiến trúc Layered Architecture. Bài thực hành Lab 2 tập trung vào việc hiện thực hóa chi tiết các thành phần trong từng phân lớp ứng dụng để hoàn thiện chức năng quản lý sinh viên.

1 Mục Tiêu

- Hiện thực hóa các thành phần trong kiến trúc: Entity, Repository, Service, Controller.
- Xây dựng API phục vụ truy vấn dữ liệu (Read).
- *Lưu ý: Các chức năng Create, Update, Delete sẽ được thực hiện trong bài thực hành Lab 4.*

Tại sao cần xây dựng API (Lab 2) trước khi phát triển giao diện (Lab 3/4)?

Trong thực tế phát triển phần mềm, 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 định dạng JSON (trong Lab 2) mang lại các lợi ích sau:

1. **Kiểm thử độc lập**: Đảm bảo logic nghiệp vụ (Service/Repository) hoạt động chính xác mà không phụ thuộc vào giao diện người dùng.
2. **Khả năng mở rộng**: Tạo điều kiện thuận lợi cho việc phát triển các ứng dụng đa nền tảng (như Mobile App với React Native/Flutter) thông qua việc tái sử dụng API.

Do đó, Lab 2 tập trung vào việc xây dựng và kiểm chứng tầng xử lý dữ liệu ("lõi" hệ thống). Trong các bài thực hành tiếp theo (Lab 3 & 4), sinh viên sẽ phát triển giao diện người dùng để tương tác với hệ thống này.

2 Thiết Kế API (API Specification)

Trước khi tiến hành hiện thực, cần thống nhất các Endpoint sẽ được 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 thực hiện trong Lab 4)
Lưu ý: Lab 3 hướng dẫn xây dựng giao diện Web (SSR) để hiển thị dữ liệu từ các API này.

3 Cơ Sở Lý Thuyết

Phần này trình bày các khái niệm cốt lõi của Spring Boot được sử dụng trong bài thực hành.

3.1 Dependency Injection (DI) & Inversion of Control (IoC)

Trong mô hình lập trình truyền thống, khi Class A phụ thuộc vào Class B, Class A thường tự khởi tạo B (ví dụ: `B b = new B()`).

- **Vấn đề**: Gây ra sự phụ thuộc chặt chẽ (Tightly Coupled) giữa các thành phần. Bất kỳ thay đổi nào trong việc khởi tạo B đều yêu cầu sửa đổi tại A, đồng thời gây khó khăn cho việc viết Unit Test.
- **Giải pháp (IoC)**: Thay vì tự khởi tạo, Class A chuyển quyền kiểm soát việc tạo instance cho một thành phần quản lý trung gian (Container). Cơ chế cung cấp instance này được gọi là **Dependency Injection (DI)**.

3.2 Annotation `@Autowired`

Annotation `@Autowired` được sử dụng để yêu cầu Spring Framework thực hiện DI.

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

- **Cơ chế hoạt động**: Khi Spring Container khởi chạy, nó sẽ tìm kiếm trong Application Context một instance phù hợp của `StudentRepository` và tự động gán vào biến `repository`.
- **Kết quả**: Lập trình viên không cần thực hiện `new StudentRepository()`, biến `repository` được đảm bảo đã được khởi tạo và sẵn sàng sử dụng.

4 Data Access Layer (Entity & Repository)

- **Entity**: Xây dựng lớp `Student` để ánh xạ (map) với bảng dữ liệu trong cơ sở dữ liệu.
- **Lưu ý**: Thuộc tính `id` được thiết lập không tự động tăng (auto-increment) nhằm mục đích làm quen với việc quản lý ID thủ công hoặc sử dụng UUID trong các bài thực hành nâng cao.

```
@Entity
@Table(name = "students")
public class Student {
    @Id
    private String id; // Su dung String (vi du MSSV hoac UUID) // Khong dung @GeneratedValue

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

- **Repository**: Xây dựng Interface `StudentRepository` kế thừa từ `JpaRepository<Student, String>`.
- **Cơ chế hoạt động**: Spring Data JPA sử dụng cơ chế Dynamic Proxy để tự động tạo một lớp implement cho Interface này tại thời điểm chạy (Runtime).
- **Lợi ích**: Lập trình viên không cần viết code triển khai chi tiết cho các thao tác CRUD cơ bản, Spring sẽ tự động sinh ra các câu lệnh SQL tương ứng.

5 Business Layer (Service)

Xây dựng lớp `StudentService` để đóng gói các logic nghiệp vụ:

```
@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);
    }
}
```

6 Controller Layer (API)

Xây dựng lớp `StudentController` đóng vai trò tiếp nhận các yêu cầu (Request) từ client và điều phối xuống tầng Service.

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

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

    // 2. API Lay chi tiet: GET http://localhost:8080/api/students/{id}
    @GetMapping("/{id}")
    public Student getStudentById(@PathVariable String id) {
        // Luu y: Can them method getById trong Service truoc
        return service.getById(id);
    }
}
```

6.1 Giải thích Annotation

- `@RestController`: Đánh dấu lớp này là một Controller chuyên xử lý các RESTful API. Dữ liệu trả về sẽ có định dạng JSON (thay vì HTML view).
- `@RequestMapping("/api/students")`: Định nghĩa tiền tố đường dẫn (endpoint prefix) chung cho tất cả các API trong Controller này.
- `@GetMapping`: Ánh xạ các HTTP GET Request vào phương thức xử lý tương ứng.
- `@PathVariable`: Trích xuất giá trị từ URL path (ví dụ: `{id}`) và gán vào tham số của phương thức.

7 Kiểm Thử Backend (Testing)

Sau khi hoàn tất việc hiện thực mã nguồn (Entity → Repository → Service → Controller), sinh viên cần khởi động lại ứng dụng để áp dụng các thay đổi.

7.1 Khởi Động Ứng Dụng

1. **Dừng ứng dụng cũ**: Nếu ứng dụng đang chạy, nhấn tổ hợp phím `Ctrl + C` tại Terminal để dừng.
2. **Khởi động lại**: Thực thi lệnh sau tại thư mục gốc của dự án:

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

3. **Kiểm tra Log**: Quan sát Terminal, đảm bảo không xuất hiện lỗi và hiển thị thông báo `Started StudentManagementApplication`.

7.2 Kiểm Tra API với Trình Duyệt

Đối với các API sử dụng phương thức `GET`, sinh viên có thể kiểm thử trực tiếp thông qua trình duyệt web (Chrome, Edge, Firefox, v.v.).

7.2.1 API Lấy danh sách sinh viên

- **Method**: `GET`
- **URL**: `http://localhost:8080/api/students`
- **Mô tả**: Truy xuất danh sách toàn bộ sinh viên (dữ liệu đã được khởi tạo từ Lab 1).
- **Kết quả mong đợi (JSON)**:

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

7.2.2 API Lấy chi tiết sinh viên

- **Method**: `GET`
- **URL**: `http://localhost:8080/api/students/1`
- **Mô tả**: Truy xuất thông tin chi tiết của sinh viên có ID là 1.
- **Kết quả mong đợi**: Trả về Object JSON chứa thông tin sinh viên tương ứng.

7.2.3 API Lấy sinh viên không tồn tại

- **Method**: `GET`
- **URL**: `http://localhost:8080/api/students/999`
- **Mô tả**: Truy xuất thông tin của một sinh viên không tồn tại trong hệ thống.
- **Kết quả mong đợi**: Trả về nội dung rỗng hoặc `null`. Status Code là `200 OK` (do chưa xử lý Exception Handling trong bài thực hành này).