



고용노동부



한국산업인력공단



국가인적자원개발컨소시엄



Spring Boot 기반 REST API 구현과 JWT인증

2024.1.24 ~ 1.25

백명숙



과정목표

- 본 과정은 Spring Boot 기반 REST API 이론과 실습을 통해서 REST API를 업무에 적용할 수 있는 역량을 기르는 과정입니다.

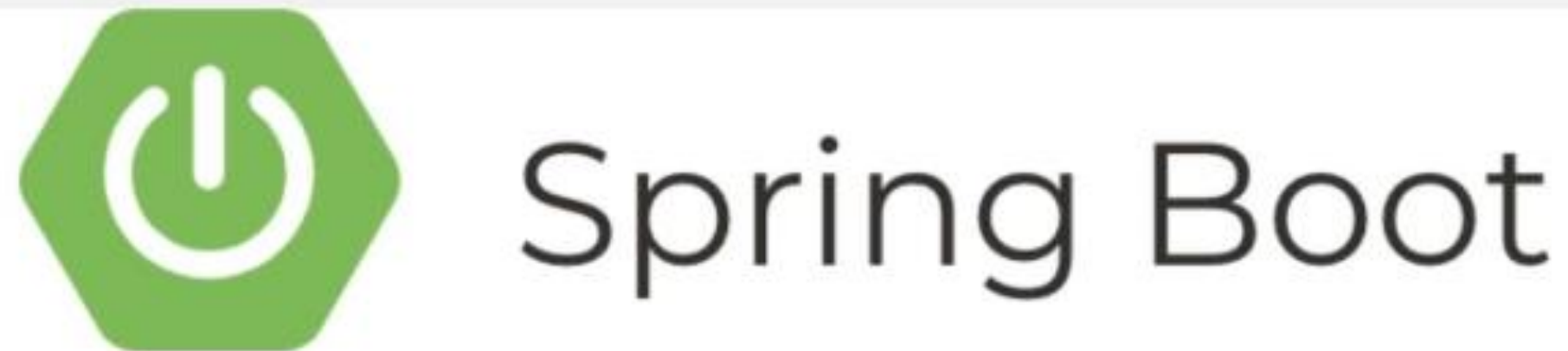
러닝 맵





LO	커리큘럼
REST API 개요	<ul style="list-style-type: none"> - REST API 소개 - 개발환경 구축
REST API 개발	<ul style="list-style-type: none"> - REST API 등록 기능 구현 - Spring Data JPA (H2, MariaDB) - ModelMapper 사용 - 입력항목 검증 Bad Request 와 Validation
HATEOAS 와 Self-Descriptive Message 적용	<ul style="list-style-type: none"> - Spring HATEOAS 소개 - Spring HATEOAS를 적용한 상태 전이 - REST API 조회 (페이징 처리), 수정 기능 구현
REST API 보안 적용	<ul style="list-style-type: none"> - Spring Security 기본 설정 - JWT 인증토큰생성/ 인증토큰 체크 필터 구현하기 - Access Token으로 인증한 사용자 정보 처리하기
REST API Docs	<ul style="list-style-type: none"> - Springdoc-OpenApi 사용하기 - Swagger-ui 설정하기

스프링 부트 기반 REST API



스웨|거



사용된 다양한 기술들

- 학습목표
 - Self-Descriptive Message와 HATEOAS를 만족하는 REST API를 이해합니다.
 - 다양한 스프링 기술을 활용하여 REST API를 개발할 수 있습니다.
- 스프링 Boot
- 스프링 Core, Data JPA, Validation
- 스프링 HATEOAS
- 스프링 Security와 JWT
- REST Docs – Spring Doc

사용된 다양한 기술들

- API Docs Urls

- Spring Framework

<https://docs.spring.io/spring-framework/docs/current/javadoc-api/index.html>

- Spring Boot

<https://docs.spring.io/spring-boot/docs/current/api/>

- Spring Data Core

<https://docs.spring.io/spring-data/commons/docs/current/api/>

- Spring Data JPA

<https://docs.spring.io/spring-data/data-jpa/docs/current/api/>

- Spring HATEOAS API docs

<https://docs.spring.io/spring-hateoas/docs/current/api/>

- Spring Security

<https://docs.spring.io/spring-security/site/docs/current/api/>

- java jwt-api

<https://javadoc.io/doc/io.jsonwebtoken/jjwt-api/latest/index.html>

- Jakarta API docs

<https://jakarta.ee/specifications/platform/9/apidocs/>

REST API

- API (Application Programming Interface)

어떠한 응용 프로그램에서 데이터를 주고 받기 위한 방법을 의미합니다. 어떤 특정 사이트에서 데이터를 공유할 경우 어떠한 방식으로 정보를 요청해야 하는지, 그리고 어떠한 데이터를 제공할 수 있을지에 대한 규격들을 API라고 하는 것입니다.

- REST (REpresentational State Transfer)

HTTP/1.0과 1.1의 스펙 작성에 참여 하였고 아파치 HTTP 서버 프로젝트의 공동설립자인 로이 필딩 (Roy Fielding)의 2000년 논문에서 처음 소개 되었다. 발표 당시의 웹이 HTTP의 설계의 우수성을 제대로 사용하지 못하고 있는 상황을 보고, 웹의 장점을 최대한 활용할 수 있는 아키텍처로서 REST를 소개 하였고, REST는 HTTP 프로토콜의 의도에 맞게 디자인 하도록 유도하고 있습니다.

REST API

- REST (REpresentational State Transfer)란 무엇입니까?

REST는 분산 시스템 설계를 위한 아키텍처 스타일이다. 아키텍처 스타일이라는 것은 제약 조건의 집합이라고 보면 됩니다.

- RESTful은 무엇입니까?

RESTful은 위의 제약 조건의 집합 (아키텍처 원칙)을 모두 만족하는 것을 의미합니다.

REST라는 아키텍처 스타일이 있고, RESTful API라는 말은 REST 아키텍처 원칙을 모두 만족하는 API이라는 뜻입니다.

REST API

- REST가 필요한 이유는 무엇입니까?

1. 분산시스템을 위해서 필요합니다.

거대한 어플리케이션을 모듈, 기능별로 분리하기 쉬워졌다. RESTful API를 서비스 하면 어떤 다른 모듈 또는 어플리케이션들이라도 RESTful API를 통해서 상호간에 통신을 할 수 있기 때문입니다.

2. Web 브라우저 이외의 클라이언트를 위해서 필요합니다.

웹페이지를 위한 HTML 및 이미지 등을 보내던 것과는 다르게, 데이터만 보내면 여러 클라이언트에서 해당 데이터를 주고 받기 때문에 자유롭고 부담없이 데이터를 이용할 수 있으며 서버도 요청한 데이터를 보내기 만 하므로 가벼워지고 유지 보수성도 좋아질 수 있습니다.

REST 구성요소

- REST의 구성요소?

HTTP URI (자원) + HTTP Method (행위)

URI는 정보의 자원을 표현해야 한다. – Resource명은 동사 보다는 명사를 사용합니다.

자원에 대한 행위는 HTTP Method (GET, POST, PUT, DELETE)로 표현한다.

```
# bad
GET /getTodos/1
GET /todos/show/1

# good
GET /todos/1
```

```
# bad
GET /todos/delete/1

# good
DELETE /todos/1
```

REST 제약조건

■ REST의 제약조건

1. Client / Server 구조 : 클라이언트와 서버가 서로 독립적이어서 별도의 진화가 가능하다.
2. Stateless (무상태) : 서버에서 클라이언트의 세션과 쿠키와 같은 context를 저장하지 않으므로 구현이 단순함
3. Cache (캐시 처리 가능) : HTTP가 가진 캐시 처리 기능을 그대로 적용 할 수 있음.
4. Layered System (계층화) : REST 서버는 다중 계층으로 구성될 수 있으며 보안, 로드 밸런싱, 암호화 계층을 추가해 구조상의 유연성을 둘 수 있음.
5. Uniform Interface (인터페이스 일관성) : 자원은 유일하게 식별 가능해야 하고, HTTP Method로 표현을 담아야 하며, 메시지는 스스로 설명 (self-descriptive) 해야 하고, 하이퍼링크를 통해서 어플리케이션의 상태가 전이(HATEOAS) 되어야 합니다.

REST 제약조건

- Uniform Interface의 제약조건

1. Identification of Resources : 리소스가 URI로 식별됩니다.
2. Manipulation of Resources through Representations : 리소스를 삭제하거나 수정할 때 HTTP 메시지에 이러한 표현을 담아서 전송해야 합니다.
3. Self-Descriptive Messages
4. Hypermedia As The Engine Of Application State(HATEOAS)

- Self-Descriptive Messages

1. 메시지 스스로 메시지에 대한 설명이 가능해야 합니다.
2. API 문서가 REST API 응답 본문에 존재해야 한다는 것이다.
: 즉, API 문서가 어디 있는지는 알려줘야 한다.

HATEOAS

- HATEOAS (**H**ypermedia **A**s **T**he **E**ngine **O**f **A**pplication **S**tate)
 1. 링크에 사용 가능한 URL을 리소스로 전달하여 client가 참고하여 사용할 수 있도록 하는 것
 2. 하이퍼미디어(링크)를 통해 애플리케이션 상태 변화가 가능해야 합니다.
 3. Hypermedia (링크)에 자기 자신에 대한 정보가 포함 되어야 한다.

전형적인 REST API의 응답 데이터

```
1 {  
2   "accountId":12345,  
3   "accountType":"saving",  
4   "balance":350000,  
5   "currency":"KRW"  
6 }
```

HATEAOS가 도입되어 자원에 대한 추가 정보가 제공되는 응답 데이터

```
1 {  
2   "accountId":12345,  
3   "accountType":"saving",  
4   "balance":350000,  
5   "currency":"KRW"  
6   "links": [  
7     {  
8       "rel": "self"  
9       "href": "http://localhost:8080/accounts/1"  
10    },  
11    {  
12      "rel": "withdraw",  
13      "href": "http://localhost:8080/accounts/1/withdraw"  
14    },  
15    {  
16      "rel": "transfer",  
17      "href": "http://localhost:8080/accounts/1/transfer"  
18    }  
19  ]  
20 }
```


HATEOAS

- Self-Descriptive Messages 해결 방법

방법1 : 미디어 타입을 정의하고 IANA에 등록하고 그 미디어 타입을 리소스를 리턴 할 때 Content-Type으로 사용합니다.

방법2 : profile 링크 헤더를 추가합니다.

브라우저들이 아직 스펙을 지원하지 않는다.

대안으로 HAL(Hypertext Application Language)의 링크 데이터 profile 링크를 추가합니다.

HAL (https://en.wikipedia.org/wiki/Hypertext_Application_Language)

- HATEOAS 해결 방법

방법1 : 데이터에 링크를 제공합니다. (HAL)

방법2 : 링크 헤더나 Location을 제공합니다.

HAL

- HAL (Hypertext Application Language)

HAL은 JSON, XMI 코드 내의 외부 리소스에 대한 링크를 추가하기 위한 특별한 데이터 타입이다.

HAL의 최초 제안자인 Mike Kelly의 HAL에 대한 정의

HAL은 API의 리소스들 사이에 쉽고 일관적인 하이퍼링크를 제공하는 방식이다. API 설계 시 HAL을 도입하면 API간에 쉽게 검색이 가능하다. 따라서 해당 API를 사용하는 다른 개발자들에게 좀 더 나은 개발 경험을 제공한다.

HAL은 두 개의 콘텐츠 타입을 갖는다.

- application/hal+json
- application/hal+xml

이 HAL 타입을 이용하면 쉽게 HATEOAS를 달성할 수 있다.

HAL

■ HAL (Hypertext Application Language) Model

Resources

: Data 필드에 해당한다.

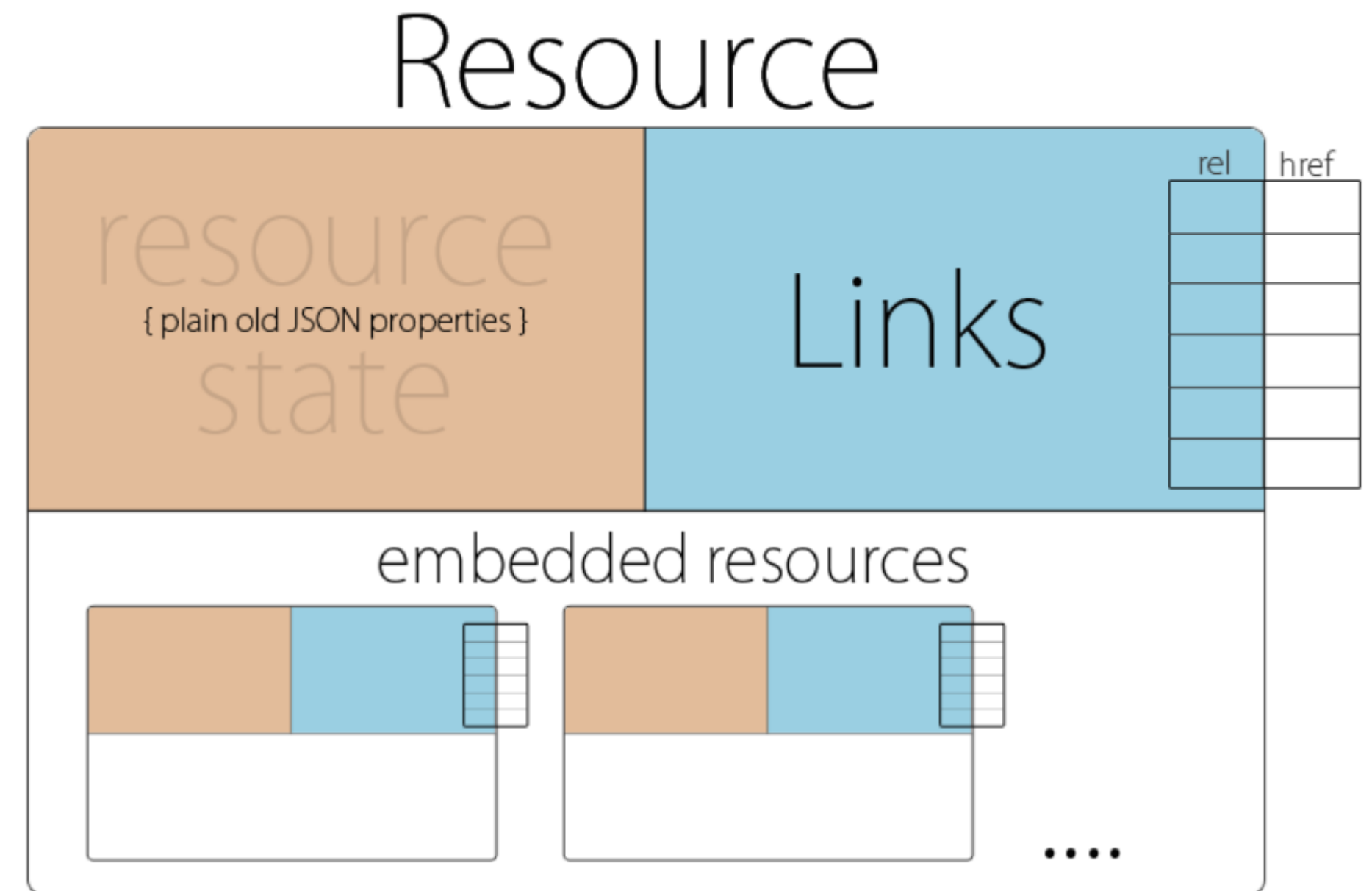
- Links (to URIs)
- Embedded Resources
- State (your bog standard JSON or XML data)

Links

: 하이퍼미디어로 _self 필드가 링크 필드가 된다.

- A target (a URI)
- A relation aka. 'rel' (the name of the link)
- A few other optional properties to help with deprecation, content negotiation, etc.

http://stateless.co/hal_specification.html



HAL

■ JSON 응답 본문 예시

```
{
  "data": { // HAL JSON의 리소스 필드
    "id": 1000,
    "name": "게시글 1",
    "content": "HAL JSON을 이용한 예시 JSON"
  },
  "_links": { // HAL JSON의 링크 필드
    "self": {
      "href": "http://localhost:8080/api/article/1000" // 현재 api 주소
    },
    "profile": {
      "href": "http://localhost:8080/docs#query-article" // 해당 api의 문서
    },
    "next": {
      "href": "http://localhost:8080/api/article/1001" // article 의 다음 api 주소
    },
    "prev": {
      "href": "http://localhost:8080/api/article/999" // article의 이전 api 주소
    }
  }
}
```

REST API 개발

Spring Boot Project 만들기

- Java 버전 17로 시작
- 스프링 부트 프로젝트 생성 (<https://start.spring.io/>)
- 추가할 의존성 (dependency)
 - Web
 - Data JPA
 - HATEOAS
 - Validation
 - H2 / MariaDB Client
 - Lombok
 - Devtools
 - Configuration Processor

REST API : Lecture 생성 API의 구현

▪ Lecture 생성 API의 구현

Lecture 클래스 : 다음의 입력 값을 받는다.

- name : 강의 이름
- description : 강의에 대한 설명
- beginEnrollmentDateTime : 등록 시작 일시
- closeEnrollmentDateTime : 등록 종료 일시
- beginLectureDateTime : 강의 시작 일시
- endLectureDateTime : 강의 종료 일시
- Location (optional) : 강의 장소 , 이 속성이 없으면 온라인 강의
- basePrice (optional) : 강의 참가비 (등록비)
- maxPrice (optional) : 강의 참가비
- limitOfEnrollment : 강의를 수강하는 제한된 인원수

REST API : Lecture 생성 API의 구현

- Lecture 도메인(엔티티) 클래스

lectures/Lecture.java

```
public class Lecture {  
    private Integer id;  
    private String name;  
    private String description;  
    private LocalDateTime beginEnrollmentDateTime;  
    private LocalDateTime closeEnrollmentDateTime;  
    private LocalDateTime beginLectureDateTime;  
    private LocalDateTime endLectureDateTime;  
    private String location;  
    private int basePrice;  
    private int maxPrice;  
    private int limitOfEnrollment;  
    private boolean offline;  
    private boolean free;  
    private LectureStatus lectureStatus = LectureStatus.DRAFT;  
}
```

REST API : Lecture 생성 API의 구현

- LectureStatus Enum 클래스

lectures/LectureStatus.java

```
public enum LectureStatus {  
  
    DRAFT, PUBLISHED, BEGAN_ENROLLMENT  
}
```

<https://gist.github.com/mysoyul/6b1e3f2b786e3f227016276132005ab7>

REST API : Lombok

- Lombok 이란?

자바에서 Model(DTO, VO, Entity) Object 를 작성할 때, 멤버필드(프로퍼티)에 대한 Getter/Setter, ToString과 멤버 필드에 주입하는 생성자 등 반복적으로 만드는 코드들을 어노테이션(@)을 통해 줄여 주는 라이브러리입니다.

- 동작원리

Lombok annotation이 부여된 Java Source를 컴파일 할 때 등록된 Lombok processor가 어노테이션을 확인하고, 그에 맞는 메서드를 자동으로 생성하여 bytecode로 변환시켜 줍니다.

- 단점

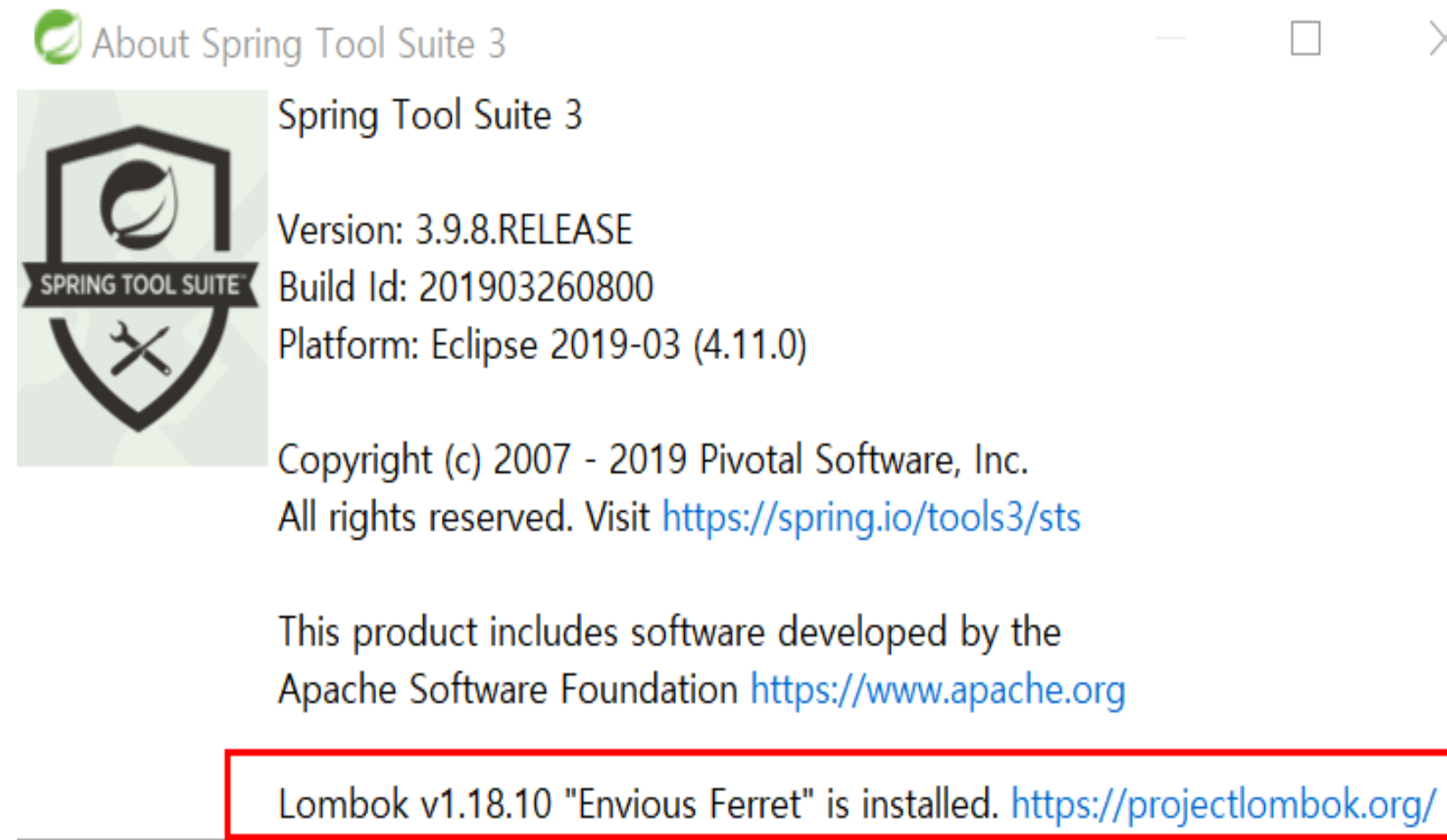
협업 모든 인원이 Lombok을 설치해야 하고, 추가적인 어노테이션을 사용 할 경우 소스코드 분석이 어려울 수도 있습니다.

REST API : Lombok

- Eclipse (STS) 에 Lombok 설치하기 (IntelliJ IDEA 를 사용하는 경우는 Skip 합니다.)
 1. lombok.jar가 다운로드 된 경로로 가서 커맨드(cmd) 창을 연다.
예를 들면 C:\Users\user\m2\repository\org\projectlombok\lombok\1.16.18
 2. lombok.jar 실행 (버전은 달라 질 수 있음)
> java -jar lombok-1.16.18.jar
 3. Installer 화면이 나오면 **Specify location** 버튼을 클릭
 4. sts-bundle 폴더에 있는 실행 파일(sts.exe) 선택
 5. Install / Update 클릭
 6. Eclipse 재 시작 해야 하고, -vm parameter로 -vmargs -javaagent: lombok.jar가 추가 되었다는 메시지가 나옴
 7. STS 폴더에 있는 STS.ini (혹은 eclipse.ini) 파일을 열어 보면 마지막에 추가 되어 있는 것을 확인

REST API : Lombok

- Lombok 설치 확인 (IntelliJ IDEA 를 사용하는 경우는 Skip 합니다.)
 - Help -> About Spring Tool Suite3 에서 확인합니다.



REST API : Lombok

- Lombok이 제공하는 어노테이션

@어노테이션	설명	세부 기능
@Getter, @Setter	getter, setter 메서드 자동생성	accessLevel : 해당 접근 제한자를 설정 lazy : 동기화를 이용하여 최초 1회만 호출
@ToString	toString 메서드 자동생성	exclude : 출력하지 않을 필드명 입력 callSuper : 상위클래스 toString 호출 여부 설정
@EqualsAndHashCode	equals, hashCode 자동생성	
@NoArgsConstructor	인자가 없는 기본 생성자 생성	
@RequiredArgsConstructor	final 변수만 있는 생성자 생성	
@AllArgsConstructor	모든 인자를 가진 생성자 생성	
@Data	@ToString, @EqualsAndHashCode , @Getter, @Setter, @RequiredArgsConstructor 자동생성	
@Builder	Builder 패턴을 적용하여 자동생성	

Lecture 도메인 구현

1. Lombok의 어노테이션 사용

- @Builder Annotation은 Class에 대한 복잡한 Builder API들을 자동으로 생성해 준다.
- Builder 패턴은 객체를 생성할 때 필수적인 인자와 선택적인 인자를 구별할 수 있는 장점이 있다.

lectures/Lecture.java

```
@Builder @AllArgsConstructor @NoArgsConstructor  
@Getter @Setter @EqualsAndHashCode(of="id")  
public class Lecture {  
    ....  
}
```

빌더 패턴

https://ko.wikipedia.org/wiki/%EB%B9%8C%EB%8D%94_%ED%8C%A8%ED%84%B4

Lecture 도메인 구현

2. Lombok을 사용한 Lecture 클래스 테스트

src/test/./lectures/LectureTest.java

```
import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.jupiter.api.Assertions.*;
public class LectureTest {
    @Test
    public void builder() {
        Lecture lecture = Lecture.builder()
            .name("Spring REST API")
            .description("REST API developmemt with Spring")
            .build();
        assertEquals("Spring REST API", lecture.getName());
    }
    @Test
    public void javaBean() {
        String name = "Lecture";           //Give
        String description = "Spring";
        Lecture lecture = new Lecture();    //When
        lecture.setName(name);
        lecture.setDescription(description);
        assertThat(lecture.getName()).isEqualTo("Lecture"); //Then
        assertThat(lecture.getDescription()).isEqualTo("Spring");
    }
}
```

Lecture 생성 API 구현 : 201 응답 받기

- 테스트 해야 할 내용
 - 입력 값 들을 전달하면 JSON 응답으로 201 결과값이 나오는지 확인
 - Http Status Code 201은 POST 나 PUT 으로 게시물 작성이나 회원 가입 등의 새로운 데이터를 서버에 생성하는(쓰는,등록) 작업이 성공했을 때 반환되는 코드 값 입니다.

<https://restfulapi.net/http-status-201-created/>

: Location 헤더에 생성된 이벤트를 조회할 수 있는 URI 가 담겨 있는지 확인 합니다.

: Id는 DB에 저장된 후에 자동 생성된 Id 값으로 나오는지 확인합니다.

Lecture 생성 API 구현 : 201 응답 받기 #1

- LectureController 클래스
 - ResponseEntity를 사용하는 이유
 - : 응답 코드, 헤더, 본문을 모두 저장 해주는 편리한 API
 - Location URI 만들기
 - : Spring HATEOAS WebMvcLinkBuilder의 `linkTo()`, `methodOn()` 사용

lectures/LectureController.java

```
import org.springframework.hateoas.server.mvc.WebMvcLinkBuilder;
import java.net.URI;

@RestController
@RequestMapping(value="/api/lectures", produces = MediaType.HAL_JSON_VALUE)
public class LectureController {

    @PostMapping
    public ResponseEntity createLecture(@RequestBody Lecture lecture) {

        lecture.setId(10);
        WebMvcLinkBuilder selfLinkBuilder = WebMvcLinkBuilder.linkTo(LectureController.class).slash(lecture.getId());
        URI createUri = selfLinkBuilder.toUri();
        return ResponseEntity.created(createUri).body(lecture);
    }
}
```

Lecture 생성 API 구현 : @JsonFormat 어노테이션

■ @JsonFormat 어노테이션 사용하기

: @JsonFormat은 Jackson 라이브러리에서 제공하는 어노테이션으로 JSON 응답값의 형식을 지정할 때 사용한다.
@JsonFormat을 이용해서 날짜 형식을 지정하려면 아래와 같이 사용하면 됩니다.

lectures/Lecture.java

```
import com.fasterxml.jackson.annotation.JsonFormat;

@JsonFormat(pattern="yyyy-MM-dd HH:mm")
private LocalDateTime beginEnrollmentDateTime;
@JsonFormat(pattern="yyyy-MM-dd HH:mm")
private LocalDateTime closeEnrollmentDateTime;
@JsonFormat(pattern="yyyy-MM-dd HH:mm")
private LocalDateTime beginLectureDateTime;
@JsonFormat(pattern="yyyy-MM-dd HH:mm")
private LocalDateTime endLectureDateTime;

private String location;
private int basePrice;
private int maxPrice;
private int limitOfEnrollment;
```

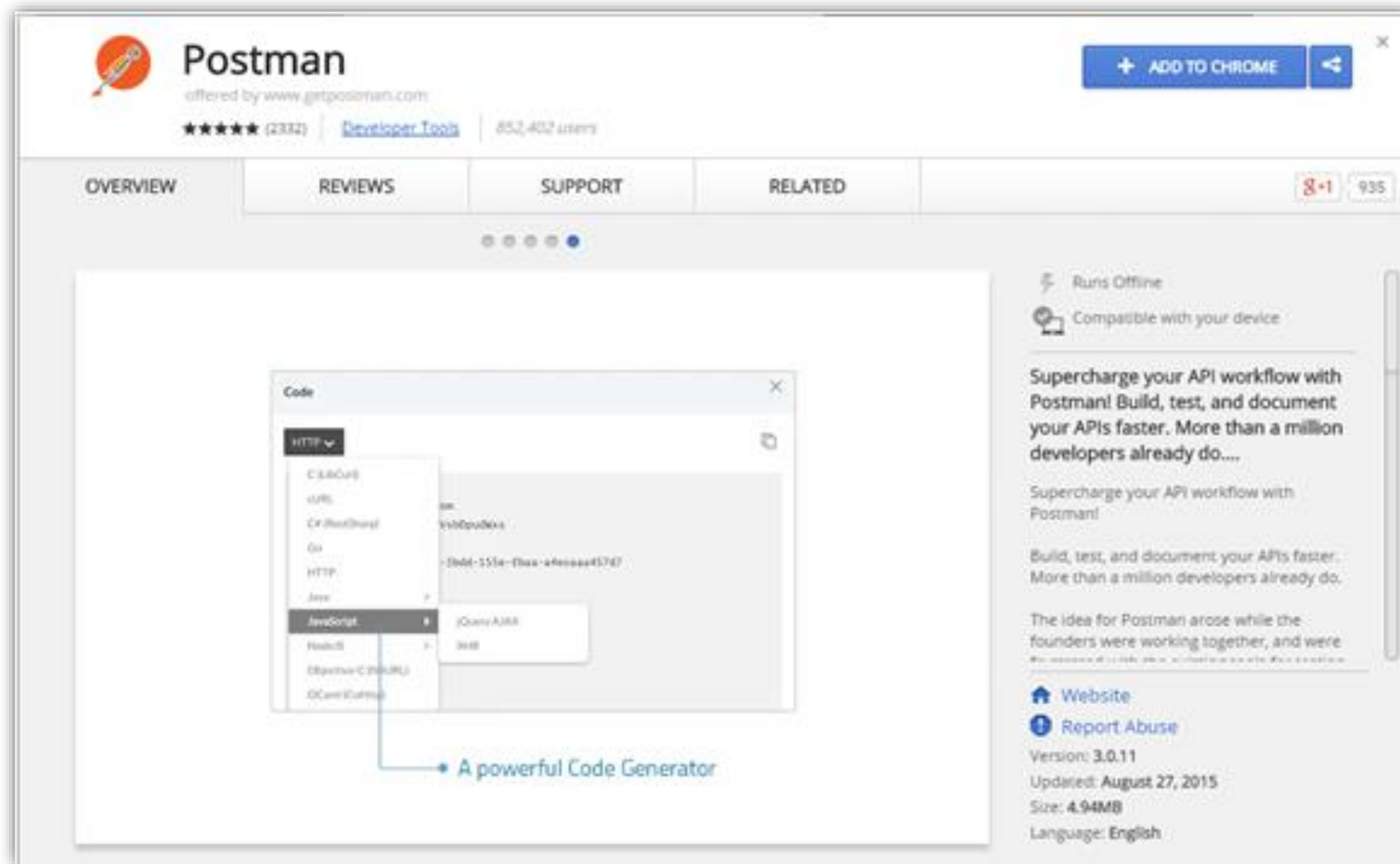
JSON 입력 데이터

```
{
  "name":"SpringBoot",
  "description":"REST API Development with Spring",
  "beginEnrollmentDateTime":"2023-02-03 10:08",
  "closeEnrollmentDateTime":"2023-02-04 10:08",
  "beginLectureDateTime":"2023-02-10 10:08",
  "endLectureDateTime":"2023-02-11 10:08",
  "location":"강의장",
  "basePrice":100,
  "maxPrice":200,
  "limitOfEnrollment":100
}
```


Lecture 생성 API 구현 : Postman 을 사용한 테스트

- Postman: REST API 테스트하는 Chrome 확장 프로그램 설치

<https://chrome.google.com/webstore/detail/postman/fhbjgbfjinjbdgggehcdcbncdddomop>



설치하기

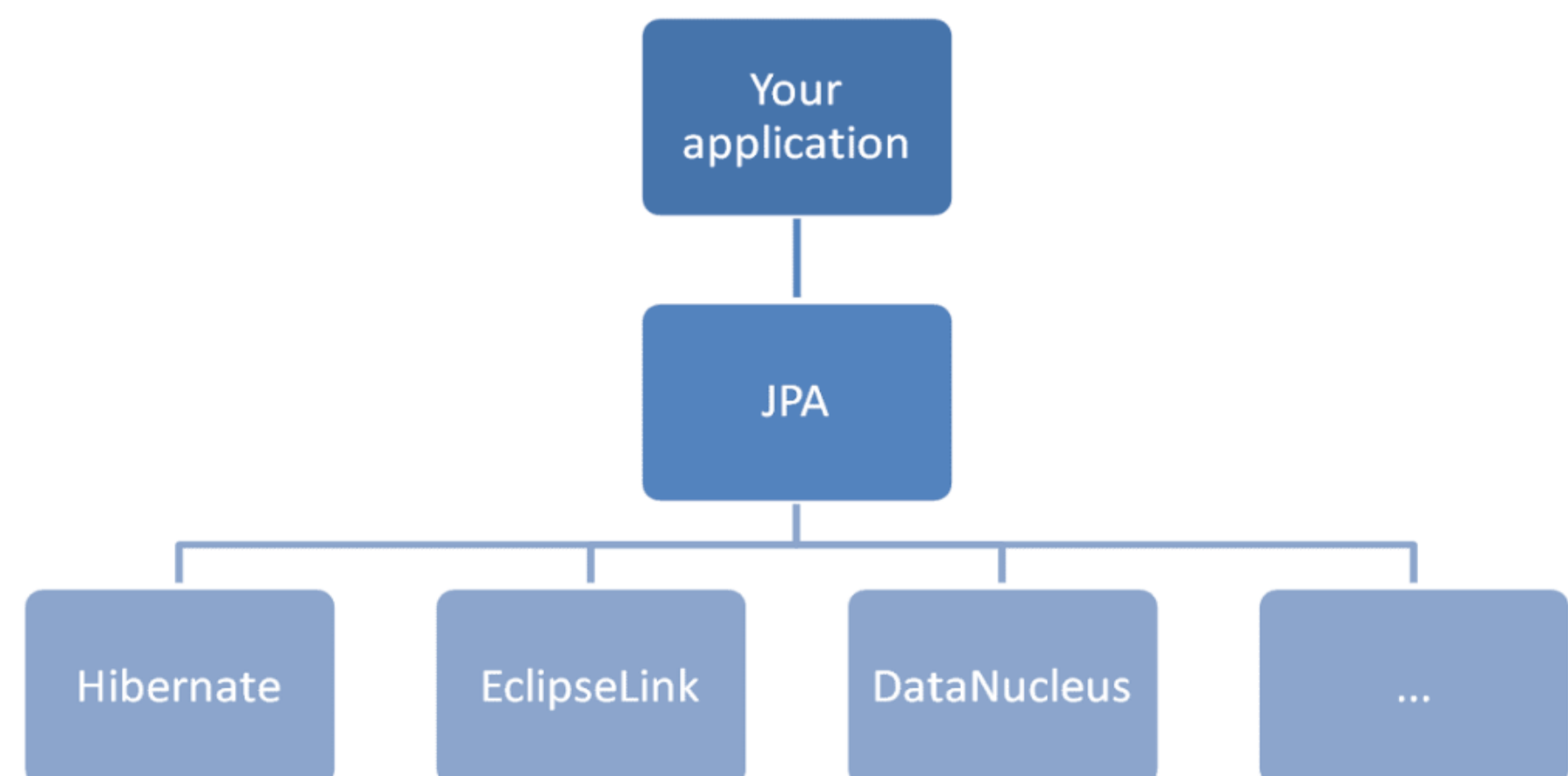
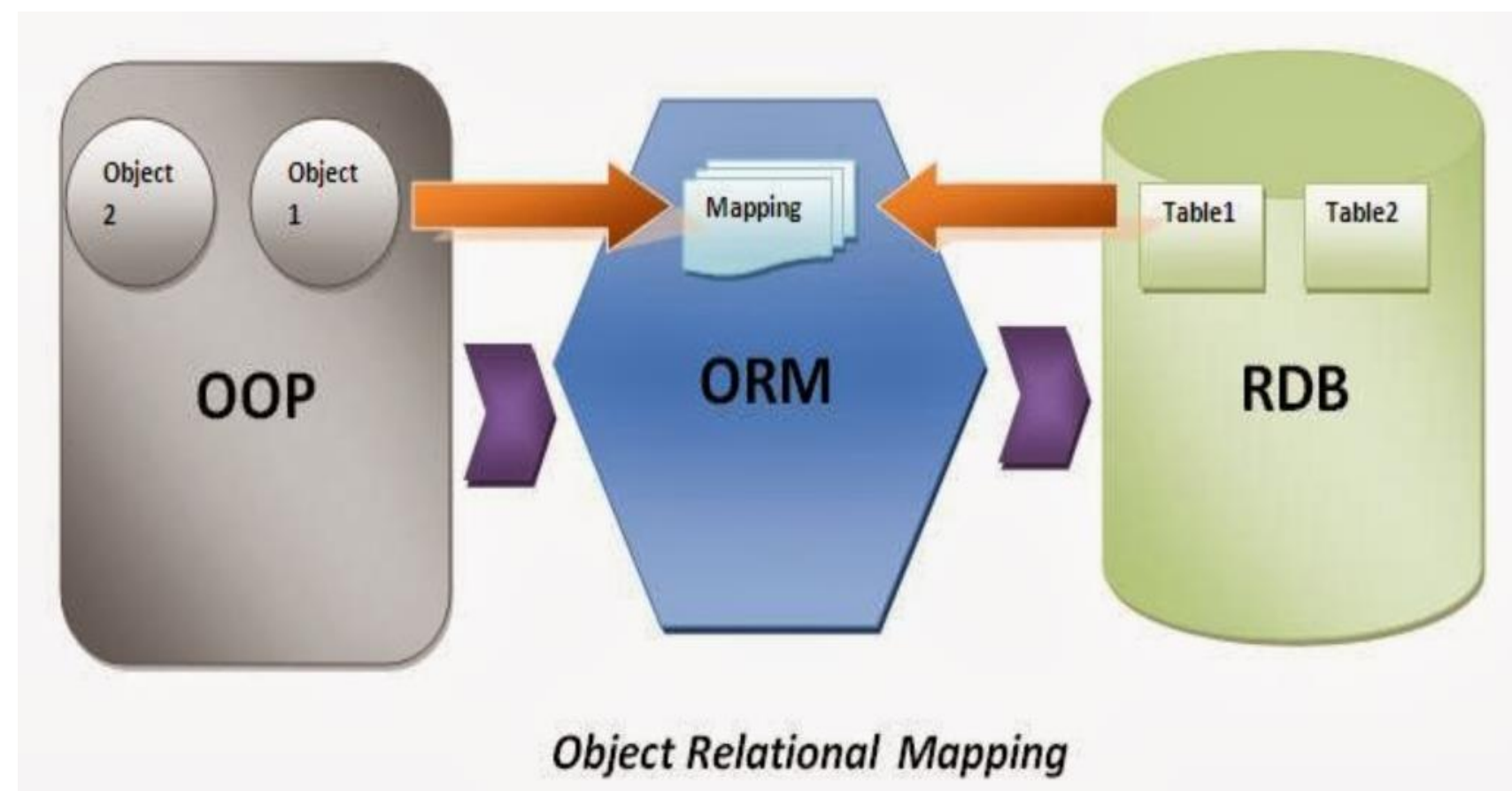
우측 상단의 ADD TO CHROME버튼 클릭

+ ADD TO CHROME

스프링 부트 데이터 (H2 DB 사용)

Spring Boot 데이터

- Spring Data : ORM과 JPA
 - ORM(Object-Relational Mapping)과 JPA (Java Persistence API)
 - : 객체와 릴레이션을 맵핑 할 때 발생하는 개념적 불일치를 해결하는 프레임워크
 - JPA (Java Persistence API) <https://jakarta.ee/specifications/platform/9/apidocs/>
 - : ORM을 위한 Jakarta 표준 스펙이다.
 - Hibernate <https://github.com/hibernate/hibernate-orm>
 - : JPA 표준스펙을 구현한 ORM 구현체



Spring Boot 데이터

▪ Spring Data : DBCP

- 스프링 부트가 지원하는 DBCP(Database Connection Pooling)
- DBCP는 Connection 객체를 만드는 것이 큰 비용을 지불하기 때문에 미리 만들어진 Connection 정보를 재사용 하기 위해 나온 테크닉입니다.
- 스프링 부트에서는 기본적으로 HikariCP 라는 DBCP를 기본적으로 제공합니다.
(<https://github.com/brettwooldridge/HikariCP#frequently-used>)
: HikariCP (기본) - `spring.datasource.hikari.*`
- 스프링에서 DBCP를 설정하는 방법은 다음과 같습니다.
: `spring.datasource.hikari.maximum-pool-size=4` 커넥션 객체의 최대 수를 4개로 설정

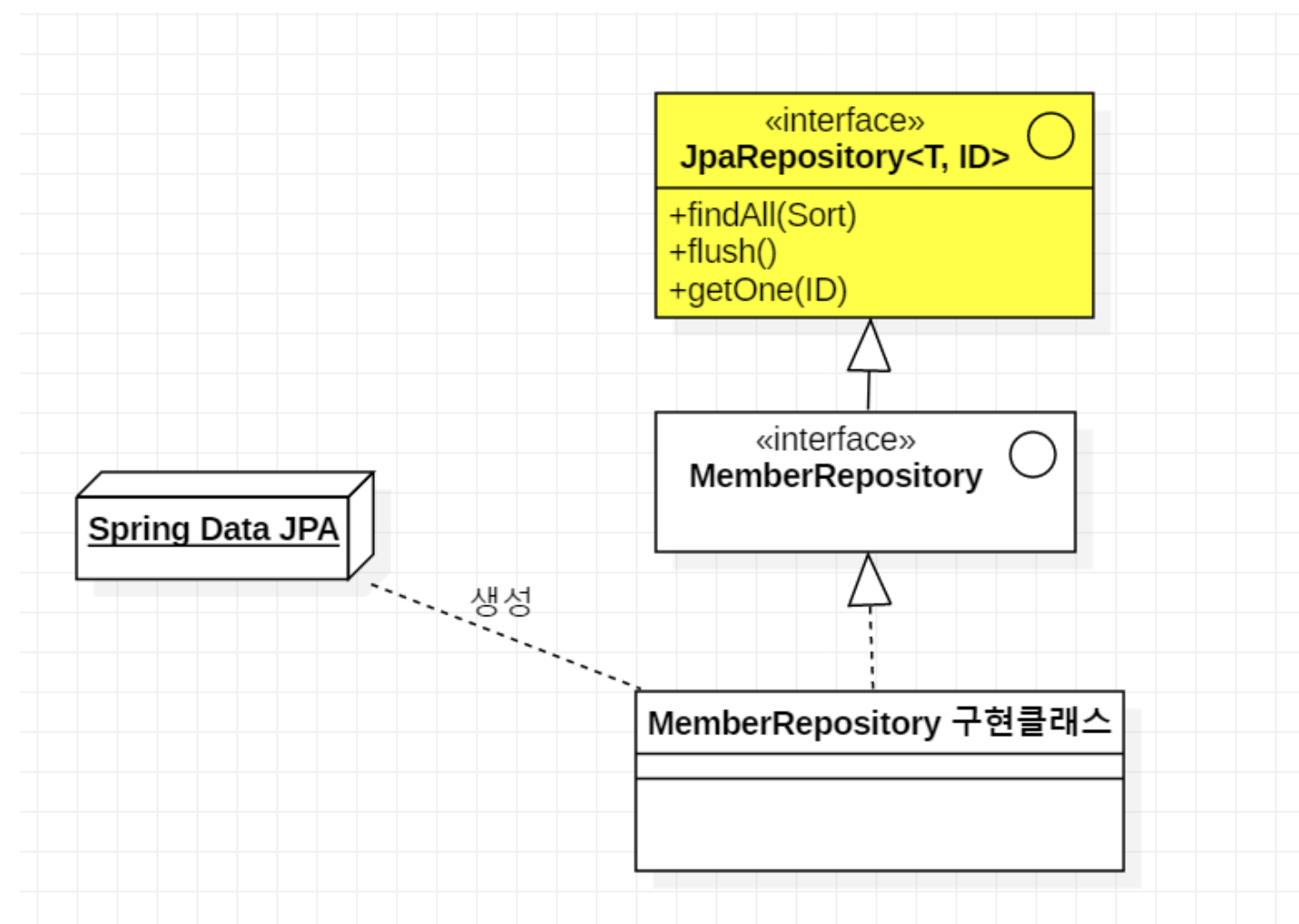
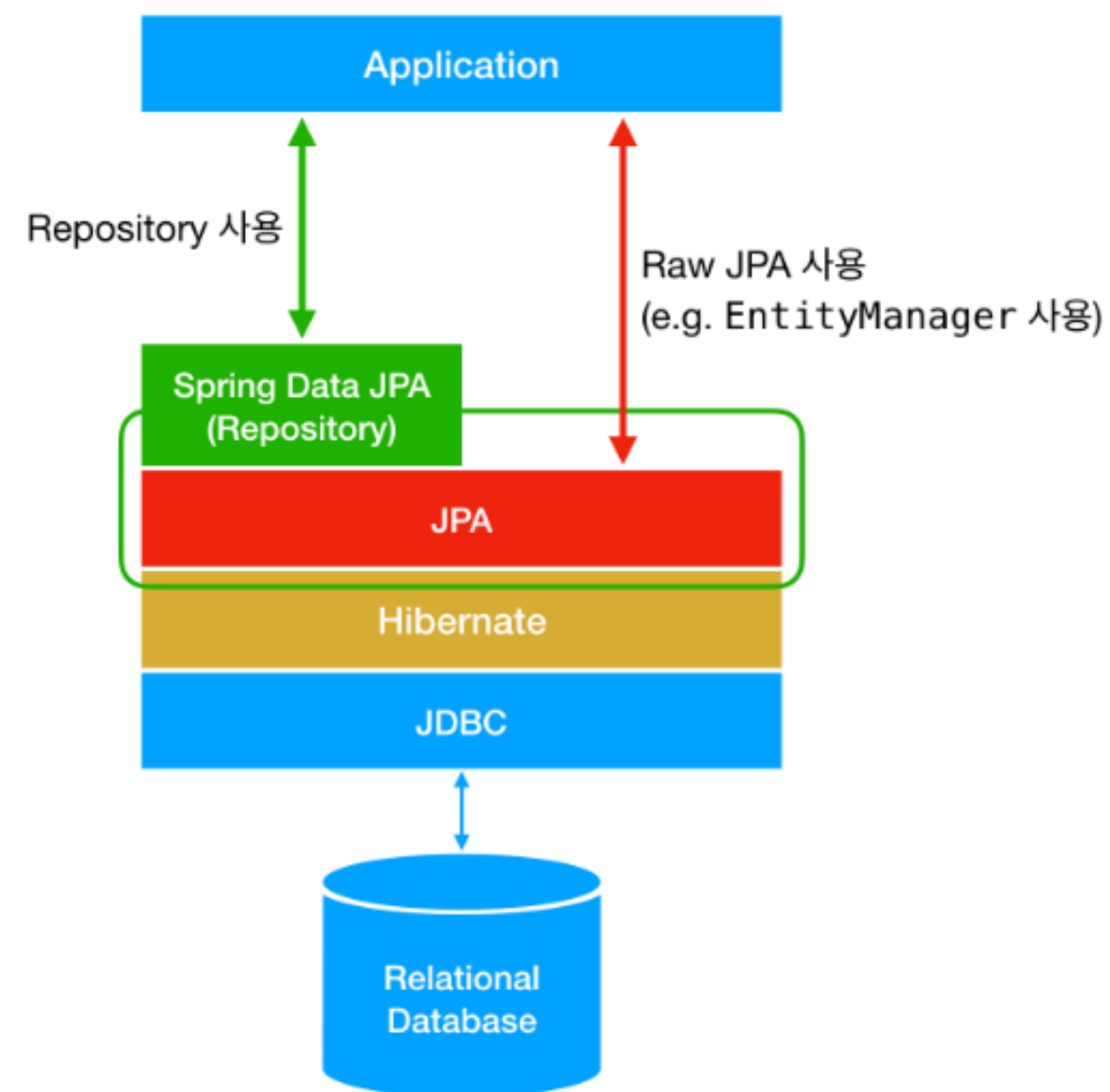
application.properties

spring.datasource.hikari.maximum-pool-size=4

Spring Boot 데이터

■ Spring Data : Spring-Data-JPA

- Spring Data JPA는 Spring에서 제공하는 모듈 중 하나로, 개발자가 JPA를 더 쉽고 편하게 사용할 수 있도록 해준다. 이는 JPA를 한 단계 추상화 시킨 Repository라는 인터페이스를 제공함으로써 이루어집니다.
- 사용자가 Repository 인터페이스에 정해진 규칙대로 finder 메서드를 작성하면, Spring이 해당 메서드 이름에 적합한 쿼리를 수행하는 구현체를 만들어서 Bean으로 등록 해준다.



Spring Boot 데이터터

▪ Spring Data : Spring-Data-JPA

- Spring-Data-JPA는 JPA를 쉽게 사용하기 위해 스프링에서 제공하고 있는 프레임워크 입니다.
- 추상화 정도는 Spring-Data-JPA -> JPA -> Hibernate -> Datasource (왼쪽에서 오른쪽으로 갈 수록 구체화) 입니다. Hibernate는 ORM 프레임워크 이며 DataSource는 스프링과 연결된 MySQL, PostgreSQL 같은 DB를 연결한 인터페이스입니다.
- 스프링 Data JPA 의존성 추가

pom.xml

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

<https://spring.io/projects/spring-data-jpa>

<https://github.com/hibernate/hibernate-orm>

Lecture 생성 API 구현 : Entity 와 Repository 작성 #1

- Entity 클래스 작성하기
 - Entity란 DB에서 영속적으로 저장된 데이터를 자바 객체로 매핑하여 '인스턴스의 형태'로 존재하는 데이터를 말합니다.
 - **@Entity** - Entity 클래스이다. **@Id** - Primary Key
 - **@GeneratedValue(strategy = GenerationType.IDENTITY)**
 - **@Column** - Entity클래스의 모든 필드는 데이터베이스의 컬럼과 매핑 되어 따로 명시하지 않아도 됩니다.
 - 하지만 매핑 될 컬럼명이 다르거나, default 값이 다른 경우에 사용합니다.
 - (이름은 카멜표기법이 소문자 스네이크 표기법으로 전환되고, length는 255, nullable은 true가 default 값입니다.)
 - **@ManyToOne** - 다른 Entity클래스와의 외래키 다대일(N:1)관계를 명시합니다.
 - Entity 클래스에서 enum을 Java Enum 타입으로 사용하려면 **@Enumerated** Annotation을 사용하면 됩니다.
 - : EnumType.ORDINAL : int형으로 DB Access. ex) 0, 1
 - : EnumType.STRING : enum명으로 DB Access. ex) Allow, Deny

Lecture 생성 API 구현 : Entity 와 Repository 작성 #2

- Lecture 엔티티 클래스와 LectureRepository 인터페이스 작성
- Lecture 엔티티 클래스

lectures/Lecture.java

```
@Entity
public class Lecture {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    .....
    @Enumerated(EnumType.STRING)
    private LectureStatus lectureStatus;
}
```


Lecture 생성 API 구현 : Entity 와 Repository 작성 #2

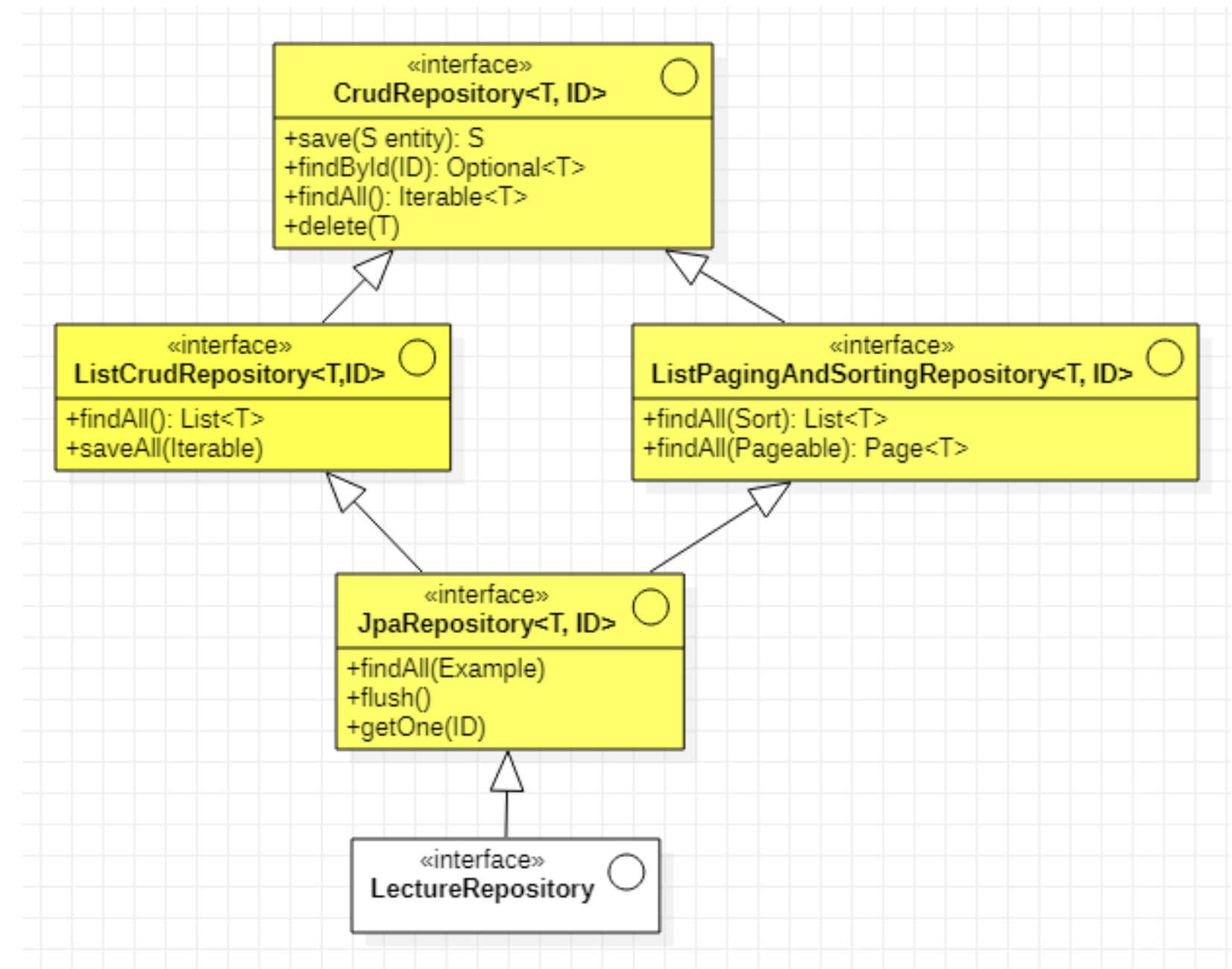
- LectureRepository 인터페이스
- Spring Data JPA는 "JpaRepository" 라는 기능을 사용하면 매우 간단히 데이터를 검색/등록 할 수 있다.
- Repository 인터페이스는 org.springframework.data.jpa.repository 패키지의 "JpaRepository"라는 인터페이스를 상속하여 만든다.

lectures/LectureRepository.java

```
public interface LectureRepository extends JpaRepository<Lecture, Integer>{  
}
```

finder 메서드 규칙

<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods>



Lecture 생성 API 구현 : H2 데이터베이스

■ In-Memory H2 데이터베이스

• H2 Console 사용하는 방법

`http://localhost:8080/h2-console`

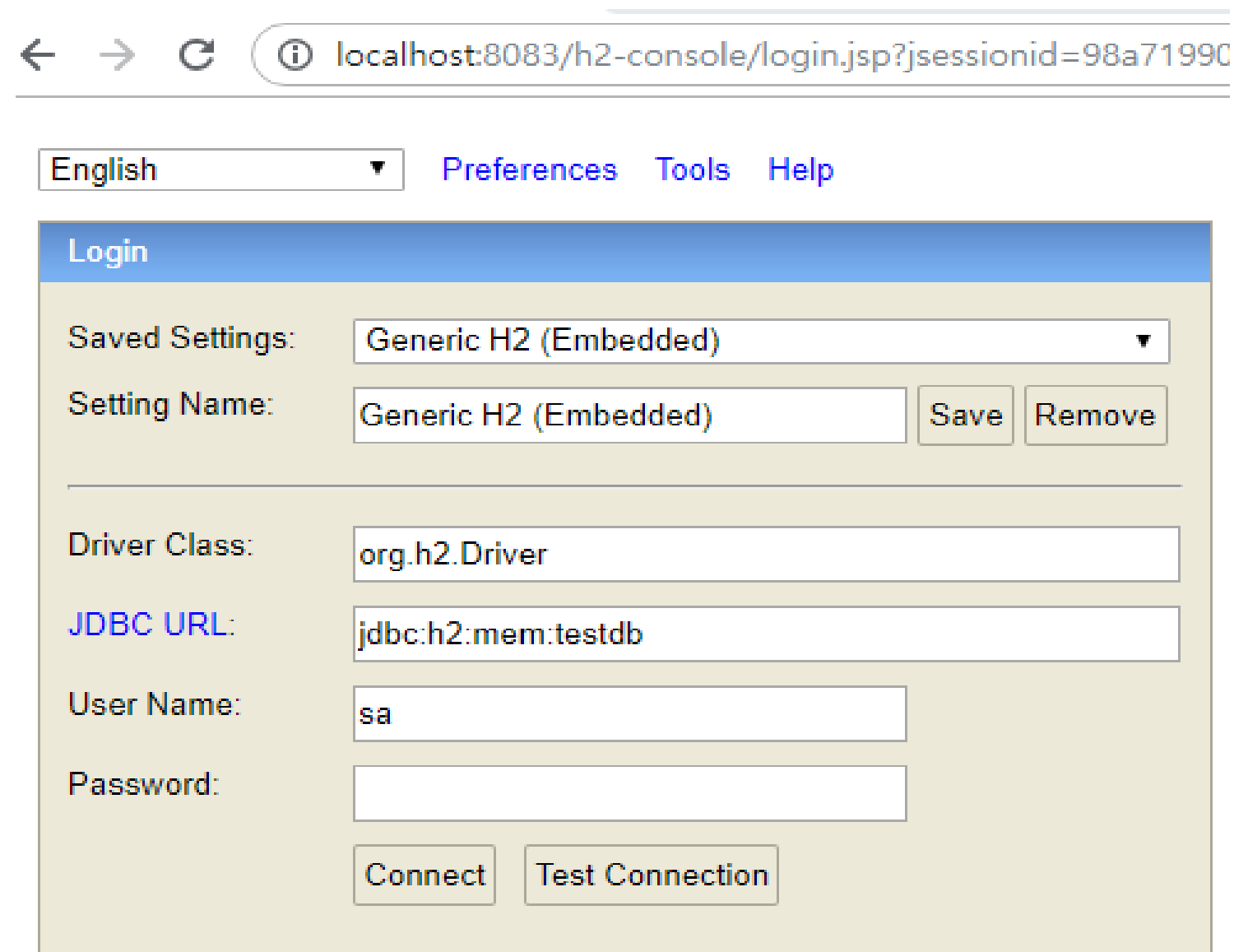
JDBC URL=> `jdbc:h2:mem:testdb` 로 설정한다.

`application.properties` 설정파일

`spring.datasource.url=jdbc:h2:mem:testdb`

`spring.datasource.driver-class-name=org.h2.Driver`

`spring.datasource.username=sa`



현재 연결된 DB를 확인할 수 있는 DatabaseRunner 클래스
<https://gist.github.com/mysoyul/002e117bf859f652466681c7422101f2>

Lecture 생성 API 구현 : Entity 와 Repository 사용

- LectureController 클래스 수정
 - 생성자를 선언하여 LectureRepository를 주입 받고, createLecture() 메서드를 수정한다.
 - lecture.setId(10); 은 제거 합니다.

lectures/LectureController.java

```
private final LectureRepository lectureRepository;

public LectureController(LectureRepository lectureRepository) {
    this.lectureRepository = lectureRepository;
}

@PostMapping
public ResponseEntity createLecture(@RequestBody Lecture lecture) {
    Lecture addLecture = this.lectureRepository.save(lecture);

    WebMvcLinkBuilder selfLinkBuilder = WebMvcLinkBuilder.linkTo(LectureController.class).slash(addLecture.getId());
    URI createUri = selfLinkBuilder.toUri();

    return ResponseEntity.created(createUri).body(addLecture);
}
```

스프링 부트 데이터 (Maria DB 사용)

Spring Boot 데이터터

- Spring Data : JDBC Driver 설정
- MariaDB Client 의존성 추가

pom.xml

```
<dependency>  
  <groupId>org.mariadb.jdbc</groupId>  
  <artifactId>mariadb-java-client</artifactId>  
</dependency>
```

Spring Boot 데이터터 – 사용자 계정과 DB 생성

root 계정으로 접속하여 사용자 계정과 DB 생성

```
mysql -u root -p
```

```
MariaDB [(none)]> show databases;
```

```
MariaDB [(none)]> use mysql;
```

```
MariaDB [mysql]> create database myboot_db;
```

```
MariaDB [mysql]> CREATE USER 'boot'@'%' IDENTIFIED BY 'boot';
```

```
MariaDB [mysql]> GRANT ALL PRIVILEGES ON myboot_db.* TO 'boot'@'%';
```

```
MariaDB [mysql]> flush privileges;
```

```
MariaDB [mysql]> select user, host from user;
```

```
MariaDB [mysql]> exit;
```

boot 사용자 계정으로 접속한다.

```
mysql -u boot -p
```

```
boot 입력
```

```
use myboot_db;
```

Spring Boot 데이터터

- Spring Data : DataSource 설정
 - MariaDB DataSource 설정

src/main/resources/application-prod.properties

```
spring.datasource.url=jdbc:mariadb://127.0.0.1:3306/myboot_db
spring.datasource.username=boot
spring.datasource.password=boot
spring.datasource.driverClassName=org.mariadb.jdbc.Driver
```

- 프로파일용 properties file
 - : application-{profile}.properties
 - : application-prod.properties / application-test.properties

src/main/resources/application.properties

```
spring.profiles.active=prod
```

Spring Boot 데이터

- Spring Data : JPA를 사용한 데이터베이스 초기화
- application.properties 파일에 JPA에 의한 데이터베이스 자동 초기화 설정

src/main/resources/application-prod.properties

```
spring.jpa.hibernate.ddl-auto=create-drop  
spring.jpa.show-sql=true  
spring.jpa.database-platform=org.hibernate.dialect.MariaDB103Dialect
```

spring.jpa.show-sql=true

: JPA가 생성한 SQL문을 콘솔에 출력합니다

Spring Boot 데이터

- Spring Data : JPA를 사용한 데이터베이스 초기화
 - `spring.jpa.hibernate.ddl-auto=create|create-drop|update|validate|none`
 - ✓ create
JPA가 DB와 상호작용할 때 기존에 있던 스키마(테이블)을 삭제하고 새로 만드는 것을 뜻한다.
 - ✓ create-drop
JPA 종료 시점에 기존에 있었던 테이블을 삭제합니다.
 - ✓ update
기존 스키마는 유지하고, 새로운 것만 추가하고, 기존의 데이터도 유지한다. 변경된 부분만 반영함
주로 개발 할 때 적합하다.
 - ✓ validate
엔티티와 테이블이 정상 매핑 되어 있는지를 검증합니다.
 - `spring.jpa.properties.hibernate.format_sql=true`
: JPA가 생성한 SQL문을 콘솔에 출력할지 여부를 알려 주는 프로퍼티입니다.

Spring Boot 데이터

- Spring Data : Dialect (방언)이란? ([Hibernate Dialect API Docs](#))
 - ANSI SQL은 모든 DBMS에서 공통적으로 사용이 가능한 표준 SQL이지만 DBMS에서 만든 SQL은 DB벤더들만의 독자적인 기능을 추가 하기 위해 만든 것으로 사용하는 특정 벤더의 DBMS에서만 사용이 가능합니다.
 - JPA는 기본적으로 어플리케이션에서 직접 JDBC 레벨의 SQL을 작성하지 않고 JPA가 직접 SQL을 생성해줍니다. JPA에 어떤 DBMS를 사용하는지를 알려주는 방법이 방언(Dialect)을 설정하는 방법입니다.
 - JPA에 Dialect를 설정할 수 있는 추상화 방언 클래스를 제공하고 설정된 방언으로 각 DBMS에 맞는 구현체를 제공합니다.
 - `hibernate.dialect=org.hibernate.dialect.MariaDBDialect`

MariaDB 10.3 버전 사용시

```
database-platform: org.hibernate.dialect.MariaDB103Dialect
```

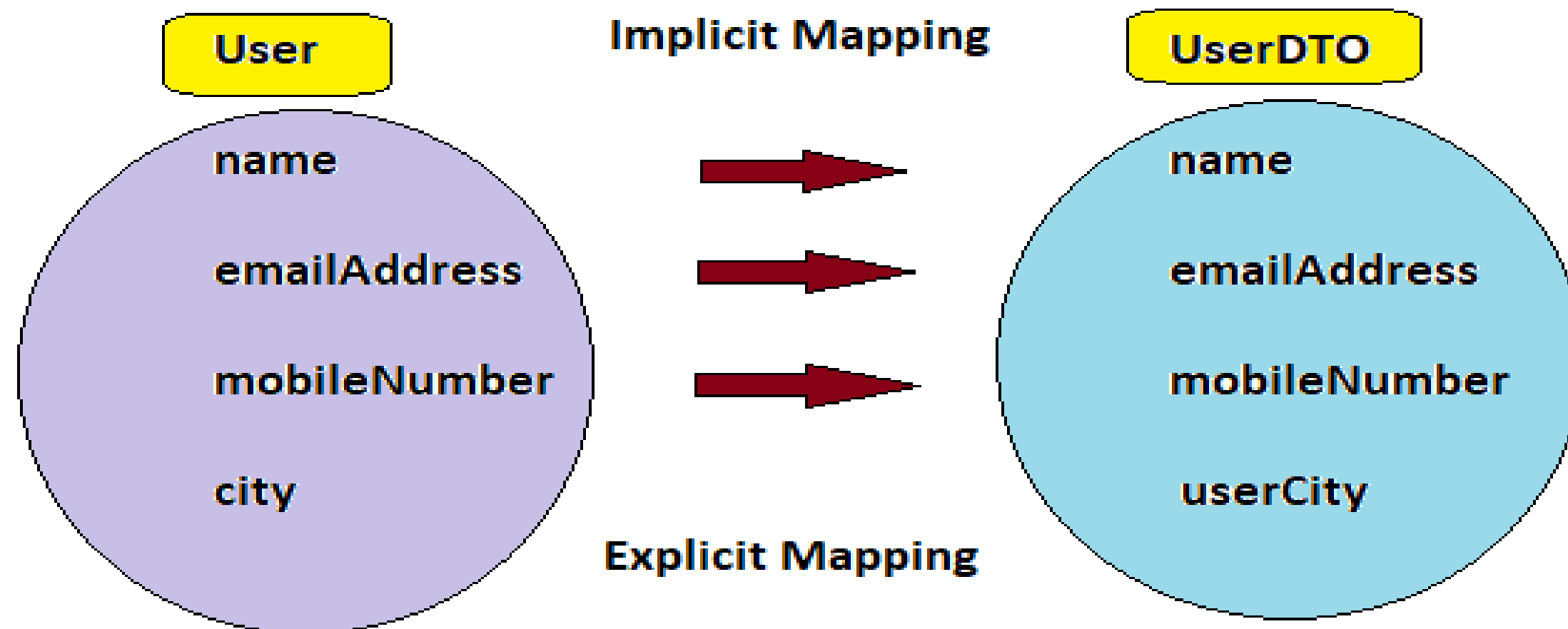
Oracle 11g 버전 사용시

```
database-platform: org.hibernate.dialect.Oracle10gDialect
```

ModelMapper

Lecture 생성 API 구현 : LectureDto 와 ModelMapper

- ModelMapper란?
 - DB Layer에는 Entity 클래스, View Layer에서 DTO 클래스를 사용하여 역할을 분리하는 것이 좋습니다.
 - Entity와 DTO를 연결할 때 ModelMapper 를 사용합니다. (<http://modelmapper.org/getting-started/>)



User to UserDTO

We can use ModelMapper to implicitly map an user instance to a new UserDTO:

```
ModelMapper modelMapper = new ModelMapper();
UserDTO userDTO = modelMapper.map(user, UserDTO.class);
```

Lecture 생성 API 구현 : LectureReqDto 와 ModelMapper

- LectureReqDto 클래스 작성 (요청을 처리하는 DTO 클래스)
 - DB Layer에는 Entity 클래스, View Layer에서 DTO 클래스를 사용하여 역할을 분리하는 것이 좋습니다.
 - Entity와 DTO 두 클래스를 어떻게 매핑 시킬 것인가?

lectures/dto/LectureReqDto.java

```
@Data @Builder
@NoArgsConstructor @AllArgsConstructor
public class LectureReqDto {
    private String name;
    private String description;
    @JsonFormat(pattern="yyyy-MM-dd HH:mm")
    private LocalDateTime beginEnrollmentDateTime;
    @JsonFormat(pattern="yyyy-MM-dd HH:mm")
    private LocalDateTime closeEnrollmentDateTime;
    @JsonFormat(pattern="yyyy-MM-dd HH:mm")
    private LocalDateTime beginLectureDateTime;
    @JsonFormat(pattern="yyyy-MM-dd HH:mm")
    private LocalDateTime endLectureDateTime;
    private String location;
    private int basePrice;
    private int maxPrice;
    private int limitOfEnrollment;
}
```

pom.xml

```
<dependency>
    <groupId>org.modelmapper</groupId>
    <artifactId>modelmapper</artifactId>
    <version>3.1.1</version>
</dependency>
```

Lecture 생성 API 구현 : LectureResDto

- LectureResDto 클래스 작성 (응답을 처리하는 DTO 클래스)

lectures/dto/LectureResDto.java

```
@Data @Builder
@NoArgsConstructor @AllArgsConstructor
public class LectureResDto {
    private Integer id;
    private String name;
    private String description;
    @JsonFormat(pattern="yyyy-MM-dd HH:mm")
    private LocalDateTime beginEnrollmentDateTime;
    @JsonFormat(pattern="yyyy-MM-dd HH:mm")
    private LocalDateTime closeEnrollmentDateTime;
    @JsonFormat(pattern="yyyy-MM-dd HH:mm")
    private LocalDateTime beginLectureDateTime;
    @JsonFormat(pattern="yyyy-MM-dd HH:mm")
    private LocalDateTime endLectureDateTime;
    private String location;
    private int basePrice;
    private int maxPrice;
    private int limitOfEnrollment;
    private boolean offline;
    private boolean free;
    private String email;
}
```

Lecture 생성 API 구현 : ModelMapper 를 Spring Bean으로

- ModelMapper 를 Bean으로 생성 : Application 클래스 수정
- ModelMapper 를 주입 받기 위해서 Spring Bean으로 등록한다.

config/AppConfig.java

```
@Configuration
public class AppConfig {

    @Bean
    public ModelMapper modelMapper() {
        ModelMapper modelMapper = new ModelMapper();
        return modelMapper;
    }
}
```

Lecture 생성 API 구현 : LectureDto 와 ModelMapper

- ModelMapper Bean 사용 : LectureController 클래스 수정
 - ModelMapper Bean을 주입 받는다.
 - ModelMapper의 map() 메서드를 호출해서 LectureReqDto 객체와 Lecture 객체를 매핑 합니다.

lectures/LectureController.java

```
private final ModelMapper modelMapper;

public LectureController(LectureRepository lectureRepository, ModelMapper modelMapper) {
    this.lectureRepository = lectureRepository;
    this.modelMapper = modelMapper;
}

public ResponseEntity createLecture(@RequestBody LectureReqDto lectureReqDto) {

    Lecture lecture = modelMapper.map(lectureReqDto, Lecture.class);

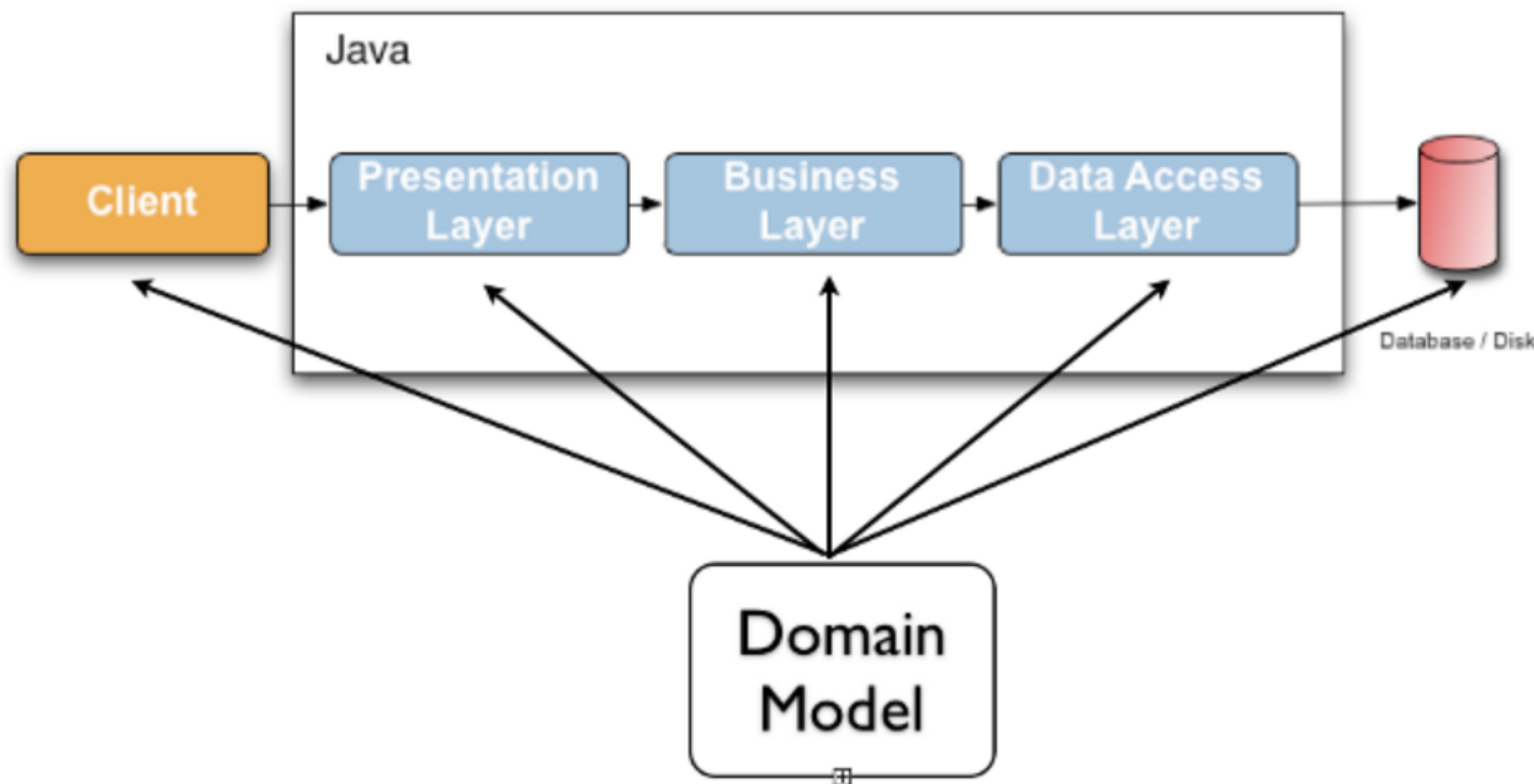
    Lecture addLecture = this.lectureRepository.save(lecture);
    ....
}
```


유효성 검증

Bad_Request와 Validation

Java Bean Validation

- Java Bean Validation ([Jakarta EE API Docs](#))
 - 데이터 검증을 위한 로직을 도메인 모델 자체에 묶어서 표현하는 방법이 있습니다.
 - Java Bean Validation 에서 데이터 검증을 위한 어노테이션 (Annotation) 을 제공하고 있습니다.



출처 [Hibernate Validator 6.0.11.Final — JSR 380 Reference Implementation: Reference Guide](#)

pom.xml

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

Java Bean Validation의 구현체 : Hibernate Validator

<https://github.com/hibernate/hibernate-validator>

<https://hibernate.org/validator/>

Hibernate Validator API

<https://docs.jboss.org/hibernate/validator/6.2/api/>

Spring Framework5 API Docs

<https://docs.spring.io/spring-framework/docs/5.3.24/javadoc-api/>

Lecture 생성 API 구현 : Bad_Request로 응답

- Bad Request로 응답하기
 - Bad_Request로 응답 vs 받기로 한 값 이외는 무시
 - 스프링 부트에서는 기본적으로 Unknown Property는 Deserialization(역직렬화) 중에 무시 됩니다.
 - 입력 값 이외에 값이 들어 왔을 때 에러(Bad Request)를 발생 시키기 위해서는
 - `spring.jackson.deserialization.fail-on-unknown-properties=true` 로 설정해야 합니다.

src/main/resources/application.properties

```
spring.jackson.deserialization.fail-on-unknown-properties=true
```

Lecture 생성 API 구현 : Bad_Request로 응답

- 입력 데이터 체크와 에러 메시지 출력
 - 입력 데이터의 값이 유효하지 않은 경우 Bad_Request로 응답
 - 비즈니스 로직으로 검사 할 수 있는 에러
 - 응답(response) 메시지에 에러 정보가 JSON 형태로 포함 되어 있어야 한다.

Lecture 생성 API 구현 : @Valid

- @Valid 어노테이션을 이용한 validation 체크
- @Valid는 JSR-303이란 이름으로 채택 된 서블릿 2.3 표준 스펙 중 하나이며, 스프링은 이 표준을 확장하고 쉽게 사용할 수 있도록 스프링만의 방식으로 재편성 해 주었습니다.
- @NotNull, @NotEmpty, @Min, @Max, @Size(min=, max=) 사용해서 입력 값을 바인딩할 때 에러를 확인할 수 있습니다. (<https://jakarta.ee/specifications/platform/9/apidocs/>)
- Errors (BindingResult)는 항상 @Valid 바로 다음 인자로 사용해야 합니다.

lectures/dto/LectureReqDto.java

```
import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.NotNull;

@NotEmpty
private String name;
@NotEmpty
private String description;
@NotNull
@JsonFormat(pattern="yyyy-MM-dd HH:mm")
private LocalDateTime beginEnrollmentDateTime;
@Min(0)
private int basePrice;
```

lectures/LectureController.java

```
import org.springframework.validation.Errors;

public ResponseEntity createLecture(@RequestBody @Valid LectureReqDto lectureDto,
Errors errors) {

    if(errors.hasErrors()) {
        return ResponseEntity.badRequest().build();
    }

}
```

Lecture 생성 API 구현 : Custom Validator

- Custom Validator 작성 : LectureValidator 클래스 작성
 - Validator 클래스를 Spring Bean 컴포넌트로 정의한다.

lectures/LectureValidator.java

```
@Component
public class LectureValidator {
    public void validate(LectureReqDto lectureReqDto, Errors errors) {
        if(lectureReqDto.getBasePrice() > lectureReqDto.getMaxPrice() &&
            lectureReqDto.getMaxPrice() != 0) {
            //Field Error
            errors.rejectValue("basePrice", "wrongValue", "BasePrice is wrong");
            errors.rejectValue("maxPrice", "wrongValue", "MaxPrice is wrong");
            //Global Error
            errors.reject("wrongPrices", "Values for prices are wrong");
        }

        LocalDateTime endLectureDateTime = lectureReqDto.getEndLectureDateTime();
        if(endLectureDateTime.isBefore(lectureReqDto.getBeginLectureDateTime()) ||
            endLectureDateTime.isBefore(lectureReqDto.getCloseEnrollmentDateTime()) ||
            endLectureDateTime.isBefore(lectureReqDto.getBeginEnrollmentDateTime()) ) {
            errors.rejectValue("endLectureDateTime", "wrongValue", "endLectureDateTime is wrong");
        }
    }
}
```

Lecture 생성 API 구현 : Custom Validator

- Custom Validator 사용 : LectureController 클래스 수정
- LectureValidator 를 주입 받고, LectureValidator의 validate() 메서드를 호출하고, Errors 객체에 유효성 에러가 포함되어 있는지 확인합니다.

lectures/LectureController.java

```
private final LectureValidator lectureValidator;

public LectureController(LectureRepository lectureRepository, ModelMapper modelMapper, LectureValidator lectureValidator) {
    this.lectureRepository = lectureRepository;
    this.modelMapper = modelMapper;
    this.lectureValidator = lectureValidator;
}

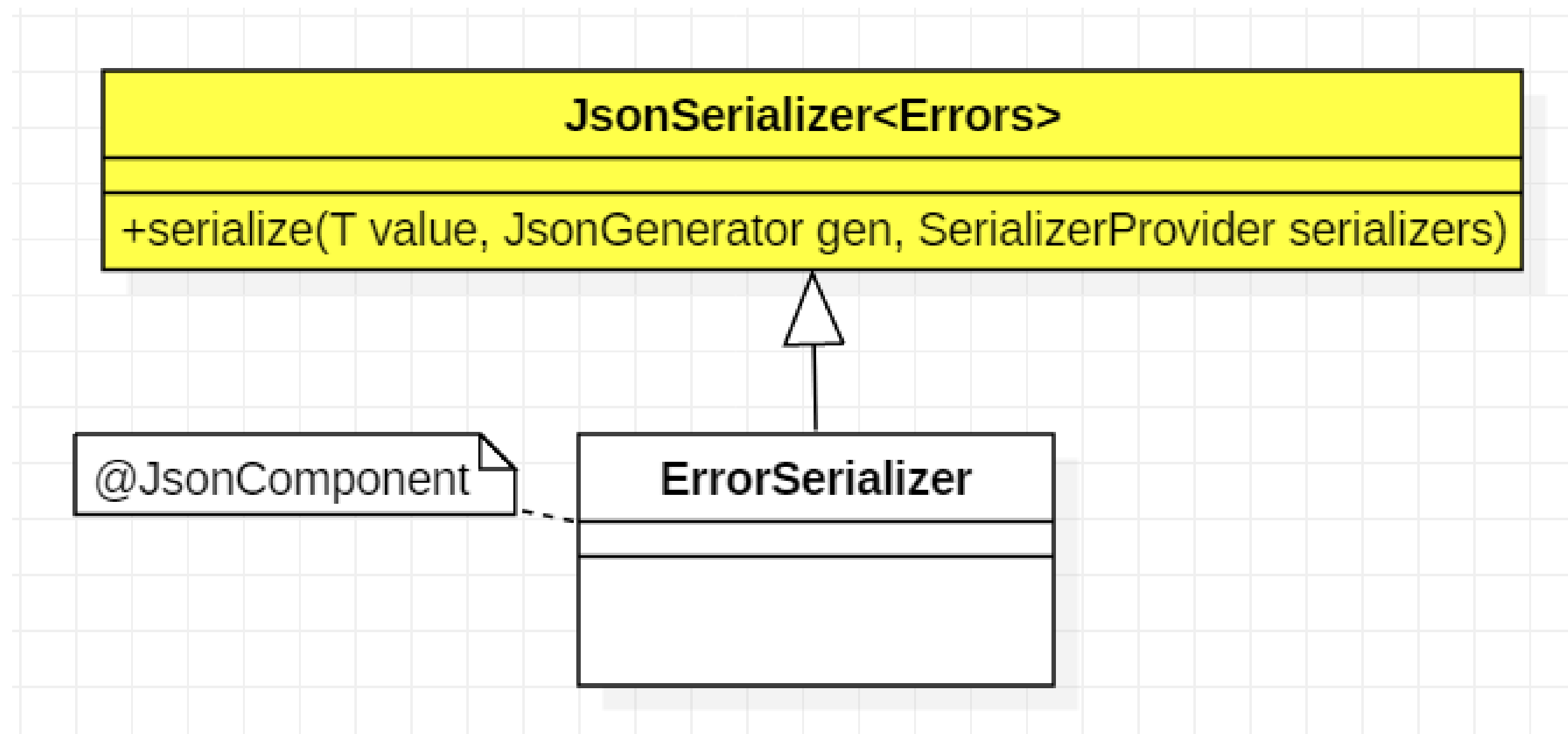
public ResponseEntity createLecture(@RequestBody @Valid LectureReqDto lectureReqDto, Errors errors) {

    this.lectureValidator.validate(lectureReqDto, errors);

    if(errors.hasErrors()) {
        return ResponseEntity.badRequest().build();
    }
    ....
}
```

Lecture 생성 API 구현 : Custom Json Serializer

- Custom JsonSerializer 작성 : JsonSerializer 클래스 작성
 - 에러 메시지를 출력하기 위해서 Custom JsonSerializer를 작성해야 합니다.
 - Jackson JSON이 제공하는 JsonSerializer를 상속 받는다. => Errors 객체를 JsonSerializer 로 Wrapping 한다.
 - Spring Boot가 제공하는 @JsonComponent 어노테이션은 Custom Serializer 클래스를 Spring Bean으로 등록해 주는 역할을 합니다.



Lecture 생성 API 구현 : Custom Json Serializer

- Custom JsonSerializer 작성 : ErrorSerializer 클래스 작성
 - Errors는 Java Bean Spec을 준수하는 객체가 아니어서 JSON 객체로 변환할 수 없다.
 - JsonSerializer를 상속 받고 serialize() 메서드를 재정의 한다.
 - Spring Boot에서 제공하는 @JsonComponent 를 선언 한다.
 - @JsonComponent annotation allows us to expose an annotated class to be a Jackson serializer and/or deserializer without the need to add it to the ObjectMapper manually.
- ErrorSerializer 작성하기
 - 1) FieldError 와 GlobalError를 둘 다 매핑해서 JSON에 담아 주어야 한다.
 - 2) ObjectMapper는 Errors라는 객체를 직렬화 (Serialization) 할 때 ErrorSerializer를 사용한다.

common/ErrorSerializer.java

```
@JsonComponent
public class ErrorSerializer extends JsonSerializer<Errors> {
    @Override
    public void serialize(Errors errors, JsonGenerator gen,
        SerializerProvider serializers) throws IOException {
        .....
    }
}
```

Lecture 생성 API 구현 : Custom JsonSerializer

- Custom JsonSerializer 작성 : JsonSerializer 클래스 작성
 - BindingError
 - ✓ FieldError 와 GlobalError (ObjectError)
 - ✓ objectName
 - ✓ defaultMessage
 - ✓ code
 - ✓ field
 - ✓ rejectedValue

Lecture 생성 API 구현 : Custom Json Serializer

- Custom JsonSerializer 작성 : ErrorsSerializer 클래스 작성

common/ErrorsSerializer.java

```
@JsonComponent
public class ErrorsSerializer extends JsonSerializer<Errors> {
    @Override
    public void serialize(Errors errors, JsonGenerator gen,
        SerializerProvider serializers) throws IOException {

        gen.writeStartArray();
        errors.getFieldErrors().forEach(e -> {
            try {
                gen.writeStartObject();
                gen.writeStringField("field", e.getField());
                gen.writeStringField("objectName", e.getObjectName());
                gen.writeStringField("code", e.getCode());
                gen.writeStringField("defaultMessage", e.getDefaultMessage());
                Object rejectedValue = e.getRejectedValue();
                if (rejectedValue != null) {
                    gen.writeStringField("rejectedValue", rejectedValue.toString());
                }
                gen.writeEndObject();
            } catch (IOException e1) {
                e1.printStackTrace();
            }
        });
    }
}
```

common/ErrorsSerializer.java

```
errors.getGlobalErrors().forEach(e -> {
    try {
        gen.writeStartObject();
        gen.writeStringField("objectName", e.getObjectName());
        gen.writeStringField("code", e.getCode());
        gen.writeStringField("defaultMessage", e.getDefaultMessage());
        gen.writeEndObject();
    } catch (IOException e1) {
        e1.printStackTrace();
    }
});
gen.writeEndArray();
}
```

Lecture 생성 API 구현 : Custom Json Serializer

- Custom JsonSerializer 사용 : LectureController 클래스 수정
- Bad_Request 에러가 발생 하면, Errors 객체를 Body에 Wrapping 해서 보내면 에러 메시지가 잘 출력됩니다.

lectures/LectureController.java

```
public ResponseEntity createLecture(@RequestBody @Valid LectureReqDto lectureReqDto, Errors errors) {  
  
    if(errors.hasErrors()) {  
        return ResponseEntity.badRequest().body(errors);  
    }  
  
    lectureValidator.validate(lectureReqDto, errors);  
    if(errors.hasErrors()) {  
        return ResponseEntity.badRequest().body(errors);  
    }  
    ....  
}
```

Lecture 생성 API 구현 : 비즈니스 로직 적용

- 비즈니스 로직 적용 : Lecture 클래스 수정
 - offline과 free 값 확인
 - isEmpty() 와 isBlank() 메서드의 차이점 (<https://stackoverflow.com/questions/51299126/difference-between-isempty-and-isblank-method-in-java-11>)

lectures/Lecture.java

```
public class Lecture {
    ...
    public void update() {
        // Update free
        if (this.basePrice == 0 && this.maxPrice == 0) {
            this.free = true;
        } else {
            this.free = false;
        }
        // Update offline
        if (this.location == null || this.location.isBlank()) {
            this.offline = false;
        } else {
            this.offline = true;
        }
    }
}
```

lectures/LectureController.java

```
public class LectureController {
    public ResponseEntity createLecture(@RequestBody @Valid
    LectureReqDto lectureReqDto, Errors errors) {
        .....
        Lecture lecture = modelMapper.map(lectureReqDto, Lecture.class);
        lecture.update();
        Lecture addLecture = this.LectureRepository.save(lecture);
    }
}
```

Before Java 11

boolean blank = string.trim().isEmpty();

After Java 11

boolean blank = string.isBlank();

select offline+0 as offline, free+0 as free from Lecture;

HATEOAS와 Self-Descriptive Message 적용

HATEOAS

■ HATEOAS 소개

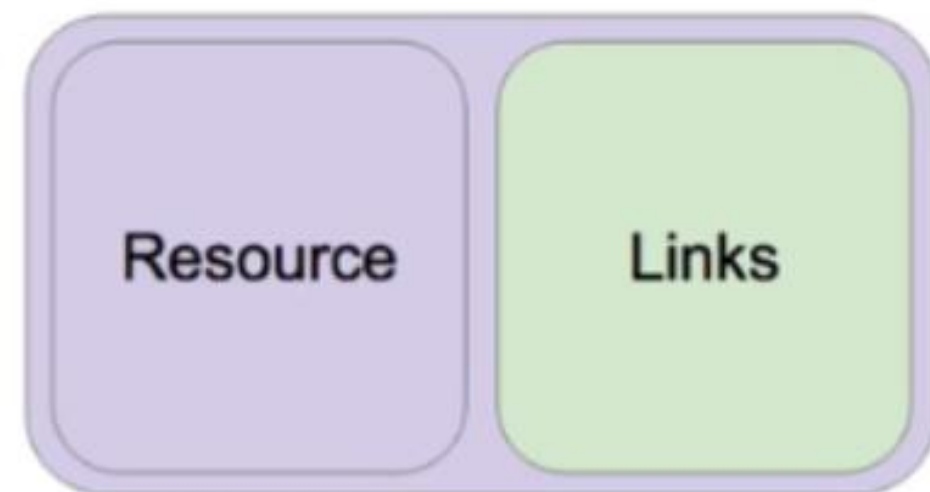
- HATEOAS (Hypermedia As The Engine Of Application state) 란? (<https://en.wikipedia.org/wiki/HATEOAS>)
- HATEOAS를 통해서 어플리케이션의 상태를 전이할 수 있는 메커니즘을 제공할 수 있습니다.
- 예를 들어, 송금 어플리케이션 Home 화면에는 입금, 출금, 송금 등 다른 화면 혹은 기능, 리소스로 갈 수 있는 링크들이 존재할 것입니다.
 - 이 링크를 통해서 다른 페이지로 가는 것을 다른 상태로 전이 한다고 보고 이 링크들에 대한 레퍼런스를 서버 측에서 전송합니다.
 - 클라이언트가 명시적으로 링크를 작성하지 않고도 서버 측에서 받은 링크의 레퍼런스를 통해 어플리케이션의 상태 및 전이를 표현할 수 있습니다. 이것이 바로 올바른 REST 아키텍처에서의 HATEOAS 구성법입니다.

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: ...
<?xml version="1.0"?>
<account>
  <account_number>12345</account_number>
  <balance currency="usd">100.00</balance>
  <link rel="deposit" href="/accounts/12345/deposit" />
  <link rel="withdraw" href="/accounts/12345/withdraw" />
  <link rel="transfer" href="/accounts/12345/transfer" />
  <link rel="close" href="/accounts/12345/close" />
</account>
```


HATEOAS : Spring HATEOAS

▪ Spring HATEOAS 소개

- Spring HATEOAS (<https://docs.spring.io/spring-hateoas/docs/current/reference/html/>)
- 스프링 진영에서는 스프링 HATEOAS라는 프로젝트를 통해 스프링 사용자들에게 HATEOAS 기능을 손쉽게 쓸 수 있도록 제공하고 있습니다. 이 프로젝트의 주요 기능은 HTTP 응답에 들어갈 유저, 게시판 글, 이벤트 등과 같은 Resource. 다른 상태 혹은 리소스에 접근할 수 있는 링크 레퍼런스인 Links 를 제공하는 것입니다.
- HATEOAS 링크에 들어가는 정보는 현재 Resource의 관계이자 링크의 레퍼런스 정보인 **REL** 과 하이퍼링크인 **HREF** 두 정보가 들어갑니다.



- HREF(Hyper Media Reference): URI나 URL 설정
- REL(Relation): 현재 이 소스와의 관계
 - **self**: 자기자신의 URL
 - **profile**: 현재 이 응답본문에 대한 설명을 가지고 있는 문서로 링크
 - **update-event**: 이벤트를 수정할 수 있는 업데이트
 - **query-events**: 이벤트를 조회할 수 있는 링크

- Spring HATEOAS (<https://docs.spring.io/spring-hateoas/docs/current/reference/html/>)
- HEATEOAS API (<https://docs.spring.io/spring-hateoas/docs/current/api/>)

HATEOAS : Resource 와 Links

- Resource와 Links를 저장하는 클래스 : LectureResource 클래스 작성
- HATEOAS 기능을 제공하는 RepresentationModel 클래스를 상속 받아 Lecture 클래스의 Resource를 저장해주는 LectureResource 클래스를 작성 합니다.
- DTO 객체는 RepresentationModel 클래스를 상속 받아야 Link를 추가(add) 할 수 있다.

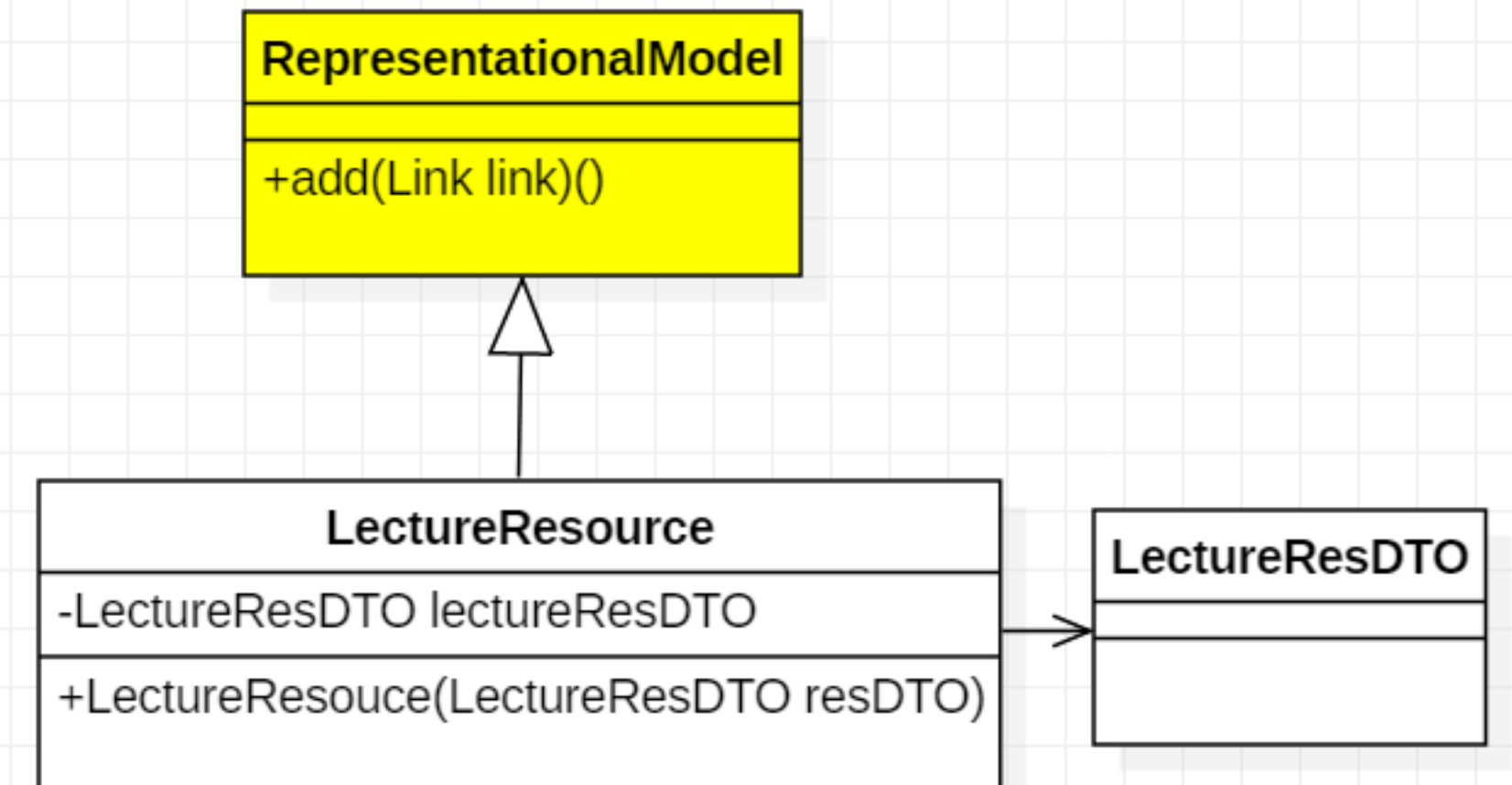
lectures/LectureResource.java

```
import org.springframework.hateoas.RepresentationModel;

public class LectureResource extends RepresentationModel<LectureResource> {
    private LectureResDto lectureResDto;

    public LectureResource(LectureResDto resDto) {
        this.lectureResDto = resDto;
    }

    public LectureResDto getLectureResDto() {
        return lectureResDto;
    }
}
```



HATEOAS : Resource 와 Links

- LectureResource 객체 사용 : LectureController 클래스 수정
 - WebMvcLinkBuilder를 사용해서 컨트롤러와 매핑된 URL + 이벤트 ID로 selfLinkBuilder 객체를 만듭니다.
링크 URL은 `http://localhost:8080/api/lectures/{id}` 와 같습니다.
 - Lecture 객체를 LectureResDto로 매핑하여 LectureResource 객체에 저장하여 HTTP 응답 메시지에 담을 수 있습니다.

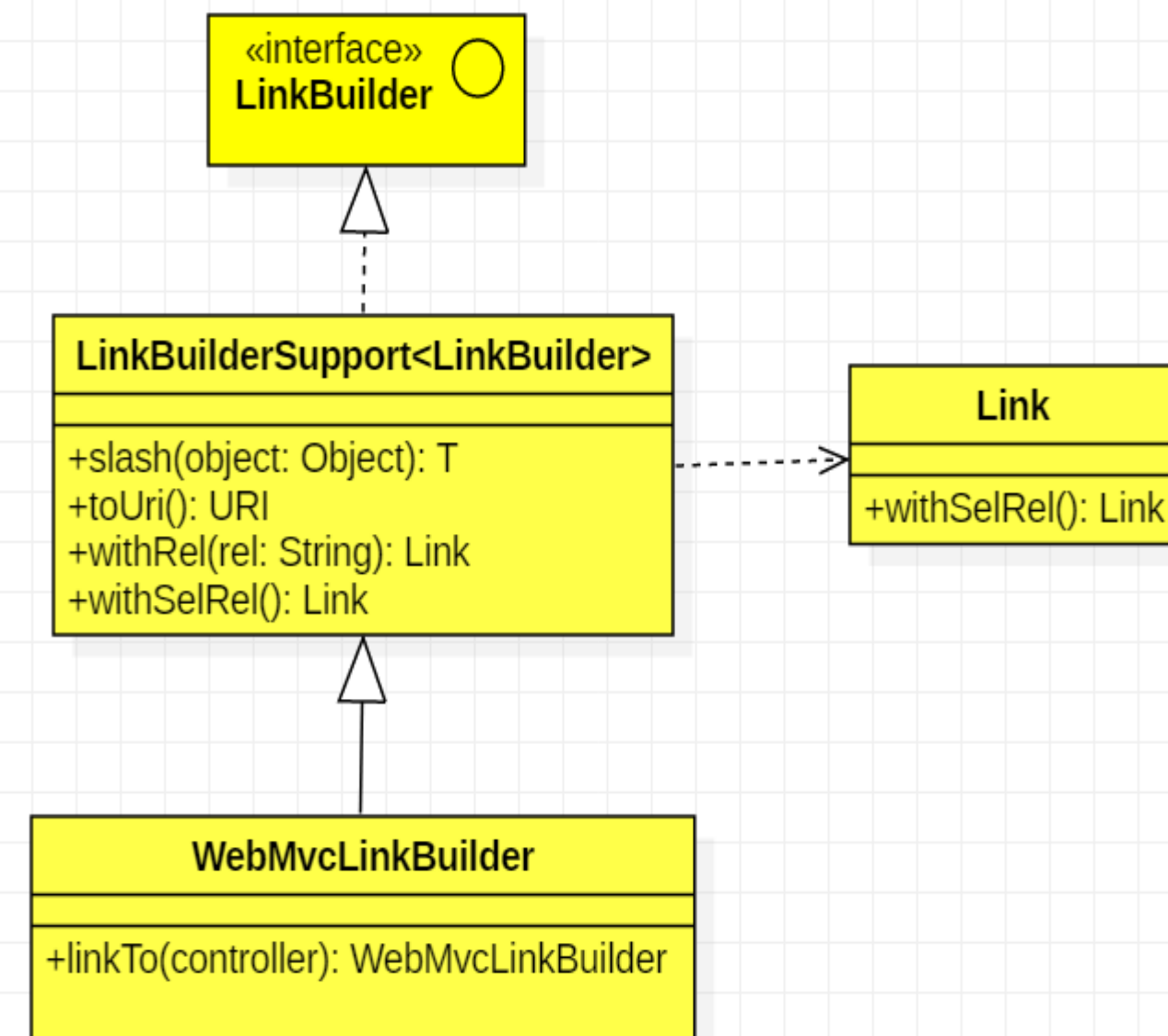
lectures/LectureController.java

```
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.*;

public ResponseEntity createLecture(@RequestBody LectureReqDto lectureReqDto, Errors errors) {
    .....
    Lecture addLecture = this.lectureRepository.save(lecture);
    LectureResDto lectureResDto = modelMapper.map(addLecture, LectureResDto.class);
    WebMvcLinkBuilder selfLinkBuilder = linkTo(LectureController.class).slash(lectureResDto.getId());
    URI createUri = selfLinkBuilder.toUri();

    LectureResource lectureResource = new LectureResource(lectureResDto);
    lectureResource.add(linkTo(LectureController.class).withRel("query-lectures"));
    lectureResource.add(selfLinkBuilder.withSelfRel());
    lectureResource.add(selfLinkBuilder.withRel("update-lecture"));

    return ResponseEntity.created(createUri).body(lectureResource);
}
```



HATEOAS : Resource 와 Links

- "lectureResDto"로 wrapping 하지 않기 : LectureResource 클래스 수정
 - @JsonUnwrapped : Jackson에게 어떤 필드의 값에 대해 Wrapping 하지 말고 펼쳐 달라는 의미

lectures/LectureResource.java

```
import com.fasterxml.jackson.annotation.JsonUnwrapped;

public class LectureResource extends RepresentationModel<LectureResource> {

    @JsonUnwrapped
    private LectureResDto lectureResDto;

    public LectureResource(LectureResDto resDto) {
        this.lectureResDto = resDto;
    }

    public LectureResDto getLectureResDto() {
        return lectureResDto;
    }
}
```

HATEOAS : Resource 와 Links

- self를 LectureResource에 추가 : LectureResource 클래스 수정
 - Lecture 객체 자신을 나타내는 self를 LectureResource에 추가 합니다.
add(linkTo(LectureController.class).slash(resDto.getId()).withSelfRel()) 는 type safe 하다.
같은 코드 => add(new Link("http://localhost:8080/api/lectures" + lecture.getId())) 는 Type Safe 하지 않다
 - Self 링크를 추가하는 코드를 LectureResource에 추가 합니다.
 - LectureController에 있는 lectureResource.add(selfLinkBuilder.withSelfRel()); 코드는 제거해도 됩니다.

lectures/LectureResource.java

```
import org.springframework.hateoas.RepresentationModel;
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.linkTo;

public class LectureResource extends RepresentationModel<LectureResource> {
    @JsonUnwrapped
    private LectureResDto lectureResDto;

    public LectureResource(LectureResDto resDto) {
        this.lectureResDto = resDto;
        add(linkTo(LectureController.class).slash(resDto.getId()).withSelfRel());
    }
}
```

HATEOAS : API Index 만들기

- API 인덱스 : IndexController 클래스 작성
 - 인덱스 핸들러

common/IndexController.java

```
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.linkTo;

@RestController
public class IndexController {

    @GetMapping("/api")
    public RepresentationModel index() {
        var index = new RepresentationModel ();
        index.add(linkTo(LectureController.class).withRel("lectures"));
        return index;
    }
}
```

GET



http://localhost:8080/api

```
{
  "_links": {
    "lectures": {
      "href": "http://localhost:8080/api/lectures"
    }
  }
}
```

HATEOAS : Index Link 만들기

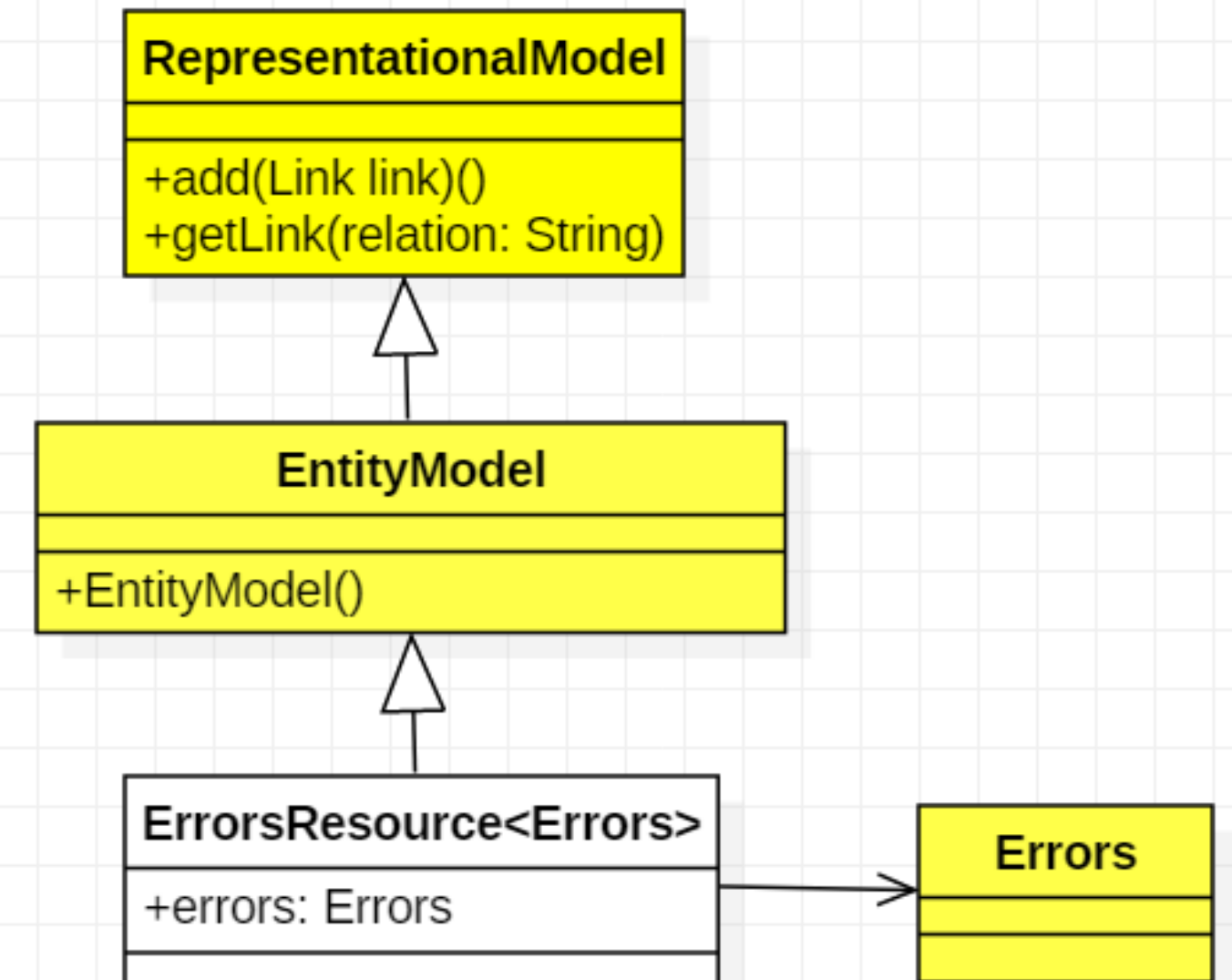
- API 인덱스 : ErrorsResource 클래스 작성 / LectureController 클래스 수정
 - 에러가 발생했을 때에도 Index 링크를 제공하기 위해서 ErrorsResource 클래스를 작성합니다.
 - LectureController에서 Errors 객체를 ErrorsResource에 Wrapping 하여 리턴 합니다.

common/ErrorsResource.java

```
import org.springframework.hateoas.EntityModel;
import org.springframework.validation.Errors;
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.linkTo;
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.methodOn;

public class ErrorsResource extends EntityModel<Errors> {
    @JsonUnwrapped
    private Errors errors;

    public ErrorsResource(Errors content) {
        this.errors = content;
        add(linkTo(methodOn(IndexController.class).index()).withRel("index"));
    }
}
```



HATEOAS : Index Link 만들기

- API 인덱스 : LectureController 클래스 수정
- LectureController에서 Errors 객체를 ErrorsResource에 Wrapping 하여 리턴 합니다.

lectures/LectureController.java

```
public class LectureController {  
    public ResponseEntity createLecture(@RequestBody @Valid LectureReqDto lectureReqDto, Errors errors) {  
        if(errors.hasErrors()) {  
            return ResponseEntity.badRequest().body(new ErrorsResource(errors));  
        }  
        LectureValidator.validate(lectureReqDto, errors);  
        if(errors.hasErrors()) {  
            return ResponseEntity.badRequest().body(new ErrorsResource(errors));  
        }  
    }  
    }  
}
```

- 반복되는 코드는 Refactoring 필요함

Lecture 생성 API 구현 : 페이징 처리 데이터 준비하기

- 페이징 데이터 준비하기 : LectureInsertRunner 클래스 작성

runner/LectureInsertRunner.java

```
@Component
public class LectureInsertRunner implements ApplicationRunner {
    @Autowired
    LectureRepository lectureRepository;
    @Override
    public void run(ApplicationArguments args) throws Exception {
        IntStream.range(0, 15).forEach(this::generateLecture);
    }
    private Lecture generateLecture(int index) {
        Lecture lecture = buildLecture(index);
        return this.lectureRepository.save(lecture);
    }
    private Lecture buildLecture(int index) {
        return Lecture.builder()
            .name(index + " event ")
            .description("test event")
            .beginEnrollmentDateTime(LocalDate.of(2022, 11, 23, 14, 21))
            .closeEnrollmentDateTime(LocalDate.of(2022, 11, 24, 14, 21))
            .beginLectureDateTime(LocalDate.of(2022, 11, 25, 14, 21))
            .endLectureDateTime(LocalDate.of(2022, 11, 26, 14, 21))
            .basePrice(100)
            .maxPrice(200)
            .limitOfEnrollment(100)
            .location(index + " 강의장")
            .free(false)
            .offline(true)
            .lectureStatus(LectureStatus.DRAFT)
            .build();
    }
}
```

[JavaSE 함수형 인터페이스 API doc](#)

[JavaSE Stream API doc](#)

람다식을 이해하기 위한 기본 테스트케이스 예제

<https://gist.github.com/mysoyul/78386ea3d1ec5fbe854a55ecb5d896f5>

<https://gist.github.com/mysoyul/d136bcf7aa7f3270a319a0d0c3152ba9>

Lecture 목록 조회 API 구현 : JPA 페이징

Spring Data JPA의 페이징과 정렬

- Spring Data JPA에서는 쿼리 메소드에 페이징과 정렬 기능을 제공하는 2가지 클래스를 제공
 - org.springframework.data.domain.Sort : 정렬 기능
 - org.springframework.data.domain.Pageable : 페이징 기능
- Pageable 인터페이스는 Paging 하기 위한 파라미터들을 담은 PageRequest 같은 객체에 접근하기 위한 역할을 합니다. 이 매개변수를 통해 Paging 하기 위한 정보를 담은 객체가 들어오게 되고 이것을 통해 자동적으로 Paging에 필요한 데이터를 처리하여 반환하게 되는 것입니다.
 - page : 검색을 원하는 페이지 번호를 나타냅니다.
 - size : 한 페이지의 보여 줄 게시물 개수를 나타냅니다.
 - sort : 정렬 방식을 나타냅니다.

Spring Data Core Api doc

<https://docs.spring.io/spring-data/commons/docs/current/api/index.html>

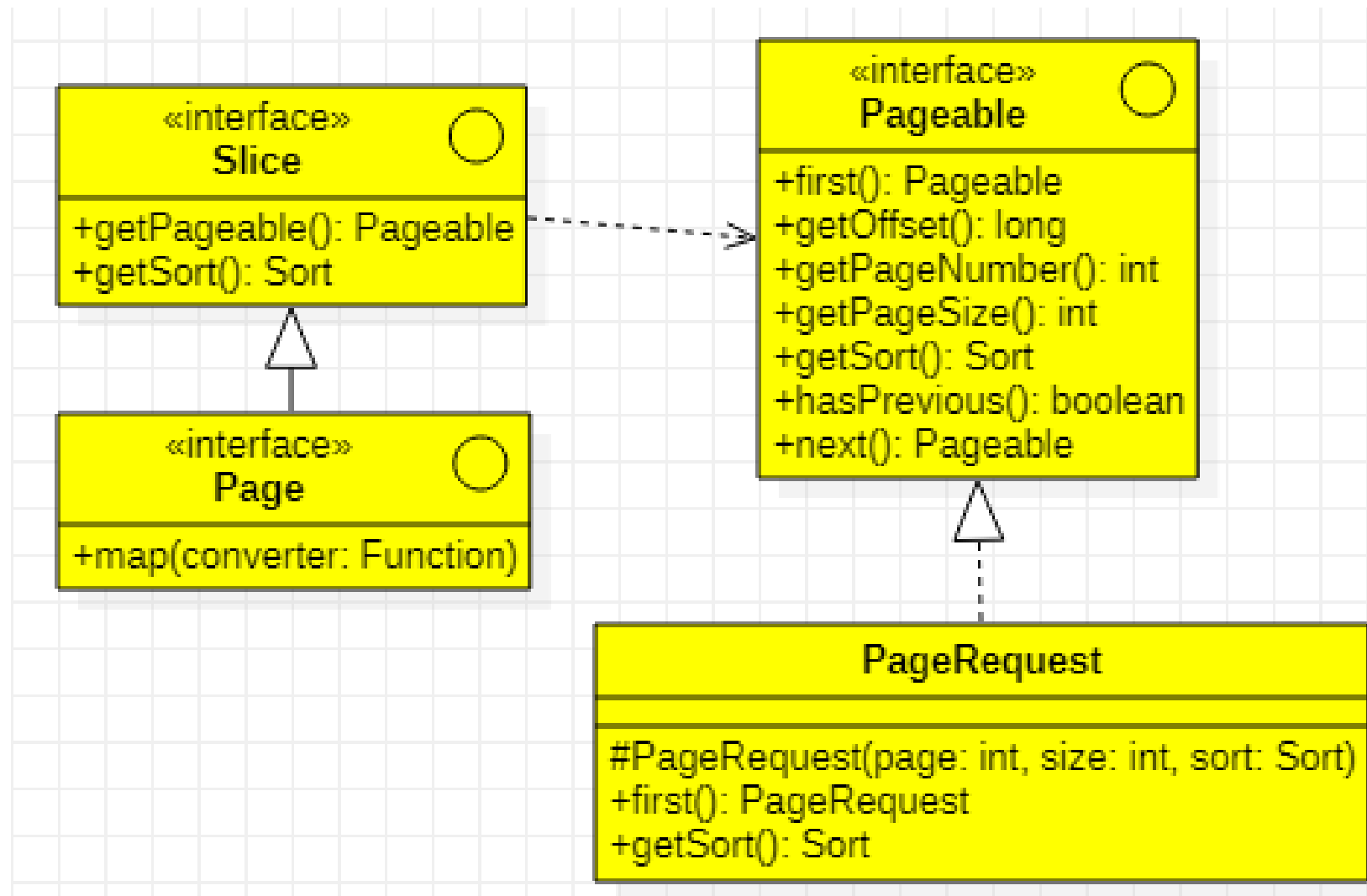
Spring Data JPA Api doc

<https://docs.spring.io/spring-data/jpa/docs/current/api/index.html>

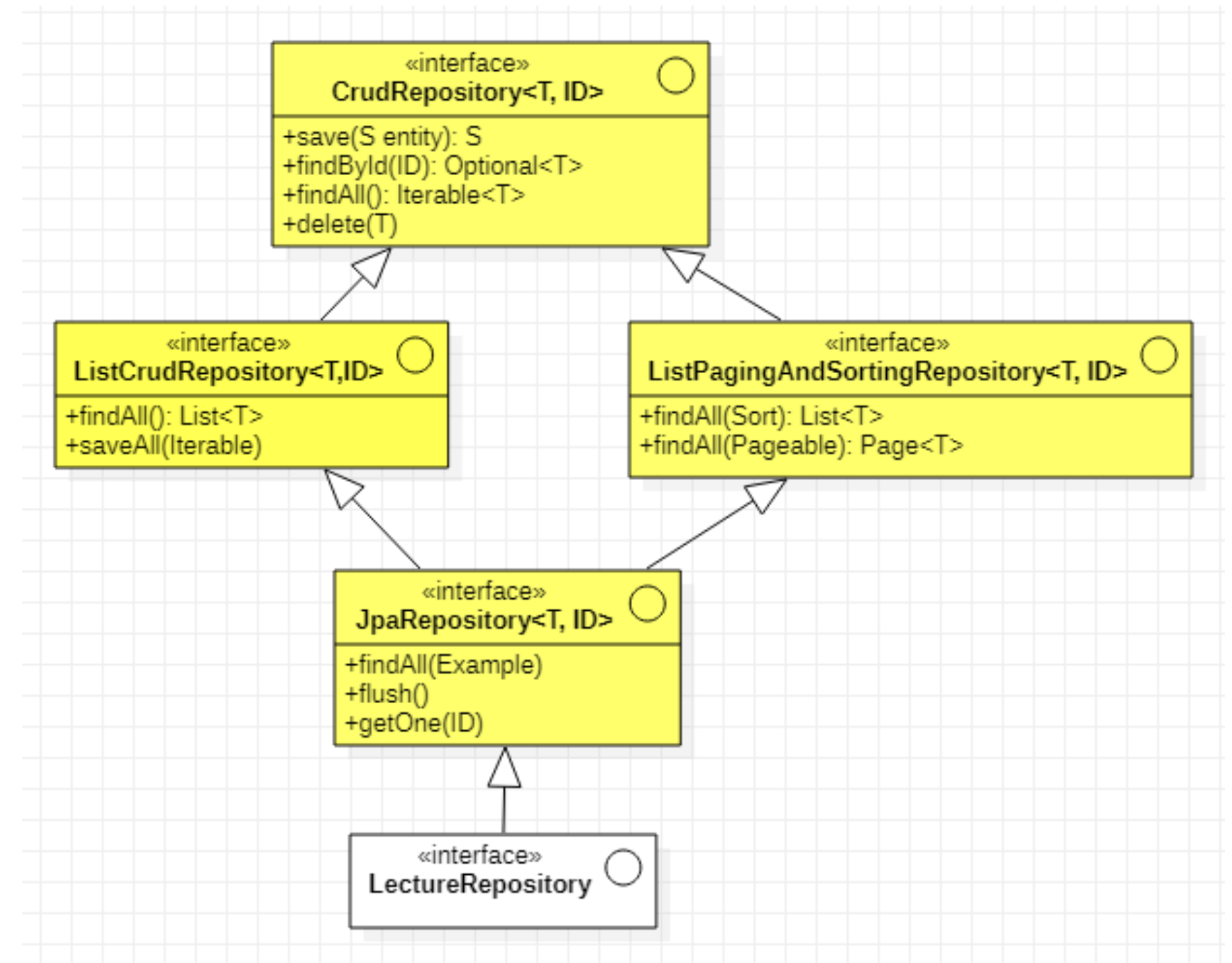
Lecture 목록 조회 API 구현 : JPA 페이징

Spring Data JPA의 페이징과 정렬

Spring Data Core



Spring Data JPA



Lecture 목록 조회 API 구현 : 페이징 처리 메서드 #1

- Lecture 목록 조회 API 구현 : LectureController 클래스 queryLectures() 메서드
- Pageable 인터페이스를 받아서 LectureRepository 메서드 findAll(pageable)로 넘기면 됩니다.

lectures/LectureController.java

```
import org.springframework.data.domain.Pageable;

public class LectureController {

    @GetMapping
    public ResponseEntity queryLectures(Pageable pageable) {
        Page<Lecture> lecturePage = this.lectureRepository.findAll(pageable);
        return ResponseEntity.ok(lecturePage);
    }
}
```

Lecture 목록 조회 API 구현 : 페이징 처리 메서드 호출하기

- Lecture 목록 조회 API 구현
- Sort과 Paging 확인
 - 15개를 만들고, 5개 사이즈로 조회 한다.
 - Lecture Name 순서대로 정렬 되어 있다.
 - 요청 파라미터
 - 1) page : 검색을 원하는 페이지 번호입니다.
 - 2) size : 한 페이지의 게시물 개수를 나타냅니다.
 - 3) sort : 정렬 방식을 나타냅니다.

GET	http://localhost:8088/api/events?page=1&size=5&sort=name,DESC	
Params	Authorization	Headers (8)
Body		
Pre-request Script		
Tests		
Query Params		
	KEY	VALUE
<input checked="" type="checkbox"/>	page	1
<input checked="" type="checkbox"/>	size	5
<input checked="" type="checkbox"/>	sort	name,DESC

http://localhost:8080/api/lectures?page=1&size=5&sort=name,DESC

Lecture 목록 조회 API 구현 : 페이징 처리 메서드 #2

- Lecture 목록 조회 API 구현 : LectureController 클래스 수정
 - "first", "prev", "self", "next", "last" 페이지로 가는 링크가 있어야 한다.
 - PagedResourcesAssembler는 Page 객체들을 손쉽게 HATEOAS의 PagedModel 객체로 변환 해주는 작업을 수행한다.
 - PagedModel은 Pageable 한 객체의 집합을 나타냅니다.
 - PagedModel 객체는 Pageable 한 객체를 모아서 전달하는 DTO 역할을 하게 됩니다.

```
"_links": {
  "first": {
    "href": "http://localhost:8080/api/lectures?page=0&size=5&sort=id,desc"
  },
  "prev": {
    "href": "http://localhost:8080/api/lectures?page=0&size=5&sort=id,desc"
  },
  "self": {
    "href": "http://localhost:8080/api/lectures?page=1&size=5&sort=id,desc"
  },
  "next": {
    "href": "http://localhost:8080/api/lectures?page=2&size=5&sort=id,desc"
  },
  "last": {
    "href": "http://localhost:8080/api/lectures?page=2&size=5&sort=id,desc"
  }
},
"page": {
  "size": 5,
  "totalElements": 15,
  "totalPages": 3,
  "number": 1
}
```

PagedResourceAssembler

PagedModel<T>

EntityModel

Lecture 목록 조회 API 구현 : 페이징 처리 메서드 #2

- Lecture 목록 조회 API 구현 : LectureController 클래스 수정

lectures/LectureController.java

```
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.web.PagedResourcesAssembler;
import org.springframework.hateoas.PagedModel;
import org.springframework.hateoas.EntityModel;

@GetMapping
public ResponseEntity queryLectures(Pageable pageable, PagedResourcesAssembler<LectureResDto> assembler) {
    Page<Lecture> page = this.lectureRepository.findAll(pageable);
    Page<LectureResDto> lectureResDtoPage = page.map(lecture -> modelMapper.map(lecture, LectureResDto.class));
    PagedModel<EntityModel<LectureResDto>> pagedResources = assembler.toModel(lectureResDtoPage);
    return ResponseEntity.ok(pagedResources);
}
```

Data Core의 클래스

PagingAndSortingRepository<T, ID>

Page

Class PagedResourcesAssembler<T>

```
public org.springframework.hateoas.PagedModel<org.springframework.hateoas.EntityModel<T>> toModel(Page<T> entity)
```

HATEOAS의 클래스들

PagedResourceAssembler

PagedModel<T>

EntityModel

Lecture 목록 조회 API 구현 : 페이징 처리 메서드 #3

- Lecture 목록 조회 API 구현 : 각각의 Lecture에 Self 링크 추가
 - 각각의 Lecture 로 전이 할 수 있는 Self 링크가 있어야 합니다.

```
{
  "id": 6,
  "name": "5 Lecture ",
  "description": "Test Lecture",
  "beginEnrollmentDateTime": "2022-11-23 14:21",
  "closeEnrollmentDateTime": "2022-11-24 14:21",
  "beginLectureDateTime": "2022-11-25 14:21",
  "endLectureDateTime": "2022-11-26 14:21",
  "location": "5 강의장",
  "basePrice": 100,
  "maxPrice": 200,
  "limitOfEnrollment": 100,
  "offline": true,
  "free": false,
  "email": null,
  "_links": {
    "self": {
      "href": "http://localhost:8080/api/lectures/6"
    }
  }
}
```

Lecture 목록 조회 API 구현 : 페이징 처리 메서드 #3-1

■ Lecture 목록 조회 API 구현 : 각각의 Lecture에 Self 링크 추가

Package `org.springframework.data.web`

Class `PagedResourcesAssembler<T>`

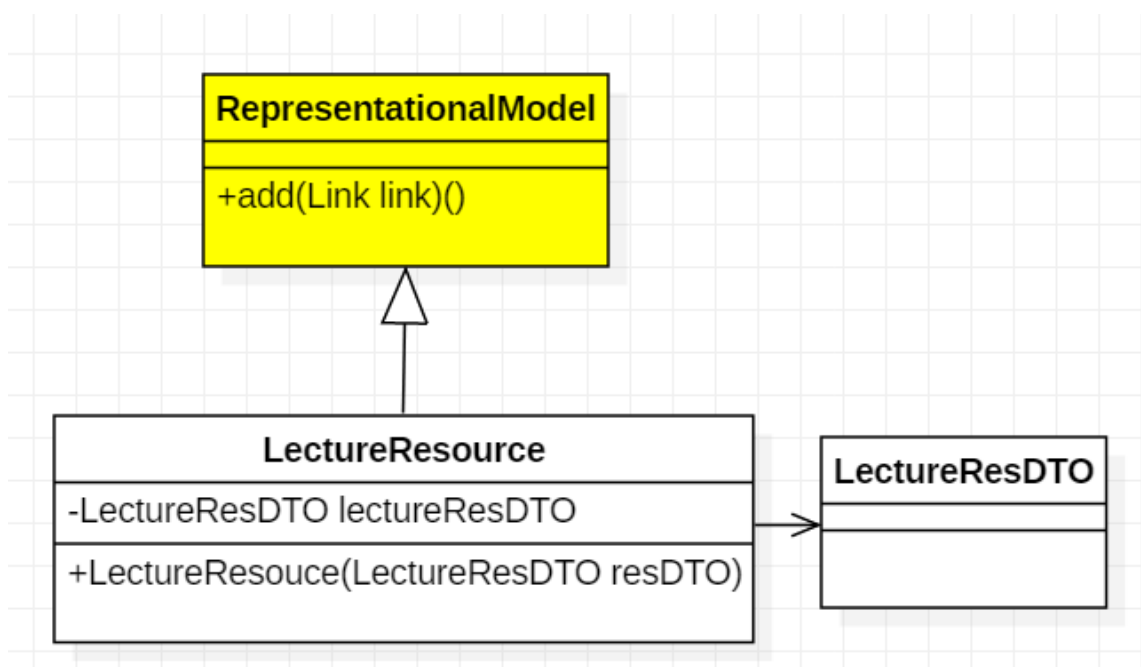
```
public <R extends org.springframework.hateoas.RepresentationModel<?>>  
org.springframework.hateoas.PagedModel<R> toModel(Page<T> page,  
                                                org.springframework.hateoas.server.RepresentationModelAssembler<T,R> assembler)
```

Package `org.springframework.hateoas.server`

Interface `RepresentationModelAssembler<T,D extends RepresentationModel<?>>`

`D toModel(T entity)`

Converts the given entity into a `D`, which extends `RepresentationModel`.



@GetMapping

```
public ResponseEntity queryLectures(Pageable pageable, PagedResourcesAssembler<LectureResDto> assembler) {  
    Page<Lecture> page = this.LectureRepository.findAll(pageable);  
    Page<LectureResDto> lectureResDtoPage = page.map(lecture -> modelMapper.map(lecture, LectureResDto.class));  
    PagedModel<LectureResource> pagedResources =  
        assembler.toModel(lectureResDtoPage, lectureResDto -> new LectureResource(lectureResDto));  
    return ResponseEntity.ok(pagedResources);  
}
```


Lecture 목록 조회 API 구현 : 페이징 처리 메서드 #3-2

- Lecture 목록 조회 API 구현 : 각각의 Lecture에 Self 링크 추가
- 각 Lecture에 Self 링크를 추가하기 위해 LectureResDto 객체를 LectureResource 객체로 Wrapping 해줍니다.

lectures/LectureController.java

```
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.web.PagedResourcesAssembler;
import org.springframework.hateoas.PagedModel;
import org.springframework.hateoas.RepresentationModel;

@GetMapping
public ResponseEntity queryLectures(Pageable pageable, PagedResourcesAssembler<LectureResDto> assembler) {
    Page<Lecture> page = this.LectureRepository.findAll(pageable);
    Page<LectureResDto> lectureResDtoPage = page.map(lecture -> modelMapper.map(lecture, LectureResDto.class));
    PagedModel<LectureResource> pagedResources =
        assembler.toModel(lectureResDtoPage, lectureResDto -> new LectureResource(lectureResDto));
    return ResponseEntity.ok(pagedResources);
}
```

Lecture 조회 API 구현 : 조회 메서드

- Lecture 조회 API 구현 : LectureController 클래스 getLecture(id) 메서드
 - 조회하는 Lecture가 있는 경우 Lecture Resource 확인
 - 조회하는 Lecture가 없는 경우 404 응답 확인

lectures/LectureController.java

```
@GetMapping("/{id}")
public ResponseEntity getLecture(@PathVariable Integer id) {
    Optional<Lecture> optionalLecture = this.lectureRepository.findById(id);
    if(optionalLecture.isEmpty()) {
        return ResponseEntity.notFound().build();
    }

    Lecture lecture = optionalLecture.get();
    LectureResDto lectureResDto = modelMapper.map(lecture, LectureResDto.class);
    LectureResource lectureResource = new LectureResource(lectureResDto);
    return ResponseEntity.ok(lectureResource);
}
```

Optional

<https://gist.github.com/mysoyu1/9497eae0c5f3dfc65f9295fd68c6e2b6>

Lecture 조회 API 구현 : 사용자정의 Exception

- Spring Boot Web MVC : 사용자 정의 Exception 클래스

exception/BusinessException.java

```
import org.springframework.http.HttpStatus;
@Getter
public class BusinessException extends RuntimeException {

    private static final long serialVersionUID = 1L;
    private String message;
    private HttpStatus httpStatus;

    public BusinessException(String message) {
        //417
        this(message, HttpStatus.EXPECTATION_FAILED);
    }

    public BusinessException(String message, HttpStatus httpStatus) {
        this.message = message;
        this.httpStatus = httpStatus;
    }
}
```

BusinessException 클래스 (exception 패키지)

<https://gist.github.com/mysoyul/c13d51d2138d8697980a3eee20755fa7>

Lecture 조회 API 구현 : 사용자정의 Exception

- Spring Boot Web MVC : 사용자 정의 Exception 클래스

exception/SystemException.java

```
@Getter
public class SystemException extends RuntimeException {
    private static final long serialVersionUID = 1L;
    private String message;
    private HttpStatus httpStatus;
    private Throwable throwable;

    public SystemException(Exception e) {
        this(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }

    public SystemException(String message) {
        this(message, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

SystemException 클래스 (exception 패키지)

<https://gist.github.com/mysoyul/608fc1fa017143fcea9e1e848615e0b4>

Lecture 조희 API 구현 : @ExceptionHandler와 @ControllerAdvice

- Spring Boot Web MVC : 예외처리
 - 스프링 부트에서는 `ExceptionHandler`를 기본적으로 등록하여 `Exception`을 처리하고 있습니다.
 - 기본 예외 처리기는 스프링에서 자동적으로 등록하는 `BasicExceptionHandler`에서 관리합니다.
(에러 발생 시 JSON 형식으로 리턴)
 - 커스텀 `Exception` 핸들러, 커스텀 `Exception` 클래스를 만들어서 예외를 처리할 수 있습니다.
- 스프링 @MVC 예외 처리 방법
- @ExceptionHandler
 - : 스프링은 컨트롤러 안에 `@ExceptionHandler`를 선언한 메서드가 존재하면 그 메서드가 컨트롤러 내부의 예외를 처리하도록 해준다.
- @ControllerAdvice
 - : `@ControllerAdvice` 애노테이션을 통해서 이 클래스의 객체가 컨트롤러에서 발생하는 `Exception`을 전문적으로 처리하는 클래스라는 것을 명시한다.

Lecture 조회 API 구현 : Advice 클래스

▪ Spring Boot Web MVC : @RestControllerAdvice

exception/advice/DefaultExceptionAdvice.java

```
@RestControllerAdvice
public class DefaultExceptionAdvice {
    private final Logger LOGGER = LoggerFactory.getLogger(DefaultExceptionAdvice.class);

    @ExceptionHandler(BusinessException.class)
    protected ResponseEntity<Object> handleException(BusinessException e) {
        Map<String, Object> result = new HashMap<String, Object>();
        result.put("message", "[안내] " + e.getMessage());
        result.put("httpStatus", e.getHttpStatus().value());

        return new ResponseEntity<>(result, e.getHttpStatus());
    }
}
```

DefaultExceptionAdvice 클래스 (exception/advice 패키지)

<https://gist.github.com/mysoyul/1cbffb8d39ad30cc58e308182d0b5131>

Lecture 수정 API 구현 : 수정 메서드

- Lecture 수정 API 구현 : LectureController 클래스 updateLecture()
 - 수정하려는 이벤트가 없는 경우 404 NOT_FOUND
 - 입력 데이터 (데이터 바인딩)가 없거나 도메인 로직으로 데이터 검증 실패하면 400 BAD_REQUEST
 - 정상적으로 수정한 경우에 LectureResource 응답 확인

lectures/LectureController.java

```
@PostMapping("/{id}")
public ResponseEntity updateLecture(@PathVariable Integer id,
                                   @RequestBody @Valid LectureReqDto lectureReqDto,
                                   Errors errors) {
    Optional<Lecture> optionalLecture = lectureRepository.findById(id);
    if (optionalLecture.isEmpty()) {
        return ResponseEntity.notFound().build();
    }

    if (errors.hasErrors()) {
        return badRequest(errors);
    }
    this.lectureValidator.validate(lectureReqDto, errors);
    if (errors.hasErrors()) {
        return badRequest(errors);
    }
}
```

lectures/LectureController.java

```
Lecture existingLecture = optionalLecture.get();
this.modelMapper.map(lectureDto, existingLecture);
Lecture savedLecture = this.lectureRepository.save(existingLecture);
LectureResDto lectureResDto = modelMapper.map(savedLecture,
LectureResDto.class);

LectureResource lectureResource = new LectureResource(lectureResDto);
return ResponseEntity.ok(lectureResource);
}
```

<https://gist.github.com/mysoyu1/5ef9f57eb7adf5bd8f97cd4f52015358>

REST API 보안 적용

스프링 Security : 개요

■ 스프링 Security 란?

스프링 security는 스프링 기반의 어플리케이션의 보안(인증과 권한)을 담당하는 프레임워크이다.

* 보안 관련 용어

- 접근 주체(Principal) : 보호된 대상에 접근하는 user
- 인증(Authenticate) : 현재 user가 누구인지 확인, 어플리케이션의 작업을 수행할 수 있는 주체임을 증명함
- 인가(Authorize) : 현재 user가 어떤 서비스, 페이지에 접근할 수 있는 권한이 있는지 검사

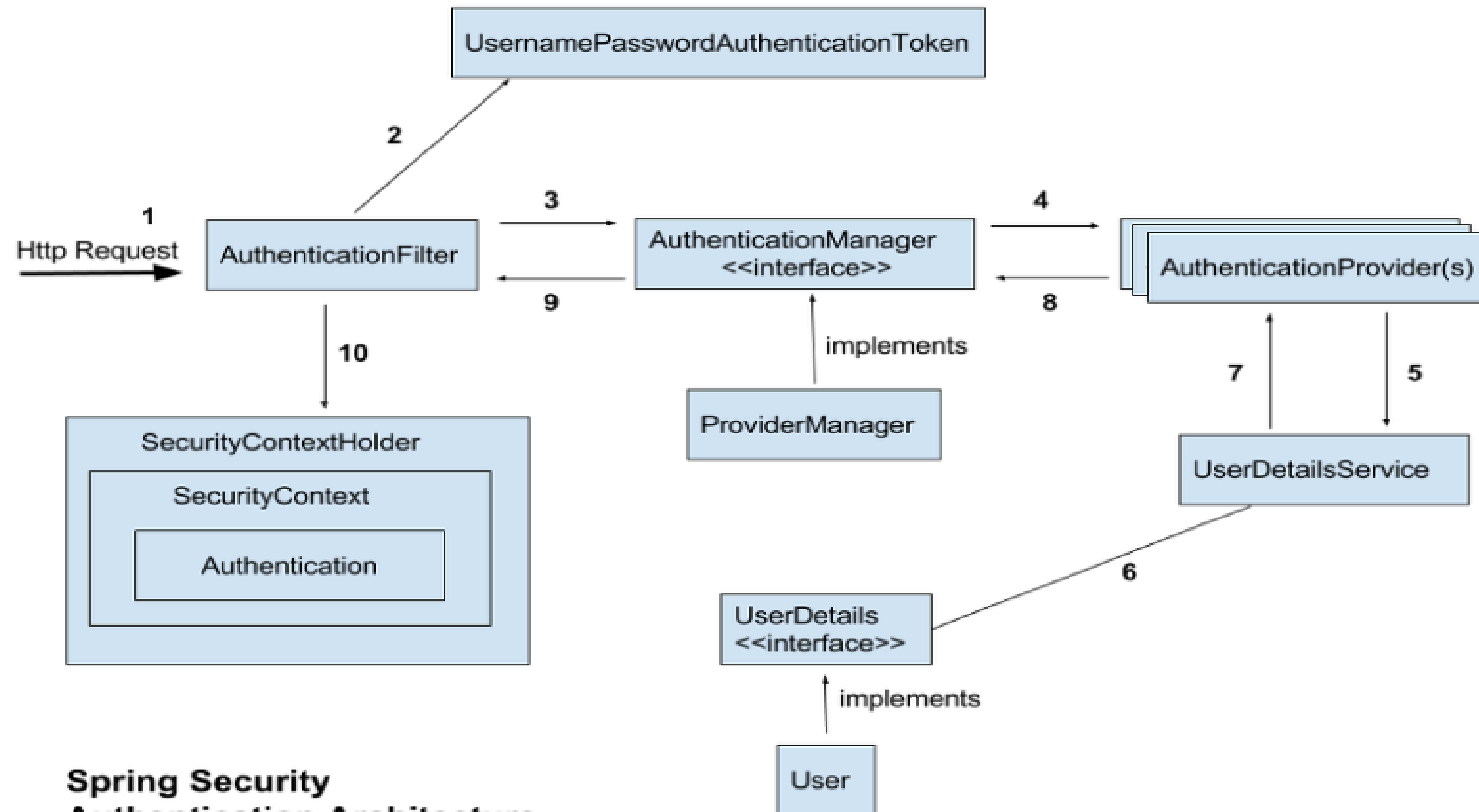
* 스프링 시큐리티가 제공하는 기능

- 웹 시큐리티 (Filter 기반 시큐리티)
- 메소드 시큐리티
- 두가지 방법 모두 Security Interceptor를 사용합니다.

: 리소스에 접근을 허용할 것이냐 말 것 이냐를 결정하는 로직이 들어 있음

스프링 Security : 인증관련 Architecture

- 인증관련 Architecture



Spring Security
Authentication Architecture

스프링 Security : 인증관련 Architecture

- 1) 사용자가 Form을 통해 로그인 정보를 입력하고 인증 요청을 보낸다.
- 2) UsernamePasswordAuthenticationFilter가 사용자가 보낸 아이디와 패스워드를 인터셉트한다.
HttpServletRequest에서 꺼낸 아이디와 패스워드로 실제 인증을 담당 할 AuthenticationManager (구현체-ProviderManager)에게 인증용 객체(UsernamePasswordAuthenticationToken)로 만들어 주어 위임한다.
- 3) AuthenticationFilter에게 인증용 객체(UsernamePasswordAuthenticationToken)을 전달받는다.
- 4) 실제 인증을 할 AuthenticationProvider에게 Authentication객체(UsernamePasswordAuthenticationToken)을 다시 전달한다.
- 5) DB에서 사용자 인증 정보를 가져올 UserDetailsService 객체에게 사용자 아이디를 넘겨주고 DB에서 인증에 사용할 사용자 정보 (사용자 아이디, 암호화된 패스워드, 권한 등)를 UserDetails라는 객체로 전달 받는다.
- 6) AuthenticationProvider는 UserDetails 객체를 전달 받은 이후 실제 사용자의 입력정보와 UserDetails 객체를 사용하여 인증을 시도한다.
- 7) 7,8,9,10. 인증이 완료되면 사용자 정보를 가진 Authentication 객체를 SecurityContextHolder에 담은 이후 AuthenticationSuccessHandler를 실행한다. (실패시 AuthenticationFailureHandler를 실행한다.)

스프링 **Security** 기본 설정 : 의존성 추가

- SpringBoot Security 의존성 추가

pom.xml

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

Spring Security without the WebSecurityConfigurerAdapter

<https://spring.io/blog/2022/02/21/spring-security-without-the-websecurityconfigureradapter>

스프링 Security 기본 설정

- 설정 파일에 Username와 Password 설정

http://localhost:8080/users/welcome

application.properties

```
spring.security.user.name=boot3
spring.security.user.password=test1234
```

userinfo/UserInfoController.java

```
@RestController
@RequestMapping("/users")
public class UserInfoController {

    @GetMapping("/welcome")
    public String welcome() {
        return "Welcome this endpoint is not secure";
    }

}
```

- InMemoryUserDetailsManager 객체 사용

config/SecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    //authentication
    public UserDetailsService userDetailsService(PasswordEncoder encoder) {
        UserDetails admin = User.withUsername("adminboot")
            .password(encoder.encode("pwd1"))
            .roles("ADMIN")
            .build();
        UserDetails user = User.withUsername("userboot")
            .password(encoder.encode("pwd2"))
            .roles("USER")
            .build();
        return new InMemoryUserDetailsManager(admin, user);
    }

}
```

스프링 Security 기본 설정

- SecurityFilterChain 사용하여 SecurityConfig 클래스 작성

config/SecurityConfig.java

```
import static org.springframework.security.config.Customizer.withDefaults;

@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    return http.csrf(csrf -> csrf.disable())
        .authorizeHttpRequests(auth -> {
            auth.requestMatchers("/users/welcome").permitAll()
            .requestMatchers("/api/lectures/**").authenticated();
        })
        .formLogin(withDefaults())
        .build();
}
```

A Review of Filters

<https://docs.spring.io/spring-security/reference/servlet/architecture.html>

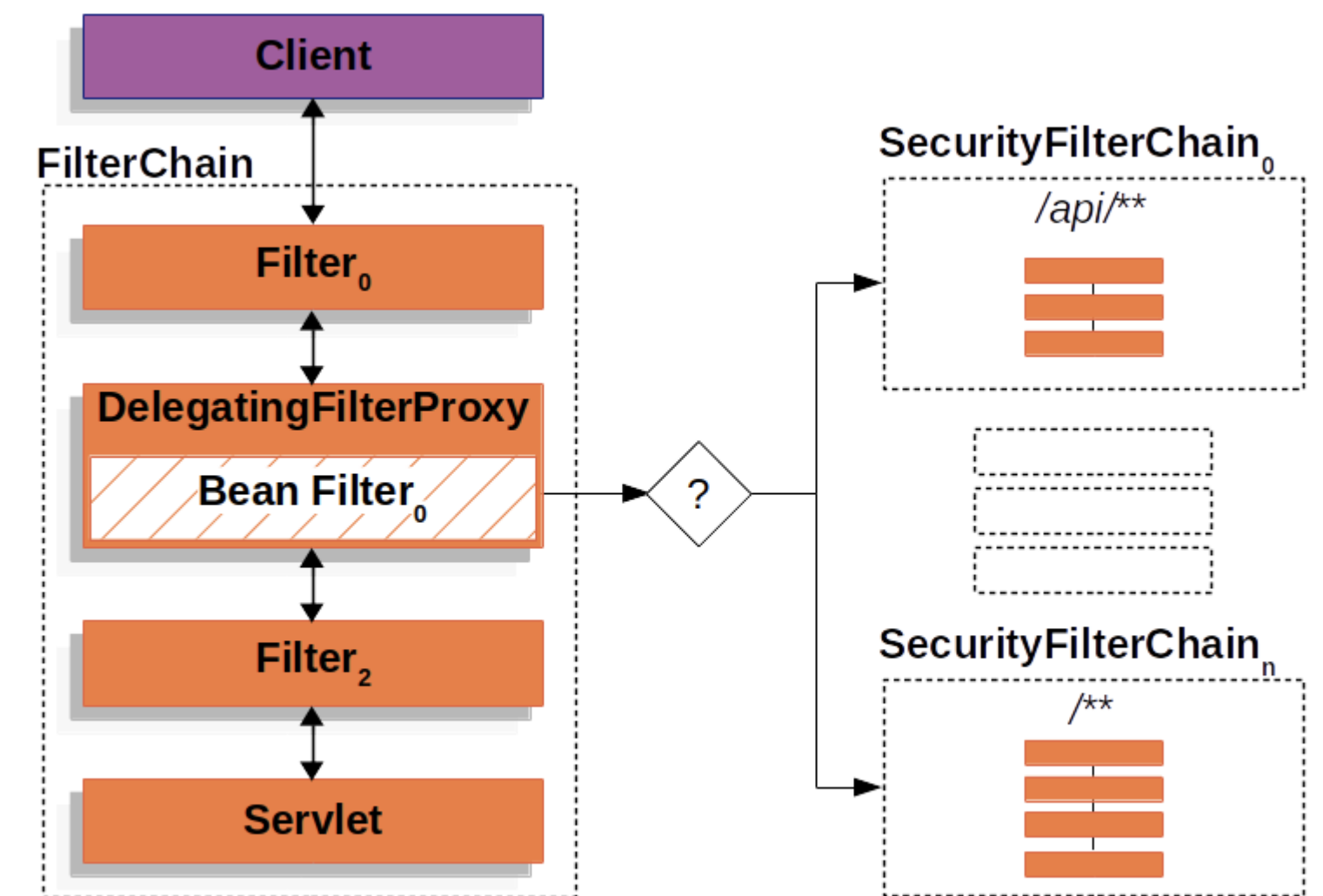


Figure 5. Multiple SecurityFilterChain

스프링 Security 기본 설정

- @EnableMethodSecurity
- @PreAuthorize는 해당 메서드가 호출되기 이전에 해당 메서드를 호출할 권한이 있는지를 확인한다.
- hasAuthority('ROLE_ADMIN') 는 ' ROLE_ ' 접두사가 자동으로 추가 되기 때문에 hasRole('ADMIN') 과 유사합니다 .
- @PreAuthorize는 관리자 권한(ROLE_ADMIN)이 없는 username으로 로그인하면 관리자 권한이 없다는 403에러를 발생시킨다.

lectures/LectureController.java

```
@RestController
@RequestMapping("/api/lectures")
public class LectureController {

    @GetMapping("/all")
    @PreAuthorize("hasAuthority('ROLE_ADMIN')")
    public ResponseEntity queryLectures(..) {
        ...
    }

    @GetMapping("/{id}")
    @PreAuthorize("hasAuthority('ROLE_USER')")
    public ResponseEntity getLecture( ..) {
        ..
    }
}
```

config/SecurityConfig.java

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {
    ....
}
```

스프링 Security 기본 설정 : UserInfo 와 UserInfoUserDetails 클래스

- UserInfo Entity 클래스와 UserInfoUserDetails 클래스 작성

userinfo/UserInfo.java

```
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class UserInfo {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Column(nullable = false)
    private String name;
    @Column(unique = true, nullable = false)
    private String email;
    private String password;
    private String roles;
}
```

userinfo/UserInfoUserDetails.java

```
import org.springframework.security.core.userdetails.UserDetails;

public class UserInfoUserDetails implements UserDetails {
    private String name;
    private String password;
    private List<GrantedAuthority> authorities;

    public UserInfoUserDetails(UserInfo userInfo) {
        name=userInfo.getEmail();
        password=userInfo.getPassword();
        authorities= Arrays.stream(userInfo.getRoles().split(","))
            .map(SimpleGrantedAuthority::new)
            .collect(Collectors.toList());
    }
}
```

userinfo/UserInfoRepository.java

```
public interface UserInfoRepository extends JpaRepository<UserInfo, Integer> {
    Optional<UserInfo> findByEmail(String username);
}
```

UserInfoUserDetails.java

<https://gist.github.com/mysoyul/a3439aae415eb4d227a523c437e6d8a3>

스프링 Security 기본 설정

- UserInfoDetailsService 클래스와 UserInfoController 클래스 작성

userinfo/UserInfoDetailsService.java

```
@Service
public class UserInfoDetailsService implements UserDetailsService {
    @Autowired
    private UserInfoRepository repository;

    @Override
    public UserDetails loadUserByUsername(String username)
        throws UsernameNotFoundException {
        Optional<UserInfo> optionalUserInfo = repository.findByEmail(username);
        return optionalUserInfo.map(userInfo -> new UserInfoUserDetails(userInfo))
            //userInfo.map(UserInfoUserDetails::new)
            .orElseThrow(() -> new UsernameNotFoundException("user not found " + username));
    }
}
```

userinfo/UserInfoController.java

```
@RestController
@RequestMapping("/users")
public class UserInfoController {
    @Autowired
    private UserInfoRepository repository;
    @Autowired
    private PasswordEncoder passwordEncoder;

    @GetMapping("/welcome")
    public String welcome() {
        return "Welcome this endpoint is not secure";
    }

    @PostMapping("/new")
    public String addNewUser(@RequestBody UserInfo userInfo){
        userInfo.setPassword(
            passwordEncoder.encode(userInfo.getPassword()));
        UserInfo savedUserInfo = repository.save(userInfo);
        return savedUserInfo.getName() + " user added!!";
    }
}
```

스프링 Security 기본 설정

- SecurityConfig 클래스 수정

config/SecurityConfig.java

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    return http.csrf(csrf -> csrf.disable())
        .authorizeHttpRequests(auth -> {
            auth.requestMatchers("/users/welcome", "/users/new").permitAll()
                .requestMatchers("/api/lectures/**").authenticated();
        })
        .formLogin(withDefaults())
        .build();
}
```

UserInfo 등록
http://localhost:8080/users/new

```
{
    "name": "adminboot",
    "password": "pwd1",
    "email": "admin@aa.com",
    "roles": "ROLE_ADMIN,ROLE_USER"
}

{
    "name": "userboot",
    "password": "pwd2",
    "email": "user@aa.com",
    "roles": "ROLE_USER"
}
```

```
MariaDB [boot_db]> select * from user_info;
```

id	email	name	password	roles
1	admin@aa.com	adminboot	\$2a\$10\$WArmJxeoR6mKIt.yvbgMHujXND.w5V0/pVi5cGvuD02aUEWuK1Pu2	ROLE_ADMIN,ROLE_USER
2	user@aa.com	userboot	\$2a\$10\$l3Y1a3hedcG9nfjvg4HloOE0dwC3/eaDy59sSkkp0Fl5B.XJGV7xe	ROLE_USER

스프링 Security 기본 설정

- SecurityConfig 클래스 수정

config/SecurityConfig.java

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {
    @Bean
    public UserDetailsService userDetailsService() {
        return new UserInfosUserDetailsService();
    }

    @Bean
    public AuthenticationProvider authenticationProvider(){
        DaoAuthenticationProvider authenticationProvider=
            new DaoAuthenticationProvider();
        authenticationProvider.setUserDetailsService(userDetailsService());
        authenticationProvider.setPasswordEncoder(passwordEncoder());
        return authenticationProvider;
    }
}
```

Please sign in

No AuthenticationProvider found
for
org.springframework.security.authentication.UsernamePasswordAuthenticationToken

Username

Password

Sign in

스프링 Security 기본 설정 : Lecture와 UserInfo 연관관계

- UserInfo 와 Lecture 연관관계 매핑 : Lecture 클래스 수정
 - @ManyToOne 어노테이션은 이름 그대로 다대일(N : 1) 관계 매핑 정보입니다.
 - Lecture 입장에서 UserInfo와 다대일 관계이므로 @ManyToOne이 됩니다.
 - 단방향 관계를 맺었고 외래키가 생겼기 때문에, Lecture에서 UserInfo 정보들을 가져올 수 있습니다.

lectures/Lecture.java

```
@Entity
public class Lecture {
    ..
    @ManyToOne
    private UserInfo userInfo;
}
```

OAuth 2.0

1. OAuth란?

- OAuth는 인증을 위한 오픈 스탠더드 프로토콜로, 사용자가 Facebook이나 트위터 같은 인터넷 서비스의 기능을 다른 애플리케이션(데스크톱, 웹, 모바일 등)에서도 사용할 수 있게 한 것이다.
- 2010년 IETF (<https://www.ietf.org/> 국제인터넷표준화기구) OAuth 워킹 그룹에 의해 IETF 표준 프로토콜로 발표된다.
- OAuth 1.0의 특징
 - : API 인증 시, 써드파티 어플리케이션에게 사용자의 비번을 노출하지 않고 인증 할 수 있다.
 - : 인증(Authentication)과 API 권한(Authorization) 부여를 동시에 할 수 있다.
- 인증 토큰의 장점
 - ✓ 사용자의 아이디 / 패스워드를 몰라도 토큰을 통해 허가 받은 API에 접근 가능
 - ✓ 필요한 API에만 제한적으로 접근할 수 있도록 권한 제어 가능
 - ✓ 저장되어 있는 인증 토큰이 유출 되더라도 트위터의 관리자 화면에서 인증 토큰의 권한 취소 가능
 - ✓ 사용자가 트위터(Service Provider)의 패스워드를 변경해도 인증 토큰은 계속 유효함
- List of OAuth Provider https://en.wikipedia.org/wiki/List_of_OAuth_providers

OAuth 2.0

2. OAuth 2.0의 개선사항

- 일단 OAuth 2.0은 1.0과 호환되지 않으며 용어 부터 많은 것이 다르다. 모바일에서의 사용성 문제나 서명과 같은 개발이 복잡하고 기능과 규모의 확장성 등을 지원하기 위해 만들어진 표준이다. 표준이 매우 크고 복잡해서 이름도 "OAuth 인증 프레임워크(OAuth 2.0 Authorization Framework)" 이다. <http://tools.ietf.org/wg/oauth/>에서 확인 가능
- OAuth2.0 용어들
 - ✓ Resource Owner : 사용자(User)
 - ✓ Resource Server : 자원을 호스팅 하는 REST API 서버
 - ✓ Authorization Server : 인증서버 (API 서버와 같을 수도 있음), Access Token 발행
 - ✓ Client : 써드파티 어플리케이션 (서비스)
- 더 많은 인증 방법을 지원

OAuth 2.0은 여러 인증 방식을 통해 웹 / 모바일 등 다양한 시나리오에 대응 가능

Access Token의 Life-time을 지정하여 만료 시간 설정 가능

OAuth 2.0

2-1. OAuth 2.0의 다양한 인증 방식(Grant types)

- client는 기본적으로 Confidential Client 와 Public Client 로 나뉘어진다.
 - : Confidential 클라이언트는 웹 서버가 API를 호출하는 경우와 같이 client 증명서 (client_secret)를 안전하게 보관할 수 있는 client를 의미한다.
 - : Public Client는 브라우저기반 어플리케이션이나 모바일 어플리케이션 같이 client 증명서를 안전하게 보관할 수 없는 client를 의미하는데 이런 경우 redirect_uri 를 통해서 client를 인증한다.
- OAuth 4가지 인증 방식
 - Authorization Code Grant – 권한 부여 승인 코드 방식
 - Implicit Grant – 암묵적 승인 방식
 - Resource Owner Password Credentials Grant – 자원 소유자 자격증명 승인 방식
 - Client Credentials Grant – 클라이언트 자격증명 승인 방식

OAuth 2.0

2-1. OAuth 2.0의 다양한 인증 방식(Grant types)

1. Authorization Code Grant : used with server-side applications

: 웹 서버에서 API를 호출하는 등의 시나리오에서 Confidential Client가 사용하는 방식이다. 서버사이드 코드가 필요한 인증 방식이며 인증 과정에서 client_secret 이 필요하다.

2. Implicit Grant : used with Mobile App or Web app

: Token과 scope에 대한 스펙 등은 다르지만 OAuth 1.0a과 가장 비슷한 인증방식이다. Public Client인 브라우저 기반의 어플리케이션(Javascript application)이나 모바일 어플리케이션에서 이 방식을 사용하면 된다. Client 증명서를 사용할 필요가 없으며 실제로 OAuth 2.0에서 많이 사용되는 방식이다.

3. Password Credentials Grant : used with trusted application

: Client에 아이디/패스워드를 저장해 놓고 아이디/패스워드로 직접 access token을 받아오는 방식이다. Client를 믿을 수 없을 때 사용하기에는 위험하므로 API 서비스의 공식 어플리케이션이나 믿을 수 있는 Client에 한해서만 사용하는 것을 추천한다.

로그인 시에 API에 POST로 grant_type=password 라고 넘긴다.

4. Client Credentials Grant : used with application api access

: 어플리케이션이 Confidential Client일 때 id와 secret을 가지고 인증하는 방식이다.

로그인 시에 API에 POST로 grant_type=client_credentials 라고 넘긴다.

OAuth 2.0

3. 다양한 Token 지원

1. Access Token

: OAuth 2.0은 기본적으로 Bearer 토큰, 즉 암호화 하지 않은 그냥 토큰을 주고 받는 것으로 인증을 한다.

기본적으로 HTTPS 를 사용하기 때문에 토큰을 안전하게 주고 받는 것은 HTTPS의 암호화에 의존한다. 또한 복잡한 signature 등을 생성할 필요가 없기 때문에 curl이 API를 호출 할 때 간단하게 Header 에 아래와 같이 한 줄을 같이 보내어 API를 테스트해 볼 수 있다.

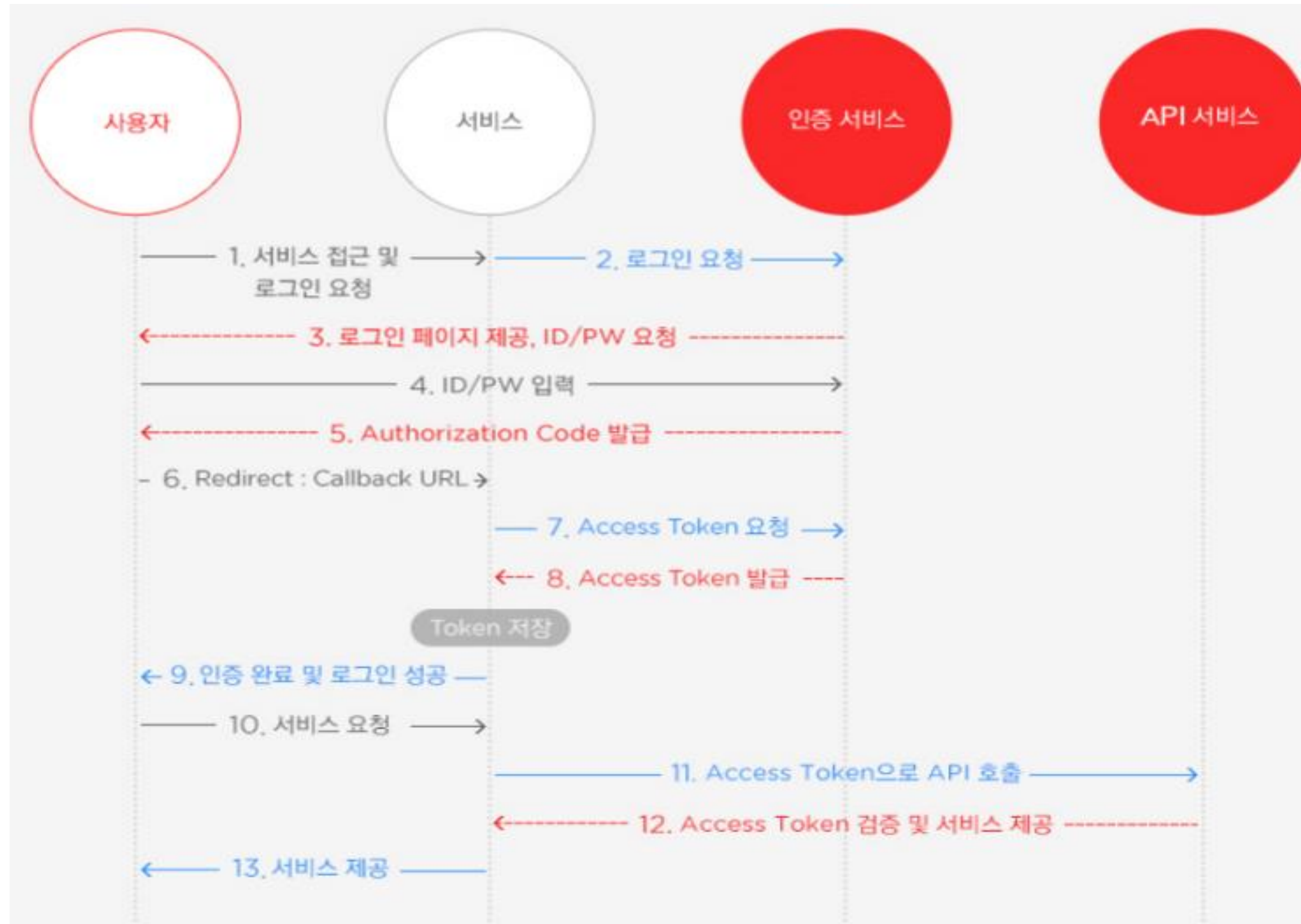
Authorization: Bearer

2. Refresh Token

: 클라이언트가 같은 access token을 오래 사용하면 해킹에 노출될 위험이 높아지므로 OAuth 2.0에서는 refresh token 이라는 개념을 도입했다. 즉, 인증 토큰(access token)의 만료 기간을 가능한 짧게 하고 만료가 되면 refresh token으로 access token을 새로 갱신하는 방법이다.

OAuth 2.0

4. OAuth2.0 프로세스



OAuth vs JWT

5.OAuth와 JWT의 관계

OAuth는 토큰을 발급하고 인증하는 오픈 스탠다드 프로토콜이며 프레임워크이다.

JWT는 이러한 프레임워크에서 발생하는 산출물로 볼 수 있다.

OAuth2.0는 하나의 플랫폼의 권한(아무 의미 없는 무작위 문자열 토큰)으로 다양한 플랫폼에서 권한을 행사할 수 있게 해줌으로써 리소스 접근이 가능하게 하는데 목적을 두고 있다.

JWT는 Cookie, Session을 대신하여 의미 있는 문자열 토큰으로써 권한을 행사할 수 있는 토큰의 한 형식이다. (로그인 세션이나 주고받는 값이 유효한지 검증할 때 주로 쓰인다.)

JWT

JWT(Json Web Token) 구조

- : Json 포맷을 이용하여 모바일이나 웹의 사용자 인증을 위해 사용하는 암호화된 토큰을 의미합니다.
- : JWT는 Header, Payload, Signature 3 부분으로 이루어진다.
- : 각각의 부분은 Base64로 인코딩 되어 표현되고 각각의 부분을 이어주기 위해 구분자를 사용하여 구분한다.

HEADER: ALGORITHM & TOKEN TYPE
<pre>{ "alg": "HS256", "typ": "JWT" }</pre>
PAYLOAD: DATA
<pre>{ "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }</pre>
VERIFY SIGNATURE
<pre>HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), your-256-bit-secret) <input type="checkbox"/> secret base64 encoded</pre>

Header : 토큰의 타입과 암호화 알고리즘으로 구성되어 있다.

Payload : 토큰에 담은 정보가 들어 있으며 Payload에 담은 정보의 한 '조각'을 클레임(claim)이라고 부르고, 이는 name/value 의 한쌍으로 이루어져 있습니다. 클레임의 종류는 등록된(registered)클레임, 공개(public)클레임, 비공개(private)클레임이 존재합니다.


Signature: 서명은 [헤더 base64 + 페이로드 base64 + SECRET_KEY] 를 사용하여 JWT 백엔드에서 발행됩니다

요청이 올 때마다 서명이 체크 됩니다. header 또는 payload의 정보가 클라이언트에 의해 변경된 경우 서명이 무효화 됩니다.


JWT

JWT(Json Web Token) 구조

<https://jwt.io/>

JWT

[Debugger](#) [Libraries](#) [Introduction](#) [Ask](#) [Get a T-shirt!](#)

Crafted by Auth0

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJiYTZmMDY4YS0wMTRiLTJiYzUyYWE4Ny01ODEyMDk2MzcwNGYiLCJleHAiOiJlMTI2NjE1NzV9.3NBfgChDav_apxl1EsaMPyrTjoHxFD3GzTxWsgvJeF7Gb8qF01IYJ14MotH8EdZjKp5LjTsFehPn7-BcH1PL5A|
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  "alg": "HS512"}
```

PAYLOAD: DATA

```
{  "sub": "ba6f068a-014b-4bc5-aa87-58120963704f",  "exp": 1612661575}
```

VERIFY SIGNATURE

```
HMACSHA512(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
) ☐ secret base64 encoded
```

Language	Library URL
NodeJS	https://github.com/auth0/node-jsonwebtoken
PHP	https://github.com/firebase/php-jwt
Java	https://github.com/jwtkt/jjwt
Ruby	https://github.com/jwt/ruby-jwt
.NET	https://github.com/AzureAD/azure-activedirectory-identitymodel-extensions-for-dotnet
Python	https://github.com/jpadilla/pyjwt

스프링 **jjwt** 기본 설정 : 의존성 추가

- jjwt 의존성 추가

pom.xml

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.12.3</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.12.3</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.12.3</version>
</dependency>
```

JWT(Json Web Token) 적용 시 고려사항

- Self-contained

토큰 자체에 정보를 담고 있으므로 토큰의 길이가 길어 질 수 있다.

토큰 길이: 토큰의 페이로드(Payload)에 3종류의 클레임을 저장하기 때문에, 정보가 많아질수록 토큰의 길이가 늘어나 네트워크에 부하를 줄 수 있다.

- Payload 인코딩

페이로드(Payload) 자체는 암호화된 것이 아니라, BASE64로 인코딩 된 것이다. 중간에 Payload를 탈취하여 디코딩 하면 데이터를 볼 수 있으므로, JWE로 암호화 하거나 Payload에 중요 데이터를 넣지 않아야 한다.

- Stateless

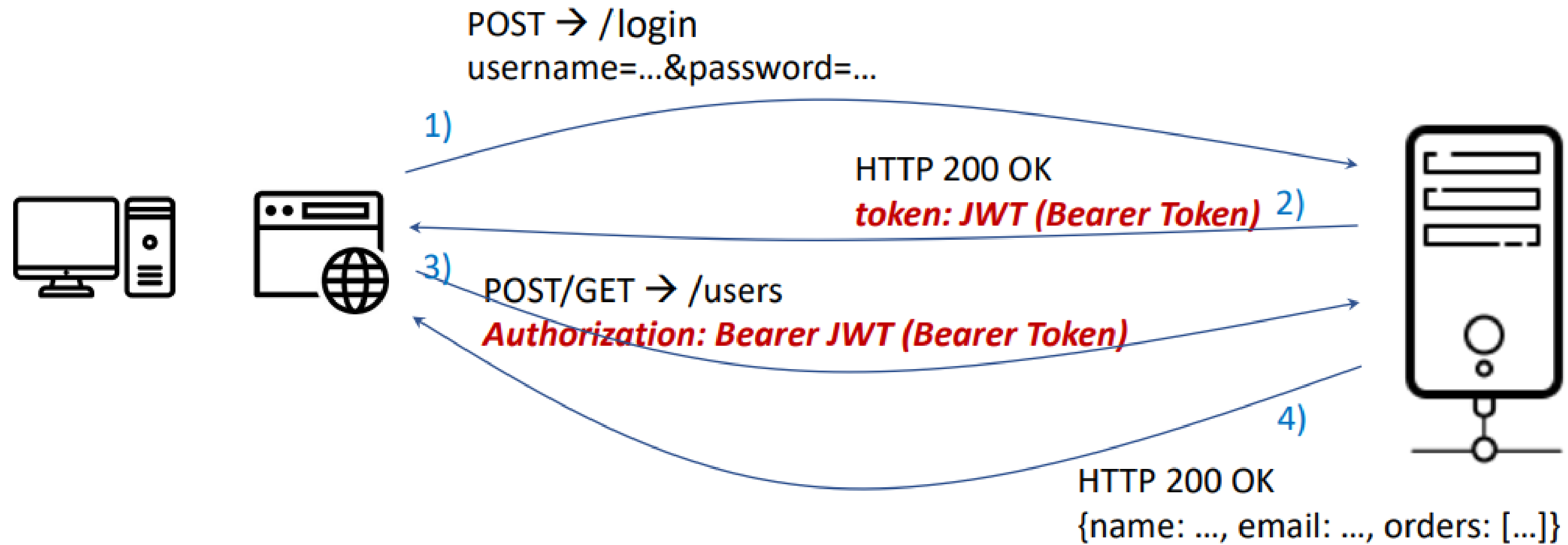
JWT는 상태를 저장하지 않기 때문에 한 번 만들어지면 제어가 불가능하다. 즉, 토큰을 임의로 삭제하는 것이 불가능하므로 토큰 만료 시간을 꼭 넣어주어야 한다.

- Token

토큰은 클라이언트 측에서 관리해야 하기 때문에, 토큰을 저장해야 한다.

Security JWT 인증 처리 과정

로그인 인증 처리과정



Security JWT 인증

인증 정보저장 DTO 클래스

- AuthRequest 클래스

security/dto/AuthRequest.java

```
@Data
public class AuthRequest {
    @NotNull(message = "Email cannot be null")
    @Size(min = 2, message = "Email not be less than two characters")
    @Email
    private String email;

    @NotNull(message = "Password cannot be null")
    @Size(min = 4, message = "Password must be equals or grater than 4 characters")
    private String password;
}
```

```
{
  .... "email": "user@email.com",
  .... "password": "user"
}
```

Security JWT 인증

- JwtService 클래스 작성

security/jwt/JwtService.java

```
@Component
public class JwtService {
    public static final String SECRET =
"5367566B59703373367639792F423F4528482B4D6251655468576D5A71347437";

    private Key getSignKey() {
        byte[] keyBytes = Decoders.BASE64.decode(SECRET);
        return Keys.hmacShaKeyFor(keyBytes);
    }

    private Claims extractAllClaims(String token) {
        return Jwts
            .parser()
            .verifyWith(KEY)
            .build()
            .parseSignedClaims(token)
            .getPayload();
    }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }
}
```

JwtService 클래스

<https://gist.github.com/mysoyul/7b6f31927640eb5ffa33376af44018c7>

Encryption Key Generator

<https://generate-random.org/encryption-key-generator>

JJWT API doc

<https://javadoc.io/doc/io.jsonwebtoken/jjwt-api/latest/index.html>

Security JWT 인증

- SecurityConfig 클래스 수정

config/SecurityConfig.java

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    return http.csrf(csrf -> csrf.disable())
        .authorizeHttpRequests(auth -> {
            auth.requestMatchers("/users/welcome", "/users/new", "/users/login").permitAll()
                .requestMatchers("/api/lectures/**").authenticated();
        })
        .formLogin(withDefaults())
        .build();
}

@Bean
public AuthenticationManager authenticationManager(AuthenticationConfiguration config)
    throws Exception {
    return config.getAuthenticationManager();
}
```

Security JWT 인증

- UserInfoController 클래스 수정

userinfo/UserInfoController.java

```
@RestController
@RequestMapping("/users")
public class UserInfoController {
    @Autowired
    private UserInfoDetailsService service;
    @Autowired
    private AuthenticationManager authenticationManager;
    @Autowired
    private JwtService jwtService;

    @PostMapping("/login")
    public String authenticateAndGetToken(@RequestBody AuthRequest authRequest) {
        Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                authRequest.getEmail(),
                authRequest.getPassword()
            ));
        if (authentication.isAuthenticated()) {
            return jwtService.generateToken(authRequest.getEmail());
        } else {
            throw new UsernameNotFoundException("invalid user request !");
        }
    }
}
```

UserInfoController 클래스

Login 인증 토큰 생성

http://localhost:8080/users/login

```
{
  "email": "admin@aa.com",
  "password": "pwd1"
}
```

[illegible]

JWT 인증 토큰 생성 요청

Token 생성 요청

<http://localhost:8080/users/login>

Request Header

Content-type: application/json

Body - row 옵션 선택

ROLE_ADMIN

```
{  
  "email": "admin@aa.com",  
  "password": "pwd1"  
}
```

ROLE_USER

```
{  
  "email": "user@aa.com",  
  "password": "pwd2"  
}
```

Security JWT 인증

JwtAuthenticationFilter 구현

- 로그인 요청 발생 시 인증 처리 작업을 해 주는 Custom Filter 클래스
- OncePerRequestFilter 인터페이스 상속

<https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/filter/OncePerRequestFilter.html>

```
@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    @Autowired
    private JwtService jwtService;

    @Autowired
    private UserInfoDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {

    }

}
```

Security JWT 인증

JwtAuthenticationFilter 구현 #1

filter/JwtAuthenticationFilter.java

```
@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {
    @Autowired
    private JwtService jwtService;
    @Autowired
    private UserInfoUserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
        FilterChain filterChain)
        throws ServletException, IOException {
        String authHeader = request.getHeader("Authorization");
        String token = null;
        String username = null;

        if (authHeader != null && authHeader.startsWith("Bearer ")) {
            token = authHeader.substring(7);
            username = jwtService.extractUsername(token);
        }
    }
}
```

JwtAuthenticationFilter 클래스

<https://gist.github.com/mysoyul/816722af8506033c857e959df1e52a05>

Security JWT 인증

JwtAuthenticationFilter 구현 #2

filter/JwtAuthenticationFilter.java

```
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
    FilterChain filterChain)
    throws ServletException, IOException {
    String authHeader = request.getHeader("Authorization");
    String token = null;
    String username = null;

    if (authHeader != null && authHeader.startsWith("Bearer ")) {
        token = authHeader.substring(7);
        username = jwtService.extractUsername(token);
    }

    if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
        UserDetails userDetails = userService.loadUserByUsername(username);
        if (jwtService.validateToken(token, userDetails)) {
            UsernamePasswordAuthenticationToken authToken =
                new UsernamePasswordAuthenticationToken(userDetails,
                    null, userDetails.getAuthorities());
            authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
            SecurityContextHolder.getContext().setAuthentication(authToken);
        }
    }
    filterChain.doFilter(request, response);
}
```


Security JWT 인증

스프링 시큐리티 Session 정책

http.sessionManagement()

- SessionCreationPolicy.ALWAYS - 스프링 시큐리티가 항상 세션을 생성
- SessionCreationPolicy.IF_REQUIRED - 스프링 시큐리티가 필요시 생성(기본)
- SessionCreationPolicy.NEVER - 스프링 시큐리티가 생성하지는 않지만, 기존 세션이 존재하면 사용
- SessionCreationPolicy.STATELESS - 스프링 시큐리티가 생성하지 않고 기존의 세션도 사용하지 않음 (JWT와 같은 토큰 방식을 쓸 때 사용한다.)

config/SecurityConfig.java

```
public class SecurityConfig {
    @Autowired
    private JwtAuthenticationFilter authenticationFilter;
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        return http.csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> {
                auth.requestMatchers("/users/**").permitAll()
                auth.requestMatchers("/api/lectures/**").authenticated();
            })
            .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authenticationProvider(authenticationProvider())
            .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class)
            .build()
    }
}
```

SecurityConfig 클래스

<https://gist.github.com/mysoyul/afe786c0475e4690dcdffc23273e13e2>

Security JWT 인증

전역 오류 처리

BusinessException 클래스 (**exception** 패키지)

<https://gist.github.com/mysoyul/c13d51d2138d8697980a3eee20755fa7>

SystemException 클래스 (**exception** 패키지)

<https://gist.github.com/mysoyul/608fc1fa017143fcea9e1e848615e0b4>

CustomAccessDeniedHandler 클래스 (**exception.security** 패키지)

<https://gist.github.com/mysoyul/121aed49a8119194ebd7acc5a05c3630>

CustomAuthenticationEntryPoint 클래스 (**exception.security** 패키지)

<https://gist.github.com/mysoyul/ea2d5614c46939dc01f6219783759463>

DefaultExceptionAdvice 클래스 (**exception.advice** 패키지)

<https://gist.github.com/mysoyul/12468b7e8af4d6fb47dfd6107251bc7d>

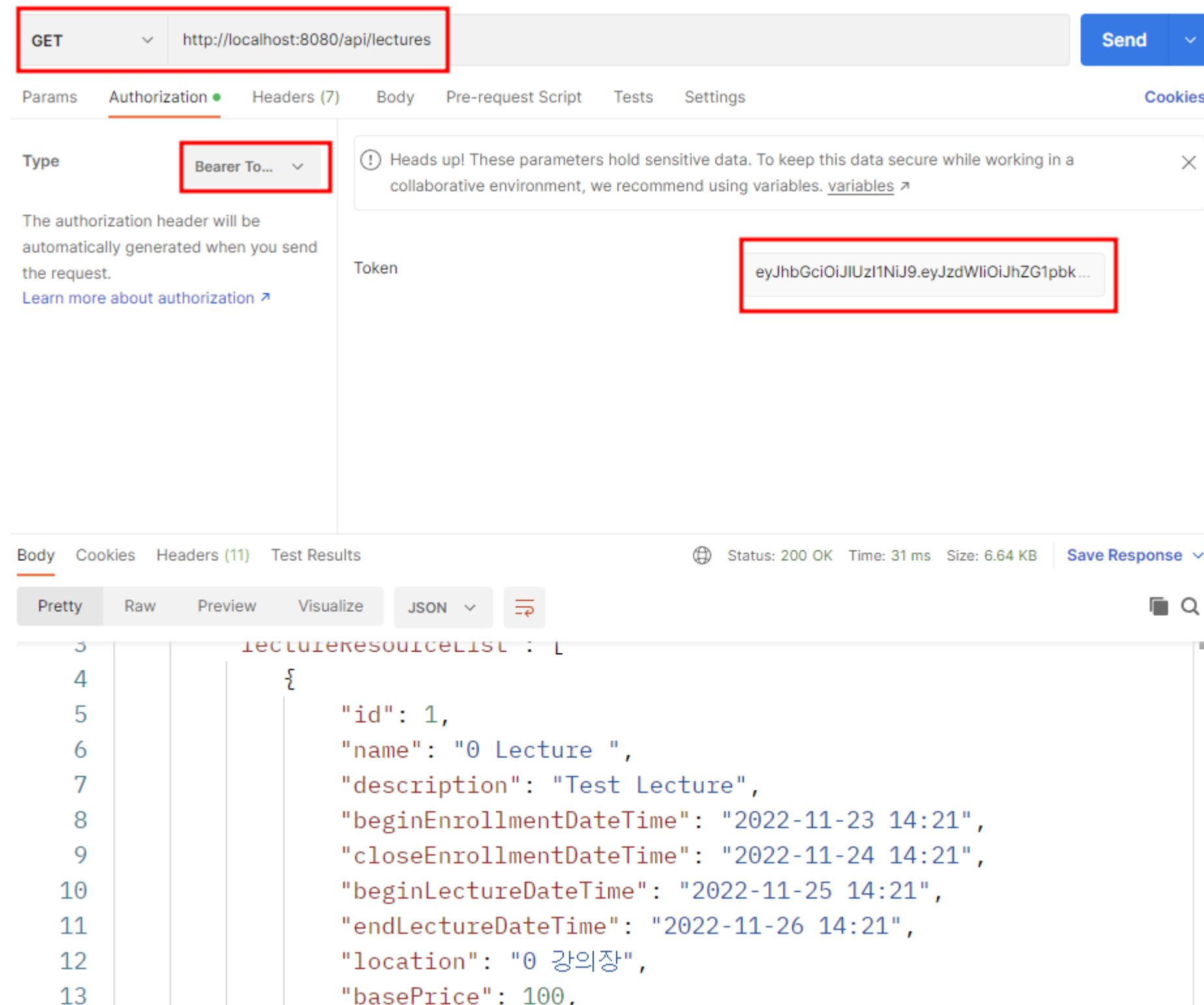
SecurityConfig 클래스 (**security** 패키지)

<https://gist.github.com/mysoyul/afe786c0475e4690dcdffc23273e13e2>

JWT 인증 토큰 사용

Admin권한 token을 사용한 Lecture 전체 목록 조회

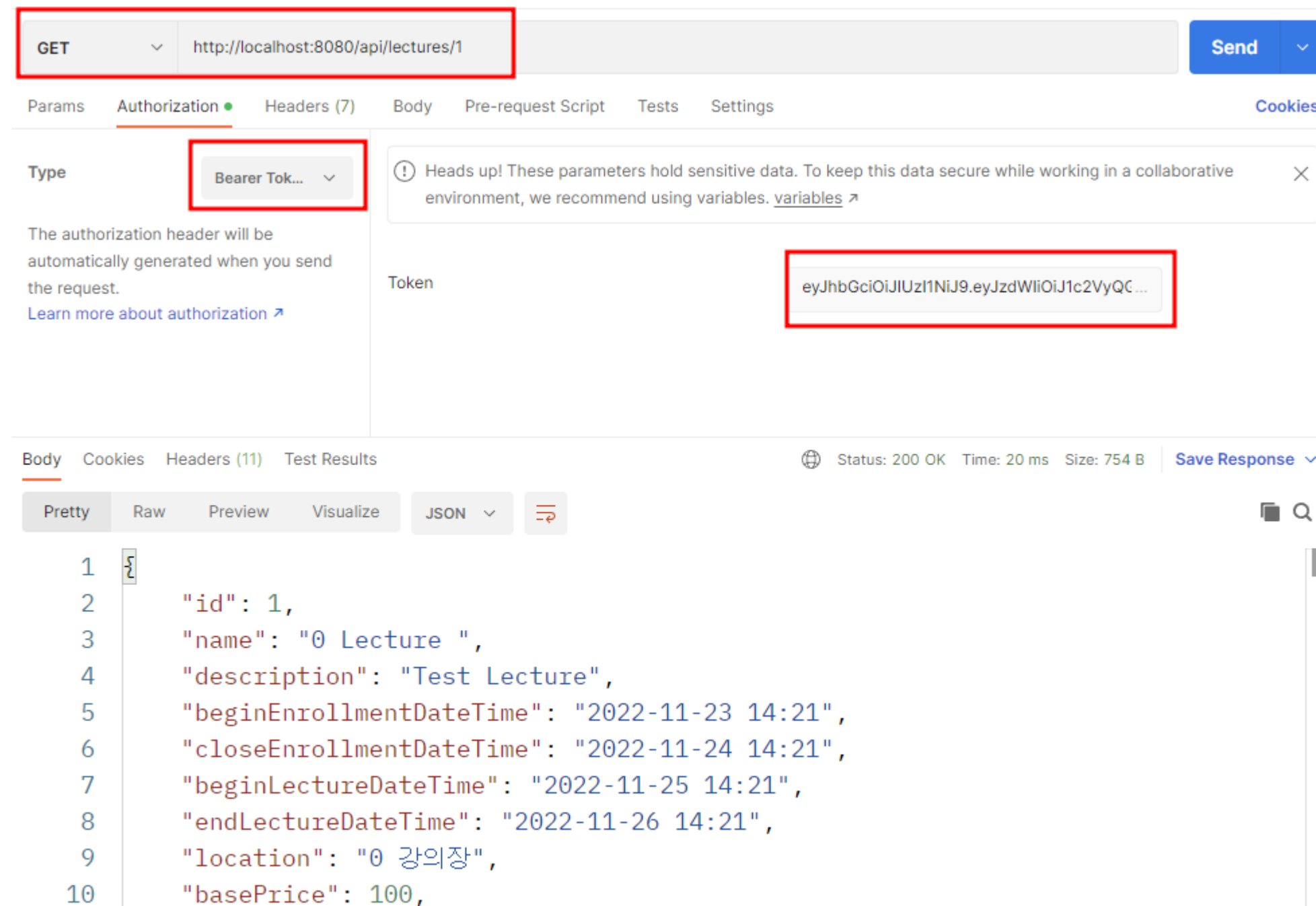
Authorization : Bearer에 Token 번호 설정



JWT 인증 토큰 사용

User권한 token을 사용한 Lecture 개별 조회

Authorization : Bearer에 Token 번호 설정



Security JWT 현재 사용자 조회 #1

1) UserInfoUserDetails를 주입 받는다 : LectureController 클래스 수정

- `@AuthenticationPrincipal` : 로그인 한 사용자의 정보를 아규먼트로 받을 수 있다.
- `@AuthenticationPrincipal` 어노테이션을 사용하면 `UserDetailsService`에서 Return 한 객체를 아규먼트로 직접 받아 사용할 수 있다.
- 로그인 한 상태이면 Lecture를 `create(생성)`하는 Link를 추가해 준다.

lectures/LectureController.java

```
import org.springframework.security.core.annotation.AuthenticationPrincipal;

public class LectureController {
    public ResponseEntity queryLectures(Pageable pageable,
                                       PagedResourcesAssembler<Lecture> assembler,
                                       @AuthenticationPrincipal UserInfoUserDetails currentUser) {
        Page<Lecture> page = this.lectureRepository.findAll(pageable);
        PagedModel<RepresentationModel<LectureResource>> pagedResources =
            assembler.toModel(page, lecture -> new LectureResource(lecture));

        if(currentUser != null) {
            pagedResources.add(LinkTo(LectureController.class).withRel("create-Lecture"));
        }
        return ResponseEntity.ok(pagedResources);
    }
}
```

Security JWT 현재 사용자 조회 #1

2) UserInfoUserDetails의 UserInfo 가져오기 : LectureController 클래스 수정

- UserInfoUserDetails에 저장된 UserInfo를 가져오기 위해서 userDetails.getUserInfo()를 사용해도 되지만 SPEL(Spring Expression Language)을 사용하여
- `@AuthenticationPrincipal(expression = "userInfo")`

lectures/LectureController.java

```
public class LectureController {
    public ResponseEntity queryLectures(Pageable pageable,
        PagedResourcesAssembler<Lecture> assembler,
        @AuthenticationPrincipal(expression = "userInfo") UserInfo currentUser) {
        Page<Lecture> page = this.lectureRepository.findAll(pageable);
        PagedResources<Resource<Lecture>> pagedResources =
            assembler.toResource(page, lecture -> new LectureResource(lecture));
        if(currentUser != null) {
            pagedResources.add(linkTo(LectureController.class).withRel("create-Lecture"));
        }
        return ResponseEntity.ok(pagedResources);
    }
}
```

Security JWT 현재 사용자 조회 #3

3) UserInfoUserDetails의 UserInfo 가져오기 : CurrentUser 어노테이션 추가

- 새로운 @CurrentUser 어노테이션 정의하기
- @Target - 어노테이션이 적용할 위치를 결정합니다. (java.lang.annotation.Target)
- @Retention - 어떤 시점에 어노테이션이 영향을 주는지를 결정합니다.

(Whether in code only, compiled into the class, or available at runtime through reflection.)

userinfo/CurrentUser.java

```
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
@AuthenticationPrincipal(expression = "#this == 'anonymousUser' ? null : userInfo")
public @interface CurrentUser {
}
```

<https://gist.github.com/mysoyul/1532ed451f45ede769b2cec3afd81e22>

lectures/LectureController.java

```
public class LectureController {
    public ResponseEntity queryLectures(
        Pageable pageable,
        PagedResourcesAssembler<Lecture> assembler,
        @CurrentUser UserInfo currentUser) {
```

queryLectures() 메서드

<https://gist.github.com/mysoyul/01a3366f59291bc7b068359d38a3681e>

Security JWT 현재 사용자 조회 #4

- 4) Lecture 등록 시 현재 사용자 정보 저장 : LectureController createLecture 메서드 수정
- Lecture 등록 할 때 UserInfo userInfo 변수에 @CurrentUser 어노테이션을 사용하여 주입 받은 currentUser를 저장 한다.

lectures/LectureController.java

```
@PostMapping
public ResponseEntity createLecture(@RequestBody @Valid LectureDto lectureDto,
                                     Errors errors, @CurrentUser UserInfo currentUser) {
    ..
    Lecture lecture = modelMapper.map(lectureDto, Lecture.class);
    lecture.update();
    lecture.setUserInfo(currentUser);
    ..
}
```

createLecture() 메서드

<https://gist.github.com/mysoyul/a43df8cc3c6c4e0c7996607e2bc0325e>

Security JWT 현재 사용자 조회 #5

5) Lecture 조회 시 로그인 상태면 update 링크 제공 : LectureController getLecture 메서드 수정

: LectureController getLecture 메서드 수정

- Lecture 한 건 조회할 때 로그인 한 상태이면 Lecture를 update 하는 Link 제공합니다.

lectures/LectureController.java

```
public ResponseEntity getLecture(@PathVariable Integer id,
    @CurrentUser UserInfo currentUser) {

    if(currentUser != null)
        lectureResDto.setEmail(lecture.getUserInfo().getEmail());

    LectureResource lectureResource = new LectureResource(lectureResDto);

    if((lecture.getUserInfo() != null) && (lecture.getUserInfo().equals(currentUser))) {
        lectureResource.add(linkTo(LectureController.class)
            .slash(lecture.getId()).withRel("update-lecture"));
    }

    ..
}
```

getLecture() 메서드

<https://gist.github.com/mysoyul/6b3408b43b4009f13d87e62cf06fd12a>

Security JWT 현재 사용자 조회 #6

- 6) Lecture 수정 시 로그인 사용자가 일치하지 않으면 UNAUTHORIZED 에러 발생 : updateLecture 메서드 수정
- 수정할 때 로그인 한 사용자가 Lecture를 등록한 사용자와 일치하지 않으면 Unauthorized 에러를 발생시킨다.

lectures/LectureController.java

```
public ResponseEntity updateLecture(@PathVariable Integer id,
                                    @RequestBody @Valid LectureDto lectureDto,
                                    Errors errors,
                                    @CurrentUser UserInfo currentUser) {

    Lecture existingLecture = optionalLecture.get();
    if((existingLecture.getUserInfo() != null) &&
        (!existingLecture.getUserInfo().equals(currentUser))) {
        return new ResponseEntity(HttpStatus.UNAUTHORIZED);
    }
    this.modelMapper.map(lectureReqDto, existingLecture);
    Lecture savedLecture = this.lectureRepository.save(existingLecture);
    LectureResDto lectureResDto = modelMapper.map(savedLecture, LectureResDto.class);
    if(currentUser != null)
        lectureResDto.setEmail(savedLecture.getUserInfo().getEmail());
}
```

updateLecture() 메서드

<https://gist.github.com/mysoyul/04fc690f88844e825cb1361201cb2200>

Security JWT 출력값 제한하기 #1

1) Lecture가 참조하는 UserInfo 정보 출력 제한하기 : UserInfoSerializer 클래스 작성
: JsonSerializer<UserInfo>를 상속받고 serialize() 메서드를 오버라이딩 합니다.
: UserInfo id와 email 만 보여지도록 합니다.

userinfo/UserInfoSerializer.java

```
import com.fasterxml.jackson.core.JsonGenerator;
import com.fasterxml.jackson.databind.JsonSerializer;
import com.fasterxml.jackson.databind.SerializerProvider;

public class UserInfoSerializer extends JsonSerializer<UserInfo>{
    @Override
    public void serialize(UserInfo userInfo, JsonGenerator gen,
        SerializerProvider serializers) throws IOException {

        gen.writeStartObject();
        gen.writeNumberField("id", userInfo.getId());
        gen.writeStringField("email", userInfo.getEmail());
        gen.writeEndObject();
    }
}
```

Security JWT 출력값 제한하기 #2

2) Lecture가 참조하는 UserInfo 정보 출력 제한하기 : Lecture 클래스 수정

: @JsonSerialize 어노테이션은 커스텀 serialize를 지정 할 때 사용한다.

: @JsonSerialize indicates a custom serializer to use when marshalling the entity.

lectures/Lecture.java

```
import com.fasterxml.jackson.databind.annotation.JsonSerialize;

public class Lecture {

    @ManyToOne
    @JsonSerialize(using = UserInfoSerializer.class)
    private UserInfo userInfo;
    ..
}
```

Swagger로 API 문서 자동화하기

Swagger

1. springdoc-openapi란? (<https://springdoc.org/>)

: Springdoc은 OpenAPI 3.0 spec을 이용하여 구현 하였다.

REST API Docs를 생성해주는 Open Source는 Springfox Swagger와 SpringDoc 이 있다.

pom.xml

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>2.2.0</version>
</dependency>
```

application.properties

```
springdoc.swagger-ui.path=/swagger-ui.html
springdoc.swagger-ui.display-request-duration=true
springdoc.cache.disabled=true
```

Swagger

2. Springdoc 관련 url 은 인증하지 않도록 설정

<http://localhost:8080/swagger-ui.html>

config/SecurityConfig.java

```
@Configuration
public class SecurityConfig {

    @Bean
    public webSecurityCustomizer webSecurityCustomizer() {
        return (web) -> web.ignoring()
            .requestMatchers("/v3/api-docs/**", "/swagger-ui.html", "/swagger-ui/**");
    }
}
```



판교 | 경기도 성남시 분당구 삼평동 대왕판교로 670길 유스페이스2 B동 8층 T. 070-5039-5805
가산 | 서울시 금천구 가산동 371-47 이노플렉스 1차 2층 T. 070-5039-5815
웹사이트 | <http://edu.kosta.or.kr> 팩스 | 070-7614-3450