photogram

포토그램

나만의 SNS 만들기

#이젠T아카데미
#4조 프로젝트

2022.02.25

INDEX

1. PROJECT OVERVIEW

팀원 소개
개발 환경
제작기간
요구사항 정의

2. DATABASE 설계

ERD
테이블 명세서

3. FRONT-END

MOCK-UP
화면 구현 소개

4. BACK-END

CODE 분석

1 PROJECT OVERVIEW

#팀원 소개 #개발환경 #프로젝트 기간 #요구사항 정의

#팀원 소개

안현욱(조장)

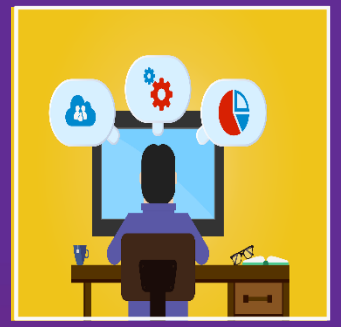
#회원가입

김도원

#프로필

김창현

#좋아요,댓글



김희윤

#구독

박현철

#스토리 페이지

서나리

#회원가입

한종현

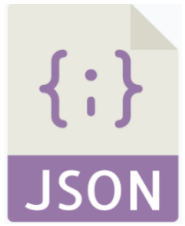
#로그인

개발환경

FRONT-END

Bootstrap

BACK-END



SERVER

1 PROJECT OVERVIEW

요구사항 정의

요구사항ID	1차분류	2차분류	3차분류	요구사항 상세내용	담당자
1	로그인 페이지	로그인	로그인	로그인	한종현
2			페이스북 로그인	페이스북 ID와 연동하여 로그인	한종현
3		회원가입	회원가입	아이디(최대30자),비밀번호,이메일,이름 필수 입력	안현욱/서나리
4	메인 페이지	게시글 조회	게시글 작성자 조회	게시글 작성자의 ID가 보이게	박현철
5			게시글 내용 조회	게시글 3개까지 보인 후 페이지 스크롤 페이징 기능 구현	박현철
6			게시글 이미지 조회	이미지 용량은 최대 2MB까지	박현철
7		댓글	댓글 쓰기	공백 입력 시 댓글입력 불가능하게 알림창 띄움	김창현
8			댓글 조회	댓글 조회	김창현
9			댓글 삭제	현재 로그인한 사용자의 댓글만 삭제가능	김창현
10		좋아요	좋아요 누르기	좋아요 표시(붉은 하트)	김창현
11			좋아요 누르기 취소	좋아요 취소 표시(빈 하트)	김창현
12			좋아요 개수 조회	좋아요 개수 만큼 likes에 표시	김창현

1 PROJECT OVERVIEW

요구사항 정의

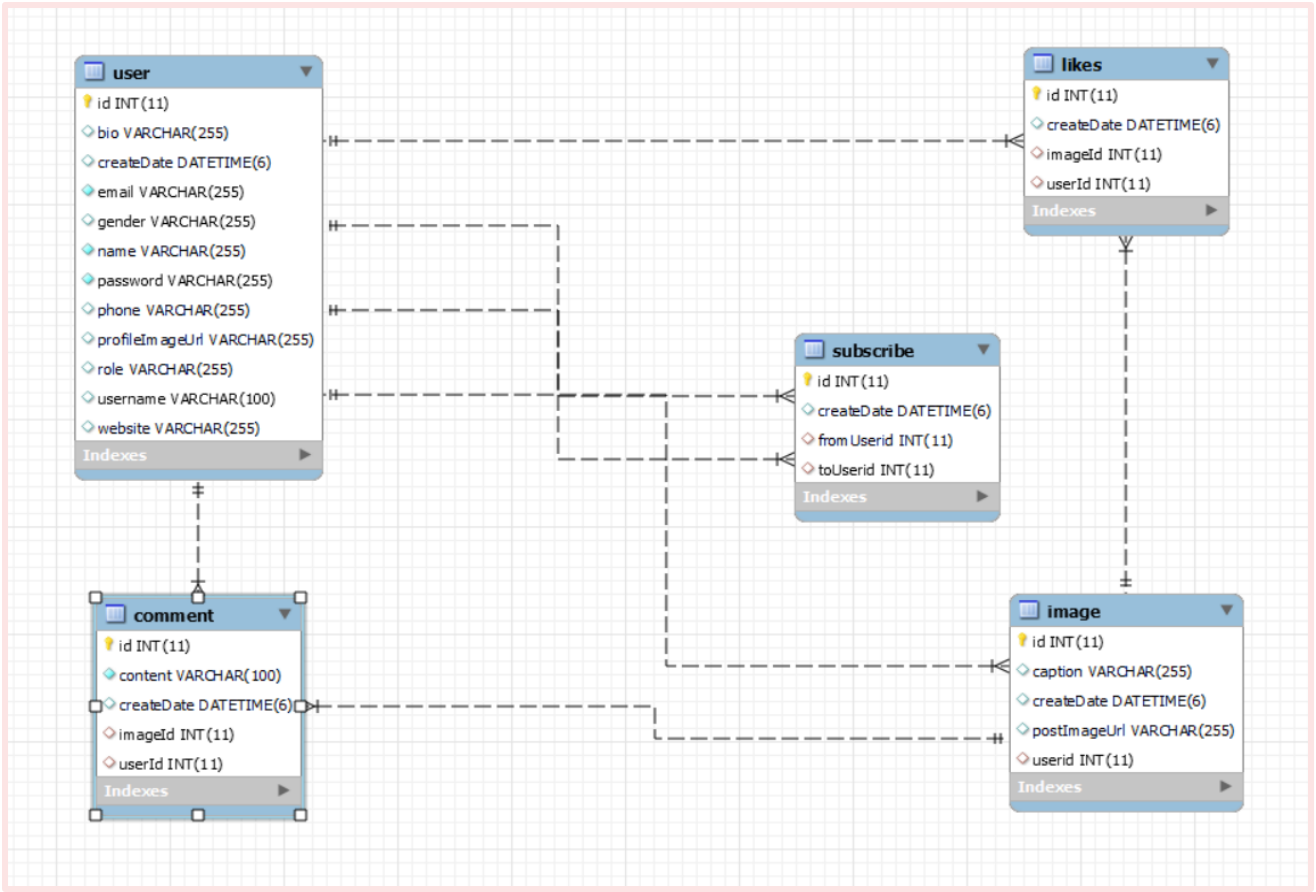
요구사항ID	1차분류	2차분류	3차분류	요구사항 상세내용	담당자
13	인기 페이지	인기 사진 보기	사진 클릭 시 유저 페이지로 이동	사진 클릭시 해당 사진 업로더 페이지로 이동	박현철
14	마이 페이지	프로필 사진 바꾸기	프로필 사진 수정	프로필 사진 수정 기능 구현	김도원
15				취소 선택시 기존 이미지로 프로필 사진 저장	김도원
17		사진등록	사진 업로드	파일 선택 버튼 클릭시 업로드 할 사진 선택하여 미리보기 구현 (이미지 용량은 최대 2MB까지)	안현욱/서나리
18			사진 설명	필요시 사진에 대한 코멘트 작성후 업로드 버튼 클릭하면 사진 등록	안현욱/서나리
20		회원정보 변경	개인정보 수정	이름, 유저네임, 웹사이트, 소개, 이메일, 전화번호, 성별 수정 가능	안현욱/서나리
21		로그아웃	로그아웃	로그아웃 후 메인페이지로 이동	한종현
22		게시물 수	게시물 수 조회	사진 게시물 수 확인 가능	김희윤
23		구독정보	구독정보 조회	내가 구독하고 있는 정보 확인 가능	김희윤
24			구독하기	구독하기 기능 구현	김희윤
25			구독취소	구독 취소 가능	김희윤
26		업로드한 사진 좋아요 확인	좋아요 개수 조회	내가 등록한 사진의 좋아요 수 확인 가능	김도원

❷ DATABASE 설계

#ERD #테이블명세서 #MariaDB

2 DATABASE 설계

ERD



테이블 명세서 -1

TableName	comment							
Description	댓글 테이블							
No	table_comment	column_name	data_type	column_type	column_key	is_nullable	column_default	extra
1	댓글 아이디	id	int	int(11)	PRI	NO	ㅋN	auto_increment
2	내용	content	varchar	varchar(100)		NO	ㅋN	
3	생성날짜	createDate	datetime	datetime(6)		YES	NULL	
4	이미지 아이디	imageId	int	int(11)	MUL	YES	NULL	
5	유저 아이디	usrId	int	int(11)	MUL	YES	NULL	

테이블 명세서 -2

TableName	Image							
Description	사진 테이블							
No	table_comment	column_name	data_type	column_type	column_key	is_nullable	column_default	extra
1	이미지 아이디	id	int	int(11)	PRI	NO	₩N	auto_increment
2	이미지 내용	caption	varchar	varchar(255)		YES	NULL	
3	생성날짜	createDate	datetime	datetime(6)		YES	NULL	
4	이미지 주소	postImageUrl	varchar	varchar(255)		YES	NULL	
5	유저 아이디	userId	int	int(11)	MUL	YES	NULL	

테이블 명세서 -3

TableName	Likes							
Description	좋아요 테이블							
No	table_comment	column_name	data_type	column_type	column_key	is_nullable	column_default	extra
1	좋아요 아이디	id	int	int(11)	PRI	NO	₩N	auto_increment
2	생성 날짜	createDate	datetime	datetime(6)		YES	NULL	
3	이미지 아이디	imageId	int	int(11)	MUL	YES	NULL	
4	유저 아이디	userId	int	int(11)	MUL	YES	NULL	

테이블 명세서 -4

TableName		Subscribe						
Description		구독 테이블						
No	table_comment	column_name	data_type	column_type	column_key	is_nullable	column_default	extra
1	아이디	id	int	int(11)	PRI	NO	₩N	auto_increment
2	생성날짜	createDate	datetime	datetime(6)		YES	NULL	
3	구독신청유저	fromUserId	int	int(11)	MUL	YES	NULL	
4	구독받는유저	ToUserId	int	int(11)	MUL	YES	NULL	

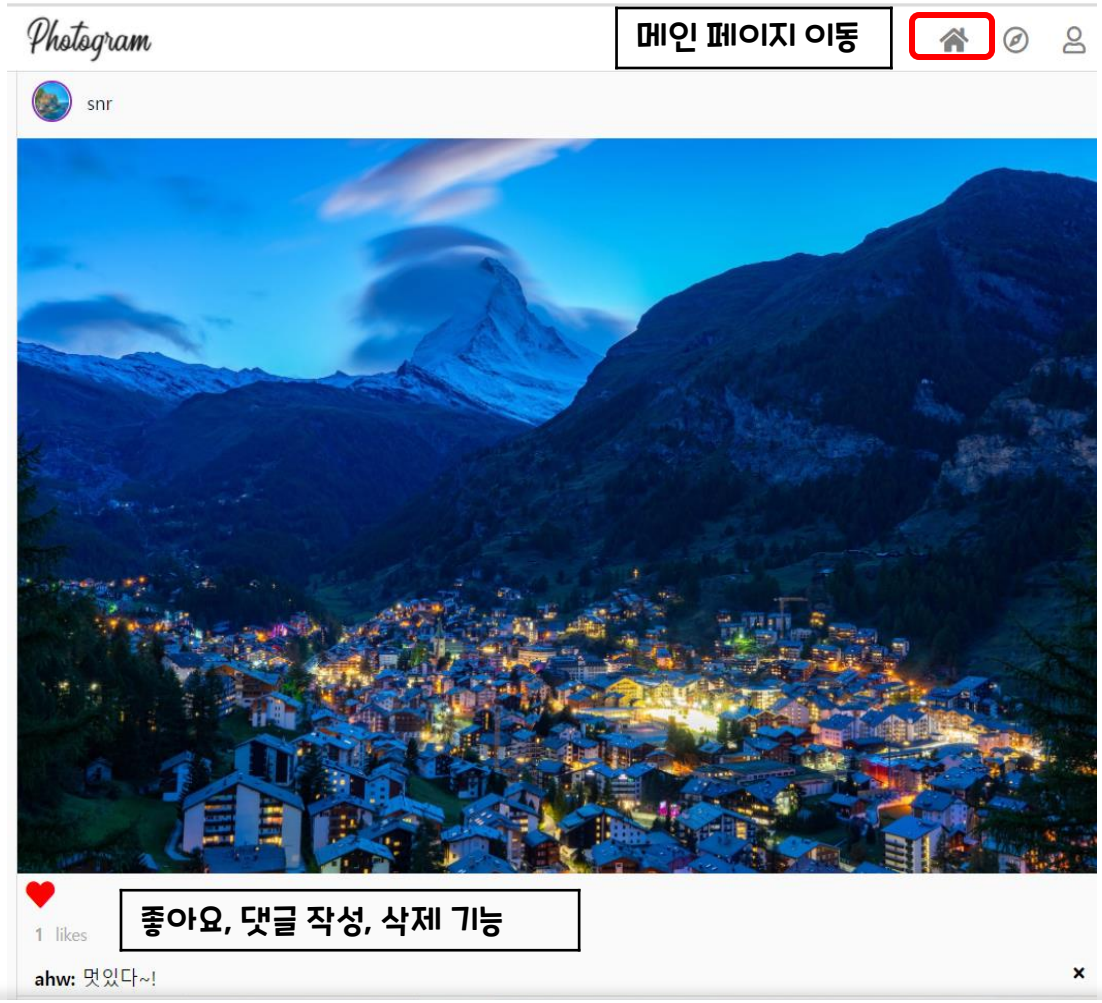
테이블 명세서 -5

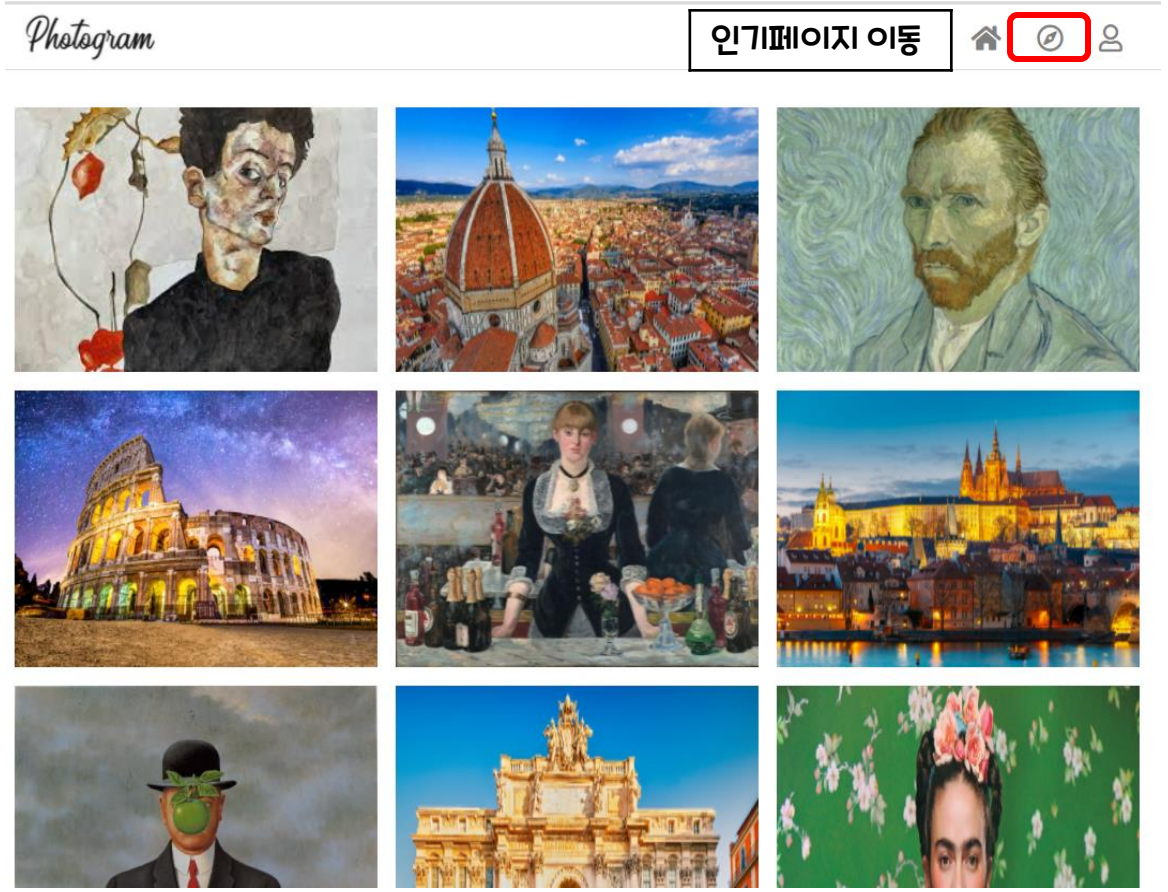
TableName	User							
Description	유저 테이블							
No	table_comment	column_name	data_type	column_type	column_key	is_nullable	column_default	extra
1	아이디	id	int	int(11)	PRI	NO	NULL	auto_increment
2	자기소개	bio	varchar	varchar(255)		YES	NULL	
3	생성날짜	createDate	datetime	datetime(6)		YES	NULL	
4	이메일	email	varchar	varchar(255)		NO	NULL	
5	성별	gender	varchar	varchar(255)		YES	NULL	
6	이름	name	varchar	varchar(255)		NO	NULL	
7	비밀번호	password	varchar	varchar(255)		NO	NULL	
8	휴대폰 번호	phone	varchar	varchar(255)		YES	NULL	
9	프로필 사진	profileImageUrl	varchar	varchar(255)		YES	NULL	
10	권한	role	varchar	varchar(255)		YES	NULL	
11	유저네임	username	varchar	varchar(100)	UNI	YES	NULL	
12	웹사이트	website	varchar	varchar(255)		YES	NULL	

3 FRONT-END

#MOCK-UP #화면 구현 소개 #BootStrap

메인 페이지(스토리)


Photogram
메인 페이지 이동
snr
1 likes
좋아요, 댓글 작성, 삭제 기능
ahw: 멋있다~!


Photogram
인기페이지 이동

프로필 페이지

Photogram

나리 사진등록

게시물 10 구독정보 1

프로필페이지 이동

나리 사진등록

회원 정보 수정

게시물 10 구독정보 1

Photogram

snr

이름 나리

유저네임 snr

패스워드 패스워드

웹사이트 웹 사이트

소개

개인정보

비즈니스나 반려동물 등에 사용된 계정인 경우에도 회원님의 개인 정보를 입력하세요. 공개 프로필에는 포함되지 않습니다.

이메일 snr@gamil.com

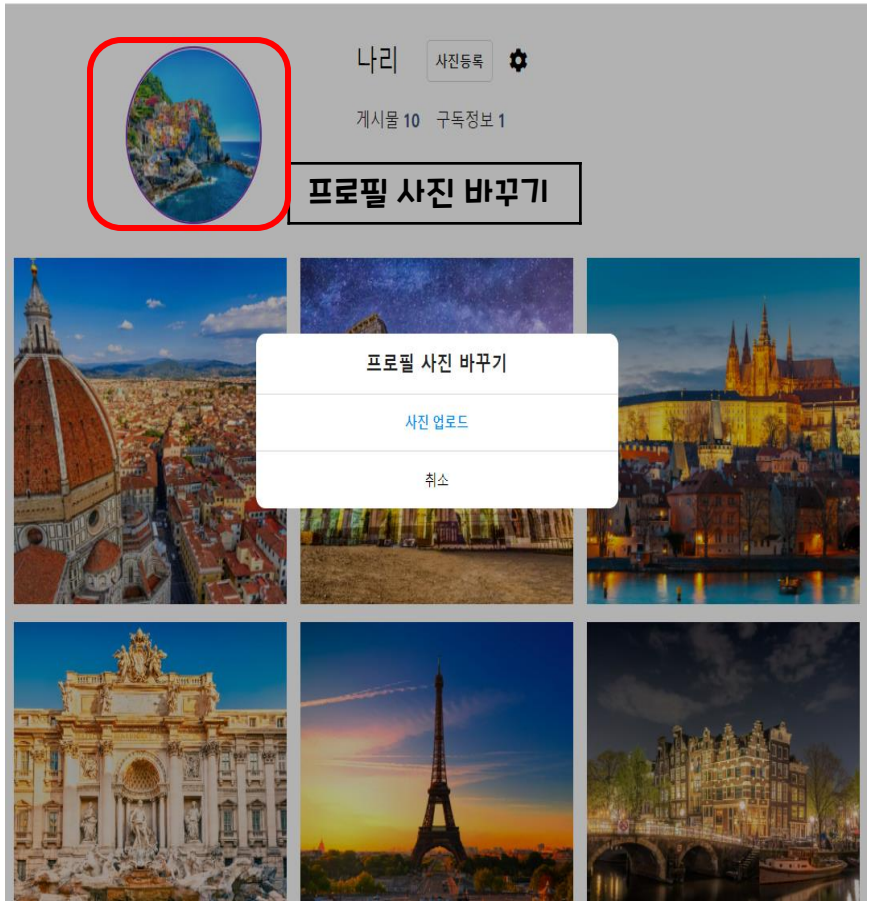
전화번호 전화번호

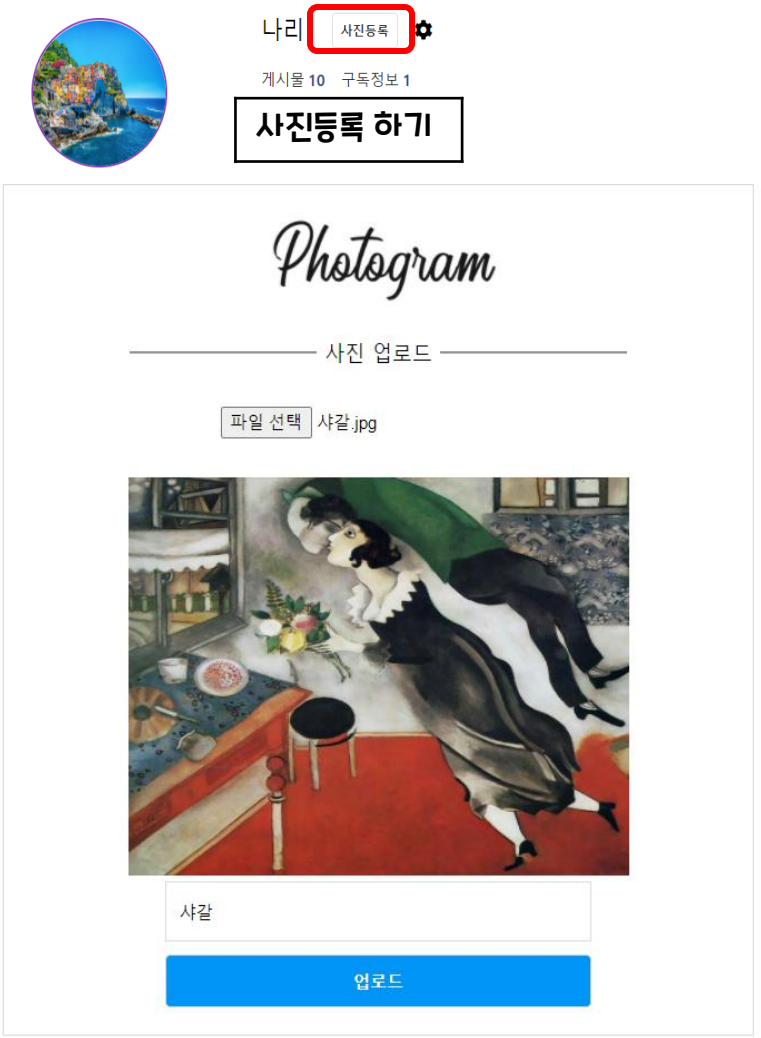
성별

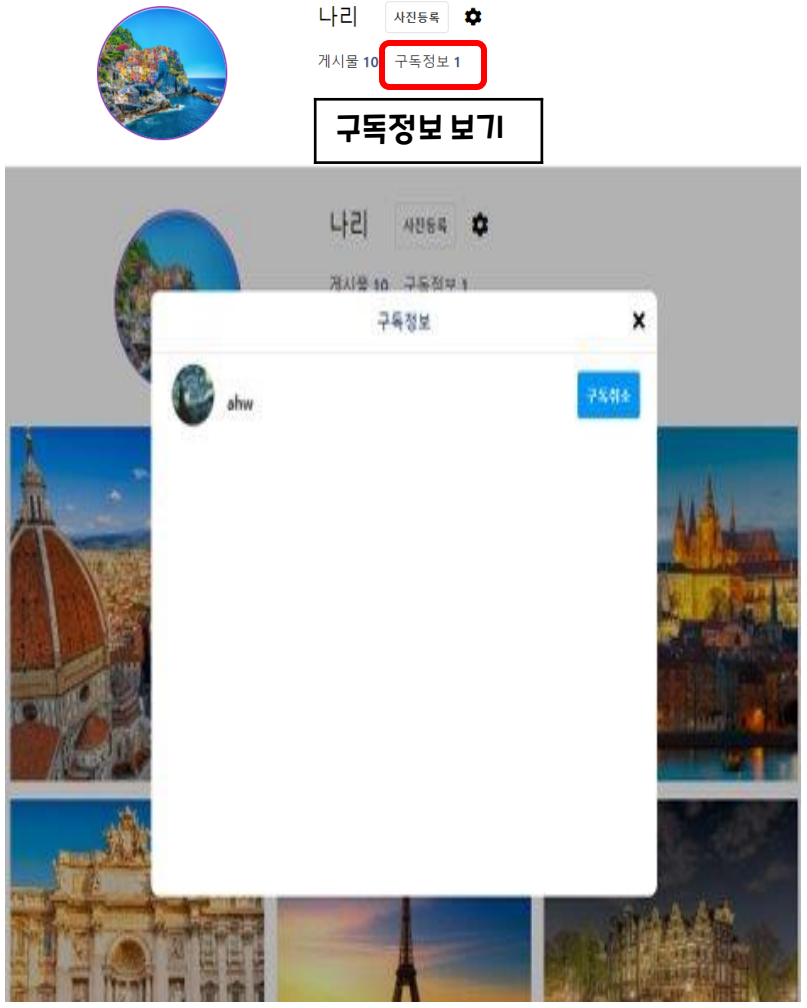
제출

3 FRONT-END

프로필 페이지







4 BACK-END

#코드 분석 #데이터 흐름 #SPRING BOOT

회원가입

JSP – method=POST 방식

```jsp
<div class="container">
<main class="loginMain">
  <!--회원가입섹션-->
  <section class="login">
    <article class="login__form__container">

      <!--회원가입 폼-->
      <div class="login__form">
        <!--로고-->
        <h1><img src="/images/logo.jpg" alt=""></h1>
        <!--로고end-->

        <!--회원가입 인풋-->
        <form class="login__input" action="/auth/signup" method="post">
          <input type="text" name="username" placeholder="유저네임" required="required" maxlength="30" />
          <input type="password" name="password" placeholder="패스워드" required="required" />
          <input type="email" name="email" placeholder="이메일" required="required" />
          <input type="text" name="name" placeholder="이름" required="required" />
          <button>가입</button>
        </form>
```

Service – 비지니스 로직 수행

```java
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@RequiredArgsConstructor
@Service  //1. IoC 2.트랜잭션 관리
public class AuthService {

  private final UserRepository userRepository;
  private final BCryptPasswordEncoder bCryptPasswordEncoder;

  @Transactional  //여러 개의 DML 명령문을 하나의 논리적인 작업 단위로 묶어서 관리하는 것 // Write(Insert, Update,Delete
  public User 회원가입(User user) { // 이 user는 외부 통신에서 받은 데이터를 담은거
    //회원가입 진행
    String rawPassword = user.getPassword();
    String encPassword = bCryptPasswordEncoder.encode(rawPassword); //암호화된 패스워드
    user.setPassword(encPassword); // 암호화된 패스워드 저장
    user.setRole("ROLE_USER"); // 관리자는 ROLE_ADMIN 으로
    User userEntity = userRepository.save(user);  //save가되면 user 리턴이된다. (디비에 들어간거) //
    return userEntity;  // userEntity는 데이터베이스에 있는 user 데이터를 받아온다.
  }
}
```

View – 회원가입 페이지

JPA – DB 테이블 생성

```java
//JPA - Java Persistnce API(자바로 데이터를 영구적으로 저장(DB)할 수 있는 API를 제공)

@Builder
@AllArgsConstructor  //전체생성자
@NoArgsConstructor  //빈생성자
@Data
@Entity // 디비에 테이블을 생성
public class User {

  @Id // 프라이머리 키
  @GeneratedValue(strategy = GenerationType.IDENTITY) //번호 증가 전략이 데이터베이스를 따라간다
  private int id;

  @Column(length = 100, unique = true) //제약조건 //length 길이제한 //OAuth2 로그인을 위해 잡힌 놀라가
  private String username;

  @Column(nullable = false)  // null 불가능
  private String password;

  @Column(nullable = false)  // null 불가능
  private String name;

  private String website; //웹사이트

  private String bio; // 자기소개

  @Column(nullable = false)  // null 불가능
  private String email;

  private String phone;

  private String gender;
```

DTO – 데이터 주고 받기

```java
//통신할때 필요한데이터를 담아두는게 D
@Data // Getter,Setter  등록
public class SignupDto {

  @Size(min = 2, max = 20)  //
  @NotBlank
  private String username;

  @NotBlank  // 비밀번호가 무조건
  private String password;

  @NotBlank
  private String email;

  @NotBlank
  private String name;


  public User toEntity() {
    return User.builder()
      .username(username)
      .password(password)
      .email(email)
      .name(name)
      .build();
```

Controller– 웹 브라우저 요청 전담

```java
  //회원가입버튼 -> /auth/signup -> /auth/signin
  // 회원가입 버튼 X  //@ResponseBody 컨트롤러리턴때 리턴타입앞에 (String같은애) 붙어 있으면 데이터를 응답한다.
  @PostMapping("/auth/signup")  //@Valid 안에 @NotBlank 와 @MAX 및 @SIZE 조건을 달아서 SignupDto에 오류가 발생하면 오류를  BindingResult getFieldErrors객체에 모아준다
  public  String signup(@Valid SignupDto signupDto, BindingResult bindingResult) {  //key=value(x-www-form-urlencoded)


    log.info(signupDto.toString());
    //User <- SignupDto
    User user = signupDto.toEntity();
    log.info(user.toString());
    User userEntity = authService.회원가입(user);
    //System.out.println(userEntity);

    //로그를 남기는 추적은
    return "auth/signin";

  }

  // 지금 페이지이동하면과 데이터리턴이 같이 있는 경우가 없다
}
```

>> 회원가입 시 입력된 정보를 Controller로 전달, Controller 에서 Service 로직 수행

>> 비밀번호 인코딩 , JPA 적용

회원가입 유효성 검사

ValidationAdvice – 유효성검사

```
@Component  //RestController, Service 모든 것들이 Component를 상속해서 만들어져 있음.
@Aspect
public class ValidationAdvice {

    //@Before = 실행되기전에
    //@After = 실행되고나서

    @Around("execution(* com.cos.photogramstart.web.*Controller.*(..))")
    public Object advice(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {

        //System.out.println("web 컨트롤러==========================");
        Object[] args = proceedingJoinPoint.getArgs();
        for (Object arg : args) {          //함수의 매개변수 접근해서 매개변수가 어떤게 있는지 뽑기
            if(arg instanceof BindingResult) {
                //System.out.println("유효성 검사를 하는 함수입니다.");
                BindingResult bindingResult = (BindingResult)arg;

                /*회원가입 유효성검사 */
                if(bindingResult.hasErrors()) {
                    Map<String, String> errorMap = new HashMap<>();

                    for(FieldError error:bindingResult.getFieldErrors()) {
                        errorMap.put(error.getField(),error.getDefaultMessage());
                        //System.out.println("============================");
                        //System.out.println(error.getDefaultMessage());
                        //System.out.println("============================");
                    }
                    throw new CustomValidationException("유효성검사 실패함",errorMap);  // 위에 조건문으로 예외검사를 한뒤에 errorMap객체에 담긴다 / CustomValidationException가 낚아챈다.
                }
            }

        }
        return proceedingJoinPoint.proceed();
    }
```

ControllerExceptionHandler – 예외처리 낚아채기

```
@RestController  // 낚아채고 응답하는거는 데이터리턴해야하니 이걸써라
@ControllerAdvice  //모든 만들어진 즉 예외처리를 낚아채려면 이걸써라
public class ControllerExceptionHandler {


    //CustomValidationException
    @ExceptionHandler(CustomValidationException.class)  // CustomValidationException에 발동되는 모든함수를  validationException 가 낚아챈다
    public String validationException(CustomValidationException e) { //CustomValidationException e 클래스에 적용


        //CMRespDto , Script 비교
        // 1.클라이언트에게 응답할 때는 Script 응용.
        // 2. Ajax통신 - CMRespDto
        //3. Android통신 - CMRespDto
        if(e.getErrorMap() == null) {
            return Script.back(e.getMessage());
        }else {
            return Script.back(e.getErrorMap().toString());
        }

    }
```

CustomValidationException – 예외처리

```
//하나가 아닌 여러개의 예외처리를 위해 추가
public class CustomValidationException extends RuntimeException{

    // 시리얼 번호는 객체를 구분할 때!  지금 수업에는 중요x
    private static final long serialVersionUID = 1L;

    //private String message;

    private Map<String,String> errorMap;

    public CustomValidationException(String message, Map<String,String> errorMap) {
        super(message);  // 리턴값 getMessage라는 함수가 실행됨 그래서 게터함수 안만들어도되고 부모한테 던지면된다.
        //this.message = message;
        this.errorMap = errorMap;
    }

    public Map<String, String> getErrorMap(){
        return errorMap;
    }

}
```

>> AuthController에서 에러 발생 시 throw new CustomValidationException("유효성 검사 실패함.", errorMap);체크 ControllerExceptionHandler에서 가져와 예외처리 진행

로그인

JSP – method = POST 방식

```
<!--로그인 인풋-->
<form class="login__input" action="/auth/signin" method="POST" >
    <input type="text" name="username" placeholder="유저네임" required="required" />
    <input type="password" name="password" placeholder="비밀번호" required="required" />
    <button>로그인</button>
</form>
<!--로그인 인풋end-->
```

Security – 로그인 설정

```
@Override
protected void configure(HttpSecurity h
    // super 삭제 - 기존 시큐리티가 가지고 있
    http.csrf().disable();
    http.authorizeRequests()
        .antMatchers("/","/user/**","/image
        .anyRequest().permitAll()
        .and()
        .formLogin()
        .loginPage("/auth/signin")  //여기 자
        .loginProcessingUrl("/auth/signin");
```

PrincipalDetails

```
import java.util.ArrayList;

@Data
public class PrincipalDetails implements UserDetails, OAuth2User{

    private static final long serialVersionUID = 1L;

    private User user;
    private Map<String, Object> attributes;

    public PrincipalDetails(User user) {
        this.user = user;
    }

    public PrincipalDetails(User user, Map<String, Object> attributes) {
        this.user = user;
    }
```

PrincipalDetailsService

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

@RequiredArgsConstructor //final 필드를 DI 할때 생성자를 생성
@Service //IoC
public class PrincipalDetailsService implements UserDetailsService{

    private final UserRepository userRepository;
    //1. 패스워드는 알아서 체킹하니까 신경 쓸 필요 없다.
    //2.리턴이 잘되면 자동으로 UserDetails 타입을 세션으로 만든다.
    //클라이언트에서 받아온 post 방식에 정보를 시큐리티 .loginProcessingUrl("/auth/signin") 에 받는다
    // 그리고 받은걸 통해 UserDetailsService가 받아서 처리하는데 PrincipalDetailsService가 상속을받고 덮어쓰여져서 이친구
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {//비밀번

        User userEntity = userRepository.findByUsername(username); //유저네임 찾은거 불러오기

        if(userEntity == null) {
            return null;
        }
        else {
            return new PrincipalDetails(userEntity);  //PrincipalDetails가 세션에 저장이 될때 userEntity을 받았으니 함수를 활용할 수 있다
            //리턴값을 userEntity 원하는 이유는 UserDetails 얘가 리턴되어야하기 때문에 상속받아놓은 PrincipalDetails 에 함수를 넣어서 리턴한다.
        }
    }
```

UserController – 세션정보 확인

```
@GetMapping("/user/{id}/update")  //@AuthenticationPrincipal 세션에 접근하고 싶을 때
public String update(@PathVariable int id, @AuthenticationPrincipal PrincipalDetails principalDetails) {

    //1.추천
    //System.out.println("세션정보" + principalDetails.getUser());



    return "/user/update";
}
```

UserReporitory – 유저 찾기

```
import org.springframework.data.jpa.repository.JpaRepository;

// 어노테이션이 없어도 JpaRepository를 상속하면 IoC 등록이 자동으로 된다.
public interface UserRepository extends JpaRepository<User, Integer>{

    //JPA query method
    //findBy - 규칙키워드
    User findByUsername(String username);
    // 쿼리로가서 유저네임을 찾겠다.

}
```

View – 로그인 페이지

View – 로그인 성공

>> 로그인 시 입력된 정보를 시큐리티로 전달, UserDetailsService에 정보를 받아 로그인을 진행

페이스북 로그인


```
<!-- Oauth 소셜로그인 -->
<div class="login__facebook">
  <button onclick="javascript:location.href='/oauth2/authorization/facebook'">
    <i class="fab fa-facebook-square"></i>
    <span>Facebook으로 로그인</span>
  </button>
</div>
<!-- Oauth 소셜로그인end -->
```

>> 로그인화면에서Facebook으로 로그인 클릭 시 페이스북 로그인 화면으로 전환

페이스북 로그인

```yaml
oauth2:
  client:
    registration:
      facebook:
        client-id: 앱아이디
        client-secret: 앱시크릿코드
        scope:
        - public_profile
        - email
```

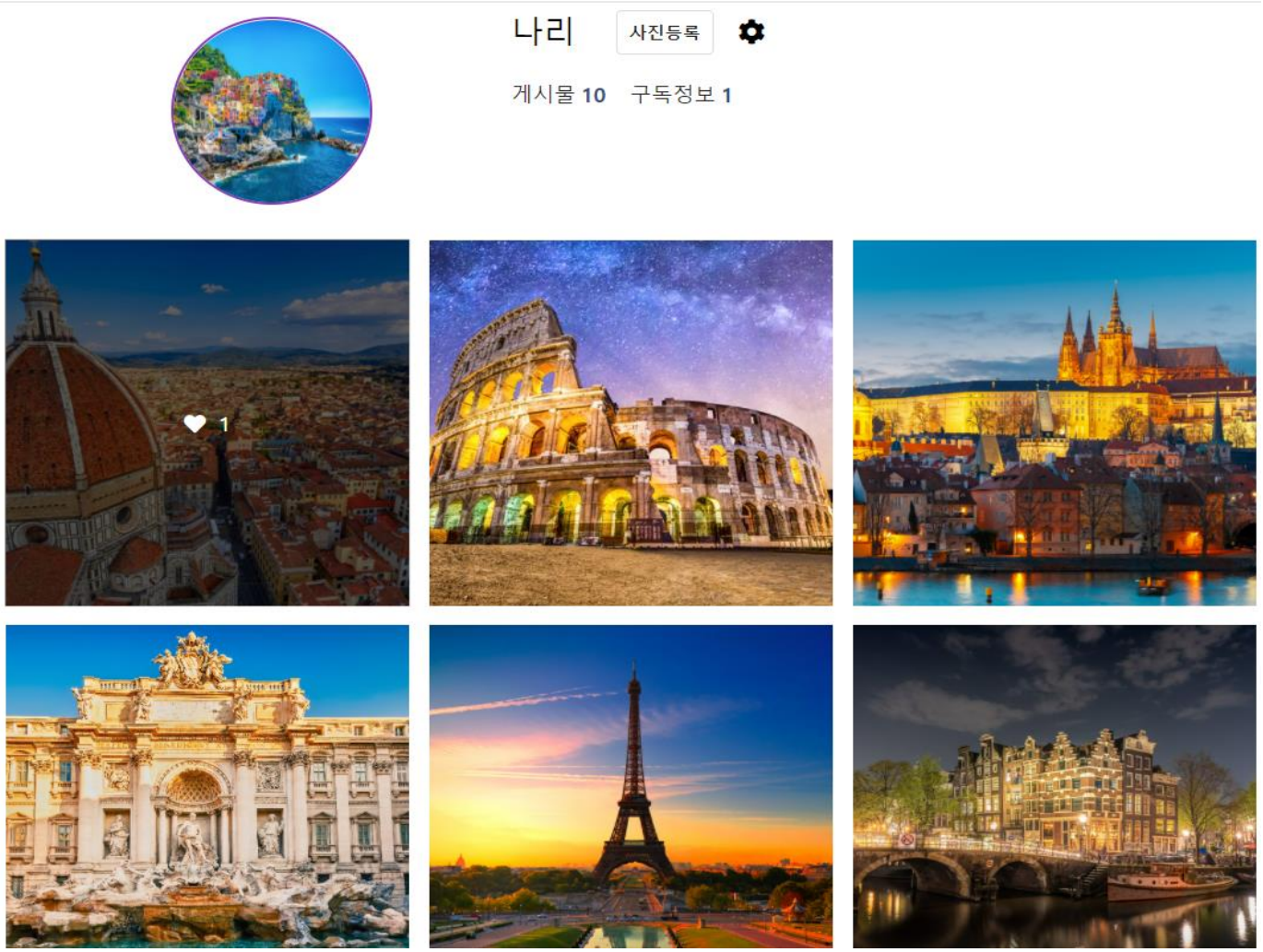
```
.and()
.oauth2Login()  // form 로그인도 하는데, oauth2 로그인도 할거야!
.userInfoEndpoint()  // oauth2 로그인을 하면 최종응답을 회원정보를 바로 받을 수 있다.
.userService(oAuth2DetailsService);
```

>> 페이스북 설정과 시큐리티 설정

```java
@Service
public class OAuth2DetailsService extends DefaultOAuth2UserService {

    private final UserRepository userRepository;

    @Override
    public OAuth2User loadUser(OAuth2UserRequest userRequest) throws OAuth2AuthenticationException {
        //System.out.println("OAuth2 서비스 탐");
        OAuth2User oAuth2User = super.loadUser(userRequest);
        //System.out.println(oAuth2User.getAttributes());

        Map<String, Object> userInfo = oAuth2User.getAttributes();

        String username = "facebook_" + (String)userInfo.get("id");
        String password = new BCryptPasswordEncoder().encode(UUID.randomUUID().toString());
        String email = (String)userInfo.get("email");
        String name = (String)userInfo.get("name");

        User userEntity = userRepository.findByUsername(username);
        if(userEntity == null) {  // 페이스북 최초 로그인
            User user = User.builder()
                    .username(username)
                    .password(password)
                    .email(email)
                    .name(name)
                    .role("ROLE_USER")
                    .build();

            return new PrincipalDetails(userRepository.save(user), oAuth2User.getAttributes());
        }
        else { // 페이스북으로 이미 회원가입이 되어 있다는 뜻
            return new PrincipalDetails(userEntity, oAuth2User.getAttributes());
        }

    }
}
```

>> 페이스북 로그인과 일반 로그인 구분하여 회원정보를 받아옴

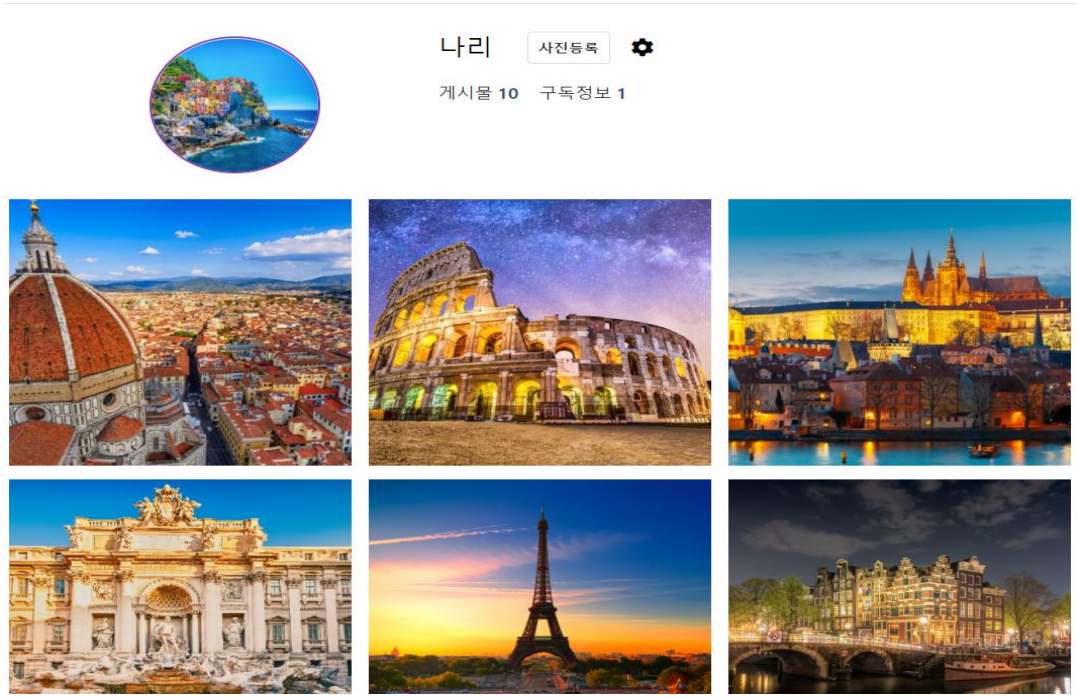
프로필 페이지

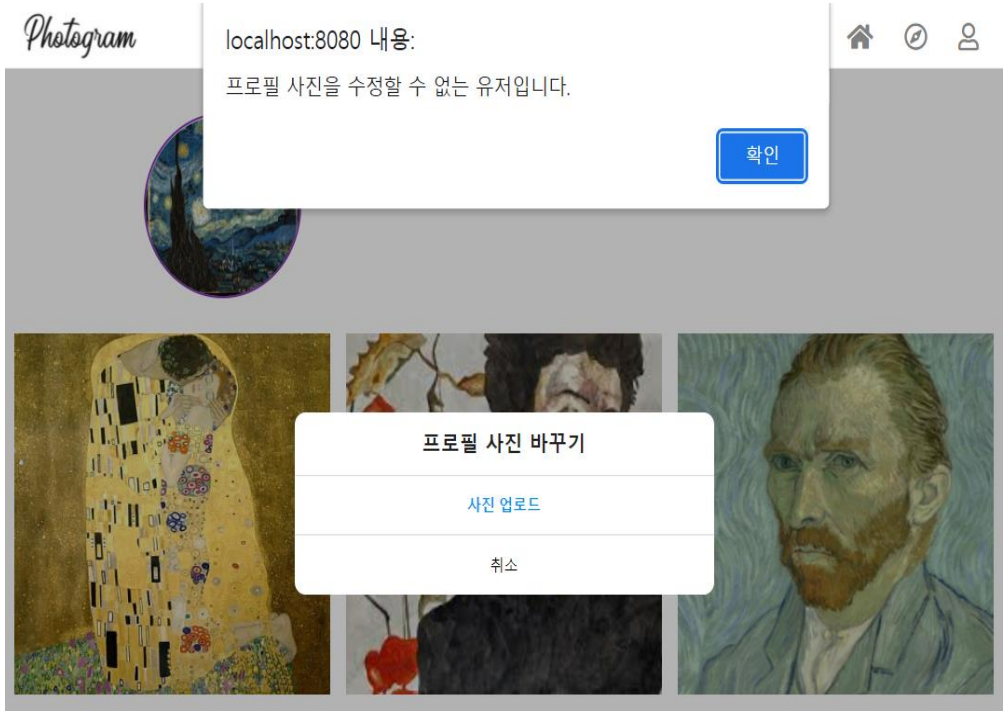



```java
//사진 업로드를 하는 함수 생성
@Transactional
public void 사진업로드(ImageUploadDto imageUploadDto, PrincipalDetails principalDetails) {
    //사진 파일의 이름 중복방지
    UUID uuid = UUID.randomUUID();// uuid
    String imageFileName = uuid+"_"+imageUploadDto.getFile().getOriginalFilename(); // 실제 파일 네임이 들어감. 1.jpg
    System.out.println("이미지 파일 이름: "+imageFileName);

    //실제 저장 경로 - 저장용도
    Path imageFilPath = Paths.get(uploadFolder+imageFileName); // 경로와 파일명

    //통신, I/O(하드디스크에 기록을 하거나 읽을때) - > 예외가 발생할 수 있다.
    try {
        Files.write(imageFilPath, imageUploadDto.getFile().getBytes());
    } catch (Exception e) {
        e.printStackTrace();
    }

    // image 테이블에 저장
    Image image = imageUploadDto.toEntity(imageFileName, principalDetails.getUser()); // 이미지 파일 이름: cde1326a
    imageRepository.save(image);
```

>> 프로필 페이지 에서 업로드한 사진과 좋아요 수, 내가 구독한 정보 확인 가능

프로필 페이지 -유저 사진 변경

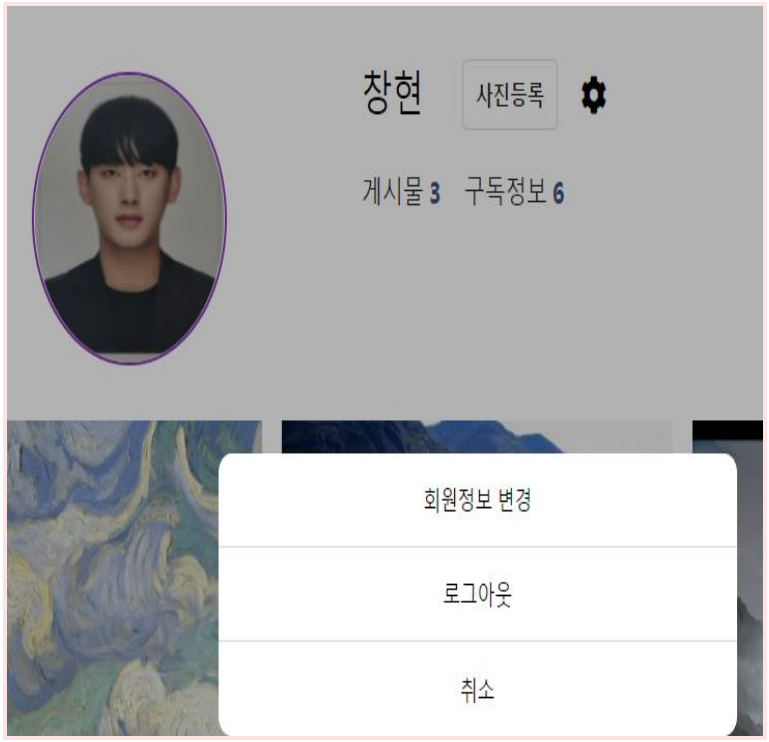


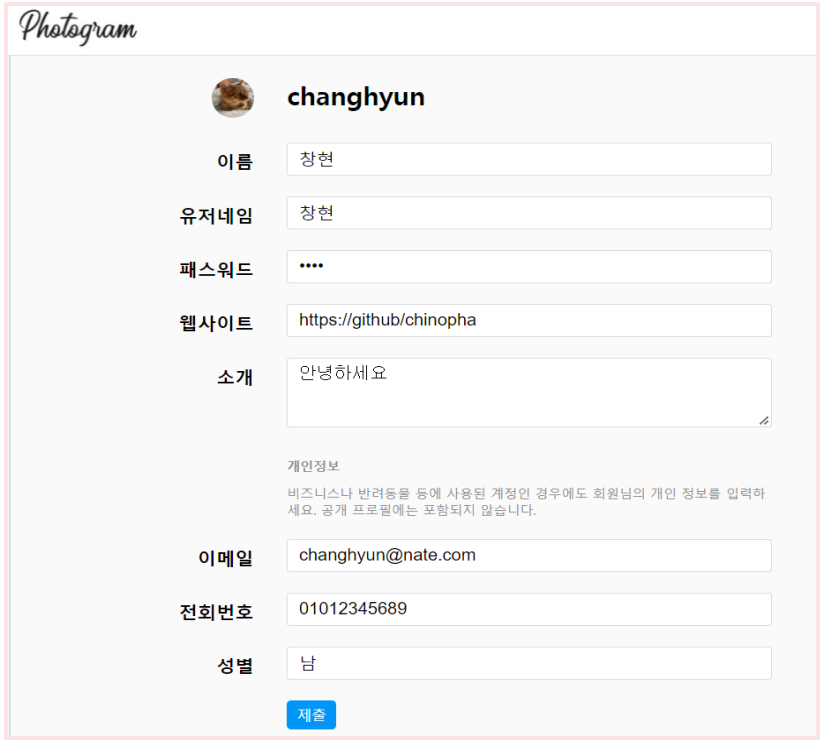



```
<!--프로필사진 바꾸기 모달-->
<div class="modal-image" onclick="modalImage()">
    <div class="modal">
        <p>프로필 사진 바꾸기</p>
        <button onclick="profileImageUpload(${dto.user.id},${principal.user.id})">사진 업로드</butto
        <button onclick="closePopup('.modal-image')">취소</button>
    </div>
</div>
```

>> 현재 로그인한 id와 페이지의 유저 id의 값을 비교 , 동일하지 않으면 사진을 수정할 수 없음

회원정보수정



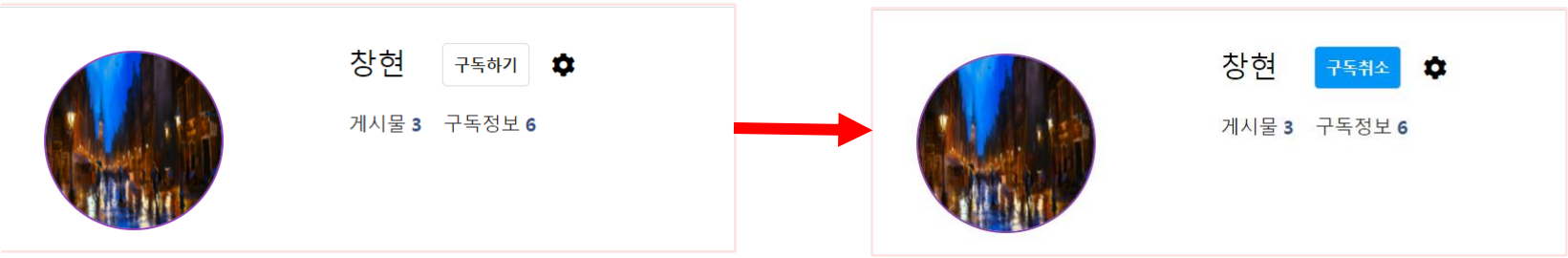



```java
@Data
public class UserUpdateDto {
    @NotBlank
    private String name; //필수
    @NotBlank
    private String password; //필수
    private String website;
    private String bio;
    private String phone;
    private String gender;

    public User toEntity() {// 필수가 아닌 데이터는 엔티티를 만드는게 위험
        return User.builder()
                .name(name)//이름을 기재 안했으면 문제!! Validation 체크
                .password(password)// 패스워드를 기재 안했으면 문제!! Validation 체크
                .website(website)
                .bio(bio)
                .phone(phone)
                .gender(gender)
                .build();
    }
```

>> 회원 정보 수정

구독하기


```
<c:choose>
  <c:when test="${dto.pageOwnerState}">
    <button class="cta" onclick="location.href='/image/upload'">사진등록</button>
  </c:when>
  <c:otherwise>
    <c:choose>
      <c:when test="${dto.subscribeState }">
        <button class="cta blue" onclick="toggleSubscribe(${dto.user.id}, this)">구독취소</button>
      </c:when>
      <c:otherwise>
        <button class="cta" onclick="toggleSubscribe(${dto.user.id}, this)">구독하기</button>
      </c:otherwise>
    </c:choose>
  </c:otherwise>
</c:choose>
```



>> 본인의 프로필에서는 구독하기 버튼이 아닌 사진등록 버튼이 나오고,
>> 다른 유저의 프로필에는 구독이 안되어 있으면 구독하기, 되어 있으면 구독 취소 버튼이 나오게 구현

구독하기

```java
@RequiredArgsConstructor
@RestController
public class SubscribeApiController {

    private final SubscribeService subscribeService;

    @PostMapping("/api/subscribe/{toUserId}")
    public ResponseEntity<?> subscribe(@AuthenticationPrincipal PrincipalDetails principalDetails, @PathVariable int toUserId) {
        subscribeService.subs(principalDetails.getUser().getId(), toUserId);
        return new ResponseEntity<>(new CMRespDto<>(1, "구독하기 성공", null), HttpStatus.OK);
    }

    @DeleteMapping("/api/subscribe/{toUserId}")
    public ResponseEntity<?> unsubscribe(@AuthenticationPrincipal PrincipalDetails principalDetails, @PathVariable int toUserId) {
        subscribeService.unsubs(principalDetails.getUser().getId(), toUserId);
        return new ResponseEntity<>(new CMRespDto<>(1, "구독취소하기 성공", null), HttpStatus.OK);
    }

}
```

```java
@Modifying  // INSERT, DELETE, UPDATE를 네이티브 쿼리로 작성하려면 해당 어노테이션이 필요하다!!
@Query(value="INSERT INTO subscribe(fromUserId, toUserId, createDate) VALUES (:fromUserId, :toUserId, now())", nativeQuery = true)
void mSubscribe(int fromUserId, int toUserId);

@Modifying
@Query(value="DELETE FROM subscribe WHERE fromUserId = :fromUserId AND toUserId = :toUserId", nativeQuery = true)
void mUnSubscribe(int fromUserId, int toUserId);
```

```java
@Transactional
public void subs(int fromUserId, int toUserId) {   // 구독하기
    try {
        subscribeRepository.mSubscribe(fromUserId, toUserId);
    } catch (Exception e) {
        throw new CustomApiException("이미 구독을 하였습니다.");
    }
}

@Transactional
public void unsubs(int fromUserId, int toUserId) {   // 구독취소하기
    subscribeRepository.mUnSubscribe(fromUserId, toUserId);
}
```

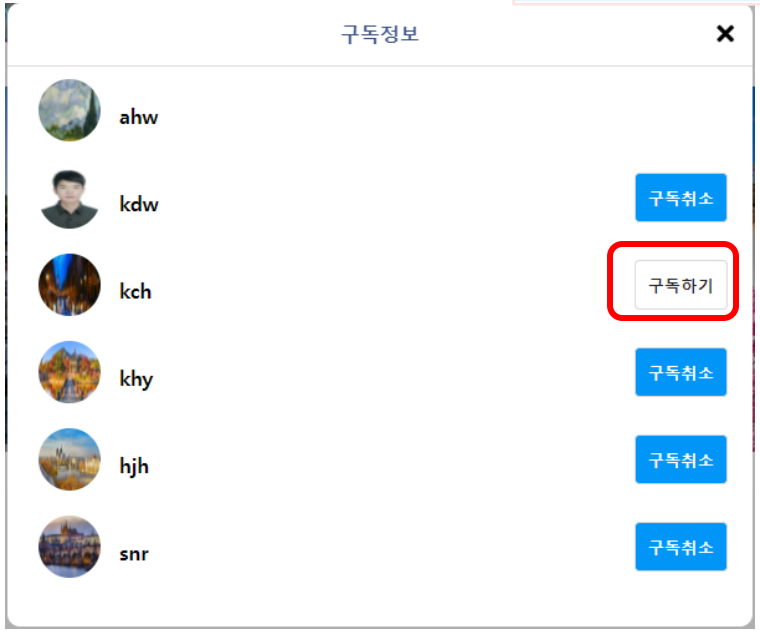
>> 네이티브 쿼리를 통해 쉽고 간단하게 구현 가능

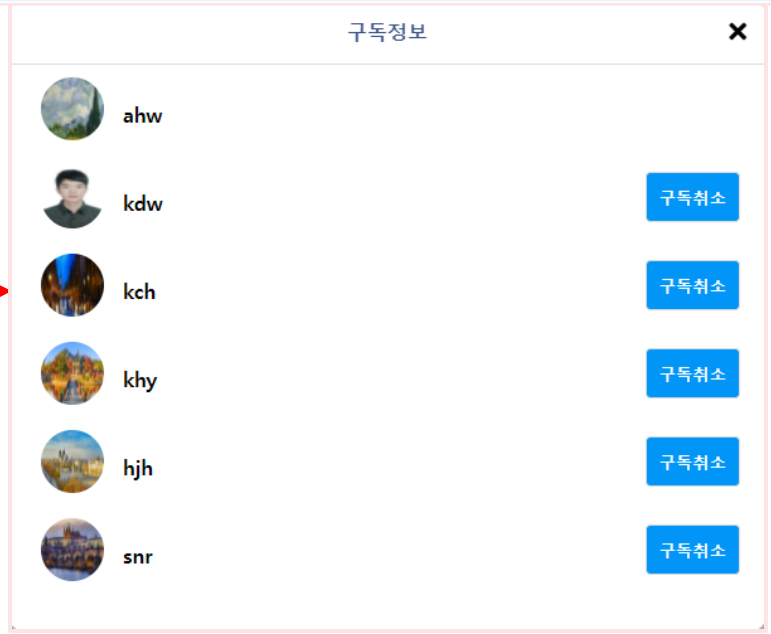
```javascript
function toggleSubscribe(toUserId, obj) {
    if ($(obj).text() === "구독취소") {

        $.ajax({
            type: "delete",
            url: "/api/subscribe/" + toUserId,
            dataType: "json"
        }).done(res=> {
            $(obj).text("구독하기");
            $(obj).toggleClass("blue");
        }).fail(error => {
            console.log("구독취소실패", error);
        });

    } else {

        $.ajax({
            type: "post",
            url: "/api/subscribe/" + toUserId,
            dataType: "json"
        }).done(res=> {
            $(obj).text("구독취소");
            $(obj).toggleClass("blue");
        }).fail(error => {
            console.log("구독하기실패", error);
        });

    }
}
```

>> 구독하기를 누르면 실행되는 toggleSubscribe 함수

구독정보

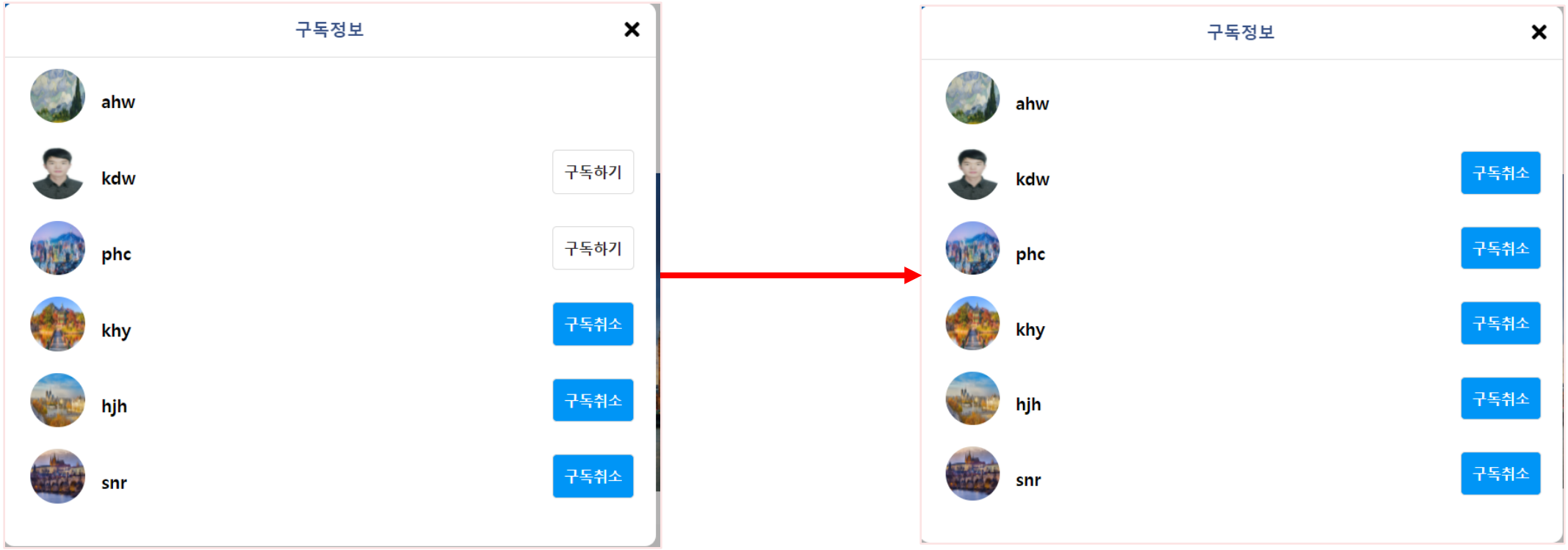

```
<li><a href="javascript:subscribeInfoModalOpen(${dto.user.id});"> 구독정보<span>${dto.subscribeCount }</span>
```





>> 구독정보를 누르면 구독하고 있는 정보가 표시되고 구독취소, 구독하기가 가능

구독정보



>> 다른 유저의 구독정보를 누르면 그 유저가 구독하고 있는 사람의 정보가 표시되고
>> 본인은 구독버튼이 안 나오고, 다른 사람을 구독하거나 구독취소 할 수 있다.

구독정보

```java
private final SubscribeRepository subscribeRepository;
private final EntityManager em;  // Repository는 EntityManager를 구현해서 만들어져 있는 구현체

@Transactional(readOnly = true) // select만 할 것 이므로 readOnly에 true를 걸어준다.
  public List<SubscribeDto> subList(int principalId, int pageUserID) { // 구독리스트

  // 쿼리 준비
  StringBuffer sb = new StringBuffer();
  sb.append("SELECT u.id, u.username, u.profileImageUrl, ");
  sb.append("if ((SELECT 1 FROM subscribe WHERE fromUserId = ? AND toUserId = u.id), 1, 0) subscribeState, ");
  sb.append("if((?= u.id), 1, 0) equalUserState ");
  sb.append("FROM user u INNER JOIN subscribe s ");
  sb.append("ON u.id = s.toUserId ");
  sb.append("WHERE s.fromUserId = ?"); // 세미콜론 첨부하면 안됨.

  // 1. 두번째줄 ?에는 principalId
  // 2. 세번째줄 ?에는 principalId
  // 3. 마지막줄 ?에는 pageUserId

  // 쿼리 완성
  Query query = em.createNativeQuery(sb.toString())
        .setParameter(1, principalId)
        .setParameter(2, principalId)
        .setParameter(3, pageUserID);

  // 쿼리 실행(qlrm 라이브러리 필요 -> DTO에 DB 결과를 매핑하기 위해서)
  JpaResultMapper result = new JpaResultMapper();
  List<SubscribeDto> subscribeDtos = result.list(query, SubscribeDto.class);
  return subscribeDtos;
```

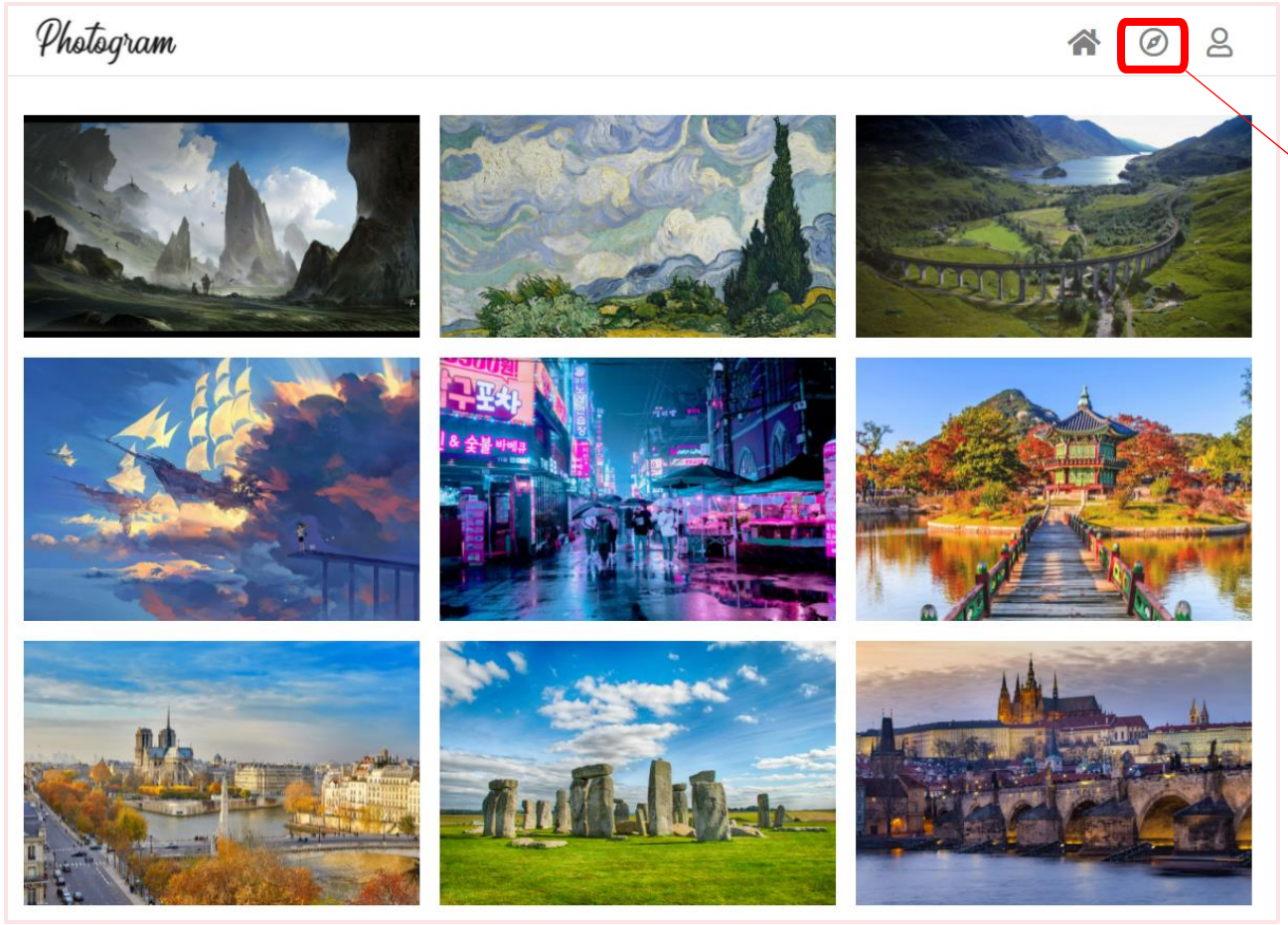
```javascript
function subscribeInfoModalOpen(pageUserId) {
  $(".modal-subscribe").css("display", "flex");

  $.ajax({
    url: '/api/user/${pageUserId}/subscribe',
    dataType: "json"
  }).done(res => {
    console.log(res.data);

    res.data.forEach((u) => {
      let item = getSubscribeModalItem(u);
      $("#subscribeModalList").append(item);
    });
  }).fail(error => {
    console.log("구독정보 불러오기 오류",error);
  });
}
```

```javascript
function getSubscribeModalItem(u) {
  let item = '<div class="subscribe__item" id="subscribeModalItem-${u.id}">
  <div class="subscribe__img">
    <img src="/upload/${u.profileImageUrl}" onerror="this.src='/images/person.jpeg'" />
  </div>
  <div class="subscribe__text">
    <h2>${u.username}</h2>
  </div>
  <div class="subscribe__btn">';

  if(!u.equalUserState) { // 동일 유저가 아닐 때 버튼이 만들어 져야함
    if(u.subscribeState) {  // 구독한 상태
      item += '<button class="cta blue" onclick="toggleSubscribe(${u.id}, this)">구독취소</button>';
    }
    else {  // 구독 안한 상태
      item += '<button class="cta" onclick="toggleSubscribe(${u.id}, this)">구독하기</button>';
    }
  }

  item += '
  </div>
</div>';

  return item;
}
```

>> 구독정보를 누르면 실행되는 subscribeInfoModalOpen 함수

>> 구독정보에 필요한 정보들을 쿼리를 통해 얻어오고, QLRM 라이브러리로 DTO에 결과를 매핑

인기페이지

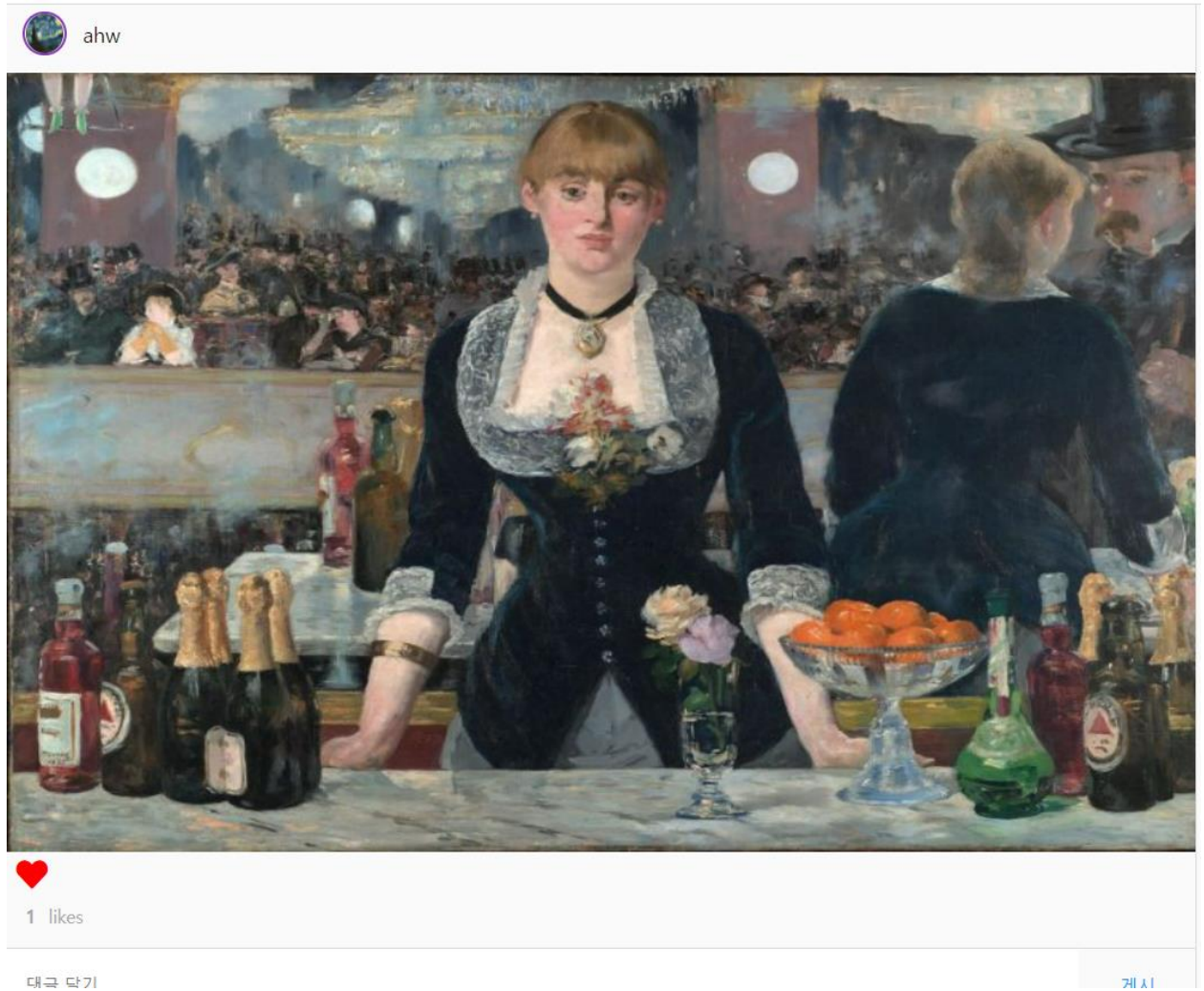



```
@GetMapping({"/image/popular"})
public String popular(Model model) {

    //api는 데이터를 리턴하는 서버!!
    List<Image> images = imageService.인기사진();
    model.addAttribute("images", images);

    return "image/popular";
}
```

```
//함수 생성
@Query(value = "SELECT * FROM image WHERE userId IN (SELECT ToUserId FROM subscribe WHERE fromUserId =:principalIc
Page<Image> mStory(int principalId, Pageable pageable);
```

>> 게시물의 좋아요 카운트에 따라 높은 순으로 상위에 노출되는 페이지를 구현

스토리 페이지

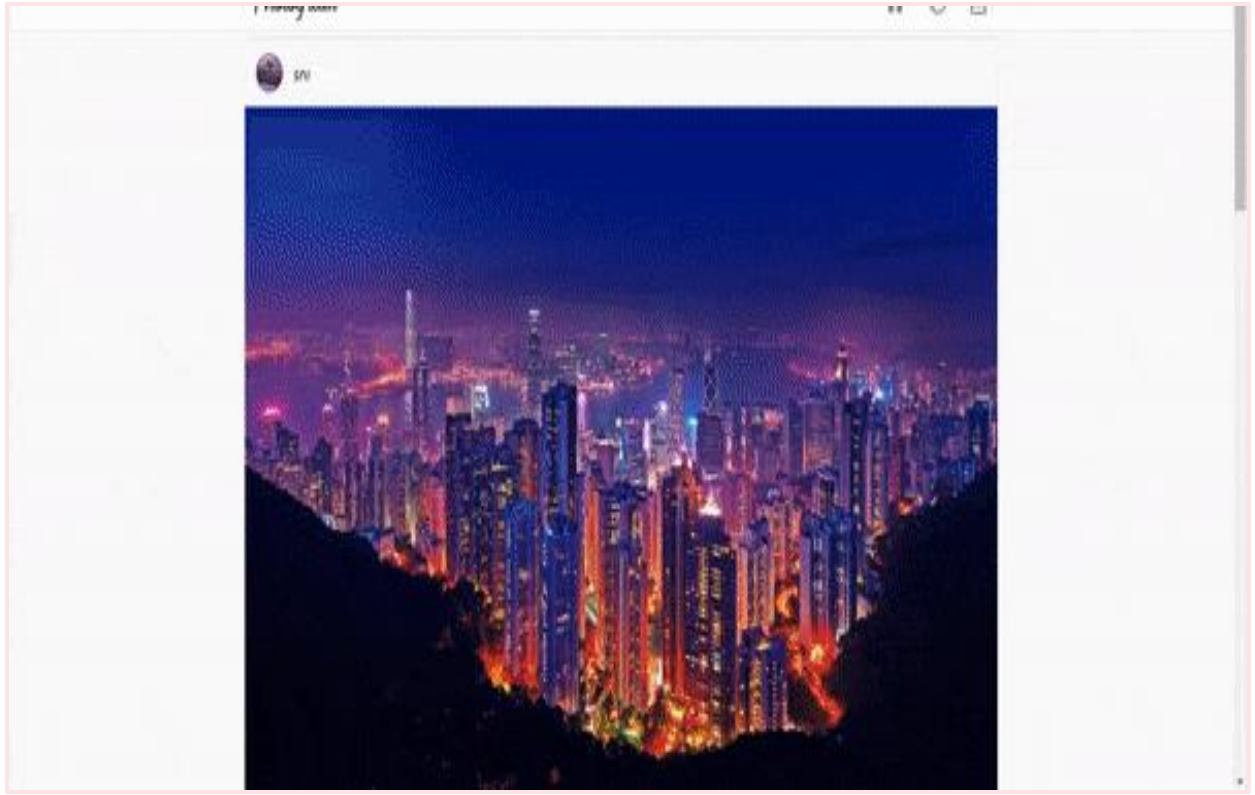



```
@RequiredArgsConstructor
@RestController
public class ImageApiController {

    private final ImageService imageService;
    private final LikesServiece likesServiece;

    @GetMapping("/api/image")
    public ResponseEntity<?> imageStory(@AuthenticationPrincipal PrincipalDetails principalDetails,
            @PageableDefault(size=3) Pageable pageable){
        Page<Image> images = imageService.이미지스토리(principalDetails.getUser().getId(), pageable);
        return new ResponseEntity<>(new CMRespDto<>(1,"성공",images),HttpStatus.OK);
    }
```

```
// (1) 스토리 로드하기
let page = 0;

function storyLoad() {
    $.ajax({
        url: `/api/image?page=${page}`,
        dataType:"json"
    }).done(res=>{
        console.log(res);
        res.data.content.forEach((image)=>{
            let storyItem = getStoryItem(image);
            $("#storyList").append(storyItem);
        });
    }).fail(error=>{
        console.log("오류", error);
    });
}

storyLoad();
```

>> 자신이 구독한 사용자들의 게시물과 정보들이 노출되는 페이지를 구현

스토리 페이징

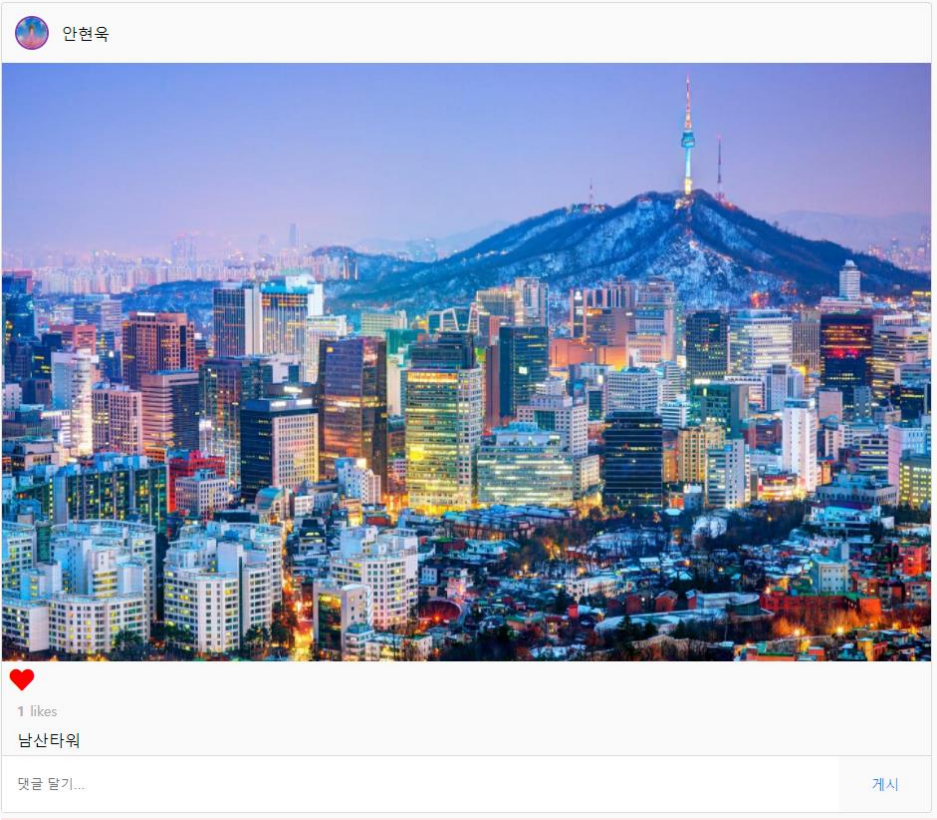


```
@RequiredArgsConstructor
@RestController
public class ImageApiController {

    private final ImageService imageService;
    private final LikesServiece likesServiece;

    @GetMapping("/api/image")
    public ResponseEntity<?> imageStory(@AuthenticationPrincipal PrincipalDetails principalDetails,
            @PageableDefault(size=3) Pageable pageable){
        Page<Image> images = imageService.이미지스토리(principalDetails.getUser().getId(), pageable);
        return new ResponseEntity<>(new CMRespDto<>(1,"성공",images),HttpStatus.OK);
    }
```

```
// (2) 스토리 스크롤 페이징하기
$(window).scroll(() => {
    /*console.log("윈도우 scrollTop", $(window).scrollTop());
    console.log("문서의 높이", $(document).height());
    console.log("윈도우 높이", $(window).height());*/

    let checkNum = $(window).scrollTop() - ($(document).height() - $(window).height() );
    console.log(checkNum);

    if(checkNum < 1 && checkNum > -1){
        page++;
        storyLoad();
    }
});
```

>> 스토리 페이지를 스크롤 페이징으로 구현

좋아요 구현



좋아요, 좋아요 취소 Controller

```
@RequiredArgsConstructor
@RestController
public class ImageApiController {

    private final LikesService likesService;


    @PostMapping("/api/image/{imageId}/likes")
    public ResponseEntity<?> likes(@PathVariable int imageId,@AuthenticationPrincipal PrincipalDetails principalDetails){
        likesService.좋아요(imageId,principalDetails.getUser().getId());
        return new ResponseEntity<>(new CMRespDto<>(1,"좋아요성공",null),HttpStatus.CREATED);
    }

    @DeleteMapping("/api/image/{imageId}/likes")
    public ResponseEntity<?> unlikes(@PathVariable int imageId,@AuthenticationPrincipal PrincipalDetails principalDetails){
        likesService.좋아요취소(imageId,principalDetails.getUser().getId());
        return new ResponseEntity<>(new CMRespDto<>(1,"좋아요취소성공",null),HttpStatus.OK);
    }

}
```

Likes.java

```
1  package com.cos.photogramstart.domain.likes;
2
3  import java.time.LocalDateTime;
24
25 @Builder
26 @AllArgsConstructor
27 @NoArgsConstructor
28 @Data
29 @Entity
30 @Table(
31     uniqueConstraints = {
32         @UniqueConstraint(
33             name="likes_uk",
34             columnNames = {"imageId", "userId"}
35         )
36     }
37 )
38 public class Likes {   // N
39     @Id
40     @GeneratedValue(strategy = GenerationType.IDENTITY)
41     private int id;
42
43     @JoinColumn(name = "imageId")
44     @ManyToOne
45     private Image image; // 1
46
47     @JsonIgnoreProperties({"images"})
48     @JoinColumn(name = "userId")
49     @ManyToOne
50     private User user; // 1
51
52     private LocalDateTime createDate;
53
54     @PrePersist
55     public void createDate() {
56         this.createDate = LocalDateTime.now();
57     }
58
59 }
60
```

LikesRepository.java

```
1  package com.cos.photogramstart.domain.likes;
2
3  import org.springframework.data.jpa.repository.JpaRepository;
6
7  public interface LikesRepository extends JpaRepository<Likes, Integer> {
8
9      @Modifying
10     @Query(value = "INSERT INTO likes(imageId, userId, createDate) VALUES(:imageId, :principalId, now())", nativeQuery = true)
11     int mLikes(int imageId, int principalId);
12
13     @Modifying
14     @Query(value = "DELETE FROM likes WHERE imageId = :imageId AND userId = :principalId", nativeQuery = true)
15     int mUnLikes(int imageId, int principalId);
16
17 }
```

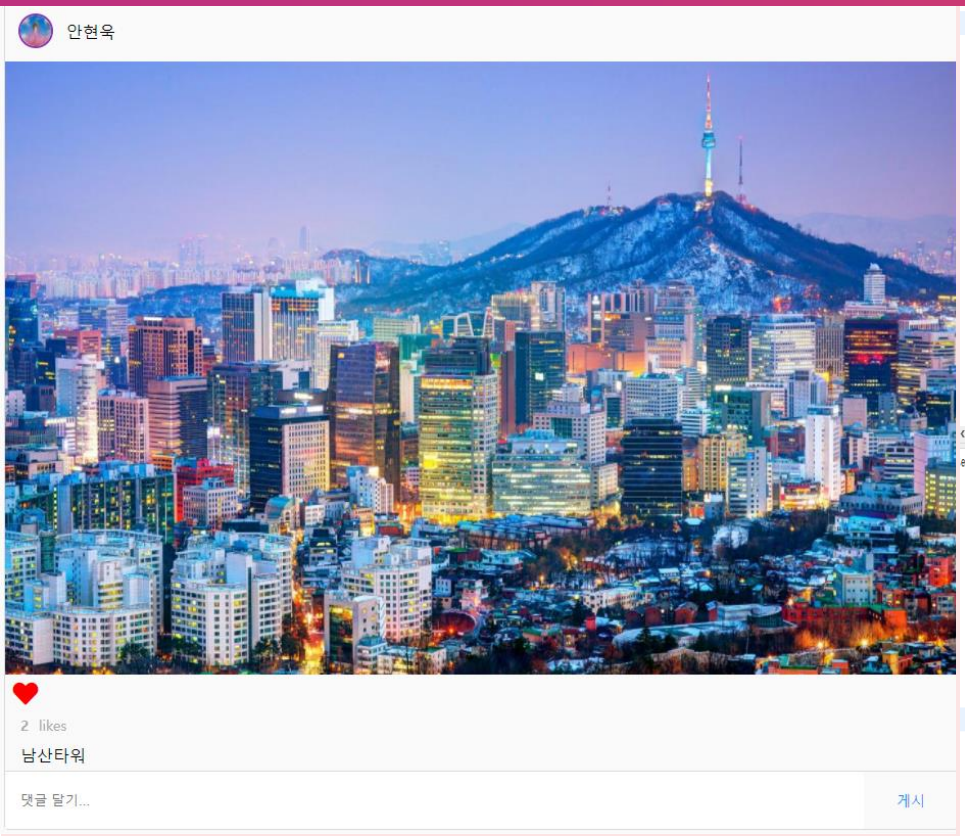
LikesService.java

```
1  package com.cos.photogramstart.service;
2
3  import org.springframework.stereotype.Service;
9
10 @RequiredArgsConstructor
11 @Service
12 public class LikesService {
13
14     private final LikesRepository likesRepository;
15
16     @Transactional
17     public void 좋아요(int imageId, int principalId) {
18         likesRepository.mLikes(imageId, principalId);
19     }
20
21     @Transactional
22     public void 좋아요취소(int imageId, int principalId) {
23         likesRepository.mUnLikes(imageId, principalId);
24     }
25 }
26
```

>> 좋아요 모델 –> 이미지와 유저의 좋아요 연관관계 형성 – 1:N

>> 좋아요 및 좋아요 취소 API를 구현한 뒤 컨트롤러에서 받아와 구현

좋아요 카운트




```
            item+=`<i class="fas fa-heart active" id="storyLikeIcon-${image.id}" onclick="toggleLike(
        }else{
            item+=`<i class="far fa-heart" id="storyLikeIcon-${image.id}" onclick="toggleLike(${imag
        }


        item+=`
    </button>
</div>

<span class="like"><b id="storyLikeCount-${image.id}">${image.likeCount} </b>likes</span>

<div class="sl__item__contents__content">
    <p>${image.caption}</p>
</div>

<div id="storyCommentList-1">
```

LikesService

```
private final ImageRepository imageRepository;

@Transactional(readOnly = true)  //영속성 컨텍스트 변경감지를 해서, 더티체킹, flush(반영) X
public Page<Image> 이미지스토리(int principalId, Pageable pageable){
    Page<Image> images = imageRepository.mStory(principalId,pageable);

    //1.2(cos) 로그인 하고 2번이 구독하고있는 팔로우들의 이미지(image)를 불러옴
    // images에 좋아요 상태 담기
    images.forEach((image)->{// 2. 그 이미지를 for문을 돌려서 하나씩 뽑아냄

        //좋아요 카운트
        image.setLikeCount(image.getLikes().size());

        image.getLikes().forEach((like)->{ //3. 첫번째 이미지를 좋아요하고있는걸 다 가져옴
            //거기에 2번 자기자신이 좋아요한걸 찾으면 된다.
            if(like.getUser().getId() == principalId) { // 해당 이미지에 좋아요한 사람들을 찾아서 현재 로그인한 사람이
                image.setLikeState(true);
```

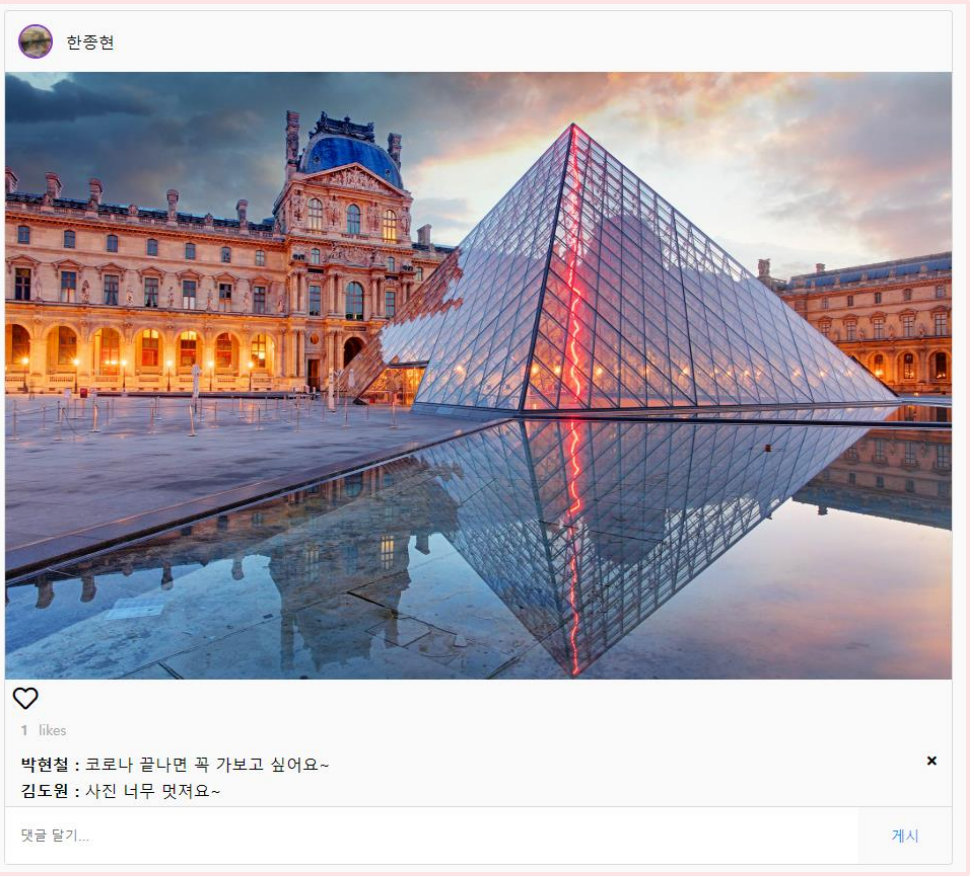
```
    private int id;

    private String caption; // ex)오늘 나 너무 피곤해!!
    private String postImageUrl; // 사진을 전송 받아서 그 사진을 서버에

    @JsonIgnoreProperties({"images"}) //images는 무시하고 User정
    @JoinColumn(name = "userId")
    @ManyToOne(fetch = FetchType.EAGER) //이미지를 select하면 2
    private User user; //1,1

    //이미지
    @JsonIgnoreProperties({"image"}) //무한참조 예방
    @OneToMany(mappedBy = "image")//연관관계주인x image만들지
    private List<Likes> likes;
    //댓글

    @Transient //DB에 컬럼이 만들어지지않는다.
    private boolean likeState;

    @Transient //좋아요카운트
    private int likeCount;


    private LocalDat

    @PrePersist //D
    public void crea
        this.createDat
    }
/*
    @Override
    public String toS
        return "Image
                + ", crea
    }
    */ //오브젝트를 콘
```

story.js

```
        //해당 아이디 접근해서 내부에 있는 텍스트를 가져온다.
        //뷰화면에서 좋아요 숫자 증가
        let likeCountStr = $(`#storyLikeCount-${imageId}`).text();
        let likeCount = Number(likeCountStr) + 1;
        $(`#storyLikeCount-${imageId}`).text(likeCount);

        likeIcon.addClass("fas");
        likeIcon.addClass("active");
        likeIcon.removeClass("far");
        }).fail(error=>{
            console.log("오류",error);
        })


    } else {  //좋아요 취소하겠다.
            $.ajax({
            type:"delete",
            url:`/api/image/${imageId}/likes`,
            dataType:"json"
        }).done(res=>{
        let likeCountStr = $(`#storyLikeCount-${imageId}`).text();
        let likeCount = Number(likeCountStr) - 1;
        $(`#storyLikeCount-${imageId}`).text(likeCount);

        likeIcon.removeClass("fas");
        likeIcon.removeClass("active");
        likeIcon.addClass("far");
        }).fail(error=>{
            console.log("오류",error);
        })
```

>> 좋아요 카운트 뷰 렌더링 -> LikseService에 좋아요 카운트를 담아서 story.js에 정보를 넣어주면 뷰화면에 좋아요 카운트 증가

댓글작성 / 삭제



댓글 모델

```java
package com.cos.photogramstart.domain.comment;

import java.time.LocalDateTime;

@Builder
@AllArgsConstructor
@NoArgsConstructor
@Data
@Entity
public class Comment {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private int id;

  @Column(length = 100, nullable = false)
  private String content;

  @JsonIgnoreProperties({"images"})
  @JoinColumn(name = "userId")
  @ManyToOne(fetch = FetchType.EAGER)  // ManyToOne에서 디폴트값은 EAGER임.
  private User user;

  @JoinColumn(name = "imageId")
  @ManyToOne
  private Image image;

  private LocalDateTime createDate;

  @PrePe
  public
    this.
  }

}
```

```java
package com.cos.photogramstart.web.api;

import java.util.HashMap;

@RequiredArgsConstructor
@RestController
public class CommentApiController {

  private final CommentService commentService;

  @PostMapping("/api/comment")
  public ResponseEntity<?> commentSave(@Valid @RequestBody CommentDto commentDto, BindingResult binc

    Comment comment = commentService.writeComment(commentDto.getContent(), commentDto.getImageId()
    return new ResponseEntity<>(new CMRespDto<>(1, "댓글쓰기성공", comment), HttpStatus.CREATED);
  }

  @DeleteMapping("/api/comment/{id}")
  public ResponseEntity<?> commentDelete(@PathVariable int id) {
    commentService.deleteComment(id);
    return new ResponseEntity<>(new CMRespDto<>(1, "댓글삭제성공", null), HttpStatus.OK);
  }
}
```

CommentService

CommentService.java

```java
package com.cos.photogramstart.service;

import org.springframework.stereotype.Service;

@RequiredArgsConstructor
@Service
public class CommentService {

  private final CommentRepository commentRepository;
  private final UserRepository userRepository;

  @Transactional
  public Comment writeComment(String content, int imageId, int userId) {   // 댓글쓰기

    // Tip (객체를 만들 때 id 값만 담아서 insert 할 수 있다)
    // 대신 return 시에 image 객체와 user 객체는 id값만 가지고 있는 빈 객체를 리턴받는다.
    Image image = new Image();
    image.setId(imageId);

    User userEntity= userRepository.findById(userId).orElseThrow(()-> {
      throw new CustomApiException("유저 아이디를 찾을 수 없습니다.");
    });

    Comment comment = new Comment();
    comment.setContent(content);
    comment.setImage(image);
    comment.setUser(userEntity);

    return commentRepository.save(comment);
  }

  @Transactional
  public void deleteComment(int id) {  // 댓글삭제
    try {
      commentRepository.deleteById(id);
    } catch (Exception e) {
      throw new CustomApiException(e.getMessage());
    }
  }
}
```

>> 댓글작성,삭제에 필요한 서비스와 컨트롤러

댓글작성 / 삭제

```javascript
// (4) 댓글쓰기
function addComment(imageId) {

  let commentInput = $('#storyCommentInput-${imageId}');
  let commentList = $('#storyCommentList-${imageId}');

  let data = {
    imageId: imageId,
    content: commentInput.val()
  }

  //console.log(data);
  //console.log(JSON.stringify(data));

  if (data.content === "") {
    alert("댓글을 작성해주세요!");
    return;
  }

  $.ajax({
    type: "post",
    url: "/api/comment",
    data: JSON.stringify(data),
    contentType: "application/json; charset=utf-8",
    dataType: "json"
  }).done(res => {
    //console.log("성공", res);

    let comment = res.data;

    let content = `
     <div class="sl__item__contents__comment" id="storyCommentItem-${comment.id}">
       <p>
         <b>${comment.user.username} :</b>
         ${comment.content}
       </p>
       <button onclick="deleteComment(${comment.id})"><i class="fas fa-times"></i></button>
     </div>
    `;
  commentList.prepend(content);
  }).fail(error=> {
    //console.log("오류", error.responseJSON.data.content);
    alert(error.responseJSON.data.content);
  });

  commentInput.val(""); // 인풋 필드를 깨끗하게 비워준다.
}
```

```javascript
// (5) 댓글 삭제
function deleteComment(commentId) {
  $.ajax({
    type: "delete",
    url: `/api/comment/${commentId}`,
    dataType: "json"
  }).done(res => {
    console.log("성공", res);
    $('#storyCommentItem-${commentId}').remove();
  }).fail(error => {
    console.log("오류", error);
  });
}
```

CommentApiController

```java
package com.cos.photogramstart.web.a

import java.util.HashMap;

@RequiredArgsConstructor
@RestController
public class CommentApiController {

    private final CommentService commentService;

    @PostMapping("/api/comment")
    public ResponseEntity<?> commentSave(@Valid @RequestBody CommentDto commentDto, BindingResult bindingResult, @AuthenticationPrincipal PrincipalDetails principalDetails) {

        Comment comment = commentService.writeComment(commentDto.getContent(), commentDto.getImageId(), principalDetails.getUser().getId());
        return new ResponseEntity<>(new CMRespDto<>(1, "댓글쓰기성공", comment), HttpStatus.CREATED);
    }

    @DeleteMapping("/api/comment/{id}")
    public ResponseEntity<?> commentDelete(@PathVariable int id) {
        commentService.deleteComment(id);
        return new ResponseEntity<>(new CMRespDto<>(1, "댓글삭제성공", null), HttpStatus.OK);
    }
}
```

CommentService

CommentService.java

```java
package com.cos.photogramstart.service;

import org.springframework.stereotype.Service;

@RequiredArgsConstructor
@Service
public class CommentService {

    private final CommentRepository commentRepository;
    private final UserRepository userRepository;

    @Transactional
    public Comment writeComment(String content, int imageId, int userId) {   // 댓글쓰기

        // Tip (객체를 만들 때 id 값만 담아서 insert 할 수 있다)
        // 대신 return 시에 image 객체와 user 객체는 id값만 가지고 있는 빈 객체를 리턴받는다.
        Image image = new Image();
        image.setId(imageId);

        User userEntity= userRepository.findById(userId).orElseThrow(()-> {
            throw new CustomApiException("유저 아이디를 찾을 수 없습니다.");
        });

        Comment comment = new Comment();
        comment.setContent(content);
        comment.setImage(image);
        comment.setUser(userEntity);

        return commentRepository.save(comment);
    }

    @Transactional
    public void deleteComment(int id) {  // 댓글삭제
        try {
            commentRepository.deleteById(id);
        } catch (Exception e) {
            throw new CustomApiException(e.getMessage());
        }
    }

}
```

>> ajax함수로 서비스와 컨트롤러에서 호출하여 뷰렌더링 구현

유효성 검사 자동화 AOP 처리

Controller

```
package com.cos.photogramstart.web.api;

import java.util.HashMap;

@RequiredArgsConstructor
@RestController
public class CommentApiController {

  private final CommentService commentService;

  @PostMapping("/api/comment")
  public ResponseEntity<?> commentSave(@Valid @RequestBody CommentDto commentDto, BindingResult bindingResult, @AuthenticationPrincipal PrincipalDetails principalDetails) {

    if(bindingResult.hasErrors()) {
      Map<String, String> errorMap = new HashMap<>();

      for(FieldError error:bindingResult.getFieldErrors()) {
        errorMap.put(error.getField(),error.getDefaultMessage());
      }
      throw new CustomValidationApiException("유효성검사 실패함",errorMap);
    }

    Comment comment = commentService.writeComment(commentDto.getContent(), commentDto.getImageId(), principalDetails.getUser().getId());
    return new ResponseEntity<>(new CMRespDto<>(1, "댓글쓰기성공", comment), HttpStatus.CREATED);
  }

  @DeleteMapping("/api/comment/{id}")
  public ResponseEntity<?> commentDelete(@PathVariable int id) {
    commentService.deleteComment(id);
    return new ResponseEntity<>(new CMRespDto<>(1, "댓글삭제성공", null), HttpStatus.OK);
  }
}
```

제거

```
import java.util.HashMap;

@Component  //RestController, Service 모든 것들이 Component를 상속해서 만들어져 있음.
@Aspect
public class ValidationAdvice {

  @Around("execution(* com.cos.photogramstart.web.api.*Controller.*(..))")
  public Object apiAdvice(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {

    //System.out.println("web api 컨트롤러======================");
    Object[] args = proceedingJoinPoint.getArgs();
    for (Object arg : args) {

      if(arg instanceof BindingResult) {
        BindingResult bindingResult = (BindingResult)arg;

        if(bindingResult.hasErrors()) {
          Map<String, String> errorMap = new HashMap<>();

          for(FieldError error:bindingResult.getFieldErrors()) {
            errorMap.put(error.getField(),error.getDefaultMessage());
          }
          throw new CustomValidationApiException("유효성검사 실패함",errorMap);
        }

      }

    }
    //ProceedingJoinPoint => profile함수의 모든 곳에 접근할 수 있는 변수
    //profile 함수보다 먼저 실행

    return proceedingJoinPoint.proceed();  //profile 함수가 실행됨.
  }
  @Around("execution(* com.cos.photogramstart.web.*Controller.*(..))")
  public Object advice(ProceedingJoinPoint proceedingJoinPoint) throws Throwable {

    //System.out.println("web  컨트롤러======================");
    Object[] args = proceedingJoinPoint.getArgs();
    for (Object arg : args) {
      if(arg instanceof BindingResult) {

        BindingResult bindingResult = (BindingResult)arg;

        if(bindingResult.hasErrors()) {
          Map<String, String> errorMap = new HashMap<>();

          for(FieldError error:bindingResult.getFieldErrors()) {
            errorMap.put(error.getField(),error.getDefaultMessage());

          }
          throw new CustomValidationException("유효성검사 실패함",errorMap);
        }
      }

    }
    return proceedingJoinPoint.proceed();
  }
}
```

>> 각각의 컨트롤러안에 있는 유효성검사를 제거하고, ValidationAdvice에 공통으로 처리

>> 공통기능들은 AOP처리를 하여 불필요한 코드를 줄임으로써 코드를 깔끔하게 정리

❺ 실행 화면

감사합니다.