

# 와플스튜디오 Spring Seminar

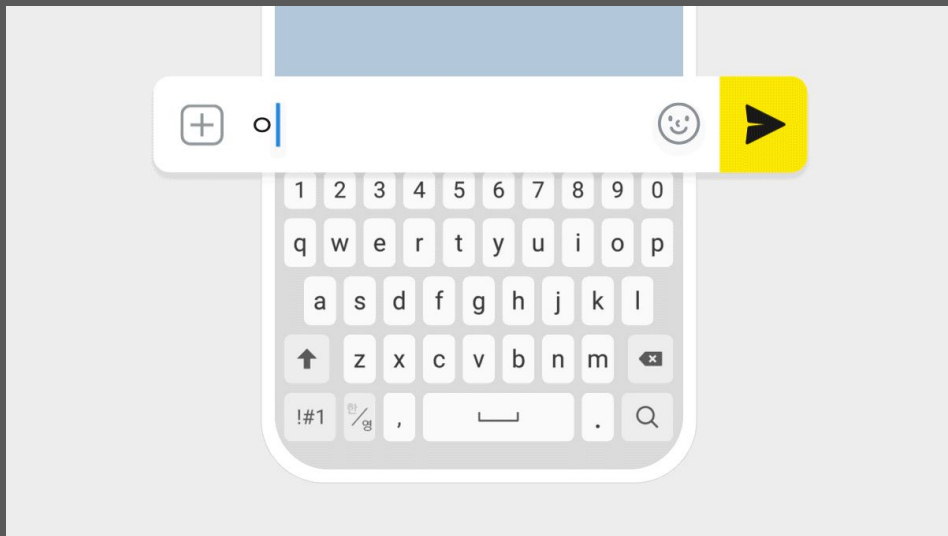
세미나장: 정원식

2023.09.06.(수) 19:00

Week0

# 세미나장 소개

- 와플스튜디오 19기(2021.03 ~)
- MXXR(2020.04 ~ 2020.12) AR/VR 안드로이드 개발
- 카카오(2020.12 ~ 현재) 이모티콘 서버 개발



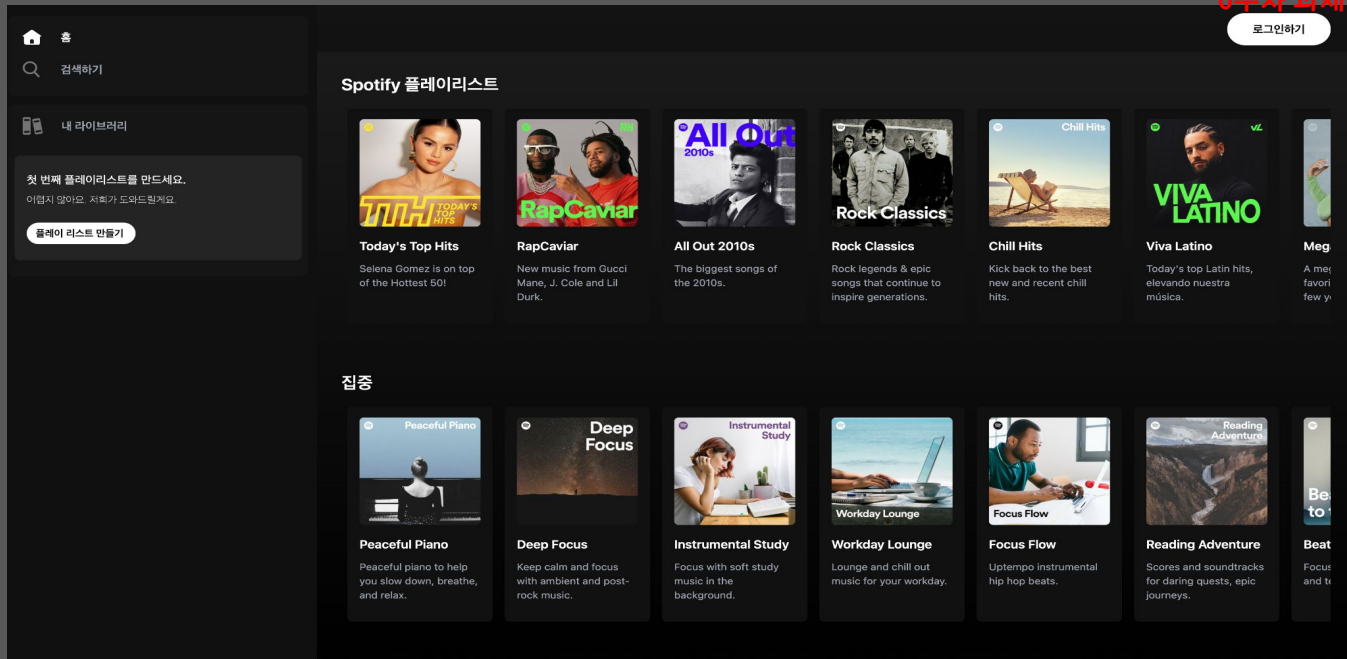
# Table of Contents

- 세미나 개괄
- Why Kotlin?
- 서버 기본 지식 학습
  - HTTP
  - REST API
- 스프링 시작하기
  - Bean, DI
  - JPA
- 세미나 과제 관련 공지

# 세미나 개괄

## Spotify 클론 코딩

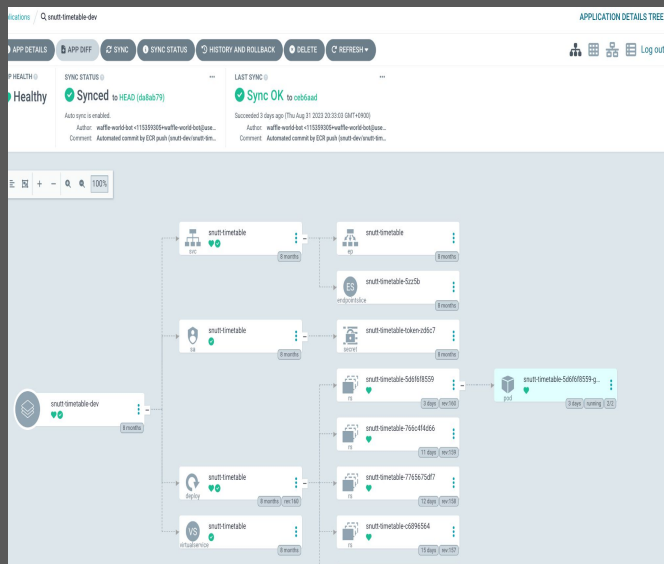
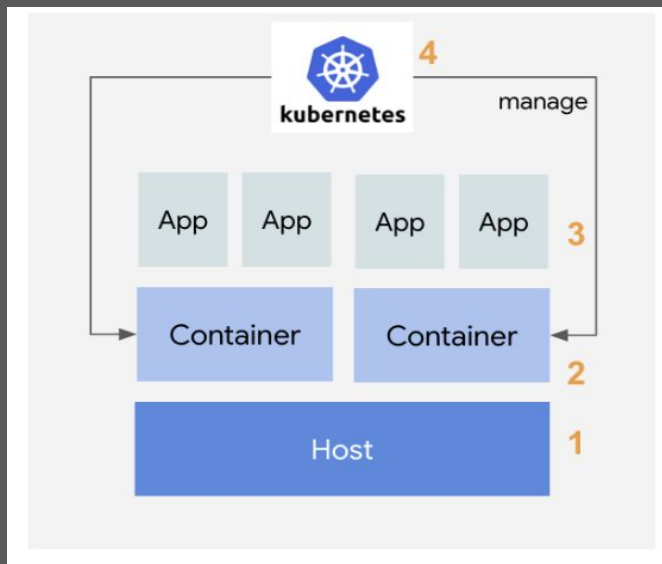
- Spotify 서버를 구현하면서 스프링 학습
- 회원 가입, 로그인, 플레이 리스트, 좋아요, 검색



# 세미나 개괄

## 쿠버네티스를 통한 서버 배포

- 컨테이너화된 어플리케이션 배포를 위해 사용하는 오픈 소스 플랫폼
- 와플 스튜디오에서도 쿠버네티스를 사용하여 서버 배포, 관리 중



# Why Kotlin?

- 간결하다.
- 안전하다.
- 기능이 많다.

```
var a = 1
```

```
a = 0
```

타입 추론

```
var b : Int = 1
```

```
b = null
```

널 불가능

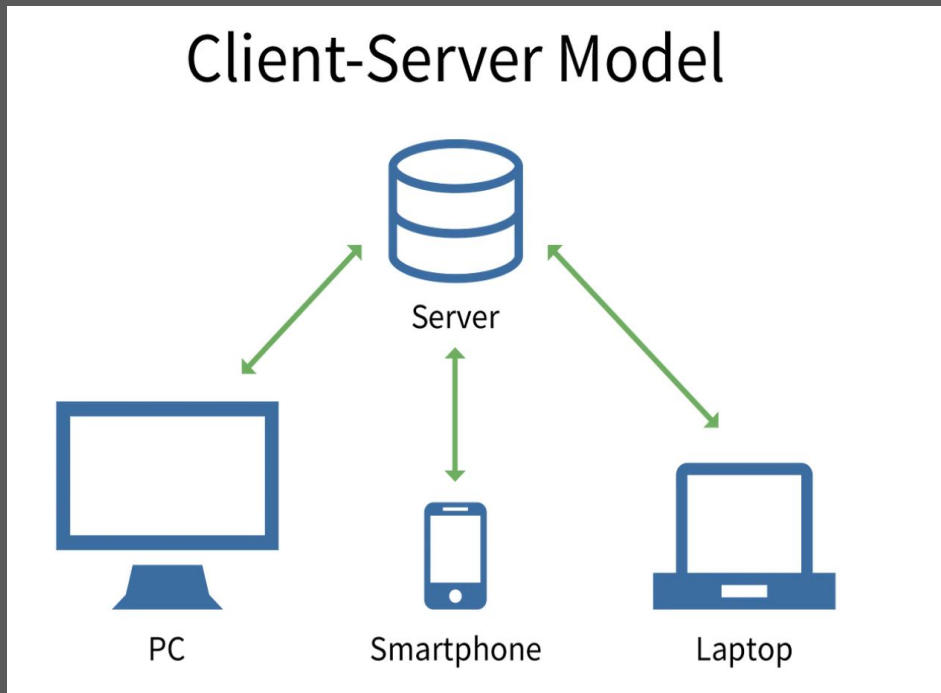
```
var c : Int? = 1
```

```
c = null
```

# 서버 기본 개념

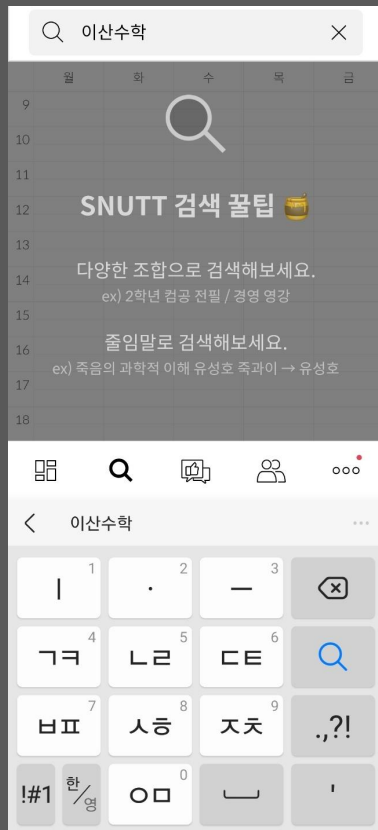
## 서버란?

- 클라이언트의 요청에 따라 데이터를 제공하거나 가공하는 역할.
- 보이지 않는 곳에서 묵묵히 일하고 있음.
- 어느 서비스에서든 필수적으로 필요하다.



# 서버 기본 개념

## 서버란?



이산수학 강의 목록을  
요청



이산수학 강의 목록을  
응답





# 서버 기본 개념

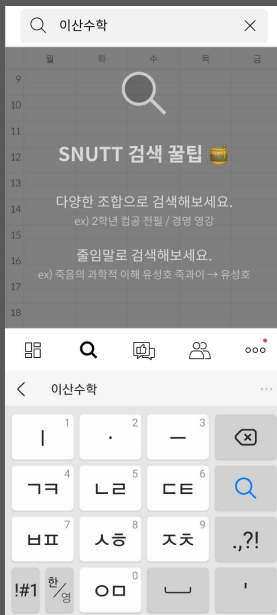
## HTTP, 클라이언트와 서버의 약속

- w3 상에서 정보를 주고받을 수 있는 프로토콜
- 클라이언트와 서버 사이에 이루어지는 요청/응답 프로토콜
- 주로 TCP를 사용하고 80번 포트를 사용

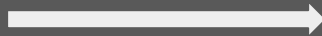
# 서버 기본 개념

## HTTP 요청

- Method: 요청 행위
  - GET: 조회
  - POST: 등록
  - PUT: 수정
  - DELETE: 삭제
- Path: 요청 경로
- Headers: 부가 정보



HTTP 1.1  
GET  
/lectures/discrete-math



# 서버 기본 개념

## HTTP 응답

- Status code: 응답 코드
  - 1xx: 조건부 응답
  - 2xx: 성공
  - 3xx: 리다이렉션
  - 4xx: 요청 오류
  - 5xx: 서버 오류
- 응답 데이터
- Headers: 부가 정보



HTTP 1.1  
200  
Content-Type: application/json  
{  
  “강사”: “이승진”  
  “학점”: “3학점”,  
  “학부”: “수리과학부”  
}



# 서버 기본 개념

## HTTP 예제 (https://weather.naver.com)

- 개발자 도구
  - 맥: option+command+I
  - 윈도우: ctrl+shift+I



▼ General

Request URL: https://weather.naver.com/today/api/nation/20230903/now

Request Method: GET

Status Code: 200

Headers Preview Response Initiator Timing Cookies

```
{
  "N04940320": {
    "naverRgnCd": "04940320",
    "regionName": "울릉/독도",
    "className": "z12",
    "lareaNm": "경상북도",
    "mareaNm": "울릉군",
    "aplymd": "20230903",
    "apltm": "22",
    "tmpr": 24.5,
    "wetrCd": "6",
    "wetrTxt": "구름많음",
    "fcstYmdt": "20230903223000",
    "hdayType": null
  },
  "N14130116": {
    "naverRgnCd": "14130116",
    "regionName": "제주",
    "className": "z17",
    "lareaNm": "제주특별자치도",
    "mareaNm": "서귀포시",
    "aplymd": "20230903",
    "apltm": "22",
    "tmpr": 24.4,
    "wetrCd": "2",
    "wetrTxt": "맑음",
    "fcstYmdt": "20230903223000",
    "hdayType": null
  },
  "N04170690": {
    "naverRgnCd": "04170690",
    "regionName": "안동",
    "className": "z13",
```

# 서버 기본 개념

HTTP 예제 (<http://localhost:8080/web/login>)

## Log in to Spotify

Email or Username

wafflestudio

Password

.....

Unknown error. (Request failed with status code 500)

Log In

Don't have an account? [Sign up for Spotify](#)

### ▼ General

Request URL:	http://localhost:8080/api/v1/signin
Request Method:	POST
Status Code:	● 500
Remote Address:	[::1]:8080
Referrer Policy:	strict-origin-when-cross-origin

# 서버 기본 개념

## REST API (Representational State Transfer)

- 자원(RESOURCE) – URI
- 행위(Verb) – HTTP METHOD (GET, POST, PUT, DELETE)
- 표현(Representations)
- 일반적으로 json 형태로 데이터 교환

```
# id가 1인 멤버 조회
GET /members/1

# id가 1인 멤버 삭제
DELETE /members/1

# 스프링 세미나의 0주차 과제 조회
GET /seminars/spring/assignments/0

# 유저 생성 요청
POST http://wafflestudio.com/members
Content-Type: application/json

{
  "username": "wafflestudio",
  "password": "spring"
}
```

# 스프링 시작하기

What is Spring framework?

- 자바기반의 오픈 소스 애플리케이션 프레임워크
- 서버 개발을 위해 필요한 대부분의 편의 기능 제공
- 한국에서 가장 많이 쓰는 서버 프레임 워크



# 스프링 시작하기

## REST API 서버 만들기

```
fun main() {  
    runApplication<SeminarApplication>()  
}  
  
@SpringBootApplication  
class SeminarApplication  
  
@RestController REST API를 처리하겠다.  
class DemoController {  
    GET 요청을 처리  
    @GetMapping("/{userId}") PATH: /users/1, /users/2..  
    fun getUser(  
        @PathVariable userId: Long,  
    ): ResponseEntity<DemoUser> {  
        val user = DemoUser(id = userId, name = "user-$userId")  
        200 ok  
        return ResponseEntity.ok(user)  
    }  
}  
  
class DemoUser(  
    val id: Long, {  
    val name: String, "id": 1,  
    "name": "user-1"  
)  
}
```



# 스프링 시작하기

## IntelliJ에서 http 요청 해보기

테스트 디렉토리에 .http 파일 만들어 테스트

The screenshot displays the IntelliJ IDEA interface. On the left, the Project view shows a directory structure with 'test' containing 'API Demo.http'. The main editor shows the content of 'API Demo.http', which includes a GET request to 'http://localhost:8080/users/2'. The bottom panel shows the 'Services' tab with an 'HTTP Request' section. It displays the executed request details, including the status '200 (327 ms)' and the response body: { 'id': 2, 'name': 'user-2' }.

```
### GET user
GET http://localhost:8080/users/2
```

GET http://localhost:8080/users/2

HTTP/1.1 200  
Content-Type: application/json  
Transfer-Encoding: chunked  
Date: Tue, 05 Sep 2023 01:35:41 GMT  
Keep-Alive: timeout=60  
Connection: keep-alive

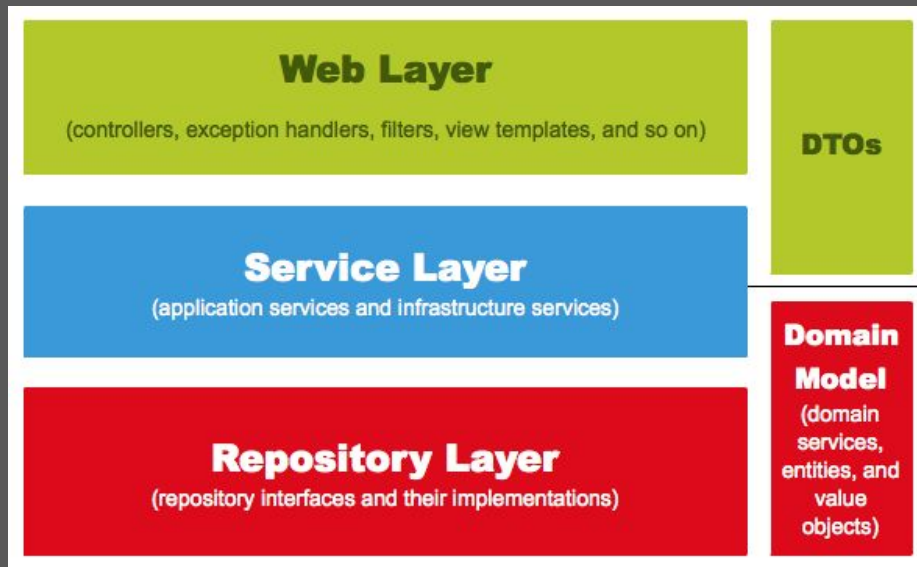
```
{
  "id": 2,
  "name": "user-2"
}
```

# 스프링 시작하기

Bean, 스프링이 관리하는 객체

- RestController: 요청에 대한 응답을 담당, REST API 처리
- Service: 비즈니스 로직을 담당
- Repository: 데이터에 대한 접근을 담당
- Component: 일반적인 케이스에 사용
- Configuration: 빈을 직접 생성

스프링에서 객체 생성과 관리를 모두 해준다.



# 스프링 시작하기

## Bean, 그리고 DI(Dependency Injection)

- Controller, Service 객체를 우리가 직접 만들지 않음
- 스프링에서 Service 객체를 만들어 이를 Controller 객체에 주입
- Bean으로 명시된 클래스끼리만 DI가 가능

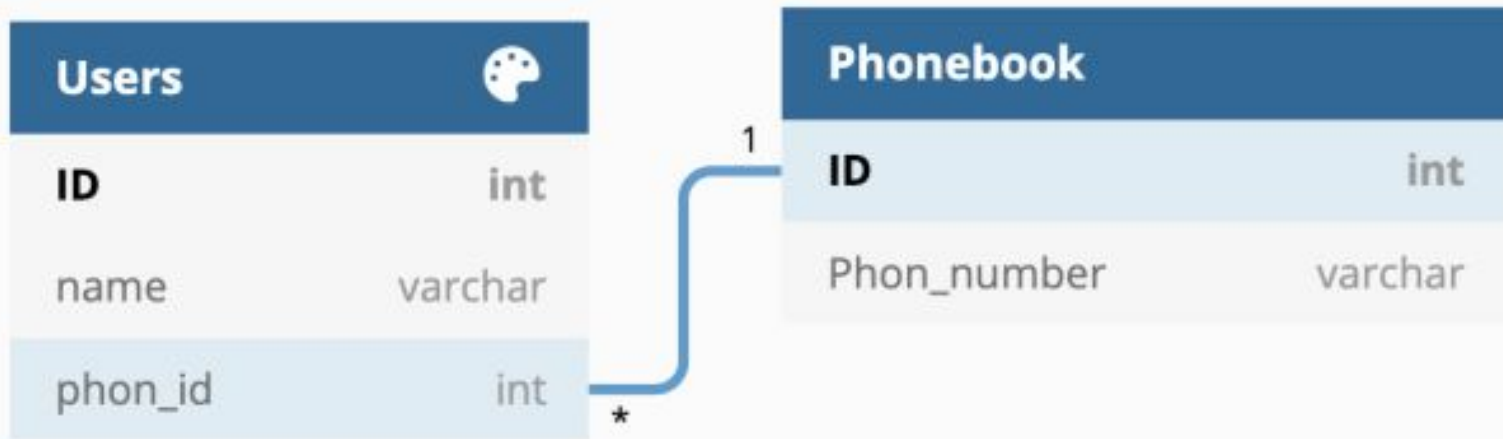
```
fun main() {  
    runApplication<SeminarApplication>()  
}  
  
@SpringBootApplication  
class SeminarApplication  
  
@RestController  
class DemoController(  
    // 생성자에 명시한 객체를 알아서 주입해준다.  
    private val service: DemoUserService,  
) {  
  
    @GetMapping("/users/{userId}")  
    fun getUser(  
        @PathVariable userId: Long,  
    ): ResponseEntity<DemoUser> {  
        return ResponseEntity.ok(service.getUser(userId))  
    }  
  
    @GetMapping("/users/v2/{userId}")  
    fun getUser2(  
        @PathVariable userId: Long,  
    ): ResponseEntity<DemoUser> {  
        val service2 = DemoUserService()  
  
        return ResponseEntity.ok(service2.getUser(userId))  
    }  
}  
  
@Service  
class DemoUserService {  
    fun getUser(userId: Long): DemoUser {  
        return DemoUser(id = userId, name = "user-${userId}")  
    }  
}
```

# 스프링 시작하기

관계형 데이터베이스, MySQL




- 각 테이블의 행과 행이 연결되는 관계를 맺을 수 있다.



# 스프링 시작하기

## ORM, Object Relational Mapping

- 테이블과 자바 객체를 맵핑

Users 	
ID	int
name	varchar
phon_id	int *

테이블명

```
@Entity(name = "users")
class User(
    @Id 기본키
    val id: Int,
    val name: String,
    val phoneId: Int,
)
```

# 스프링 시작하기

## JPA, 자바의 대표적인 ORM

- JpaRepository를 상속
- 그럴듯한 함수명을 가진 함수를 생성. 알아서 쿼리를 날려준다. [함수명 짓는 방법](#)

```
@Entity(name = "users")
class User(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    val id: Long = 0L,
    val name: String,
)

interface UserRepository : JpaRepository<User, Long> {
    fun findByName(name: String): User?
}
```

```
Hibernate:
    /* <criteria> */ select
        d1_0.id,
        d1_0.name
    from
        users d1_0
    where
        d1_0.name=?
```

# 스프링 시작하기

## TDD, 테스트 주도 개발

- 테스트 코드 먼저 짜고 그 다음 개발을 진행한다.
- 배포 전 심리적 안정감을 가질 수 있다.
- 단위 테스트: 클래스 수준의 테스트 ex) UserServiceTest.kt
- 통합 테스트: 말 그대로 전체 테스트 ex) UserIntegrationText.kt

Test Results		793 ms
UserServiceTest		793 ms
✖	유저 식별을 위한 인증 토큰이 정확해야한다()	775 ms
✖	이미 존재하는 유저 이름으로 가입할 수 없다()	6 ms
✖	로그인시 유저 이름과 비밀번호가 정확해야 한다()	5 ms
✖	유저 이름과 비밀번호는 4글자 이상이어야 한다()	7 ms

Test Results		896 ms
UserIntegrationTest		896 ms
!	회원가입시에 유저 이름 혹은 비밀번호가 정해진 규칙에 맞지 않는 경우 400 응답을 내려준다()	879 ms
!	회원가입시에 이미 해당 유저 이름이 존재하면 409 응답을 내려준다()	7 ms
!	로그인 정보가 정확하지 않으면 404 응답을 내려준다()	5 ms
!	잘못된 인증 토큰으로 인증시 401 응답을 내려준다()	5 ms

# 세미나 과제 관련 공지

- 2023 스프링 세미나에서는 총 4번의 과제가 주어질 예정입니다. (week 0, 1, 3, 4)
- 모든 과제는 다음 세미나가 있는 주의 월요일까지 제출해야 합니다.
  - ex) week0 과제는 2023-09-18 23:59:59까지 제출
- 과제 미제출 시, 루키 과정에서 탈락합니다.
- 추가 과제 제출 시, 이들의 grace day가 부여됩니다.