

[실습 1] JSON 기반 RESTful API 예제 프로젝트 (SpringBootLab02)

주요 구성 요소

단계	내용	비고
1	H2 DB 연동 & JPA 만 사용	- application.yml 에서 H2 설정 - spring-boot-starter-data-jpa, spring-boot-starter-h2 의존성
2	Validation 및 Exception Handling 학습	- @Valid, @NotBlank 등으로 입력값 검증 - @ControllerAdvice 로 글로벌 예외 처리
3	API Versioning 예제 (예: /v1/members, v2/members)	- URL Path Versioning 방식 - 컨트롤러를 버전별로 나눠서 관리
4	JUnit 테스트	- @SpringBootTest , - @Test - @AutoConfigureMockMvc : MockMvc 테스트 자동 설정

디렉토리 구조

SpringBootLab02/
├── src/
│ ├── main/
│ │ ├── java/com/lab02/
│ │ │ ├── controller/
│ │ │ ├── service/
│ │ │ ├── entity/
│ │ │ ├── repository/
│ │ │ ├── config/
│ │ │ └── SpringBootLab02Application.java
│ │ └── resources/application.yml
│ └── test/
│ └── java/com/lab02/StudentControllerTest.java
└── pom.xml

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
```

1) Member 엔티티

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

    @NotBlank(message = "이름은 필수입니다.")
    private String name;

    @NotBlank(message = "이메일은 필수입니다.")
    @Email(message = "올바른 이메일 형식이어야 합니다.")
    private String email;
}
```

2) MemberRepository

```
@Repository
public interface MemberRepository extends JpaRepository<Member, Long> {
    // 필요하면 커스텀 쿼리 메소드 추가 가능
}
```

3) Service: MemberService

기능	메소드명: 리턴타입	설명
모든 회원 조회	getAllMembers(): List<Member>	전체 회원 목록을 조회한다.
회원 ID 로 단일 조회	getMemberById(Long id): Member	특정 ID 의 회원을 조회한다.
회원 저장	saveMember(Member member): Member	새로운 회원 정보를 저장한다.
회원 정보 수정	updateMember(Long id, Member member): Member	기존 회원의 정보를 수정한다.
회원 삭제	deleteMember(Long id): void	특정 ID 의 회원을 삭제한다.

4) MemberController : @RestController 어노테이션 사용 (JSON 응답)

기능	메소드명: 리턴타입	HTTP 메소드	URL
전체 회원 조회	getAllMembers(): List<Member>	GET	/members
단일 회원 조회	getMemberById(Long id): Member	GET	/members/{id}
회원 등록	createMember(Member member): Member	POST	/members
회원 수정	updateMember(Long id, Member member): Member	PUT	/members/{id}
회원 삭제	deleteMember(Long id): void	DELETE	/members/{id}

4) Validation 및 Exception Handling

1. **Member 엔티티**에 이미 @NotBlank, @Email 등으로 제약 조건이 정의되어 있음
2. **컨트롤러의 파라미터**에는 @Valid 를 붙여서 유효성 검증을 수행
3. 검증 실패 시, **MethodArgumentNotValidException** 발생 → **Exception 처리**로 해결!

단계	내용
1	Validation: 요청 데이터 유효성 검증을 위해 javax.validation 어노테이션(@NotNull, @Email, @Size 등)과 @Valid 를 사용한다.
2	Exception Handling: 발생 가능한 예외를 @RestControllerAdvice 나 @ControllerAdvice 에서 처리한다.
3	일관된 에러 응답 포맷: 예외 발생 시 표준 JSON 형태로 에러 메시지를 반환한다.

javax.validation.constraints 패키지 주요 어노테이션

어노테이션	설명
@NotNull	null 을 허용하지 않음 (" "은 허용)
@NotBlank	null 및 공백문자(" ")를 허용하지 않음 (문자열 전용)
@NotEmpty	null 및 ""(빈 문자열, 공백은 허용)을 허용하지 않음
@Size(min=, max=)	문자열/컬렉션/배열 크기 제한
@Min(value=)	숫자 최소값 지정
@Max(value=)	숫자 최대값 지정
@Positive	0 보다 큰 숫자만 허용
@PositiveOrZero	0 이상만 허용
@Negative	0 보다 작은 숫자만 허용
@NegativeOrZero	0 이하만 허용
@Pattern(regexp="정규식")	문자열 정규식 검사
@Email	이메일 형식인지 검사 (@ 및 기본 이메일 구조)
@Future / @Past	날짜/시간이 미래인지, 과거인지 검사

Exception Handling : @RestControllerAdvice 와 @ExceptionHandler 를 사용

src/main/java/com/lab02/exception/GlobalExceptionHandler.java 패키지.class 추가

@RestControllerAdvice

public class GlobalExceptionHandler {

 //(1) 유효성 검증 실패 처리

 @ExceptionHandler(MethodArgumentNotValidException.class)

 public ResponseEntity<String>

 handleValidationException(MethodArgumentNotValidException ex) {

 String errorMessage = ex.getBindingResult()

 .getAllErrors()

 .get(0)

 .getDefaultMessage();

 return ResponseEntity

 .badRequest()

 .body(errorMessage);

 }

 // (2) 엔티티를 찾지 못했을 때 처리

 @ExceptionHandler(EntityNotFoundException.class)

 public ResponseEntity<String> handleEntityNotFound(EntityNotFoundException
ex) {

 return ResponseEntity

 .status(HttpStatus.NOT_FOUND)

 .body(ex.getMessage());

 }

 // (3) 그 외의 예외 처리

 @ExceptionHandler(Exception.class)

 public ResponseEntity<String> handleOtherExceptions(Exception ex) {

 return ResponseEntity

 .status(HttpStatus.INTERNAL_SERVER_ERROR)

 .body("알 수 없는 오류가 발생했습니다: " + ex.getMessage());

 }

}

5) JUnit : Validation 검증 MemberControllerTest.java

테스트 목적	메서드	기대 결과
정상 등록 테스트	createMember_성공()	201 Created, JSON 응답 검증
이름 누락 예외	createMember_실패_이름누락()	400 Bad Request, 이름 오류 메시지
이메일 형식 오류	createMember_실패_이메일형식()	400 Bad Request, 이메일 오류 메시지

```

@Test
void createMember_성공() throws Exception {
    String requestBody = "{\"name\":\"홍길동\", \"email\":\"hong@test.com\"}";

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

```

```

@Test
void createMember_실패_이름누락() throws Exception {
    String requestBody = "{\"email\":\"hong@test.com\"}";

    mockMvc.perform(post("/members")
        .contentType(MediaType.APPLICATION_JSON)
        .content(requestBody))
        .andExpect(status().isBadRequest())
        .andExpect(jsonPath("$.message").value("이름은 필수입니다."));
}

```

```
@Test
void createMember_실패_이메일형식() throws Exception {
    String requestBody = "{\n" + "name\n" + "홍길동\n" + ",\n" + "email\n" + "wrong-email\n" + "}";

    mockMvc.perform(post("/members")
        .contentType(MediaType.APPLICATION_JSON)
        .content(requestBody))
        .andExpect(status().isBadRequest())
        .andExpect(jsonPath("$.message").value("올바른 이메일 형식이어야 합니다."));
}
```

[실습 2] API Versioning 예제 (URL Path Versioning)

/v1/members, /v2/members 로 버전별로 컨트롤러를 나눠서 관리한다.

디렉토리 구조

```
src/main/java/com/lab02/
├── controller/
│   ├── v1/
│   │   └── MemberControllerV1.java // 기존 엔티티 전체 조회
│   ├── v2/
│   │   └── MemberControllerV2.java // /v2/members - DTO 로만 id, name 조회
```

1) V2 - MemberControllerV2.java

```
@RestController
@RequestMapping("/v2/members")
public class MemberControllerV2 {
    private final MemberService memberService;

    public MemberControllerV2(MemberService memberService) {
        this.memberService = memberService;
    }

    @GetMapping
    public List<MemberDto> getAllMembers() {
        // V2 에서는 DTO 로 응답
        List<Member> members = memberService.getAllMembers();
        return members.stream()
            .map(member -> new MemberDto(member.getId(),
                member.getName()))
            .collect(Collectors.toList());
    }

    // 그 외 메서드는 V1 과 동일하게 구현 (예제 간단화)
}
```


2) DTO 정의 (V2 용)

```
@Data
@AllArgsConstructor
public class MemberDto {
    private Long id;
    private String name;
}
```

3) API Versioning (V1, V2) 컨트롤러 JUnit 테스트

src/test/java/com/lab02/MemberControllerVersioningTest.java

```
@Test
void getAllMembersV1() throws Exception {
    mockMvc.perform(get("/v1/members")
        .accept(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$").isArray());
}

@Test
void getAllMembersV2() throws Exception {
    mockMvc.perform(get("/v2/members")
        .accept(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        // V2 에서는 DTO 만 반환하므로 id 와 name 만 검증
        .andExpect(jsonPath("$[0].id").exists())
        .andExpect(jsonPath("$[0].name").exists());
}
}
```

[추가 실습] 모든 예외가 일관된 JSON 응답으로 처리하도록 해보자.**src/main/java/com/lab02/exception/ErrorResponse.java 생성**

```
@Data
@AllArgsConstructor
public class ErrorResponse {
    private int status;
    private String message;
    private LocalDateTime timestamp;
}
```

GlobalExceptionHandler 수정

```
@RestControllerAdvice
public class GlobalExceptionHandler {

    // (1) 유효성 검증 실패 처리
    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<ErrorResponse>
    handleValidationException(MethodArgumentNotValidException ex) {
        String errorMessage =
ex.getBindingResult().getAllErrors().get(0).getDefaultMessage();
        ErrorResponse errorResponse = new ErrorResponse(
            HttpStatus.BAD_REQUEST.value(),
            errorMessage,
            LocalDateTime.now()
        );
        return ResponseEntity.badRequest().body(errorResponse);
    }

    // (2) 엔티티를 찾지 못했을 때 처리
    @ExceptionHandler(EntityNotFoundException.class)
    public ResponseEntity<ErrorResponse>
    handleEntityNotFound(EntityNotFoundException ex) {
```

```
        ErrorResponse errorResponse = new ErrorResponse(
            HttpStatus.NOT_FOUND.value(),
            ex.getMessage(),
            LocalDateTime.now()
        );
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(errorResponse);
    }

    // (3) 그 외의 예외 처리
    @ExceptionHandler(Exception.class)
    public ResponseEntity<ErrorResponse> handleOtherExceptions(Exception ex) {
        ErrorResponse errorResponse = new ErrorResponse(
            HttpStatus.INTERNAL_SERVER_ERROR.value(),
            "알 수 없는 오류가 발생했습니다: " + ex.getMessage(),
            LocalDateTime.now()
        );
        return
        ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(errorResponse);
    }
}
```

클라이언트가 잘못된 데이터를 보낼 경우 확인

```
{
    "status": 400,
    "message": "이름은 필수입니다.",
    "timestamp": "2025-05-25T15:00:00"
}
```