SpringBoot 시작하기

환경 세팅

JDK

Windows (OpenJDK)

https://jdk.java.net/archive/

MacOS (Oracle JDK)

https://www.oracle.com/java/technologies/downloads/#java11-mac

IntelliJ

https://www.jetbrains.com/ko-kr/idea/download/#section=windows

Maven과 Gradle의 차이

참고 : https://www.geeksforgeeks.org/difference-between-gradle-and-maven/

(정확한 해석이 아님…)

Gradle의 장점

1. 커스터마이징에 더 용이하다.
2. 성능이 훨신 빠르고 효율적이다. Maven보다 대략 2배 빠른 속도를 가진다.
3. 플러그인을 만드는 툴이며 유연하게 동작한다.
4. UX를 더 좋게 하기 위해 다양한 IDE를 제공한다.

Gradle의 단점

1. 빌드하는 데에 전문적인 지식이 필요하다.
2. Ant 프로젝트 구조와 사용할 수 없다
3. Doc 문서가 광범위하다.

4. ??

Maven의 장점

1. 빌드 과정이 단순하고 잘 정돈되어있다.
2. Jar 파일과 다른 종속성에 대해 자동으로 다운로드를 받는다.
3. POM 파일에 새로운 종속성을 추가하는 것이 쉽다.
4.

Maven의 단점

1. working system을 설치해야한다.
2. 종속성에 대한 Maven 코드가 없다면 종속성을 추가할 수 없다.
3. 느린 편이다.

⇒ 아무튼 **Gradle**을 써보기로 했다. (Maven에서 Gradle로 넘어가는 추세로 보임)

읽어보기 : https://hyojun123.github.io/2019/04/18/gradleAndMaven/

▼ `build.gradle` 파일

```
plugins {
    id 'org.springframework.boot' version '2.7.1'
    id 'io.spring.dependency-management' version '1.0.11.RELEASE'
    id 'java'
}

group = 'com.example'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'

configurations {
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

dependencies {
```

```
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-oauth2-client'
    implementation 'org.springframework.boot:spring-boot-starter-oauth2-resource-serve
r'
    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-web-services'
    implementation 'org.mybatis.spring.boot:mybatis-spring-boot-starter:2.2.2'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'org.mariadb.jdbc:mariadb-java-client'
    annotationProcessor 'org.springframework.boot:spring-boot-configuration-processor'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.springframework.security:spring-security-test'
}

tasks.named('test') {
    useJUnitPlatform()
}
```

application.properties와 application.yml의 차이

참고 : https://www.baeldung.com/spring-boot-yaml-vs-properties

yml 파일은 가독성이 좋고 계층 구조가 명확히 드러난다.

아직 사용해보진 않았지만 고려해야될 점이 좀 있어보여서 우선은 기본적으로 생성된 properties 파일로 해볼 것이다.

테스트 프로젝트 생성

IntelliJ에 SpringBoot 프로젝트 생성

(참고 : https://ifuwanna.tistory.com/294)

Maria DB에 간단한 스키마, 테이블 생성

기본적으로 MySQL을 기반으로 해서 쿼리가 다르지 않은 듯하다.

```
create database test;use test;create table user (        idx int not null AUTO_INCREMENT pr
imary key,    email varchar(50) not null,    name varchar(50) not null);insert into user (e
mail, name) values ('a@naver.com', 'a');insert into user (email, name) values ('b@naver.co
m', 'b');insert into user (email, name) values ('c@naver.com', 'c');
```

Dependencies를 빌드 명세에 추가하기

`build.gradle`

```
dependencies {
     implementation 'org.springframework.boot:spring-boot-starter-web'
     implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
     runtimeOnly 'org.mariadb.jdbc:mariadb-java-client:2.7.4'
     compileOnly 'org.projectlombok:lombok'
     annotationProcessor 'org.projectlombok:lombok'
     testImplementation('org.springframework.boot:spring-boot-starter-test') {
         exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'
     }
}
```

DB 및 logging 설정 추가

`application.properties`

```
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.datasource.url=jdbc:mariadb://localhost:3306/DB명
spring.datasource.username=userName
spring.datasource.password=password

#update the schema with the given values.
spring.jpa.hibernate.ddl-auto=update
#To beautify or pretty print the SQL
spring.jpa.properties.hibernate.format_sql=true
#show sql
spring.jpa.properties.hibernate.show-sql=true
#show parameter binding
logging.level.org.hibernate.type.descriptor.sql=DEBUG

logging.level.org.hibernate.SQL=DEBUG
```

가장 간단하게 JPA 맛보기

참고 : https://kim-oriental.tistory.com/20

▼ `User` Entity 파일

```
package com.example.helloworld.entity;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
@Getter @Setter
public class User {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long idx;

    private String name;

    private String email;

}
```

▼ `UserRepository` Repository 파일

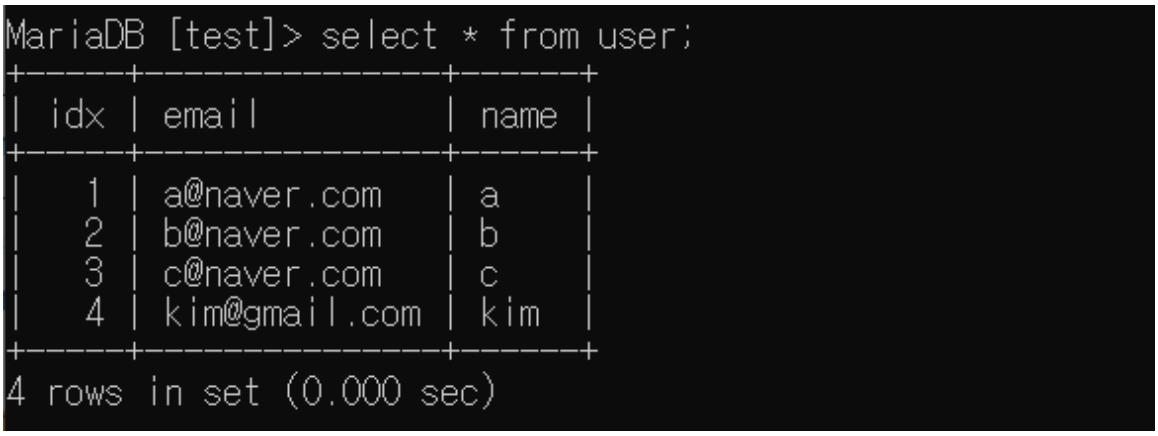
```
package com.example.helloworld.repository;

import com.example.helloworld.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
}
```

▼ `HelloWorldApplicationTests` Test 파일

```
package com.example.helloworld;

import com.example.helloworld.entity.User;
import com.example.helloworld.repository.UserRepository;
import org.junit.jupiter.api.MethodOrderer;
import org.junit.jupiter.api.Order;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestMethodOrder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.List;
```

```
@SpringBootTest
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
class HelloWorldApplicationTests {

    @Autowired
    UserRepository userRepository;

    @Test
    @Order(1)
    void save(){
        User user = new User();
        user.setName("kim");
        user.setEmail("kim@gmail.com");
        userRepository.save(user);
    }

    @Test
    @Order(2)
    void select(){
        List<User> userList = userRepository.findAll();
        for(User u : userList){
            System.out.println("[FindAll] : " + u.getIdx() + " " + u.getName());
        }
    }
}
```

결과



조금 더 디벨롭...

참고 : https://goddaehee.tistory.com/209

▼ `UserService` Service 파일

```
package com.example.helloworld.service;

import com.example.helloworld.entity.User;
import com.example.helloworld.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;

    public List<User> findAll(){
        List<User> users = new ArrayList<>();
        userRepository.findAll().forEach(e -> users.add(e));
        return users;
    }

    public Optional<User> findByIdx(Long idx){
        Optional<User> user = userRepository.findByIdx(idx);
        return user;
    }

    public void deleteByIdx(Long idx){
        userRepository.deleteById(idx);
    }

    public User save(User user){
        userRepository.save(user);
        return user;
    }

    public void updateById(Long idx, User user){
        Optional<User> e = userRepository.findByIdx(idx);

        if(e.isPresent()){
            e.get().setName(user.getName());
            e.get().setEmail(user.getEmail());
            userRepository.save(user);
        }
    }
```

```
}
```

▼ `UserController` Controller 파일

```
package com.example.helloworld.controller;

import com.example.helloworld.entity.User;
import com.example.helloworld.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.servlet.http.HttpServletRequest;
import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping(path = "/user")
public class UserController {

    @Autowired
    UserService userService;

    @GetMapping(produces = {MediaType.APPLICATION_JSON_VALUE})
    public ResponseEntity<List<User>> getAllUsers(){
        System.out.println("GET ALL USERS");
        List<User> user = userService.findAll();
        return new ResponseEntity<List<User>>(user, HttpStatus.OK);
    }

    @GetMapping(value = "/{idx}", produces = {MediaType.APPLICATION_JSON_VALUE})
    public ResponseEntity<User> getUser(@PathVariable("idx") Long idx){
        Optional<User> user = userService.findByIdx(idx);
        return new ResponseEntity<User>(user.get(), HttpStatus.OK);
    }

    @DeleteMapping(value = "/{idx}", produces = {MediaType.APPLICATION_JSON_VALUE})
    public ResponseEntity<Void> deleteUser(@PathVariable("idx") Long idx){
        userService.deleteByIdx(idx);
        return new ResponseEntity<Void>(HttpStatus.NO_CONTENT);
    }

    @PutMapping(value = "/{idx}", produces = {MediaType.APPLICATION_JSON_VALUE})
    public ResponseEntity<User> updateUser(@PathVariable("idx") Long idx, User user){
        userService.updateById(idx, user);
        return new ResponseEntity<User>(user, HttpStatus.OK);
```

```
    }

    @PostMapping
    public ResponseEntity<User> save(User user){
        return new ResponseEntity<User>(userService.save(user), HttpStatus.OK);
    }

    @RequestMapping(value = "/saveUser", method = RequestMethod.GET)
    public ResponseEntity<User> save(HttpServletRequest req, User user){
        return new ResponseEntity<User>(userService.save(user), HttpStatus.OK);
    }
}
```

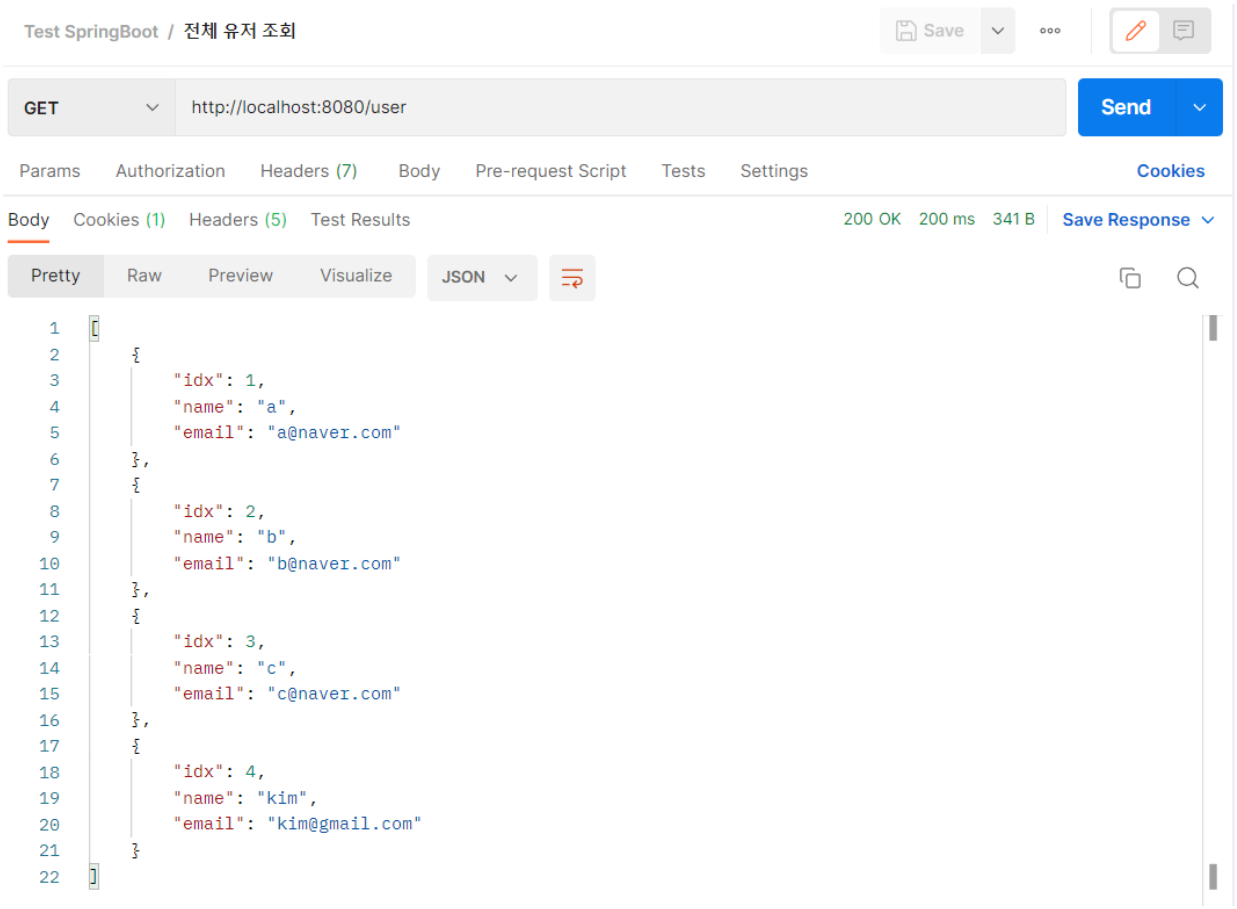
▼ `HelloWorldApplication`

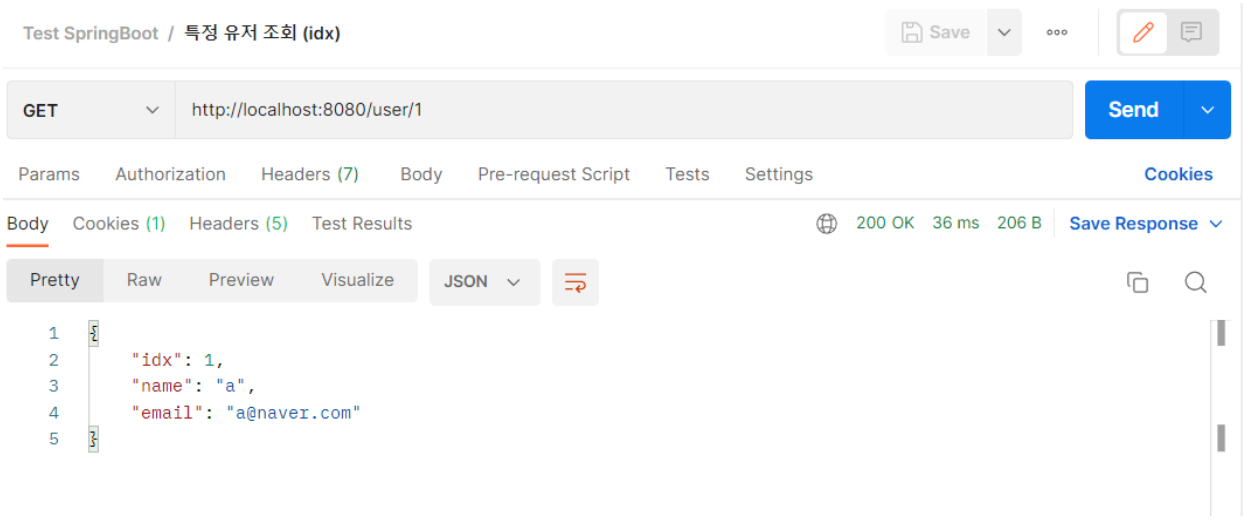
```
package com.example.helloworld;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguration;

@SpringBootApplication(exclude = SecurityAutoConfiguration.class)
public class HelloWorldApplication {

    public static void main(String[] args) {

        SpringApplication.run(HelloWorldApplication.class, args);
    }
}
```

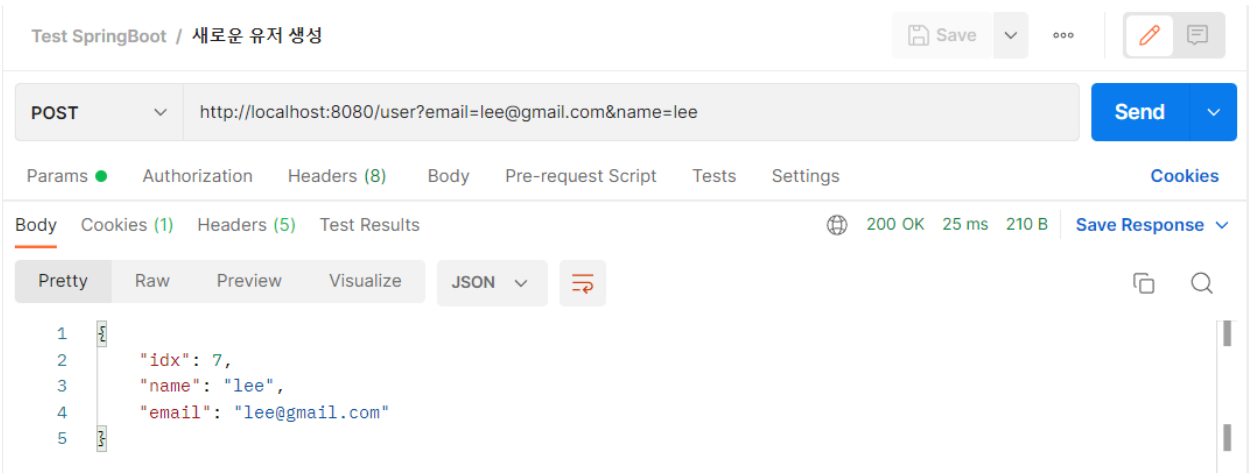
자꾸 401 Unauthorized 에러가 나서 찾아보니…

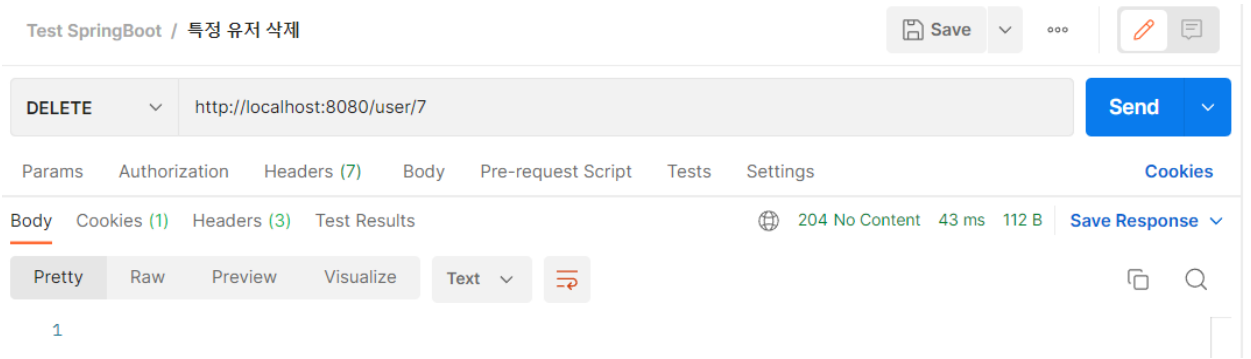
→ build.gradle에서는 Spring Security 의존성을 추가했지만, 아직 인증단계를 개발하지 않은 경우 Application에서 exclude를 이용해 Security 기능을 꺼둘 수 있다!

결과


Test SpringBoot / 전체 유저 조회
Save
GET
http://localhost:8080/user
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Body
Cookies (1)
Headers (5)
Test Results
200 OK
200 ms
341 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"idx": 1,
"name": "a",
"email": "a@naver.com"
},
{
"idx": 2,
"name": "b",
"email": "b@naver.com"
},
{
"idx": 3,
"name": "c",
"email": "c@naver.com"
},
{
"idx": 4,
"name": "kim",
"email": "kim@gmail.com"
}
]


Test SpringBoot / 특정 유저 조회 (idx)
Save
GET
http://localhost:8080/user/1
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Body
Cookies (1)
Headers (5)
Test Results
200 OK
36 ms
206 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"idx": 1,
"name": "a",
"email": "a@naver.com"
}





JPA

참고 : https://docs.oracle.com/javaee/6/tutorial/doc/bnbqa.html

Java Persistence API는 관계형 데이터를 다루는ORM (object/relational mapping) facility를 제공한다.

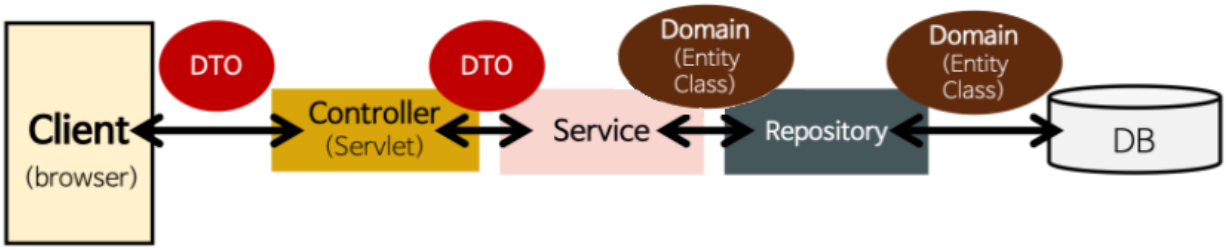
Entity

- a lightweight persistence domain objcet
- 보통 entity 하나는 관계형 데이터베이스의 table 하나를 나타낸다. 각 entity 객체는 table의 row와 일치한다.
- Requirements, Persistent Fields and Properties, …
- Primary Keys
  - 각 entity는 유일한 identifier = primary key가 있다
  - Id annotation을 사용한다

- Multiplicity
  - One-to-one
  - One-to-many
  - Many-to-one
  - Many-to-many
- Direction
  - Bidirectional Relationships vs Unidirectinoal Relationships
  - Cascade Operations and Relationships

Entity VS DTO

- 데이터베이스 테이블 → Entity
- 각 기능마다 필요로 하는 필드들을 모아 → DTO
- Service단에서 DAO로 데이터를 저장시키기 위해 DTO를 Entity로 변환하는 과정 필요 ⇒ Request DTO에서 처리, Builder 패턴 활용
- 클라이언트에게 Response를 보내기 위해 entity를 Response로 변경해주는 작업 ⇒ entity 클래스가 처리
- DTO 클래스를 사용하는 이유
  - 서비스 단은 데이터베이스와 독립적이어야 한다
  - Entity 클래스를 DTO로 재사용하는 일은 코드가 더러워질 수 있다. (결합력이 높아짐)



Repository

1. Repository<T, ID>

2. CrudRepository<T, ID>
3. PagingAndSortingRepository<T, ID>
4. JpaRepository<T, ID>

OAuth2

Spring Security

더 알아볼 것들…

- Serializable
- Transactional