

개요

프로젝트명	Python&Django MTV SNS 웹 어플리케이션 '마이스타그램'
개발기간	2019.11.11 ~ 2019.11.21
참여인원	3인
구현	메인포스트, 포스트작성, 각종 알림, 회원기능
개발환경	Window10, Ubuntu 18.04
사용 도구	Pycharm, SQLite DB Browser, Dbeaver, Github
사용 기술	Python, Django, SQL(RDS), Github, HTML, CSS, JS, JQUERY, Ajax, JSON, Python Library (Pillow, Exif etc.)
사용 외부 API	Kakao map API

요약흐름도

SNS관리

- 팔로잉 끊기
- 팔로잉 시작
- 새로운 친구 추천
- 알림 기능
- 좋아요(하트)

회원관리

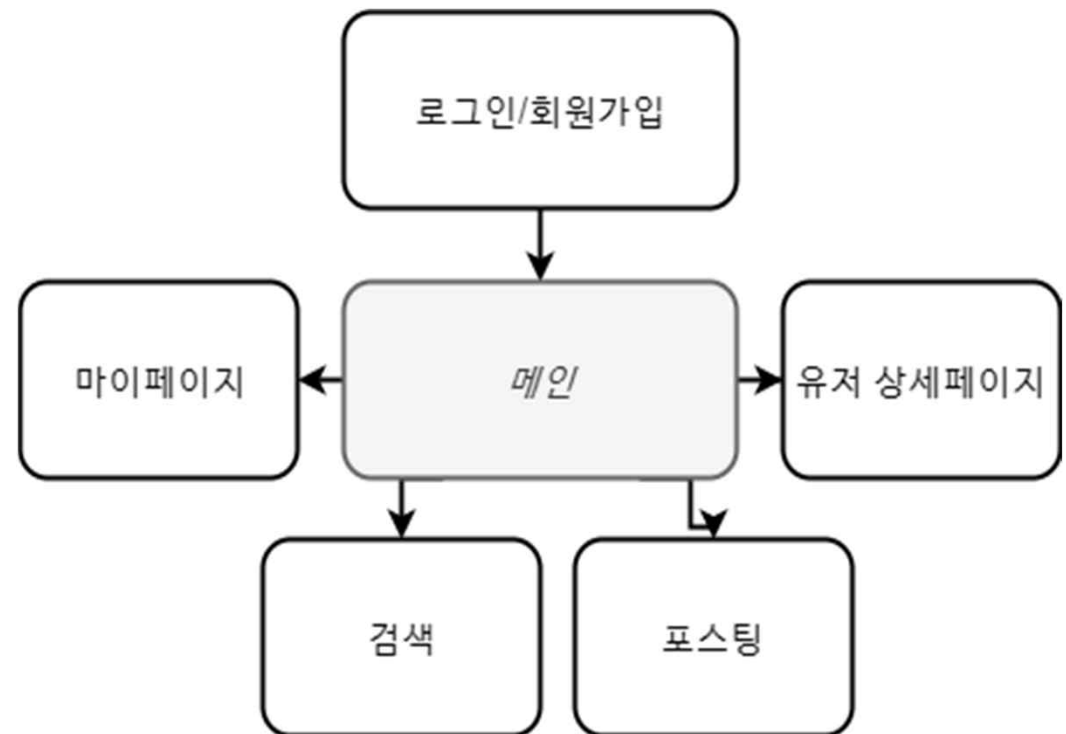
- 로그인
- 로그아웃
- 회원가입
- 내 정보 수정(탈퇴)

포스팅관리

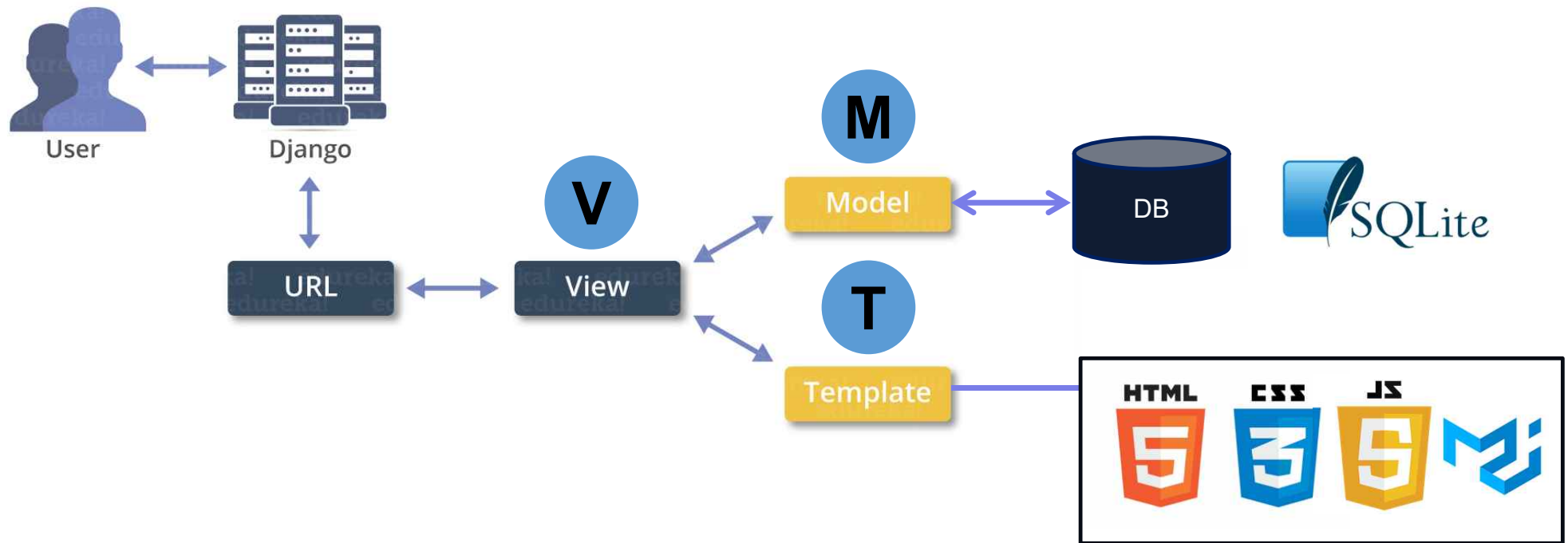
- 새글쓰기
- 포스팅 수정
- 포스팅 삭제
- 포스트 출력
- 특정 유저의 포스트 보여주기

부가기능

- 이미지의 위치정보 표시
- 검색창
- 좋아요(하트)
- 새로운 친구 추천

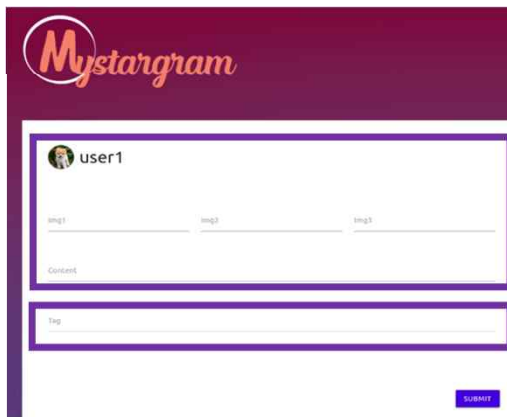


시스템구성도



1)포스트 작성

- 포스트 작성



post테이블로 db저장처리

```
def submit_post(request): # 글쓰기 insert.

    post = Post(

        img1=request.POST["img1"],

        img2=request.POST["img2"],

        img3=request.POST["img3"],

        content=request.POST["content"],

        user_id=User.objects.filter(user_id=request.session["id"]).get()) # 세션에 담겨있는 유저아이디로 저장.

    post.save() # 포스트테이블 부분 서버로 저장완료.
```

Tags필드데이터: db형식에 맞게 처리

```
def split_tags(tags): # 글쓰기 입력시, 태그필드 별도 저장처리.

    output = [] # 배열로 받을 준비.

    splitted = tags.split("#") # 샵을 기준으로 들어온 인자를 쪼개서 splitted 변수 안에

    for split in splitted:

        stripped = split.strip() # 반복문 통해서 앞뒤의 공백을 없애고,

        if stripped: # 공백제거 처리된후 output 배열로 넣어준다,

            # false인 경우는 패스(' ' 와 같은 경우 공백제거로 false가 처리, 자동패스됨)

            output.append(stripped)

    return output # 결과값 리턴.
```

```
tags = split_tags(request.POST["tag"]) # 태그부분에 담긴 글을 split_tag 함수를 통해 쪼개 후 tags 변수 안에 담기

for tag in tags:

    dto = PostTag(word=tag, post_id=post)

    dto.save() # 포스트태그_테이블로 단어와, 포스트id를 반복적으로 디비저장처리.

return redirect("home") # 메인 페이지 보내기.
```

2)포스트 조회 (마이페이지)



def mypage(request): # mypage로 이동.

ssid = request.session["id"]

user_data = User.objects.filter(user_id=ssid) # 세션의 아이디 값으로 db에 있는 내용 불러옴.

my_followee = read_my_followees_with_name(ssid) #나의 팔로워 데이터 읽기 함수 호출.

my_follower = read_my_followers_with_name(ssid) #나의 팔로워 데이터 읽기 함수 호출.

my_all_post = read_my_all_post(ssid) #나의 포스팅 데이터 읽기 함수 호출.

dic_user = {"id": user_data[0].user_id, "name": user_data[0].name, "pw": user_data[0].pw,
"intro": user_data[0].intro, "favorite": user_data[0].favorite, "userpic": user_data[0].userpic,
"my_followee": my_followee, "my_follower": my_follower, "my_post": my_all_post}

return render(request, "mypage.html", dic_user)

템플릿 상 탭에 따라 노출처리,
필요한 모든 데이터는 현 함수 내 확보하여 전달처리

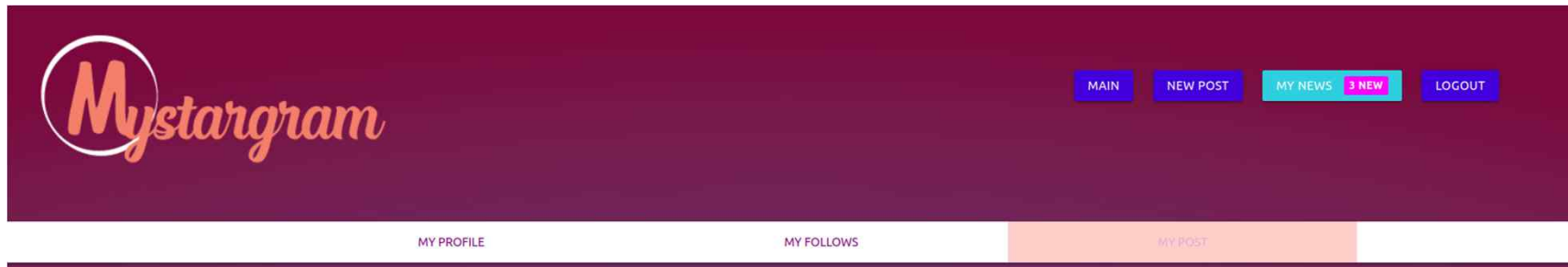
마이페이지 구성: 3탭.

1. 나의프로필 수정/탈퇴.

② 나의 포스팅 수정삭제를
위한 조회.

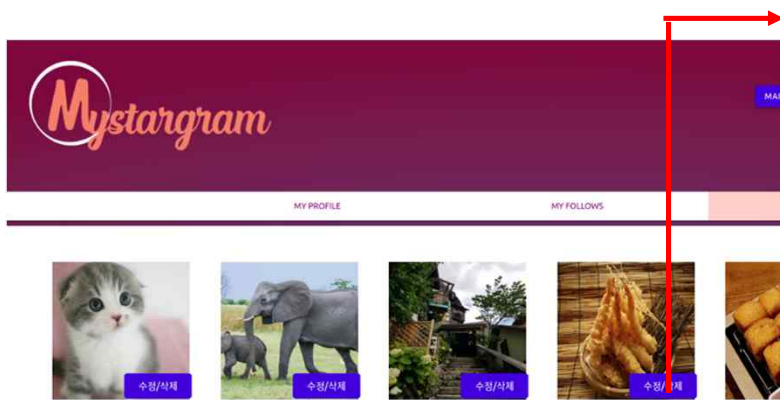
③ 나의 팔로우현황.

2)포스트 조회 (마이페이지)



```
]def read_my_all_post(id): # 나의페이지 - 나의 포스팅 불러오기.  
    result = Post.objects.filter(user_id=id) #인자로받은 나의 아이디를 넣어 필터링 후 담기.  
    my_post = [] #배열변수선언.  
    for x in result: #읽어온 포스트의 대표이미지(img1)가 될 데이터만 담기.  
        if x.img1:  
            my_post.append(x)  
    print(my_post)  
    return my_post
```

2)포스트 조회 (마이페이지)



```
def read_post_detail(request): #나의글 수정/삭제를 위한 해당포스팅 데이터 읽어오기.
```

```
pid = request.POST.get("pid") #포스팅의 id를 받아서.
```

```
dto = Post.objects.get(id=pid) #해당 내용을 불러오고.
```

```
tag_list = list(PostTag.objects.filter(post_id=pid).values_list('word', flat=True)) #태그불러오기.
```

```
tags = '#'.join(tag_list) #단어로 분리되어 DB로 삽입된 태그를 조인.
```

```
return render(request, "edit_post.html", {"dto": dto, "tags": tags})
```

포스트의 읽기는 저장과 마찬가지로,
Post테이블과 postTag 테이블에서 각각 불러오며,
태그에서 읽어온 리스트를 '#'을 삽입해 조인으로 묶어
출력 할 수 있도록 처리.



<!--수정/삭제.요청 버튼.-->

```
<div>
```

```
<input class="btn" style="..." type="button"
```

```
value="delete" onclick="post_delete()>
```

```
<input class="btn" style="..." type="button"
```

```
value="edit" onclick="post_update()>
```

```
</div>
```


3)포스트 수정 (마이페이지)

user1

46.jpg 50.jpg 51.jpg

그림판은 풍경들

#서울#가고싶은곳

EDIT DELETE

```
<input class="btn" style="..." type="button"
value="edit" onclick="post_update()">
```

```
function post_update() {
    if (confirm("내용을 수정하시겠습니까?")) {
        document.confirm_edit_delete.action = "post_update";
        document.confirm_edit_delete.submit();
    }
}
```

```
def post_update(request): #나의 글 업데이트.
    id = request.POST.get("pid")
    dto = Post(id=id,
        img1=request.POST["img1"],
        img2=request.POST["img2"],
        img3=request.POST["img3"],
        content=request.POST["content"],
        user_id=User.objects.filter(user_id=request.session["id"]).get(),
        time=timezone.now()) # 글을 수정받고, 시간을 새로운 시간으로 갱신.
    dto.save()
    PostTag.objects.filter(post_id=id).delete() #기존의 태그데이터를 삭제하고.
    tags = split_tags(request.POST["tag"]) #새로운 태그를 받아 split_tags 함수로 정리 후,
    for tag in tags: #반복문을 통해 태그테이블로 세이브처리.
        id = request.POST.get("pid")
        tag = PostTag(word=tag, post_id=id)
        tag.save()
    return redirect("mypage") # 메인 페이지 보내기.
```

Time컬럼의 시간을 새롭게 갱신처리.
Tag의 경우 기존 postTag의 데이터를 삭제 후
새로운 태그데이터와 동일한 처리.

4)포스트 삭제 (마이페이지)



```
<input class="btn" style="..." type="button"
      value="edit" onclick="post_update()">
```

```
<input class="btn" style="..." type="button"
      value="delete" onclick="post_delete()">
```

```
<script>
```

```
function post_update() {
    if (confirm("내용을 수정하시겠습니까?")) {
        document.confirm_edit_delete.action = "post_update";
        document.confirm_edit_delete.submit();
    }
}
```

```
function post_delete() {
```

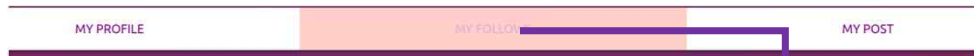
```
    if (confirm("삭제하시겠습니까?")) {
        document.confirm_edit_delete.action = "post_delete";
        document.confirm_edit_delete.submit();
    }
}
```

```
</script>
```

(포스트 삭제 시)
포스트와 해당 포스트의 태그테이블 역시 함께 삭제처리.

```
④ def post_delete(request): #나의글 삭제
    id = request.POST.get("pid") #포스트의 id를 받아와서.
    Post.objects.get(id=id).delete() #해당포스트를 삭제.
    PostTag.objects.filter(post_id=id).delete() #해당 포스트의 태그를 삭제.
    return redirect("mypage")
```

5) 팔로우조회



나의 팔로우스 (팔로워/팔로워 각각 읽어오기)

My Followee

ID: user7 이름:정병수	팔로잉취소
ID: user8 이름:김갑동	팔로잉취소
ID: user10 이름:안도영	팔로잉취소
ID: user4 이름:홍길동	팔로잉취소
ID: user9 이름:하성우	팔로잉취소
ID: user2 이름:정민경	팔로잉취소

My Follower

ID: user4 이름:홍길동
ID: user7 이름:정병수
ID: user9 이름:하성우

```
def read_my_followees_with_name(id): # 나의 팔로우스- 나의 **팔로워** 불러오기.  
    result = [] #리스트변수.  
    for x in Follows.objects.filter(follower=id): #인자로 전달받은 id의 팔로워 매칭데이터를  
        result.append((x, x.followee.name)) #위의 리스트안으로 이름과 함께 반복넣기.  
    return result
```

팔로잉**신청**은 친구추천or검색을
통해 가능.
마이페이지 내에서는
팔로잉취소만 처리하도록.

```
def read_my_followers_with_name(id): # 나의 팔로우스- 나의 **팔로워** 불러오기.  
    result = [] #리스트변수.  
    for x in Follows.objects.filter(followee=id): #인자로 전달받은 id의 팔로워 매칭데이터를  
        result.append((x, x.follower.name)) #위의 리스트안으로 이름과 함께 반복넣기.  
    return result
```

6)팔로잉취소

MY PROFILE	MY FOLLOWS	MY POST
------------	------------	---------

My Followee

ID: user7 이름:정병수	팔로잉취소
ID: user8 이름:김갑동	팔로잉취소
ID: user10 이름:안도영	팔로잉취소
ID: user4 이름:홍길동	팔로잉취소
ID: user9 이름:하성우	팔로잉취소
ID: user2 이름:정민경	팔로잉취소

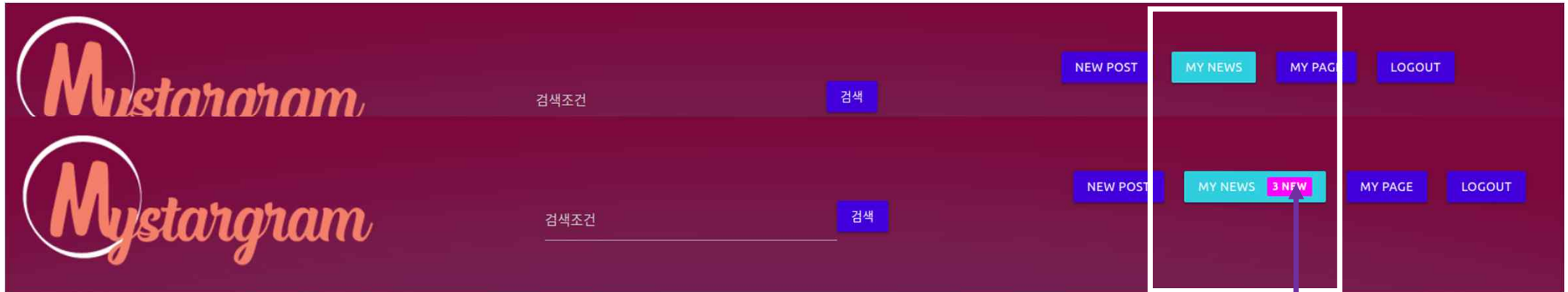
My Follower

ID: user4 이름:홍길동
ID: user7 이름:정병수
ID: user9 이름:하성우

```
def del_followship(request): #팔로워 취소
    id = request.POST.get("fid") #취소할 친구의 아이디를 포스트로 받기.
    ssid=request.session.get("id") #나의 세션정보.
    Follows.objects.filter(follower_id=ssid, followee_id=id).delete() #위와 매칭데이터 삭제.
    return redirect("mypage")
```

```
{% for x, name in my_followee %} {# 나의 팔로워 데이터의 출력 #}
<form name="del_followship" action="del_followship" method="POST">
    {% csrf_token %}
    <input type="hidden" name="fid" value="{{ x.followee_id }}">
    ID: {{ x.followee_id }} 이름:{{ name }}
    <input type="submit" value="팔로잉취소" class="btn"
        style="background-color: cornflowerblue; float: right; "
    ><br>
</form>
{% endfor %}
<script>
function del_fwee() { {# 나의 팔로워 취소신청 확인 #}
    if (confirm("팔로잉을 취소하시겠습니까?")) {
        document.del_followship.action = "del_followship";
    }
    console.log(document.del_followship);
    document.del_followship.submit();
}
```

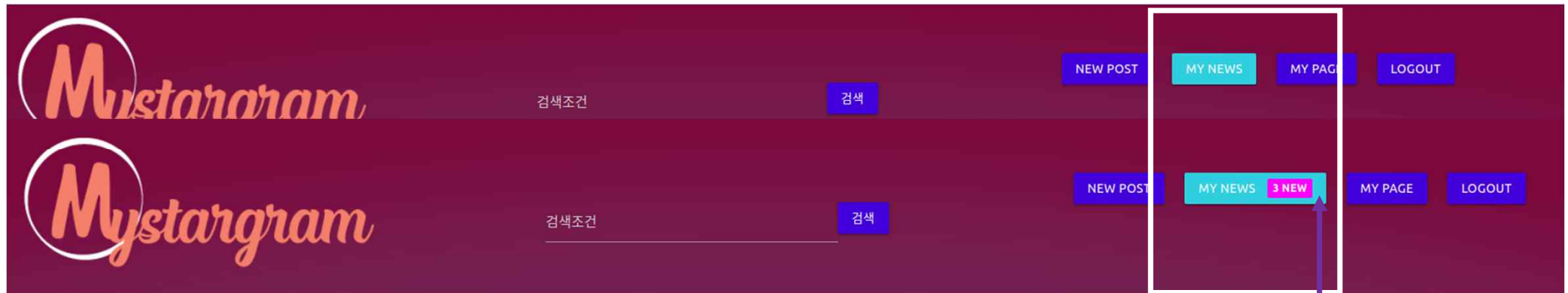
7)나의뉴스(뉴스배지)



`$(document).ready(function ()` //페이지 모두 읽고난다음에 아래의 함수를 실행.

```
{  
  var badge_text = $("#news_badge").text() //뉴스배지 컴포넌트를 변수에 담기.  
  var notification_counter = parseInt(badge_text); //위의 변수타입을 숫자로 넣고, 새로운 변수명 지정.  
  
  if (badge_text.length > 0 && notification_counter > 0) { // 0개 이상일때만 보여주기.  
    $("#news_badge").show(); //0개 이상이면 뉴스배지가 보여지고,  
  } else {  
    $("#news_badge").hide(); //아닌경우 보이지 않음.  
  }  
}
```

7)나의뉴스(뉴스배지)



```
setInterval(function () { //정해진 시간마다 자동호출 (매 10초로 설정)
```

```
$.get("/notification/count", function (data) { //notification/count 함수를 호출
```

```
if (data.notification_counter > 0) { //카운팅 결과값이 0보다 큰 경우,
```

```
$("#news_badge").text(data.notification_counter); //뉴스배지의 텍스트 값으로 주어지고,
```

```
$("#news_badge").show(); //뉴스배지가 보이게 처리.
```

```
});
```

```
}, 10 * 1000); // 10초마다
```

```
def count(request): #나의 뉴스 갯수 카운팅함수.
```

```
return JsonResponse({
```

```
    "notification_counter": Notification.objects.filter(userid__user_id=request.session["id"], seen=0).count()
```

```
}) #JsonResponse로 숫자를 리턴값으로 지정.
```


7)나의뉴스(뉴스배지)

```
var popOpen = false; //팝오픈 변수의 기본값을 false로 설정

$("#my_news").on("click", function () { //마이뉴스 버튼을 클릭할 때,
    popOpen = !popOpen; //팝오픈의 상태가 반대로 변경되고, (False -> True, True->False)
    if (popOpen) { //팝오픈이 True 상태일때,
        $.get("/notification/read_all", function (data) { // notification/read_all 함수를 호출, 성공하면 콜백함수(success)
            var notifications = data.notifications.map(function (notification) {
                return " " + notification.fromid + "님이 회원님의 게시물을 좋아합니다!";
            }).join("\n"); //notifications 변수에 map으로 리턴값을 필요데이터인 값을 골라내어, join으로 줄바꿈을 처리.

            $("#popoverb").text(notifications); //위 변수에 담긴 notifications를 팝오버 텍스트로 표시.
            $("#news_badge").hide(); // 뉴스배지다시숨긴다.
        });
        $("#popoverb").fadeTo(500, 1) //fadeTo를 이용해 투명도조절을 통해 보여지고,
    } else {
        $("#popoverb").fadeTo(500, 0) //해당상태가 아닌경우 투명하게 처리.
    }
}
```

Html에 포함되어있는JS스크립트 - >urls->
views.Function(&models) 함수 거쳐 dto 확인->
값을 다시 스크립트로 전달 받은 후 - >html로 출력



클릭되는 순간, 필요한 정보를 읽어오고,
Notification의 seen 컬럼을 1로 업데이트처리.
업데이트 후 더 이상의 count의 수는 0이므로
더 이상 카운팅 배지는 보여지지 않음.

```
def read_all(request): #나의 뉴스부분 해당데이터 읽기를 위한 함수.

    notification_objects = Notification.objects.filter(userid__user_id=request.session["id"], seen=0)
    #seen의 컬럼 조건을 boolean타입의 초기값을 0으로 할당, 해당데이터만 읽어옴.

    notifications = [] #리스트변수 선언.
    for notification in notification_objects:
        notifications.append({
            "fromid": notification.fromid.user_id,
            "note_type": notification.note_type
        }) #받은 객체를 반복문을 통해 배열에 담음.

    notification_objects.update(seen=1) #위의 처리 된 객체에 대해 db테이블의 seen컬럼 업데이트를 처리.
    return JsonResponse({"notifications": notifications})

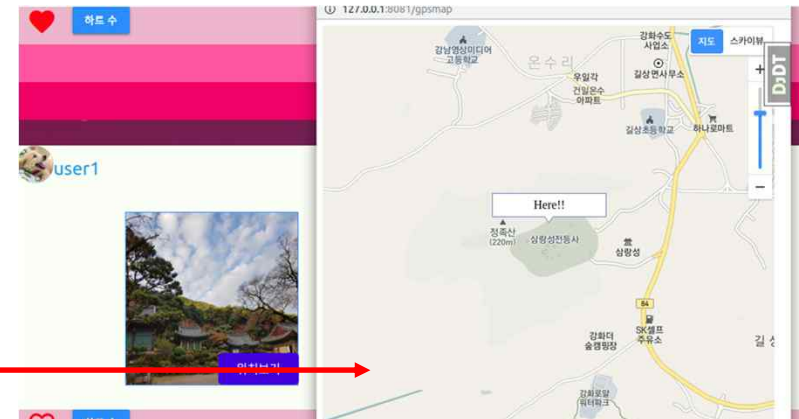
#카운팅과 마찬가지로 JsonResponse 형태로 리턴값을 지정함.
```

8) 이미지GPS추출



위치보기버튼 클릭->
urls에 있는 views의 함수에 도달(**def get_gps**)

함수에서 위도와 경도의 값을 얻어,
template html로 이동->
준비하고 있는 지도에서 데이터를 넘김.



```
<form action="{% url 'gpsmap' %}" method="POST" target="print_popup">
    {# name=gpsmap으로 연결, 팝업창으로 출력페이지 연결 설정. #}
    onsubmit="window.open('about:blank','print_popup','width=630px,height=630px');">
    {# 연결페이지로 오픈될 윈도우의 크기설정 #}
    {% csrf_token %}
    <input value="{{ post.img1 }}" type="hidden" name="the_pic"> {# views 함수로 넘길 img의 이름설정. #}
    <div class="bt_on_img_container"
        style="float: left; position: relative; ">
         {# 이미지출력. #}
        {% if 'gps' in post.img1%} {# 임의설정 이미지명에 gps가 포함되어 있는 경우, #}
        <button type="submit" class="bt_on_img_btn" style="">위치보기</button>
        {% endif %} {# 위치보기 버튼을 추가처리 #}
    </div>
</form>
```


8) 이미지GPS추출

① Urls를 통해 연결받은 views의 함수

```
def get_gps(request): #이미지의 위도경도 추출 시 호출함수.
    the_pic = request.POST.get("the_pic") #이미지의 파일명지정.
    ② info = get_exif('static/img/' + the_pic) #메타데이터 추출함수에 인자로 전달.
    ③ long = Longitude(info) #위의 함수 리턴값을 경도,
    ④ lat = Latitude(info) #위도에 각각 넣어 확보.
    result = decimal_form(lat, long) #최종 [위도, 경도]
    ⑤ gps = {'lat': result[0], 'long': result[1]} #최종값을 딕셔너리 형태로 전달.
    return render(request, 'init_map.html', gps) # 맵 출력을 위한 팝업창을 연결, 데이터전달.
```

```
③ def Longitude(y): #경도계산을 위한 함수.
    e = y["GPSInfo"]["GPSLongitude"]
    ref = y["GPSInfo"]["GPSLongitudeRef"]
    Longitude = (e[0][0] / e[0][1] +
                 e[1][0] / e[1][1] / 60 +
                 e[2][0] / e[2][1] / 3600
                 ) * (-1 if ref in ['S', 'W'] else 1)
    print("Longitude:", Longitude)
    return Longitude

④ def Latitude(y): #위도계산을 위한 함수.
    e = y["GPSInfo"]["GPSLatitude"]
    ref = y["GPSInfo"]["GPSLatitudeRef"]
    Latitude = (e[0][0] / e[0][1] +
                e[1][0] / e[1][1] / 60 +
                e[2][0] / e[2][1] / 3600
                ) * (-1 if ref in ['S', 'W'] else 1)
    print("Latitude:", Latitude)
    return Latitude
```

② def get_exif(filename): #이미지의 메타데이터 추출 라이브러리 사용 함수.

```
exif = Image.open(filename)._getexif() #함수파라미터값으로 들어온 이미지를 오픈.
if exif is not None:
    for key, value in exif.items(): #이미지에 포함된 메타의 키와 밸류를 분리하여
        name = TAGS.get(key, key) #각각 전달 처리.
        exif[name] = exif.pop(key)
    if 'GPSInfo' in exif: #필요한 정보의 키를 특정하여, (현 경우는 GPSInfo)
        for key in exif["GPSInfo"].keys(): #특정된 키 범위의 데이터를
            name = GPSTAGS.get(key, key) #오브젝트형태로 다시반복확보처리.
            exif["GPSInfo"][name] = exif["GPSInfo"].pop(key)
```

```
#받은 GPS의 자료로 위도, 경도를 계산하기위해 필요한 값은 있으나 해당 키의 명칭이 부정확하게
#표기되는 것을 발견하여, 해당 값에 필요한 명칭(키)를 정의하여 데이터를 추가삽입처리.
exif["GPSInfo"]["GPSLatitude"] = exif["GPSInfo"][2]
exif["GPSInfo"]["GPSLongitudeRef"] = exif["GPSInfo"][3]
return exif #GPS가 포함된 데이터를 리턴.
```