[실습 1] RESTful API 예제 프로젝트 (SpringBootLab01)

RESTful API 를 Spring Boot 로 단계별로 구축해보는 실습을 하려고 한다.

1. com.lab01 패키지에서 단계별로 Controller/Service/Repository 를 설계하고, H2 DB 연동 (JPA 만 사용!)
2. Swagger 를 통해 API 문서화를 확인
3. JUnit 으로 각 API 를 테스트 (@SpringBootTest,@Test 등)
4. JPA 만 사용해서 H2 데이터베이스 연동를 진행

단계별 실행 진행

단계	내용
1 단계	Controller/Service/Repository 구성 및 H2 DB 연결 확인
2 단계	Swagger (springfox-swagger) 연동
3 단계	HATEOAS 를 이용한 하이퍼미디어 링크 추가 연습
핵심 스타터	• spring-boot-starter-web,• spring-boot-starter-data-jpa • spring-boot-starter-h2, spring-boot-starter-test

디렉토리 구조

```
src/
├── main/java/com/lab01/
│   ├── controller/
│   ├── service/
│   ├── entity/
│   ├── repository/
│   ├── config/
│   └── Springbootlab01Application.java
├── main/resources/
│   └── application.yml
└── test/java/com/lab01/
    └── StudentControllerTest.java
```

H2 DB 연결 _application.yml

```
spring:
  datasource:
    url: jdbc:h2:mem:testdb
    driver-class-name: org.h2.Driver
    username: sa
    password:
  h2:
    console:
      enabled: true
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
```

방식

방식	H2 활용 방식	특징
JPA+Hibernate	spring-boot-starter-data-jpa	엔티티/레포지토리 사용, ORM 편리
JDBC 만 사용시	spring-boot-starter-jdbc + H2 driver	직접 SQL, JdbcTemplate 으로 작업

1) Student 엔티티

```
@Entity
@Data
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private int grade;
}
```

2) StudentRepository

```
public interface StudentRepository extends JpaRepository<Student, Long> {

}
```

3) StudentService

메소드명: 리턴타입	설명
findAll(): List<Student>	저장된 모든 Student 데이터를 조회한다.
save(Student student): Student	새로운 Student 를 저장하거나 기존 데이터를 수정한다.
findById(Long id): Student	특정 ID 의 Student 를 조회한다.
delete(Long id): void	특정 ID 의 Student 를 삭제한다.

4) StudentController

기능	메소드명: 리턴타입	HTTP 메소드	URL
학생 목록 조회	getAllStudents(): List<Student>	GET	/students
학생 등록	createStudent(Student student): Student	POST	/students
학생 상세 조회	getStudentById(Long id): Student	GET	/students/{id}
학생 정보 업데이트	updateStudent(Long id, Student student): Student	PUT	/students/{id}
학생 삭제	deleteStudent(Long id): void	DELETE	/students/{id}

5) JUnit

```
@SpringBootTest
public class StudentRepositoryTest {

    @Autowired
    private StudentRepository studentRepository;

    @Test
    public void testSaveStudent() {
        Student student = new Student();
        student.setName("John");
        student.setGrade(2);

        Student savedStudent = studentRepository.save(student);

        assertThat(savedStudent.getId()).isNotNull();
        assertThat(savedStudent.getName()).isEqualTo("John");
    }
}
```

Mock = 가짜 객체

- 실제 객체를 대신해서 테스트를 수행할 수 있도록 만들어진 **테스트용 객체.**
- 테스트에서는 **DB, 네트워크, 외부 시스템**에 실제로 연결하지 않고도 동작을 검증힌다.
- **MockMvc** 는 **스프링 MVC 웹 애플리케이션을 실제로 실행하지 않고** (서버를 띄우지 않고) **HTTP 요청과 응답을 테스트**할 수 있게 해주는 **테스트용 유틸리티**다.

Mock 주요 특징

- 컨트롤러를 통해 요청을 보내고, 실제처럼 **HTTP 요청/응답**을 테스트
- **상태코드**(200, 400 등), **응답 JSON** 등을 검증할 수 있다
- @AutoConfigureMockMvc 어노테이션과 함께 사용
- mockMvc.perform() 메서드를 통해 **GET/POST/PUT/DELETE** 등 다양한 요청 테스트 가능

주요 어노테이션

어노테이션	설명
@SpringBootTest	스프링 부트 애플리케이션 컨텍스트를 로드해서 통합 테스트를 수행한다. 전체 애플리케이션을 테스트할 수 있다.
@AutoConfigureMockMvc	MockMvc 를 자동으로 구성해준다. 컨트롤러의 요청-응답을 테스트할 수 있다.
@Autowired	필요한 스프링 빈(예: MockMvc, StudentRepository)을 주입한다.
@BeforeEach	각 테스트 메소드 실행 전에 실행되는 초기화 메소드에 사용된다.
@Test	JUnit5 에서 테스트 메소드로 지정하는 어노테이션이다.

6) springdoc-openapi 가 자동으로 /swagger-ui/index.html 을 제공하므로 별도 설정 필요 없이 확인

http://localhost:8080/swagger-ui/index.html

7). HATEOAS 를 이용한 하이퍼미디어 링크 추가 후 확인

HATEOAS(Hypermedia As The Engine Of Application State)

- RESTful API 에서 클라이언트가 API 를 통해 어떤 추가 작업을 할 수 있는지를 하이퍼미디어 링크를 통해 안내해주는 기술
- HATEOAS 는 응답 자체에 링크를 포함시켜 다음에 호출할 수 있는 URL, 가능한 액션 등을 제공
- https://docs.spring.io/spring-hateoas/docs/current/reference/html/

[구현 방식]

1. 엔티티나 DTO 를 EntityModel, RepresentationModel 로 감싼다
   - 응답 객체를 EntityModel.of(객체)로 감싸서 하이퍼미디어를 위한 래퍼로 만든다.
2. linkTo()와 methodOn()으로 링크를 생성한다

   - linkTo(): 링크를 생성하는 도구
   - methodOn(): 링크에 들어갈 컨트롤러 메소드 참조
   - linkTo(methodOn(StudentController.class).getStudentById(1)).withSelfRel()

3. 응답에 링크를 포함해서 반환한다

   - EntityModel 또는 CollectionModel 형태로 반환
   - JSON 응답에 _links 필드로 하이퍼미디어 정보가 추가된다

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-hateoas</artifactId>
</dependency>
```

```
// Controller 메소드에서 하이퍼미디어 링크 추가
//기존 코드
@GetMapping
public List<Student> getAllStudents() {
    return studentService.findAll();
}
```

```
// 리턴 타입을 CollectionModel<EntityModel<Student>>로 바꾸고,
 // 학생마다 링크를 추가!
// (1) 전체 학생 목록 조회 - HATEOAS 링크 포함
 @GetMapping
 public CollectionModel<EntityModel<Student>> getAllStudents() {
     List<Student> students = studentService.findAll();

     List<EntityModel<Student>> studentModels = students.stream()
         .map(student -> EntityModel.of(student,
                 linkTo(methodOn(StudentController.class)
                             .getStudentById(student.getId())).withSelfRel(),
                 linkTo(methodOn(StudentController.class)
                             .getAllStudents()).withRel("all-students")
         ))
         .toList();

     return CollectionModel.of(studentModels,
             linkTo(methodOn(StudentController.class).getAllStudents()).withSelfRel());
 }
```

```
  // (2) 특정 학생 조회 - HATEOAS 링크 포함
  @GetMapping("/{id}")
  public EntityModel<Student> getStudentById(@PathVariable Long id) {
      Student student = studentService.getStudentById(id);
      return EntityModel.of(student,
              WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder
                                       .methodOn(StudentController.class)
                                       .getStudentById(id)).withSelfRel(),
              WebMvcLinkBuilder.linkTo(WebMvcLinkBuilder.
                                        methodOn(StudentController.class)
                                        .getAllStudents()).withRel("students"));
  }
```

8) /students 요청 시 각 학생마다 하이퍼미디어 링크가 포함된 JSON 확인

[추가 실습 01] 학생 등록 기능 테스트 추가하기

요구사항

1. POST /students 엔드포인트를 호출해서 새로운 학생을 등록하는 테스트 코드를 작성한다
2. 요청 바디는 {"name": "홍길동", "email": "hong@test.com"} 형식의 JSON 이다.
3. 등록이 성공하면 **상태코드 201 (Created)**를 기대한다.
4. 응답으로 등록된 학생의 정보가 반환되는지 검증한다.
5. StudentControllerTest 클래스에 작성한다.
6. src/test/java/com.lab01.controller.StudentControllerTest

```
@Test
void createStudent() throws Exception {
    String newStudentJson = "{ ₩"name₩": ₩"홍길동₩", ₩"email₩":
₩"hong@test.com₩" }";

    mockMvc.perform(post("/students")
            .contentType(MediaType.APPLICATION_JSON)
            .content(newStudentJson))
            .andExpect(status().isCreated())
            .andExpect(jsonPath("$.name").value("홍길동"))
            .andExpect(jsonPath("$.email").value("hong@test.com"));
}
```

[추가 실습 02] 학생 정보 수정 기능 테스트 추가하기

요구사항

1. PUT /students/{id} 엔드포인트를 호출해서 학생의 이메일을 수정하는 테스트 코드를 작성한다.
2. 요청 바디는 {"email": "newemail@test.com"} 형식의 JSON 이다.
3. 수정 후 응답으로 반환된 학생의 이메일이 요청과 같은지 검증한다.
4. 아래 코드를 참조해서 StudentControllerTest 클래스에 작성한다.
5. src/test/java/com.lab01.controller.StudentControllerTest

```
mockMvc.perform(put("/students/{id}", savedStudent.getId())
                .contentType(MediaType.APPLICATION_JSON)
                .content(updatedJson))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.name").value("김길동"))
                .andExpect(jsonPath("$.email").value("kim@test.com"));
```

HATEOAS 의 주요 메소드 (Spring HATEOAS 기준)

메소드	역할 및 설명
EntityModel.of()	단일 리소스(객체)를 감싸는 EntityModel 을 생성한다.
CollectionModel.of()	여러 리소스(컬렉션)를 감싸는 CollectionModel 을 생성한다.
linkTo()	컨트롤러 메소드와 관련된 Link 를 생성한다.
methodOn()	컨트롤러 메소드를 참조하기 위한 프록시 객체를 반환한다. (linkTo()와 함께 사용)
withSelfRel()	현재 리소스의 self 링크를 추가한다.
withRel(String rel)	특정 관계(Relation) 이름을 가진 링크를 추가한다.

주요 andExpect 메소드 종류

메소드	설명
andExpect(status().isOk())	응답 상태 코드 검증 (200 OK)
andExpect(status().isCreated())	응답 상태 코드 검증 (201 Created)
andExpect(jsonPath(...))	JSON 응답의 특정 필드나 값 검증
andExpect(content().string())	응답 바디 문자열 전체 비교
andExpect(header().string())	응답 헤더 검증