학습 내용 정리 - 1월 3주차

박시준

목차

- 스프링 가이드에서 주어진 예제 실습
- 프론트 엔드에 연결
- 테스트 프로젝트 작성
- 최소 신장트리, 크루스칼 알고리즘, Union-find 알고리즘

스프링 가이드에서 주어진 예제 실습

예제로 REST API 실습

- https://spring.io/guides/gs/rest-service/에 있는 예제를 바로 따라서 만들어 봄
- Localhost:8080/greeting?name=User로 이동하면 아래와 같은 JSON 객체가 웹사이트에 표시됨

```
http://localhost:8080/greeting?name=User
```

COPY

The `name` parameter value overrides the default value of `World` and is reflected in the response, as the following listing shows:

```
{"id":1,"content":"Hello, User!"}
```

COPY

데이터 모델과 컨트롤러

[예제 코드] 주어진 대로 그대로 타이핑 한다

데이터 모델

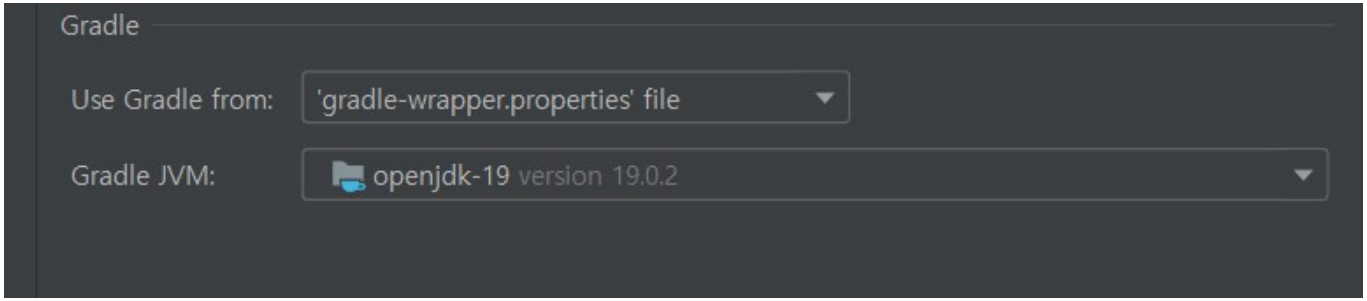
```java
package com.example.restservice;

public record Greeting(long id, String content) { }
```

컨트롤러

```java
@RestController
public class GreetingController {

    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

    @GetMapping("/greeting")
    public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {
        return new Greeting(counter.incrementAndGet(), String.format(template, name));
    }
}
```

Gradle 3.0.1 Error

- Java 17버전 이상을 사용하지 않아서 그런 것.
- IntelliJ의 Setting에서 빌드 도구 – Gradle 에서 Gradle JVM이 자바 11로 되어있었는데 이것을 19로 바꾸니 정상 동작하였다.
- Gradle 3.0은 JAVA 17버전 이상부터 호환이 된다.



모델 작성 중 의문 사항

- Record가 뭐지? 왜 class 대신 있는거지?
- 필드, 생성자, Getter, Setter 대신 해당 클래스의 매개변수 위치에 필드 값을 정의해서 입력하면, 생성자, Getter, Setter 등 잡다한 역할이 모두 알아서 구현된다.
- record에선 매개변수라고 하지 않고, 컴포넌트라고 한다.
- toString(), hashCode() 등 모든 메서드가 포함되어있음.
- JDK16부터 정식 스펙으로 포함됨.

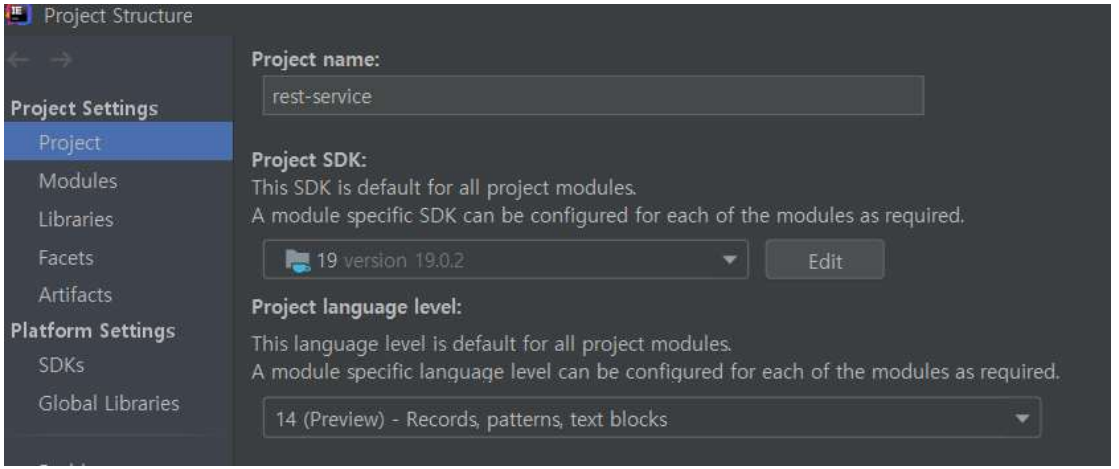
데이터 모델

```
package com.example.restservice;

public record Greeting(long id, String content) { }
```

JAVA code Error – record

- Intellij 설정 문제
- Project language level에서 15가 아닌 14로 설정해야 함.
- Project language level이란?

=> 현재 코드 어시스턴트가 지원해야할 언어 레벨을 의미한다.
현재 사용하고 있는 JDK 버전과 지원해야할 JAVA 코드의 버전과 다를 때 설정해서 사용해야 한다.



컨트롤러 작성 중 의문 사항

- 왜 private을 사용하는 걸까?

=> 외부 접근 제한을 막으려고, 본 템플릿을 고치려면 setter와 같은 메서드를 활용해야한다. 물론 여기선 setter가 없어서 못 고침. 그냥 정해진 템플릿만 쓰는 거.

- 왜 static을 사용하는 걸까?

=> Heap이 아닌 static영역에 저장. 프로세스 종료 시까지 살아있음.

인스턴스 필드 같은 경우는 생성될 때 값이 제각각 다르다. 하지만 static으로 지정하면, 클래스가 생성될 때 static 메모리 영역에 값이 생성되어 그 값이 공통적으로 쓰인다. 하지만 static으로 지정해두면 인스턴스에서는 이 변수에 할당을 할 순 없다.

변수	생성 시기	소멸 시기	저장 메모리	사용 방법
클래스 변수	클래스가 메모리에 올라갈 때	프로그램이 종료될 때	메소드 영역	클래스이름.변수이름
인스턴스 변수	인스턴스가 생성될 때	인스턴스가 소멸할 때	힙 영역	인스턴스이름.변수이름
지역 변수	블록 내에서 변수의 선언문이 실행될 때	블록을 벗어날 때	스택 영역	변수이름

컨트롤러 작성 중 의문 사항

- AtomicLong은 뭐지?

=> Long 자료형을 갖고 있는 Wrapping 클래스

```java
@RestController
public class GreetingController {

    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

    @GetMapping("/greeting")
    public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {
        return new Greeting(counter.incrementAndGet(), String.format(template, name));
    }
}
```

- Wrapping 클래스는 뭐지?

⇒기본 타입 데이터를 객체로 포장해주는 클래스다. 사용하는 이유는 메소드의 인자로 객체가 요구되면 기본 타입 데이터를 객체로 바꿔야함. 이것의 편의를 위해서 사용하는 클래스다.

예) Int -> Integer, char -> Character, long -> Long

```java
Integer num = new Integer(17); // 박싱
int n = num.intValue();      // 언박싱
```

AtomicLong? Atomic?

- AtomicLong vs Long

=> AtomicLong은 Long 자료형이지만, 그 값이 Atomic하게 update 될 수 있다.

- Atomic한 게 뭐지?

=> read, write의 절대적 순서가 모든 스레드에서 지켜지는 자료구조.

=> 여러 스레드에서 접근 할 수 있는 변수는 Atomic하게 처리해줘야 잘못된 간섭이 일어나지 않는다. 멀티 스레드 환경에서는 Atomic 하지 않은 변수에는 여러 스레드가 접근하면서 값을 고치게 되면 원하지 않은 결과값이 출력될 수 있음.

멀티 스레드에서 동시에 접근하는 변수

- 왜 멀티 스레드에서 원하지 않는 값이 출력?

=> 단일 스레드라면 오른 쪽의 코드가 완벽하게 작동한다. 하지만, 멀티 스레드 환경에서는 하나의 전역변수에 여러 메서드가 동시에 접근할 수 있다. 여러 스레드 작업이 동시에 일어나며 스레드의 실행 순서는 특정 지을 수 없기 때문에 출력되는 값을 확정 짓지 못함. 아래 예시는 2를 원했으나 1이 출력.

```
public class Counter {
    int counter;

    public void increment() {
        counter++;
    }
}
```

[원하는 동작]

1. `thread 1` 이 `result` 의 값을 레지스터에 저장한다.
2. `thread 1` 의 레지스터의 값을 1 증가시킨다.
3. `result` 에 증가시킨 값을 저장한다.
4. `thread 2` 가 `result` 의 값을 레지스터에 저장한다.
5. `thread 2` 의 레지스터의 값을 1 증가시킨다.
6. `result` 에 증가시킨 값을 저장한다.

[실제 일어날 수도 있는 동작]

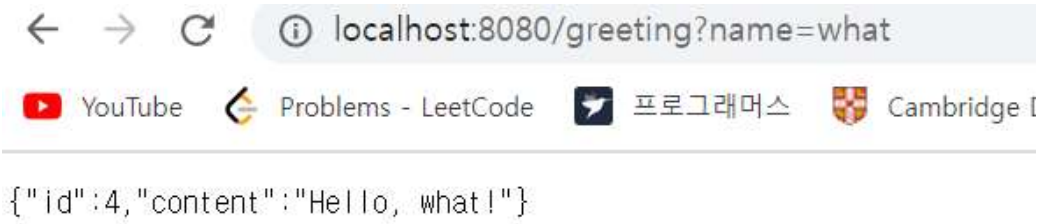
1. `thread 1` 이 `result` 의 값을 레지스터에 저장한다.
2. `thread 2` 가 `result` 의 값을 레지스터에 저장한다.
3. `thread 1` 의 레지스터의 값을 1 증가시킨다.
4. `thread 2` 의 레지스터의 값을 1 증가시킨다.
5. `result` 에 증가시킨 값을 저장한다.
6. `result` 에 증가시킨 값을 저장한다.

Lock과 Atomic

- 원하지 않는 값이 출력되지 않도록, 하나의 스레드 작업 중에는 다른 스레드가 작업을 하지 못하도록 Lock을 걸 수 있음. (파이썬에는 Lock 거는 메서드 이름이 threading.Lock().)
- 하지만 이렇게 Lock을 걸면 다른 스레드에서는 해당 작업을 하지 못함.
- 이런 Lock을 걸지 않고도 동기화 문제를 해결하기 위한 방법이 바로 Atomic.
- Atomic의 동작 원리는 CAS 알고리즘이다.
- CAS 알고리즘은 CPU 캐시와 메모리에 있는 변수를 비교하는 방법인데, 현재 쓰레드(캐시)에 저장된 값과 메모리에 저장된 값을 비교하여, 일치하는 경우 새로운 값으로 교체되고, 불일치하면 실패 처리하고 다시 재시도한다.

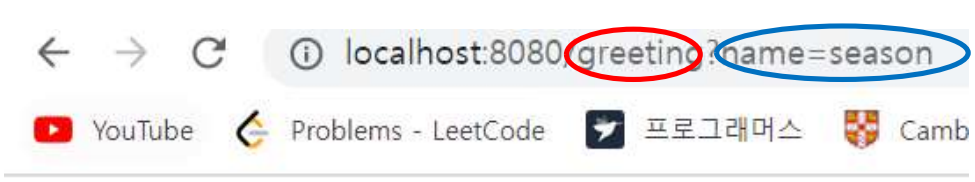
Atomic의 incrementAndGet()

- CAS 알고리즘을 적용하는 메서드(get(), set() 모든 메서드도 마찬가지)
- incrementAndGet() 내부적으로 comparedAndSet()을 실행, 메모리에 있는 변수와 캐시에 있는 변수를 비교하여 같을 경우에만 값을 업데이트 할 수 있게 한다.
- 특별히, incrementAndGet()은 값을 1 증가시키고 그 값을 리턴함.
- 서버를 구동하는 중에 클라이언트에서 파라미터를 바꿔서 새로 접속할 때마다 id의 넘버가 올라간다.
- 서버는 계속 살아있기 때문



예제 실행 결과

[실행 결과]



{"id":2,"content":"Hello, season!"}

{"id":3,"content":"Hello, World!"}

```java
@RestController
public class GreetingController {

    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

    @GetMapping("/greeting")
    public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {
        return new Greeting(counter.incrementAndGet(), String.format(template, name));
    }
}
```

예제를 통해 복습 – JAVA

- String의 format 메서드
- Format은 String의 Static 메서드다. (Static 메서드라 하면, 인스턴스화 시켜서 사용하는 게 아니라 그냥 클래스 상태에서 메서드 사용)
- 정해진 프레임을 짜놓고, 그 프레임에 있는 포맷 지정자에 변수를 집어넣어서 돌려주는 메서드
- 포맷 지정자의 형식과 변수 타입이 서로 일치해야한다.

```
"name", defaultValue = "World") String name) {
(), String.format(template, name));
```

```
1  int i = 23;
2
3  System.out.println(String.format("%d_", i));
```

```
String str = "tete";

System.out.println(String.format("%s_", str));
```

예제를 통해 복습 – @RESTController

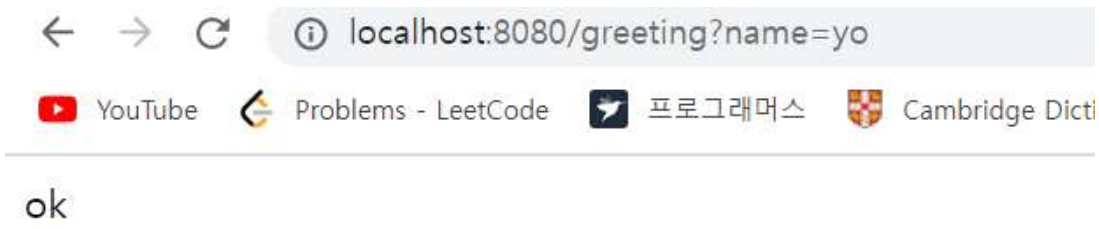
- @RESTController는 @ResponseBody와 @Controller를 합친 것이다.
- 그러므로 현재 컨트롤러의 모든 메서드가 @ResponseBody를 쓸 것이면, 그것을 생략하고 @RESTController를 쓰는 게 편함

```java
@Controller
public class GreetingController {

    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

    @ResponseBody
    @GetMapping("/greeting")
    public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {
        return new Greeting(counter.incrementAndGet(), String.format(template, name));
    }
}
```

예제를 통해 복습 - @ResponseBody

- 예제 코드의 @ResponseBody 메서드의 반환 값은 객체.
- 사실 @ResponseBody는 매우 유연해서 String으로도 반환해도 된다.

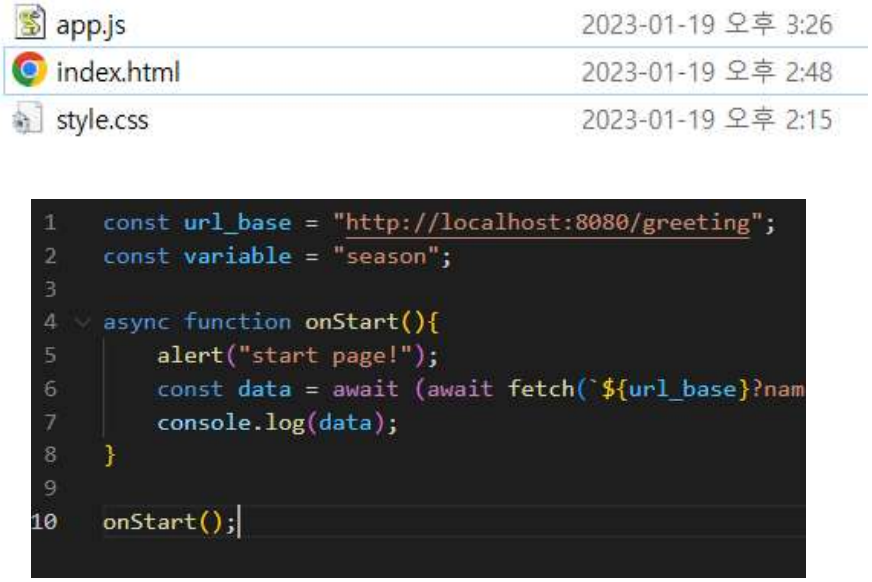
```java
@RestController
public class GreetingController {

    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

    @GetMapping("/greeting")
    public String greeting(@RequestParam(value = "name",
        return "ok";
    }
}
```



프론트엔드 작성해서 API 받아보기

프론트엔드 작성

- 간단한 HTML 파일과 JS 코드로 테스트 한다.
- 별도의 서버 포팅 없음. 컴퓨터에 있는 HTML 파일로 그냥 테스트 한다.

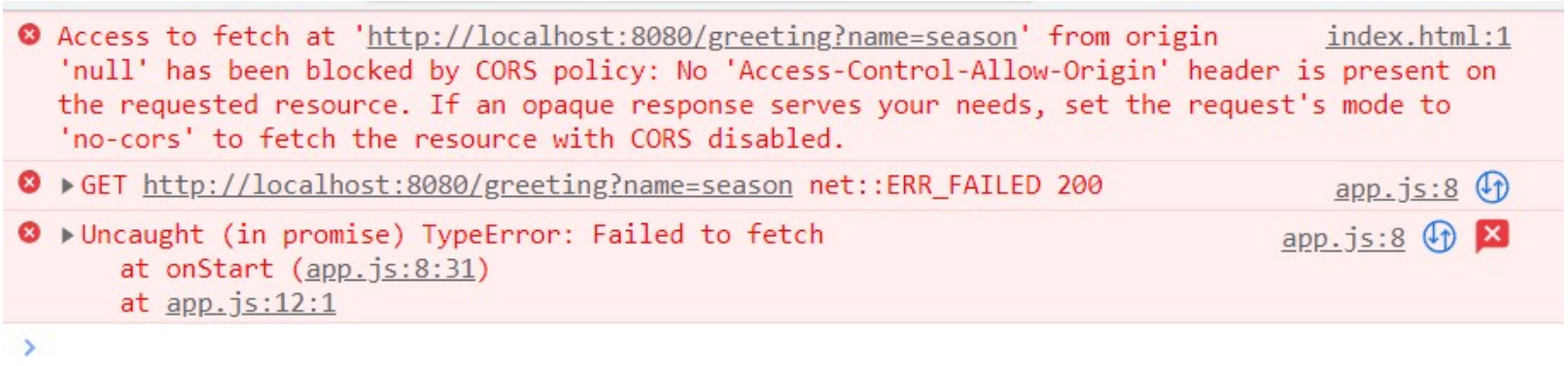



```
const url_base = "http://localhost:8080/greeting";
const variable = "season";

async function onStart(){
    alert("start page!");
    const data = await (await fetch(`${url_base}?nam
    console.log(data);
}

onStart();
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <title>Document</title>
</head>
<body>
    <h2>Welcome to API test!</h2>
    <script src="app.js">
    </script>
</body>
</html>
```

Error – No ‘Access-Control-Allow-Origin’

- 작성이 끝나고 index.html 파일로 접속하여 콘솔창을 켜봤으나, 원하는 콘솔은 안 나오고 No ‘Access-Control-Allow-Origin’이라는 경고가 나왔다.
- JavaScript 엔진 표준 스펙에 있는 동일 출처 정책(Same-Origin Policy)이라는 보안규칙으로 인해 발생한다. 같은 출처 (프로토콜, 호스트명, 포트) 의 페이지에만 접근 가능하다.
- 거부당한 이유는 프론트쪽에서 http://localhost:8080으로 접근했기 때문. (서버 단에서는 http프로토콜이 제외된, localhost:8080으로만 포팅한다.



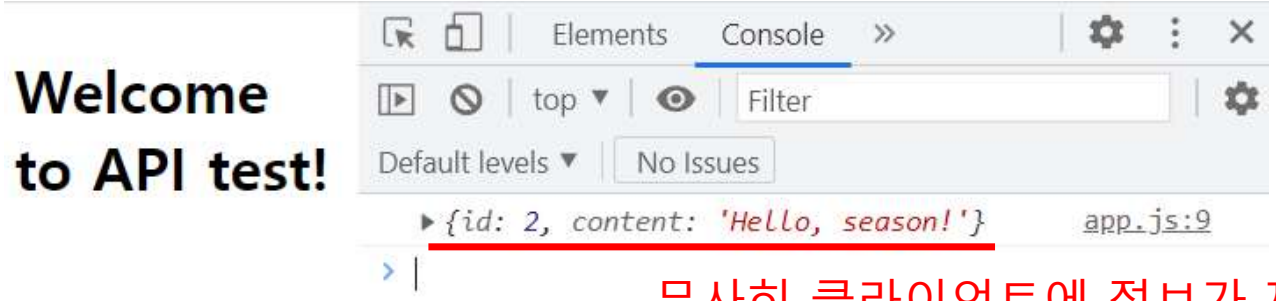
Error – No ‘Access-Control-Allow-Origin’

- 외부 요청을 허용하게 해주는 CORS 정책이 있다.
- CORS(Cross-Origin Resource Sharing)
- 다른 Origin의 데이터를 읽고 싶으면 CORS 표준을 지켜서 헤더에 별도로 설정을 해야 한다.
- CORS 요청을 하기 위해서 프론트 단에서는 할 것이 없다. 브라우저가 다 알아서 한다.
- CORS 응답을 하기 위해서는 서버 단에서는 추가적인 작업이 필요하다. (스프링 자체적인 해결법 있음)
- 지역적, 전역적 방법이 있음
- 지역적인 설정은 @CrossOrigin

```
@CrossOrigin
@GetMapping("/greeting")
public Greeting greeting(@RequestParam(val
    return new Greeting(counter.incrementA
}
```

JS Error – No 'Access-Control-Allow-Origin'

- @CrossOrigin을 붙이니 한 방에 해결



무사히 클라이언트에 정보가 제공되었다.

```
const url_base = "http://localhost:8080/greeting";
const variable = "season";

async function onStart(){
    alert("start page!");
    const data = await (await fetch(`${url_base}?name=${variable}`)).json();
    console.log(data);
}

onStart();
```

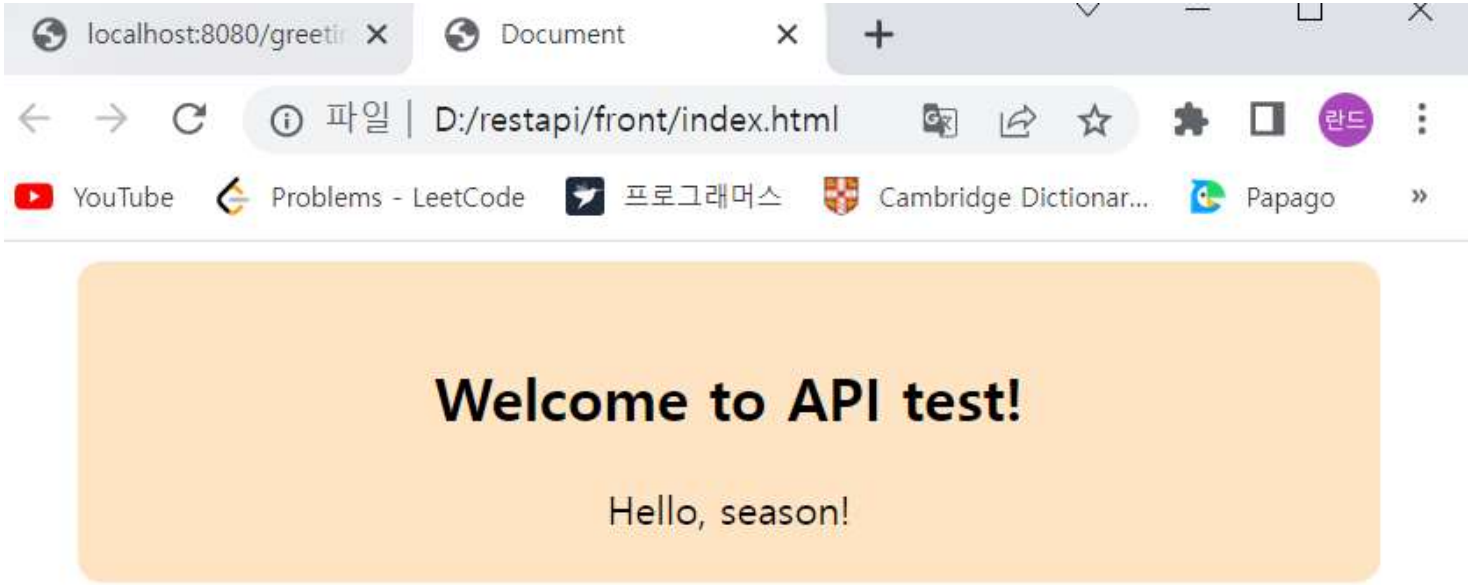
[프론트엔드]

```
@CrossOrigin
@GetMapping("/greeting")
public Greeting greeting(@RequestParam(value = "name",
    return new Greeting(counter.incrementAndGet(), Str
}
```

[백엔드]

프론트엔드 API 활용 완료

- 프론트엔드에서 API를 사용해서 다음과 같이 렌더링 한다.



테스트 프로젝트 생성

-영상 편집 어플리케이션을 서비스한다고 가정

서비스 개요 및 기능

- 영상을 편집하는 어플리케이션
- 홈 화면에서 작업 중인 영상 목록을 볼 수 있음
- 영상 목록에서 영상을 누르면 편집 화면으로 갈 수 있음
- 사용자는 로그인할 수 있음
- 로그인 한 사용자는 멤버십에 가입할 수 있음
- 멤버십에 가입한 사용자는 광고 없이 이용할 수 있음

비즈니스 요구사항 정리

- 데이터

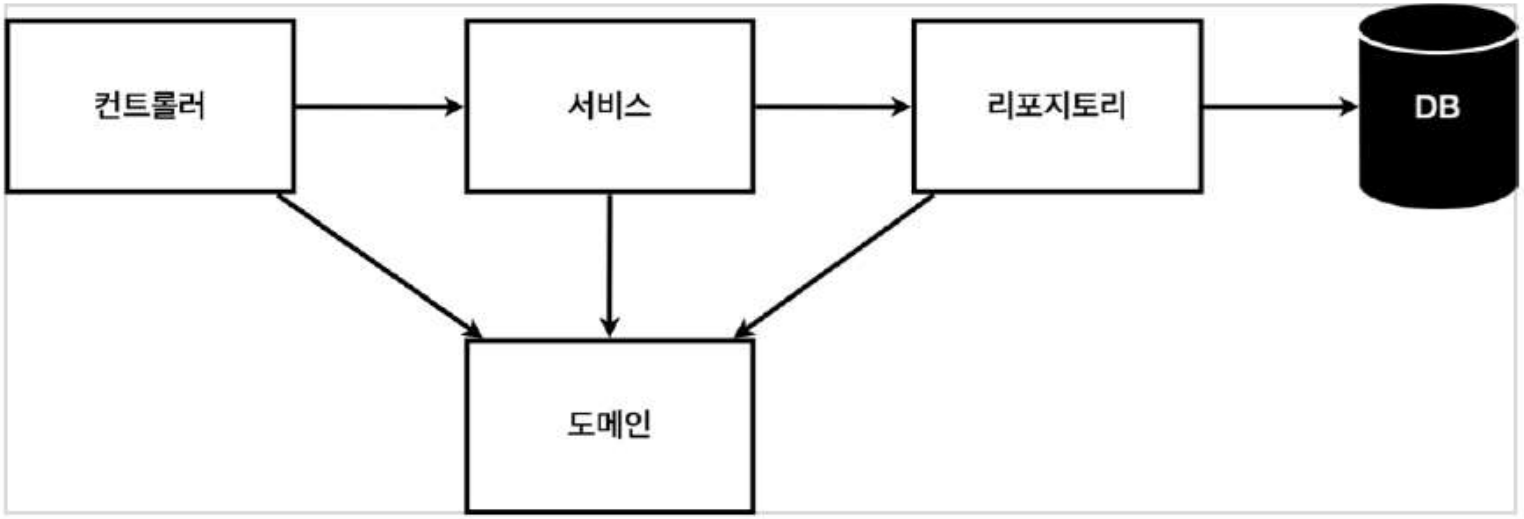
1) 고객 (id, 이름, 멤버십 유무)
2) 영상 (id, 마지막 편집날짜)

- 기능

1) 로그인
2) 작업 파일 목록 조회
3) 파일 등록
4) 파일 편집

웹 어플리케이션 작성 순서

일반적인 웹 애플리케이션 계층 구조



- 컨트롤러: 웹 MVC의 컨트롤러 역할
- 서비스: 핵심 비즈니스 로직 구현
- 리포지토리: 데이터베이스에 접근, 도메인 객체를 DB에 저장하고 관리
- 도메인: 비즈니스 도메인 객체, 예) 회원, 주문, 쿠폰 등등 주로 데이터베이스에 저장하고 관리됨

- 도메인 -> 리포지토리 -> 서비스 -> 컨트롤러

Error - InaccessibleObjectException

- 빌드 도중 InaccessibleObjectException
- 구글링 결과 'Scala'라고 하는 Plugin에 문제가 있는듯 함
- JDK16이후로 많은 클래스의 형태가 바뀌었는데, 이것이 Scala 라이브러리와 호환이 안 되면서 발생하는 문제였음.
- 해결책: Scala Plugin 해제

도메인, 리포지토리, 서비스 구현

- 리턴 값을 가져와야 할 때 리턴 값의 형태를 알 수 있도록 DB와 가장 가까운 곳에서 부터 바텀 업 구조로 작성한다.
- 데이터 모델(엔티티) -> 리포지토리(DAO) -> 서비스 -> 컨트롤러 순서로 작성한다.

엔티티 구현

- 멤버와 비디오를 담기 위한 데이터 모델을 먼저 작성
- Getter와 setter는 별도 메서드를 사용하지 않아도 Lombok 라이브러리를 사용하면 알아서 자동 생성해준다.

```java
@Getter @Setter
public class Member {
    private Long id;
    private String name;
    private String email;
    private String password;
    private Boolean membership;
}
```

```java
@Getter @Setter
public class Video {
    private Long id;
    private String name;
    private String date;
}
```

리포지토리 구현

- DAO = Data Access Object
- DB(현재는 메모리만)에 접근하기 위한 객체를 별도로 구현

```java
@Repository
public class MemoryMemberRepository implements MemberRepository {

    private static Map<Long, Member> store = new HashMap<>();
    private static long sequence = 0L;

    @Override
    public void make(Member member) {
        member.setId(sequence++);
        store.put(member.getId(), member);
    }

    @Override
    public Optional<Member> findById(Long id) { return Optional.ofNullable
```

```java
@Repository
public class MemoryVideoRepository implements VideoRepository{

    private static Map<Long, Video> store = new HashMap();
    private static Long sequence = 0L;

    @Override
    public void enroll(Video video) {
        video.setId(sequence++);
        store.put(video.getId(), video);
    }
```

서비스 구현

- 서비스를 제공하기 위한 핵심 로직을 작성
- 서비스는 리포지토리를 갖고 논다. DAO를 통해서 DB의 데이터를 가져온다.

```java
@Service
public class MemberService {

    private final MemberRepository memberRepository;

    @Autowired
    public MemberService(MemberRepository memberRepository) {
        this.memberRepository = memberRepository;
    }

    public Long join(Member member){
        validatedDuplicateMember(member);
        memberRepository.make(member);
        return member.getId();
    }
```

```java
@Service
public class VideoService {
    private final VideoRepository videoRepository;

    @Autowired
    public VideoService(VideoRepository videoRepository) {
        this.videoRepository = videoRepository;
    }
```

컨트롤러 vs 서비스 ?

- 데이터 모델(엔티티)와 리포지토리는 역할이 확실히 구분되고 이해됨.
- 근데 컨트롤러와 서비스는 왜 구분하는 것일까? 어떻게 구분하는 것일까?

```java
@Controller
public class MemberController {

    private final MemberService memberService;

    @Autowired
    public MemberController(MemberService memberService) { this.memberS

    @GetMapping("/members/new")
    public String createForm() { return "members/createMemberForm"; }

    @PostMapping("/members/new")
    public String create(MemberForm form) {
        Member member = new Member();
        member.setName(form.getName());

        memberService.join(member);

        return "redirect:/";
    }
```

[컨트롤러]

```java
public class MemberService {

    private final MemberRepository memberRepository;

    public MemberService(MemberRepository memberRepository) {
        this.memberRepository = memberRepository;
    }

    //회원가입
    public long join(Member member){
        validatedDuplicateMember(member); // 중복 회원 검증
        memberRepository.save(member);
        return member.getId();
    }
```

[서비스]

컨트롤러 vs 서비스

- 컨트롤러는 웹 계층을 처리하기 위한 코드를 작성함
- 서비스는 웹 계층 처리를 제외한 비즈니스 로직을 넣는다. 웹 없이 콘솔에서만 동작하는 어플리케이션을 작성한다고 할 때, 이 서비스 부분으로 해결할 수 있어야 한다.
- 그러므로 웹 주소를 매핑하고 API 데이터를 가져오고 보내는 것은 모두 컨트롤러에서 일어난다고 보면 됨.
- 계산에 필요한 알고리즘 로직은 모두 서비스에서 일어남.

엔티티 vs 리포지토리

- 엔티티는 DB에 쓰이는 데이터 형태를 그대로 투사한 것.
- 데이터 베이스의 하나의 column이 엔티티의 필드가 된다.
- 리포지토리는 왜 사용하는 것이라까?

=> 엔티티를 선언함으로써 데이터베이스의 전체적인 틀을 잡았으나, 이것을 가만히 놔두면 만든 의미가 없다. 값을 넣고 꺼내 써야하는데, 이 역할하는 객체를 독립적으로 구분해놓았다. 이것이 리포지토리다.

텍스트가 안 나온다... 왜지..

- 컴포넌트 스캔도 달고 컨트롤러도 달았으나, 화이트레이블 에러페이지만 나온다.
- 빈 등록에서 빠진 게 있는지 점검했으나, 발견 못함.

```java
@ComponentScan
@Configuration
public class AppConfig {

}
```

```java
@Controller
public class HomeController {

    @GetMapping("/")
    public String home(){
        return "welcome";
    }
}
```

@ResponseBody 또는 @RestController

- 며칠 전 그렇게 실습 해놓고..
- 심지어 RestController로 예제도 만들어놓고 똑같은 실수를
- @ResponseBody나 @RestController 사용해서 해결

```
@RestController
public class HomeController {

    @GetMapping("/")
    public String home(){
        return "welcome";
    }
}
```

컨트롤러에서 요청 매핑, GET 테스트

- 로그인 양식을 예제로 테스트한다.
- 로그인 입력창에서 username(id), password를 입력 받음.
- 실제로는 프론트엔드에서 HTML placeholder로 만들겠지만,
  그냥 테스트 용도로 GET으로 passive 입력 값을 전송함.

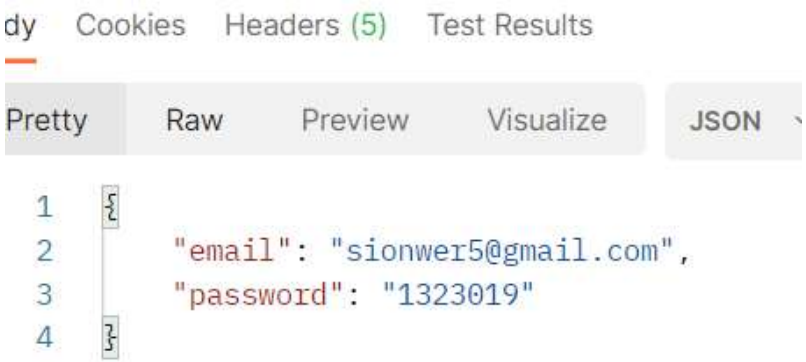

```java
@ResponseStatus(HttpStatus.OK)
@GetMapping("/login")
public LoginForm login(){
    LoginForm loginForm = new LoginForm();
    loginForm.setUsername("Input your user id");
    loginForm.setPassword("Your password");
    return loginForm;
}
```

컨트롤러에서 요청 매핑, POST 테스트

- 사용자가 특정 데이터를 입력했다고 가정하고, POSTMAN으로 테스트 함. 로그가 아래와 같이 성공적 출력.

```java
@ResponseStatus(HttpStatus.OK)
@GetMapping("/login")
public LoginFrame loginForm(){
    LoginFrame loginFrame = new LoginFrame();
    loginFrame.setEmail("Input your user email");
    loginFrame.setPassword("your password");
    return loginFrame;
}

@PostMapping("/login")
public LoginFrame login(@RequestBody LoginFrame data) {
    log.info("email={}, password={}",
            data.getEmail(), data.getPassword());
    return data;
```

```
: name=season, email=sionwer5@gmail.com, pass
: email=sionwer5@gmail.com, password=1323019
```



회원가입 기능도 REST API로 테스트

- 회원가입도 똑같이 컨트롤러 생성하여 GET, POST 테스트.

```java
@GetMapping("/join")
public JoinFrame joinForm(){
    JoinFrame joinFrame = new JoinFrame();
    joinFrame.setName("your name");
    joinFrame.setEmail("your email");
    joinFrame.setPassword("your password");
    joinFrame.setPasswordConfirm("password confirm");
    return joinFrame;
}

@PostMapping("/join")
public JoinFrame join(@RequestBody JoinFrame data){
    log.info("name={}, email={}, password={}, password confirm={}",
            data.getName(), data.getEmail(), data.getPassword(), data.getPasswordConfirm());
    return data;
}
```

```
: name=season, email=sionwer5@gmail.com, password=1323019, password confi
: name=season, email=sionwer5@gmail.com, password=1323019, password confi
```

POST http://localhost:8080/join

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1  {"name":"season", "email": "sionwer5@gmail.com", "password":"1323019", "passwordConfirm": "1323019"}
```

Error – bean could not be found.

- Parameter 0 of constructor in … that could not be found.
- @Repository를 등록하지 않아서 발생한 문제.
- @Repository나 @Component로 빈 등록을 해야 컨테이너가 @Autowired를 통해 자동주입을 할 수 있다.

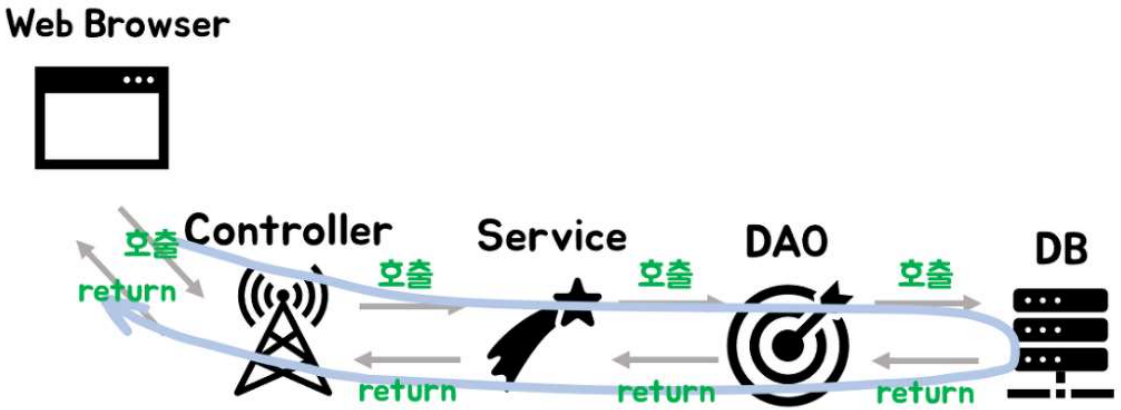
```
***************************
APPLICATION FAILED TO START
***************************

Description:

Parameter 0 of constructor in training.restapi.service.VideoService required a bean of type 'training.restapi.repository.VideoRepository' that could not be found.
```

```
public class MemoryVideoRepository implements VideoRepository{

    private static Map<Long, Video> store = new HashMap();
    private static Long sequence = 0L;
```

```
@Repository
public class MemoryVideoRepository implements VideoRepository{

    private static Map<Long, Video> store = new HashMap();
    private static Long sequence = 0L;
```

그런데.. 컨트롤러와 서비스는 어떻게 연결하지?

- 컨트롤러의 GET, POST 매핑과 서비스와 리포지토리도 만들어봤으나, 정작 컨트롤러와 서비스. 이 둘의 연결고리는 어떻게 만들어야 하는지 모름.
- 구글링 검색 결과, 서비스 부분에서 리포지토리를 가져올 때처럼 똑같이 하면 된다는 것을 알게 됨.



컨트롤러에서 서비스 객체를 활용

- 컨트롤러 객체에 서비스 객체를 호출하여 메서드 사용.
- 인터페이스 없이 서비스 클래스로만 구현했지만, 단순한 기능이기 때문에 문제 없을 것이라 생각.

```
@PostMapping("/join")
public JoinFrame join(@RequestBody JoinFrame data){
    log.info("name={}, email={}, password={}, password confirm={}",
            data.getName(), data.getEmail(), data.getPassword(), data.getPasswordConfirm());
    Member member = new Member();
    member.setName(data.getName());
    member.setEmail(data.getEmail());
    member.setPassword(data.getPassword());
    memberService.join(member);
    return data;
}
```

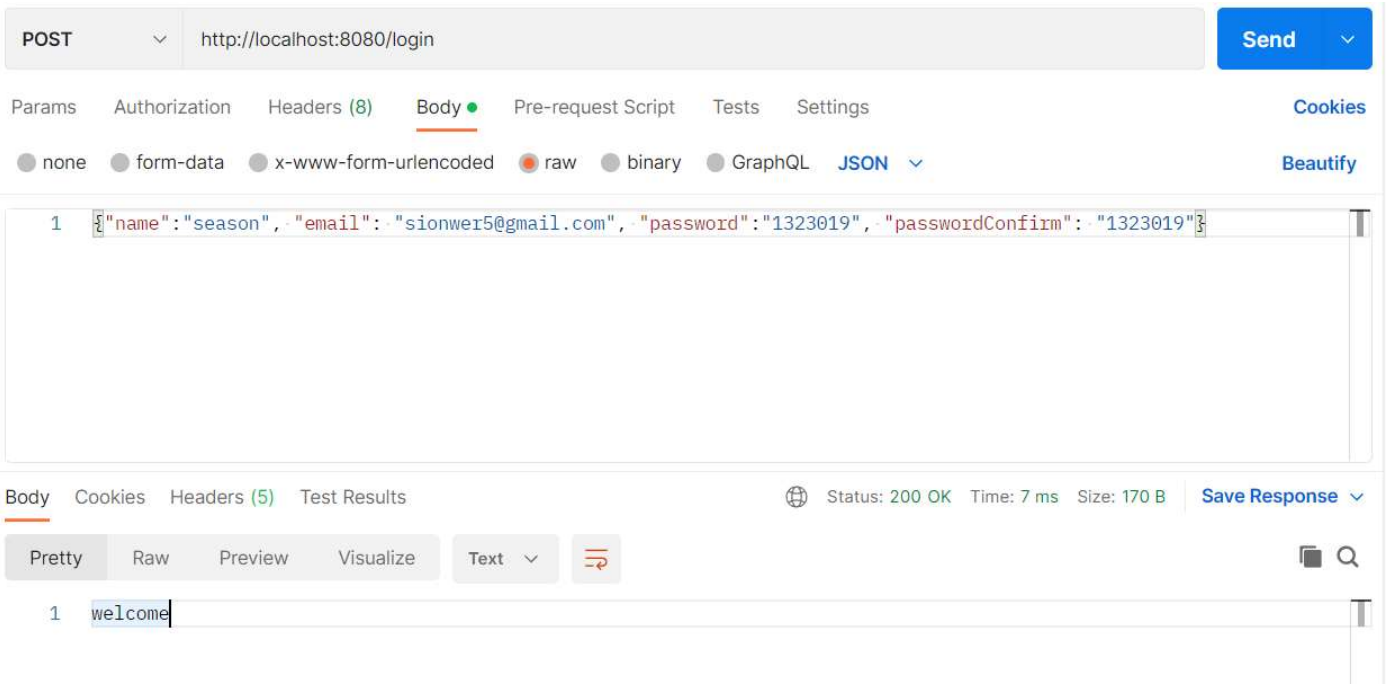
로그인 컨트롤러에서 redirect

- 로그인이 알맞게 되면 (id와 패스워드가 일치하면) 홈으로 리다이렉트
- 로그인이 틀렸으면(id나 패스워드가 틀렸으면) 틀렸다는 경고 문구와 함께 다시 login 화면으로 리다이렉트

```
@PostMapping("/login")
public String login(@RequestBody LoginFrame data, RedirectAttributes redirectAttributes) {
    log.info("email={}, password={}",
            data.getEmail(), data.getPassword());
    if(!memberService.login(data.getEmail(), data.getPassword())){
        return "redirect:/login/error";
    }
    return "redirect:/";
}
```

로그인 컨트롤러에서 redirect

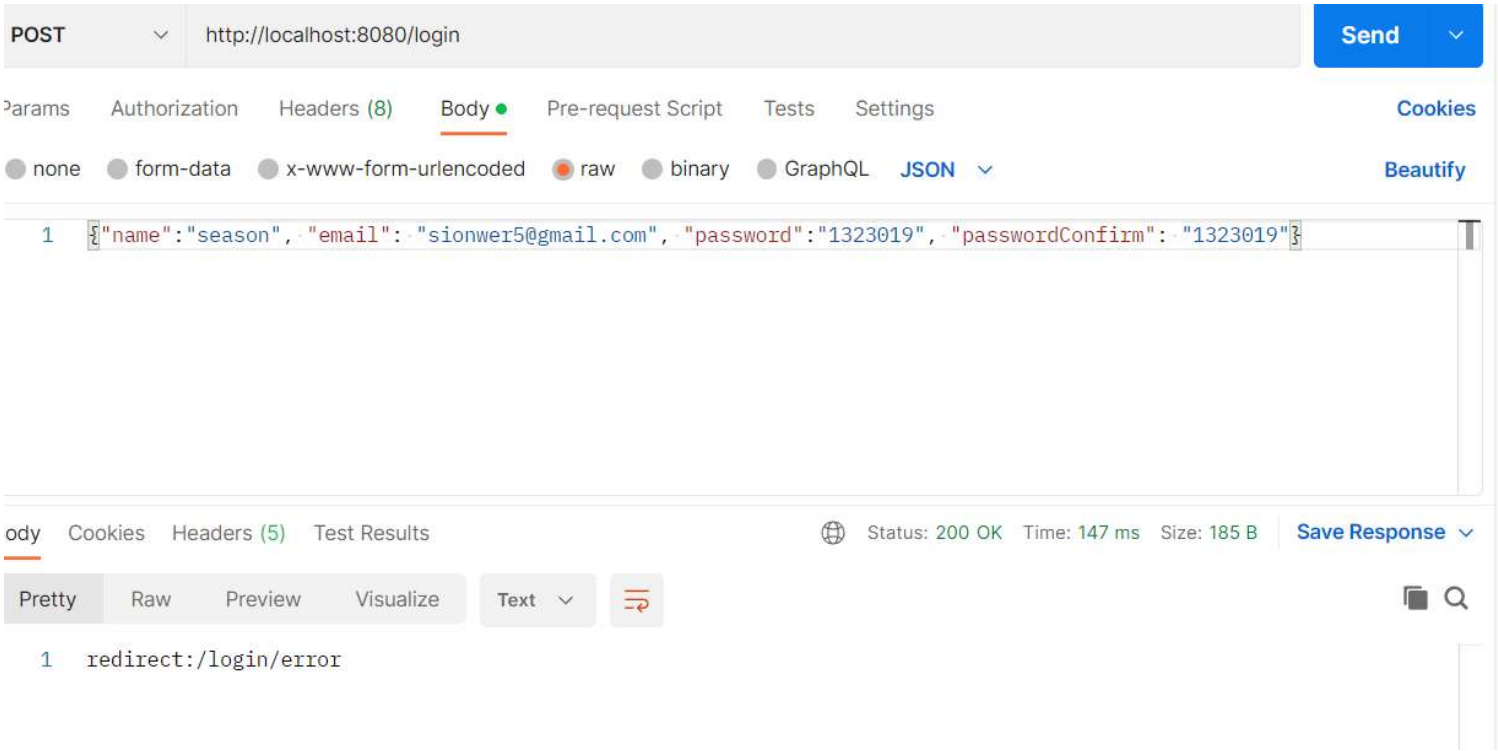
- 리다이렉트를 사용하기 위해서는 RESTController가 아닌 일반 controller를 사용해야 하고, 각 컨트롤러 메서드의 인자로 RedirectAttribute 객체를 가져와야 한다.
- 로그인에 성공해서 홈 화면으로 리다이렉트 됨. 홈화면에 나오는 “welcome”




```java
@RestController
public class HomeController {

    @GetMapping("/")
    public String home(){
        return "welcome";
    }
}
```

RestController와 redirect

- RestController를 써버리면 아래와 같이 나온다. Redirect 기능을 못 쓰고, 리턴문에 나와있는 문자 그대로 렌더링함.



RestController와 redirect

- Redirect가 필요한 메서드를 제외한 나머지 메서드들은 REST API방식을 사용하는데, 모든 메서드에 @ResponseBody를 붙이는 건 번거로운 일이다.
- @RestController를 사용하면서도 redirect를 하는 방법이 있다. 그건 바로 HttpServletResponse 클래스를 사용하면 된다. (스프링이 활용하는 서블릿 객체를 사용하면 @RestController에 종속받지 않고, redirect 기능을 사용할 수 있다.)
- HttpServletResponse 클래스에는 sendRedirect()라는 메서드를 사용하면 redirec 시켜준다. 해당 메서드의 인자에는 String형태로 url을 넣어주면된다.

```
@GetMapping("/google")
public void oauthLogin(HttpServletResponse response){
    String redirect_uri="http://www.google.com";
    response.sendRedirect(redirect_uri);
}
```

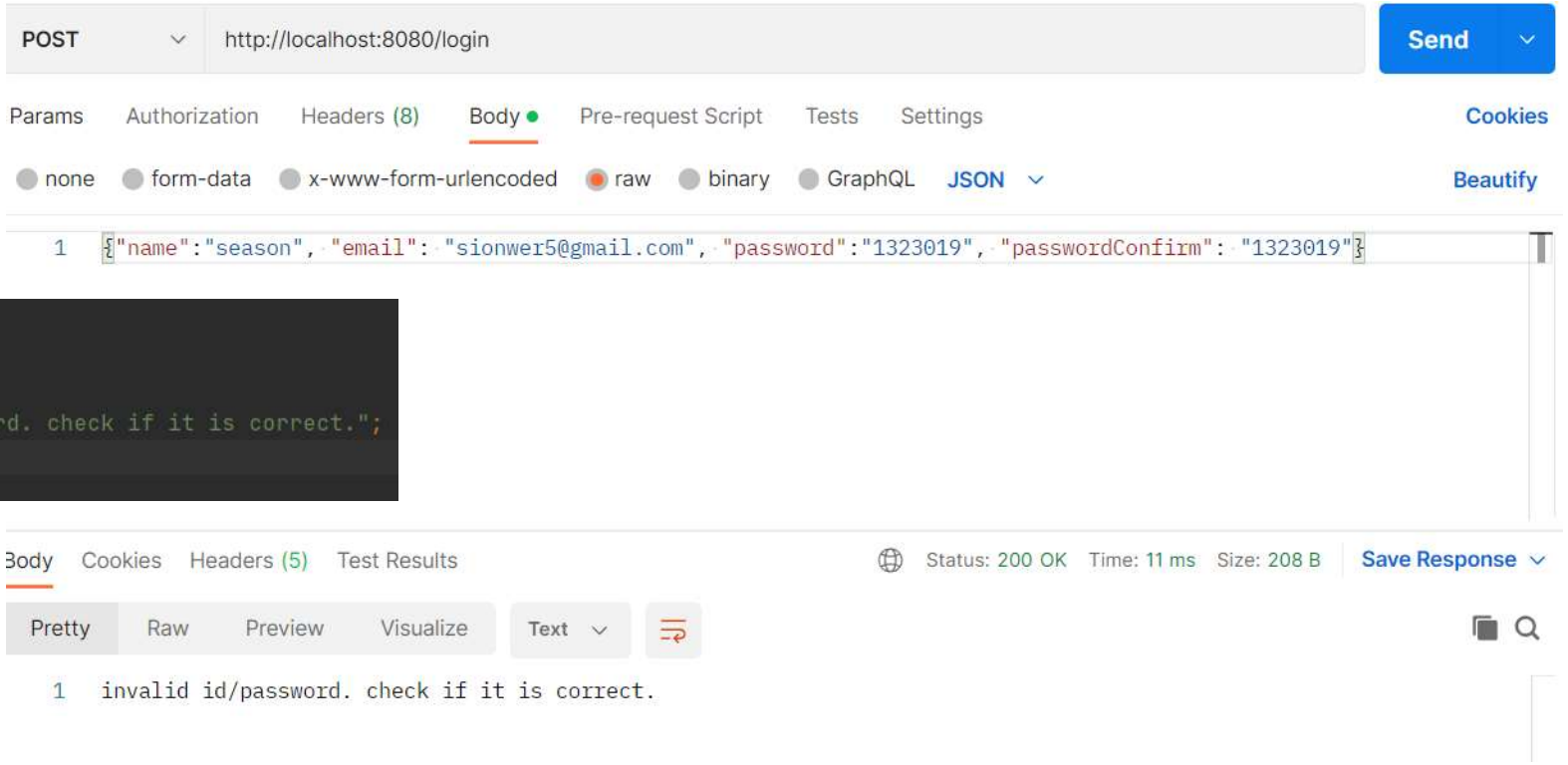
HttpServletResponse

- HttpServletResponse를 사용하여 redirect 구현
- Return 문이 필요 없다.

```
@PostMapping("/login")
public void login(@RequestBody LoginFrame data, HttpServletResponse response) throws IOException {
    log.info("email={}, password={}",
            data.getEmail(), data.getPassword());
    if(!memberService.login(data.getEmail(), data.getPassword())){
        response.sendRedirect( location: "/login");
    }else{
        response.sendRedirect( location: "/");
    }
}
```

생성 안 된 계정으로 로그인

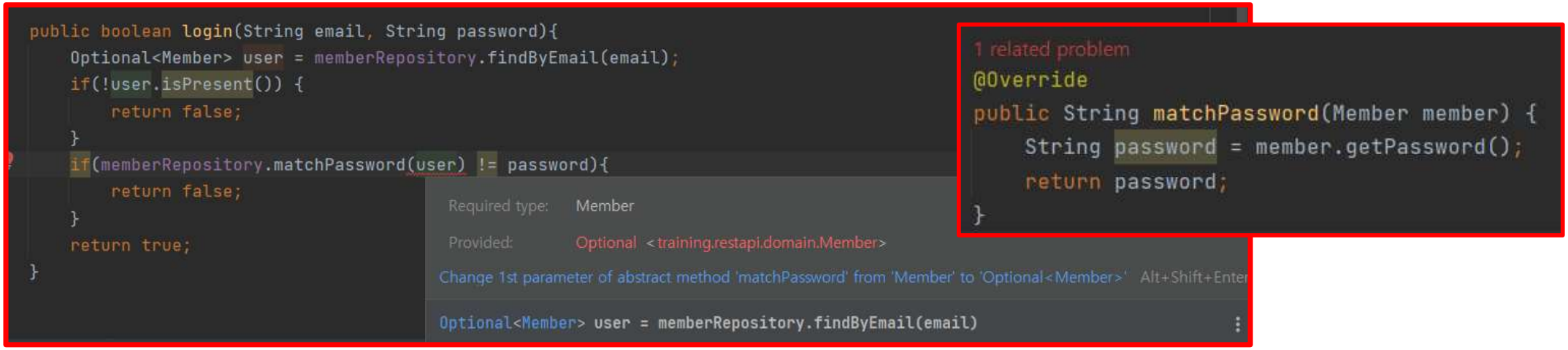
- Join을 하지 않고(리포지토리에 저장하지 않고) 로그인 하려고 하면 다음과 같이 login/error로 리다이렉트됨




```java
@GetMapping("/login/error")
@ResponseBody
public String loginError(){
    return "invalid id/password. check if it is correct.";
}
```

패스워드 매칭 ???

- Service에서 login 메서드는 아래와 같이 해당 Email을 사용하는 유저가 있는지 검사한 후, 입력된 패스워드까지 검사한다.
- 패스워드 매칭에서 Optional 객체를 사용하면서 문제가 발생.
- Optional 객체를 어떻게 일반 객체처럼 사용할 수 있을까



Optional 객체에서 Member객체 꺼내기

- Optional 자체가 특정 타입의 객체의 Wrapper 클래스이므로 그 포장을 벗겨내면 원래의 객체를 얻어낼 수 있다.
- Optional.get()을 하면 Member 객체를 꺼내 쓸 수 있음.

```
public boolean login(String email, String password){
    Optional<Member> user = memberRepository.findByEmail(email);
    if(!user.isPresent()) {
        return false;
    }
    Member member = user.get();
    if(memberRepository.matchPassword(member) != password){
        return false;
    }
    return true;
}
```

- User.isPresent()로 존재 유무를 확인했으므로 바로 .get()을 사용해도 무관하다.

Wrapper Class(=Reference Type)

- 기본 데이터 타입인 primitive type을 객체 형태로 포장해주는 클래스
- 프로그램에 따라 데이터 타입을 객체로 취급해야하는 경우 사용한다.
- Boxing: 기본 타입의 값을 포장 타입으로 바꿔주는 것
- UnBoxing: 포장 타입에서 기본 타입 값을 가져오는 것

```
Integer num = new Integer(17); // 박싱
int n = num.intValue(); //언박싱
```

- Optional타입에서 .get()메서드를 사용하면 기존 타입 데이터를 Unboxing할 수 있다.

Wrapper Class(=Reference Type)

- Wrapper Class끼리 비교할 때는 동등연산자(==)가 아닌 .equals() 메서드를 사용해야한다.
- 동등연산자를 사용할 경우, 객체의 값이 아니라 주소값을 비교하게 됨.
- Wrapper class도 객체의 일종이므로 동등연산자를 사용하면 주소값 비교가 일어남.

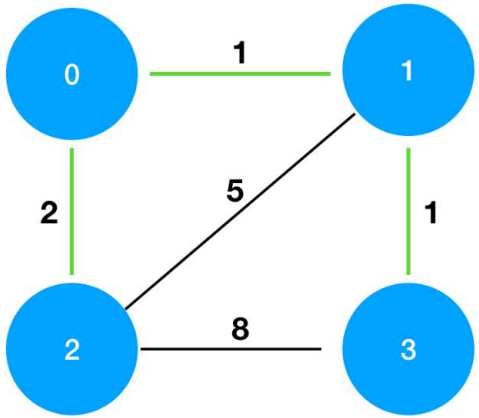
isPresent() vs ifPresent()

- 둘 다 존재 유무를 확인하지만, 두 메서드의 반환 값이 다르다.
- IsPresent()는 Boolean 값이 반환된다.
- ifPresent()는 아무 값도 반환되지 않고, 객체 값이 존재할 경우 괄호 안의 메서드가 실행된다.

크루스칼 알고리즘
Union-find 알고리즘

그리디 알고리즘 문제

- [프로그래머스] 섬연결하기
- n개의 섬 사이에 다리를 건설하는 비용(costs)이 주어질 때, 최소의 비용으로 모든 섬이 서로 통행 가능하도록 만들 때 필요한 최소 비용을 return하기



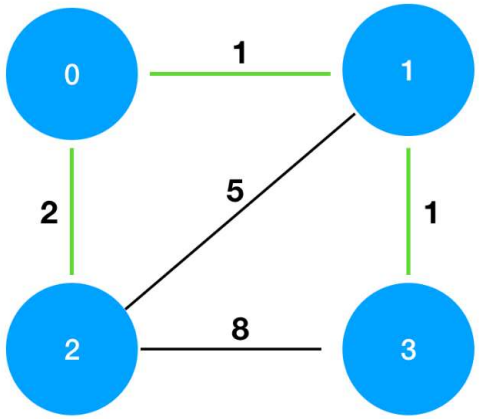
- 예시와 같은 경우 녹색 선이 최소 비용 다리 건설.

입출력 예

n	costs	return
4	[[0,1,1],[0,2,2],[1,2,5],[1,3,1],[2,3,8]]	4

Minimum Spanning Tree 최소 신장 트리

- 배열 X, 스택/큐 X, DFS/BFX X, DP X
- 아무리 방법을 생각해도 도저히 해법이 떠오르지 않음.
- 알고 보니 MST(Minimum Spanning Tree)문제



- MST문제는 크루스칼 알고리즘 또는 프림 알고리즘으로 풀 수 있다.
- 크루스칼 알고리즘을 사용하기 위해서 Union-find 알고리즘도 알아야 한다.

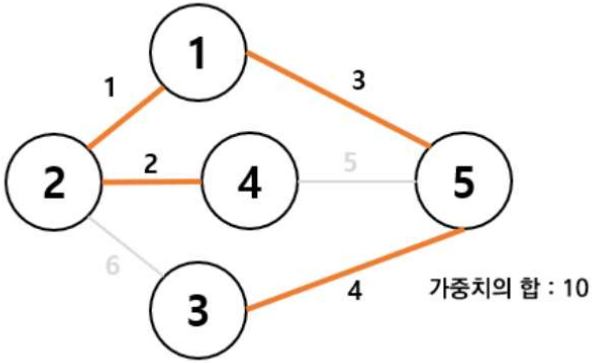
크루스칼 알고리즘

- 모든 정점을 가장 적은 비용의 간선으로 연결하기 위해서 사용.
- 최소신장트리를 구하는 알고리즘이다.
- 신장트리란?

=> 그래프에서 모든 정점을 포함하고, 정점 간 서로 연결되어 사이클을 형성하지 않음.

- 최소신장트리란?

=>모든 간선들의 가중치의 합이 최소가 되는 신장트리



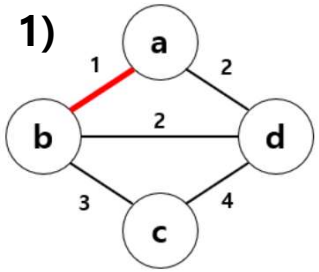
크루스칼 알고리즘

- 알고리즘 푸는 원리

1) 그래프 간선(link)들을 기준으로 배열을 오름차순으로 정렬한 뒤,

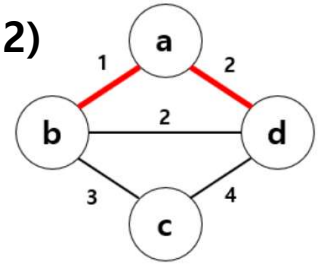
2) 사이클을 형성하지 않는 선에서 정렬된 순서대로 간선을 가져온다. (**Union-find**)

1)



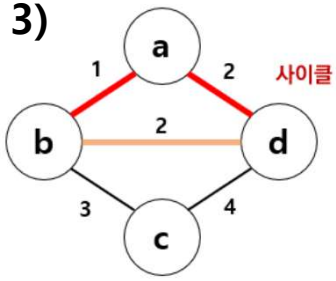
a-b	a-d	b-d	b-c	c-d
1	2	2	3	4

2)



a-b	a-d	b-d	b-c	c-d
1	2	2	3	4

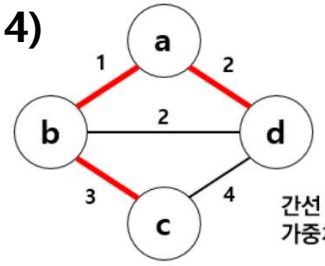
3)



사이클 형성

a-b	a-d	b-d	b-c	c-d
1	2	2	3	4

4)



a-b	a-d	b-d	b-c	c-d
1	2	2	3	4

간선 (n-1)개 (4-1=3개) 선택 완료 ->
가중치 합 6인 최소 신장 트리 구하기 종료

Union-find 알고리즘

- 두 개의 노드의 부모 노드를 확인하여 현재 같은 집합에 속하는지 확인하는 알고리즘.
- 사이클을 형성하는지 아닌지 판단하기 위해서 이 알고리즘을 사용한다
- Union-find란?

⇒ 서로소 집합을 표현하기 위한 자료구조

⇒ Union: 서로 다른 집합을 병합

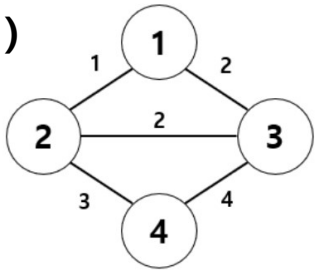
⇒ Find: 원소가 어떤 집합에 포함되어있는지 찾음

Union-find 알고리즘

- 부모 노드 배열을 사용하여 각 노드를 연결 짓는다.

1) 초기 셋팅은 각 노드 값을 부모 노드 값으로
2) for문 돌리면서 부모 노드 비교 (main)
3) 두 노드의 루트 노드 값을 찾아온다. (find)
4) 루트 노드 값이 같으면 패스
5) 루트 노드 값이 다르면, 더 작은 값으로 루트 노드 값 단일화 (union)

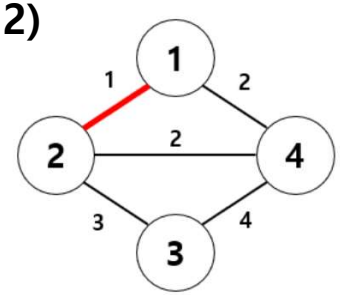
1)



parent 배열

1	2	3	4
1	2	3	4

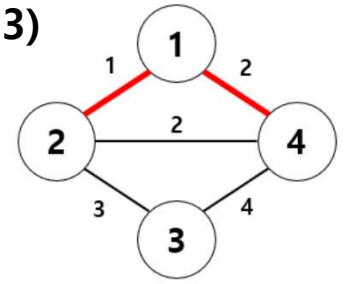
2)



1	2	3	4
1	1	3	4

2의 부모 정점은 1이다

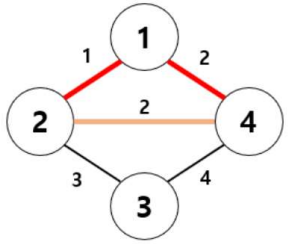
3)



1	2	3	4
1	1	3	1

4의 부모 정점은 1이다

4)



1	2	3	4
1	1	3	1

2와 4를 연결하려 했더니,
둘의 부모가 1로 동일하므로 사이클을 형성한다

Union-find 알고리즘 구현

- Find에서 주어진 노드의 루트 노드를 반환한다.
- Union에서 Find()함수를 호출해서 두 노드의 루트 노드를 각각 가져와서 값을 비교해서 같으면 그냥 패스, 다르면 더 작은 루트 값을 두 부모 노드에 지정해줌.
- Main 함수에서 parent 배열 만들고 union 함수 호출하여 병합 실행

```
def find(parent, node):  # 루트 노드를 찾아준다.
    if parent[node] == node:
        return node
    parent[node] = find(parent, parent[node])
    return parent[node]


def union(parent, a, b): # 두 노드를 합친다.
    a = find(parent, a)
    b = find(parent, b)
    if a == b:
        return False
    if a <= b:
        parent[b] = a
    else:
        parent[a] = b
    return True

def main():
    parent = [i for i in range(10)]

    union(parent, 1, 2)
    union(parent, 2, 3)
    union(parent, 4, 5)
    union(parent, 5, 6)

    print(union(parent, 1, 3))
```

```
================== RESTART: D:/Algorithm/method/union_find.py
False
>>>
```

크루스칼 알고리즘에서 Union-find 활용

- 크루스칼 알고리즘을 다시 정리하면..

0) 간선을 구성하는 노드와 간선의 가중치로 구성된 배열이 주어진다.

1) 간선의 가중치를 기준으로 오름차순으로 정렬한다. (sort)

2) 각 간선의 노드 별로 루트 노드를 가져온다.(find) – (union 내 함수)

3) 배열의 간선 별로 각 노드가 같은 그룹에 있는지 판별한다 (union)

3-1) 같은 루트 노드를 가지면 패스

3-2) 다른 루트 노드를 가지면, 같은 루트 노드를 가지도록 저장

4) 모든 배열에 대해서 3번 과정 반복