

[Voaltilty]

Second Django Project

[FBV을 이용한 CRUD]

안 병 욱

2019. 11. 22

목 차

1. 프로젝트 개요	p.4
1.1 프로젝트 목적	p.4
2. 프로젝트 요구분석	p.4
2.1 요구분석	p.4
2.2 개발에 필요한 도구	p.4
2.2.1 S3 Image Server	p.4
2.2.2 Bootstrap	p.4
2.2.3 Heroku	p.4
3.프로젝트 설계	p.5
3.1 DB	p.5
3.2 Work Breakdown Structure	p.6
3.3 DFD	p.7
3.3.1 Post DFD	p.7
3.3.2 Comments DFD	p.8
3.3.3 HitCount DFD	p.8

4.구현	p.9
4.1 Front-end	p.9
4.1.1 구현	p.9
4.1.2 구현 간 애로사항	p.9
4.2 Back-end	p.10
4.2.1 구현	p.10
4.2.2 구현 간 애로사항	p.19
4.3 Dev-Ops	p.20
4.3.1 구현	p.20
4.3.2 구현 간 애로사항	p.22
5. 프로젝트 마무리	p.23
6. 참고	p.23

1. 프로젝트 개요

1.1 프로젝트 목적

Function Base View에 대해 숙달, 다른 배포 환경 경험과 이전 프로젝트에서 부족했던 부분 보완하는 것에 목적을 두고 있습니다.

2. 프로젝트 요구분석

2.1 요구분석

CRUD를 FBV만을 이용한 View 구성, 조회수 기능, STATIC파일을 위한 Image server, Heroku.

2.2 개발에 필요한 도구

2.2.1 S3 Image Server

STATIC image file upload 속도 개선 목적.

2.2.2 BootStrap

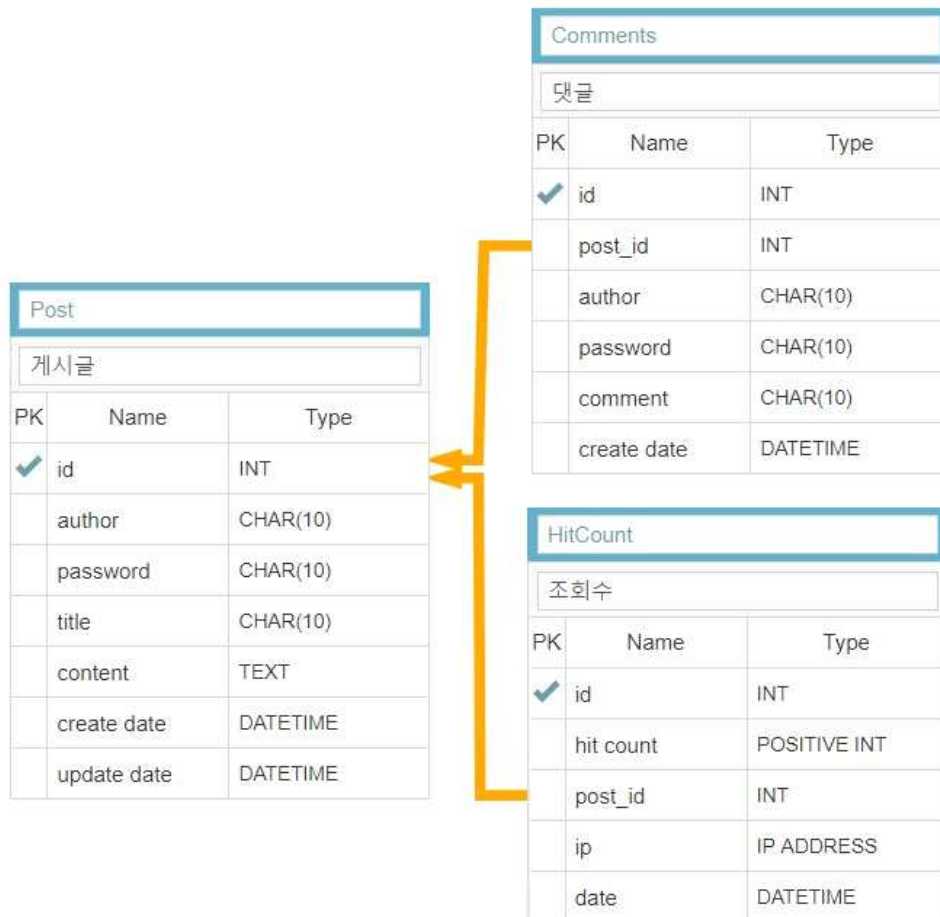
front-end 구성을 위해 사용.

2.2.3 heroku

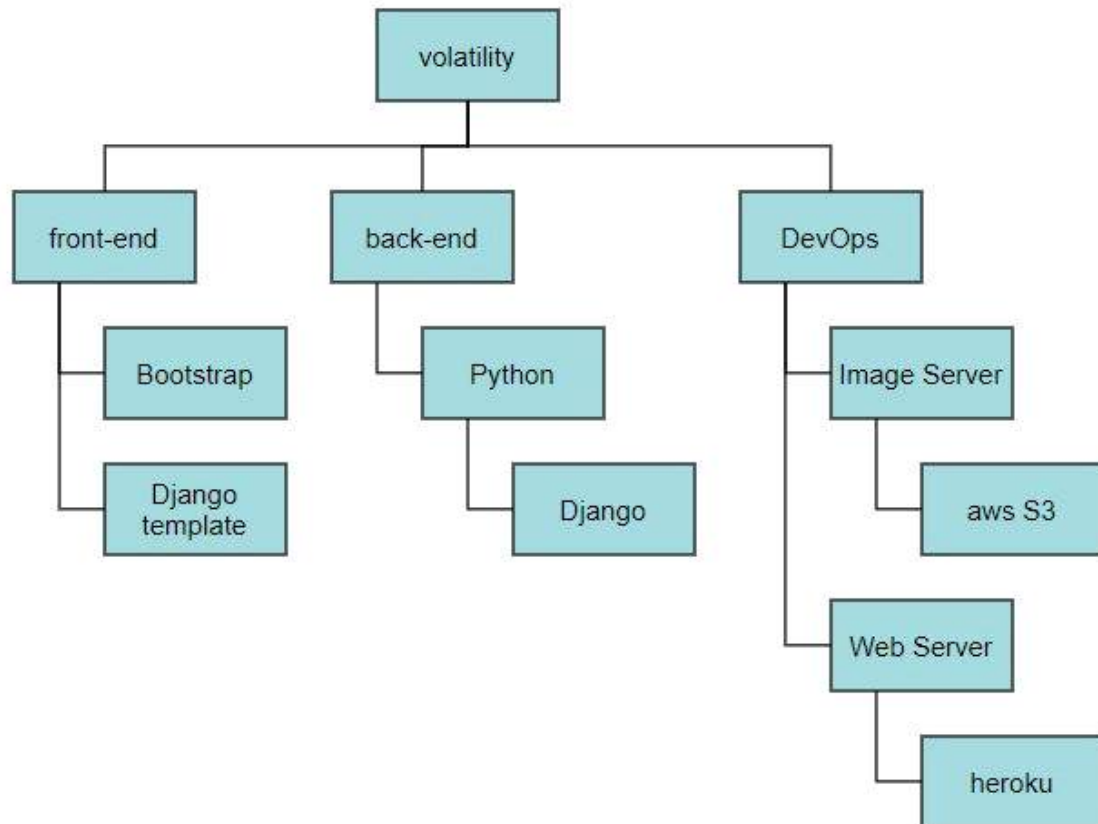
배포 Web Server로 heroku를 사용.

3.프로젝트 설계

3.1 DATABASE

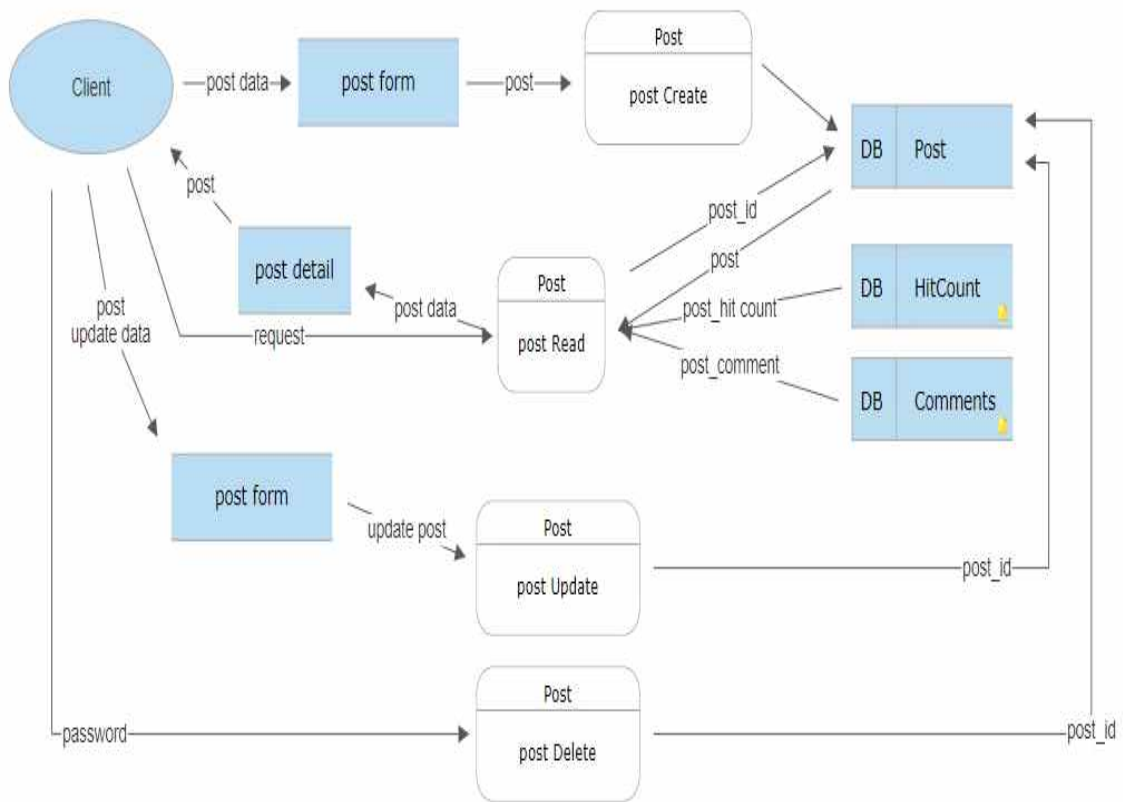


3.2 Work Breakdown Structure

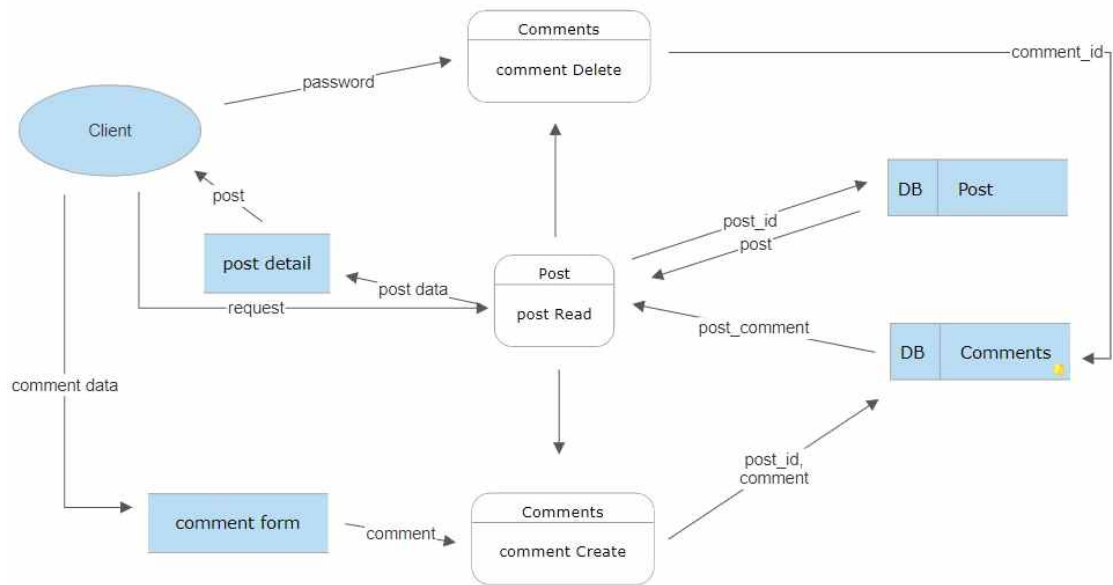


3.3 DFD

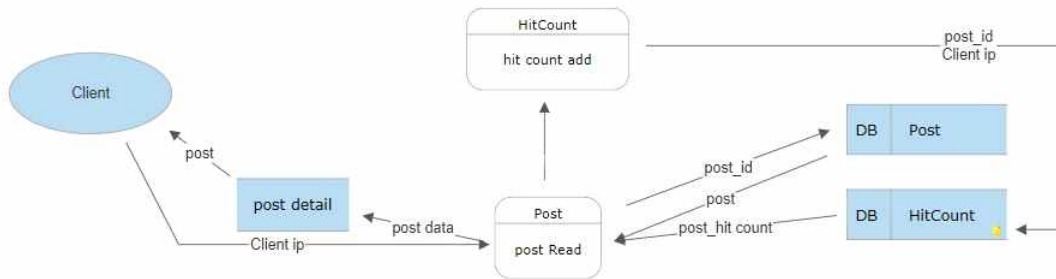
3.3.1 Post DFD



3.3.2 Comments DFD



3.3.3 HitCount DFD



4. 구현

4.1 front-end

4.1.1 구현

```
{% if messages %}
{% for message in messages %}
<script>alert('{{ message }}')</script>
{% endfor %}
{% endif %}

<!--삭제를 요청 할 경우 prompt로 입력받아서 form에 자동 저장후 submit.
      삭제 요청을 취소할경우 뒤로가기 -->
<script>

var delete_check = confirm('정말로 지우시겠습니까?');
if(delete_check){
    var password=prompt('글 작성시 입력한 비밀번호 입력하세요. ');
    document.getElementById('check_password').value=password;
    document.forms['passwordForm'].submit();
}
else {
    history.go(-1);
}
```

view에서 특정 상황 발생 시 messages 모듈을 통해 메시지 전달, front에서 메시지가 있을 경우 alert로 출력합니다.

prompt로 입력받은 값을 view로 전달하기 위해 form에 대입해주고 form을 submit해줍니다.

4.1.2 구현 간 애로사항

보통 prompt값을 view에 넘겨주는 건 ajax를 이용하기에 따로 추가적인 공부 필요성 느끼게 되었습니다.

4.2 Back-end

4.2.1 구현

```
def get_client_ip(request):  
    """클라이언트의 IP를 가져 오는 함수."""  
    x_forwarded_for = request.META.get('HTTP_X_FORWARDED_FOR')  
    if x_forwarded_for:  
        ip = x_forwarded_for.split(',')[0]  
    else:  
        ip = request.META.get('REMOTE_ADDR')  
    return ip
```

Client측의 IP를 META데이터에서

‘HTTP_X_FORWARDED_FOR’ get하기 위한 함수입니다.

존재할 경우 IP표기법 형식에 맞게 수정, 없을 경우

‘REMOTE_ADDR’ get해오게 됩니다.

```
def check_ip(request, model, pk):  
    """해당 IP로 조회수를 올린 적 있는지 확인, 만약 있다고 하면  
    조회수 Field는 그대로, 없으면 조회수를 올림."""  
    ip = get_client_ip(request)  
    queryset = model.objects.get_or_create(  
        ip=ip,  
        post_id=pk,  
        defaults={'ip': ip,  
                  'post_id': pk  
                }  
    )  
    return queryset
```

해당 IP로 조회 수 추가합니다. 이미 추가하였을 경우 get,
추가되지 않은 경우 create로 조회 수 추가합니다.

```
def index(request):

    today = timezone.now()
    yesterday = today - timezone.timedelta(hours=24)
    #지금을 기준으로 24시간 안에 작성된 Post만 필터링
    post = Post.objects.filter(update_date__range=(yesterday, today))
    #공지 사항 느낌.
    admin_post = Post.objects.filter(author='admin')
    form = SearchForm()
```

게시 글 READ 범위를 작성 시간 기준 24시간으로 제한 하지만, 관리자의 게시 글일 경우에는 항상 READ합니다.

```
if request.method == 'POST':
    form = SearchForm(request.POST or None)

    if form.is_valid():
        schType = '%s' % request.POST['search_type']
        schWord = form.cleaned_data['search_word']
        #schWord = '%s' % request.POST['search_word'] 와 같은 의미이지만 . form.cleaned_data
        # 더 안전하다고 하다. python에 맞게 형식도 바꿔준다고 함.
```

검색을 위한 form 검증과정입니다, schType은 검색키워드, schWord는 검색어. form.cleaned_data가 request.POST[]보다 안전하며 python에 맞게 형식변환.

```

def type_check(model, schType):
    if schType == 'author':
        search_list = model.objects.filter(Q(author__icontains=schWord),
                                             update_date__range=(yesterday, today))

    elif schType == 'title':
        search_list = model.objects.filter(Q(title__icontains=schWord),
                                             update_date__range=(yesterday, today))

    elif schType == 'content':
        search_list = model.objects.filter(Q(content__icontains=schWord),
                                             update_date__range=(yesterday, today))

    return search_list

context={}
context['admin_post'] = admin_post
context['form'] = form
context['post'] = post
context['search_word'] = schWord
context['search_list'] = type_check(Post, schType)

if not context['search_list'].exists():
    """만약 주어진 키워드로 검색된 데이터가 없다면
    message 출력 하기 위해 message를 프론트단에 보내줌."""
    messages.warning(request, schWord + '에 관한 검색결과가 없습니다.')
    return redirect('/')

return render(request, 'blog/index.html', context)

```

검색하려는 키워드 작성자, 제목, 내용 중에 확인합니다. 만약 검색 결과가 없을 경우 messages모듈을 통해 에러 메시지 template에 전달하게 됩니다.

```

def detail(request, pk):
    post = get_object_or_404(Post, pk=pk)
    #외래키로 외래모델의 기본키를 가져올때는 '필드명_id'
    comments = Comment.objects.filter(post_id=pk)
    check_ip(request, HitCount, pk)
    hit = HitCount.objects.filter(post_id=pk)
    random_name = str(round(random.random() * 10000))
    # 글쓴이의 이름을 익명+random숫자로 지정함

    if request.method == 'POST':
        form1 = CommentForm(request.POST or None)

        if form1.is_valid():
            form1.instance.post_id = pk
            form1.save()
        else:
            form1 = CommentForm(initial={'author': '익명'+random_name})

    return render(request, 'blog/detail.html', {'post': post, 'comments': comments,
                                                'form1': form1, 'hit': hit})

```

게시 글 detail을 위한 view입니다. 게시 글뿐만 아니라
게시 글 PK값으로 저장되어 있는 댓글과 조회 수를 불러오며
댓글 이름 기본값은 '익명 + 랜덤 숫자 4자리' 입니다.

```
def post_create(request):
    random_name = str(round(random.random() * 10000))
    # 글쓴이의 이름을 익명+random숫자로 지정함

    if request.method == 'POST':
        form = PostForm(request.POST or None)
        if form.is_valid():
            form.save()
            return redirect('blog:index')
    else:
        form = PostForm(initial={'author': '익명' + random_name})

    return render(request, 'blog/form.html', {'form': form})
```

게시 글 작성 view

```
def post_update(request, pk):
    post = get_object_or_404(Post, pk=pk)

    if request.method == 'POST':
        form = PostForm(request.POST or None)

        if form.is_valid():
            form.save()
            return redirect('blog:index')
    else:
        # instance로 이미 작성된 post를 보내줌
        form = PostForm(instance=post)

    return render(request, 'blog/form.html', {'form': form})
```

게시 글 수정 view. 이미 작성된 게시 글을 instance로 전달합니다.

```

def post_delete(request, pk):
    post = get_object_or_404(Post, pk=pk)
    form = PasswordCheck(request.POST or None)

    # 비밀번호를 주고 받기때문에 정석적인 방법이다.
    if request.method == 'POST':
        form = PasswordCheck(request.POST or None)
        if form.is_valid():
            if post.password == form.cleaned_data['check_password']:
                post.delete()
                return redirect('/')
            else:
                messages.warning(request, '비밀번호가 틀립니다')

    return render(request, 'blog/delete.html', {'post': post, 'form': form})

```

prompt로 입력받은 password를 form에 대입 후 view로 전달받아 form 검증하는 절차입니다. 틀릴 경우 messages 모듈을 통해 메시지 전달합니다.


```
def comment_delete(request, pk):
    comment = get_object_or_404(Comment, pk=pk)

    #비밀번호를 주고 받는 것이기 때문에 GET통신은 보안적으로 매우 취약하다고 한다.
    #post_delete()에서 정석적인 방법을 사용하고있다.
    if request.method == 'GET':
        password = request.GET.get('password')
        if comment.password == password:
            comment.delete()
        else:
            messages.warning(request, '비밀번호가 틀립니다.')

    return redirect('blog:detail', pk=comment.post.pk)
```

댓글 삭제 view입니다. detail에서 입력받은 password를 전달받아 검증합니다. 일반적인 경우 외부에 노출되지 않아야 할 정보는 GET method를 사용하지 않는 것이 정석입니다.

```
CHOICES = [
    ('author', '작성자'),
    ('title', '제목'),
    ('content', '내용'),
]

class SearchForm(forms.Form):
    search_type = forms.ChoiceField(choices=CHOICES, widget=forms.Select, label='')
    search_word = forms.CharField(label='Search Word')
```

검색 form. search_type을 통해 어떤 키워드로 검색할지 입력받습니다. 그에 따른 선택 값들은 CHOICES로 정의되어 있습니다.


```

class PostForm(forms.ModelForm):

    class Meta:
        model = Post
        exclude = ['created_date', 'update_date']
        #fields = ['author', 'password', 'title', 'content'] 와 같은의미

        #bootstrap에 맞는 class를 사용하기 위해 widgets에서 form의 class를 정의해놓음.
        widgets = {
            'author': forms.TextInput(
                attrs={'class': 'form-control'}
            ),
            'password': forms.PasswordInput(
                attrs={'class': 'form-control', 'placeholder': 'Your Password'}
            ),
            'title': forms.TextInput(
                attrs={'class': 'form-control', 'placeholder': 'Title'}
            ),
            'content': forms.Textarea(
                attrs={'class': 'form-control', 'placeholder': 'Content'}
            ),
        }

```

게시 글 작성 form입니다. create date와 update date는 입력받지 않으며, 각각의 폼에 bootstrap에서 사용하는 class값을 주기 위해 attrs 정의하고 있습니다.

```
def clean_author(self):
    """author가 admin인 것은 관리자이기 때문에 사용자는
    admin을 사용하지 못하게 하는 메소드.
    author필드를 확인"""
    cd = self.cleaned_data
    if cd['author'] == 'admin':
        raise forms.ValidationError("'admin'이라는 닉네임 사용하지 마세요.")
    return cd['author']
```

django에서 이미 정의된 메소드로 clean_tablename으로 내부함수를 선언할 경우 author에 대해 추가적인 기능을 따로 만들겠다는 의미를 가지고 있습니다.

여기서는 작성하는 자가 관리자 닉네임으로 글을 작성하지 못하게 하려는 목적입니다.

```
class CommentForm(forms.ModelForm):
    password = forms.CharField(label='Password', max_length=10, widget=forms.PasswordInput(attrs={'size': 10}))

    class Meta:
        model = Comment
        exclude = ['post', 'create_date']
        widgets = {
            'author': forms.TextInput(attrs={'size': 8}),
            'comment': forms.TextInput(attrs={'size': 35})
        }

    def clean_author(self):
        """author가 admin인 것은 관리자이기 때문에 사용자는
        admin을 사용하지 못하게 하는 메소드.
        author필드를 확인"""
        cd = self.cleaned_data
        if cd['author'] == 'admin':
            raise forms.ValidationError("'admin'이라는 닉네임 사용하지 마세요.")
        return cd['author']
```

댓글 작성 form

```
class PasswordCheck(forms.Form):
    check_password = forms.CharField(max_length=10,
                                     widget=forms.PasswordInput(attrs={'id': 'check_password'}),
                                     label="작성시 사용한 비밀번호 입력하세요")
```

비밀번호가 일치하는지 확인하기 위한 form입니다. 비밀번호 form이기에 widget으로 PasswordInput을 이용합니다.

4.2.2 구현 간 애로사항

보통 조회 수 기능은 인터넷 쿠키를 이용하지만, 인터넷 쿠키를 이용할 경우 쿠키 삭제 후 다시 조회를 시도할 경우 조회 수가 증가하는 단점. 이에 IP로 조회 수 기능을 구현하였지만, 각각의 장치들이 Mac 주소처럼 고유의 IP를 가진 것이 아니기에 문제 발생할 수 있다고 생각합니다.

검색기능은 대부분 GET 방식을 이용하며 password check 같은 보안이 필요한 정보는 POST 방식을 이용해야 된다고 합니다.

4.3 Dev-Ops

4.3.1 구현

1. AWS S3 bucket 생성
2. IAM 사용자 그룹 등록, ACCESS KEY, SECRET ACCESS KEY 저장.

```
"""settings.py"""
AWS_ACCESS_KEY_ID = 'IAM 액세스 키 ID'
AWS_SECRET_ACCESS_KEY = '비밀 액세스 키'
AWS_REGION = 'ap-northeast-2'
AWS_STORAGE_BUCKET_NAME = '버킷 이름'
AWS_S3_CUSTOM_DOMAIN = '%s.s3.%s.amazonaws.com' % (AWS_STORAGE_BUCKET_NAME, AWS_REGION)
AWS_S3_OBJECT_PARAMETERS = {
    'CacheControl': 'max-age=86400',
}
AWS_DEFAULT_ACL = 'public-read'
AWS_LOCATION = 'static'

STATIC_URL = 'https://%s/%s/' % (AWS_S3_CUSTOM_DOMAIN, AWS_LOCATION)
STATICFILES_STORAGE = 'storages.backends.s3boto3.S3boto3Storage'
```

3.django project settings.py에 정의.

4.heroku 계정 생성, heroku, git 설치

5.django module 추가 설치.

1.dj-database-url : 데이터베이스 환경 변수를 설정할 수 있게 도와주는 유틸리티

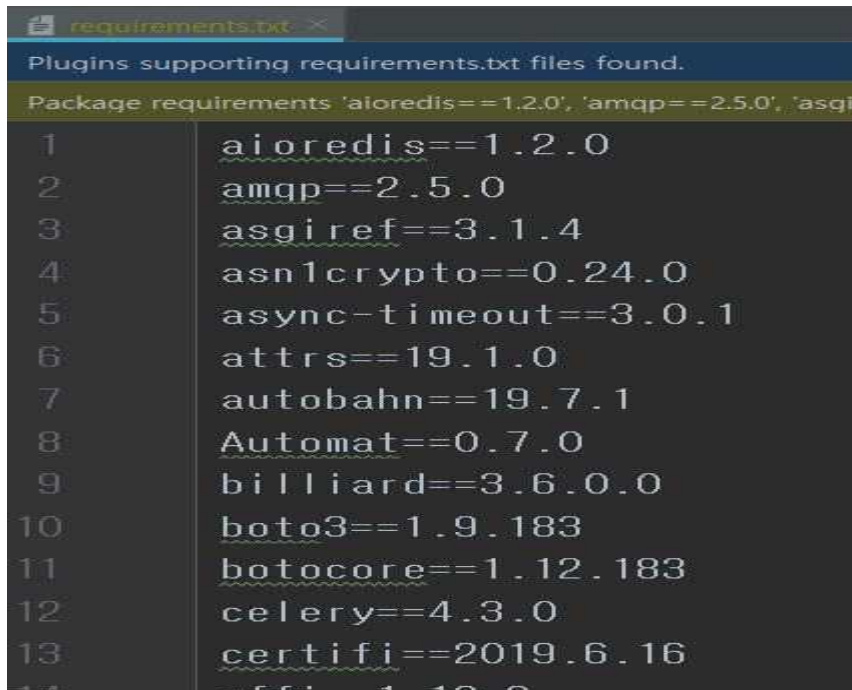
2.gunicorn : WSGI middleware

3.whitenoise: 정적 파일의 사용을 돕는 미들웨어

4.psycopg2-binary : PostgreSQL을 사용하기 위한 모듈

6.project에 필요한 module 문서화

pip freeze > requirements.txt



```
requirements.txt
Plugins supporting requirements.txt files found.
Package requirements 'aioredis==1.2.0', 'amqp==2.5.0', 'asgi
1  aioredis==1.2.0
2  amqp==2.5.0
3  asgiref==3.1.4
4  asn1crypto==0.24.0
5  async-timeout==3.0.1
6  attrs==19.1.0
7  autobahn==19.7.1
8  Automat==0.7.0
9  billiard==3.6.0.0
10 boto3==1.9.183
11 botocore==1.12.183
12 celery==4.3.0
13 certifi==2019.6.16
14
```

7.settings.py 설정

```
###settings.py###
import dj_database_url
DEBUG = False
ALLOWED_HOSTS = ['*']
DATABASES['default'].update(dj_database_url.config(conn_max_age=500))
MIDDLEWARE = [
    'whitenoise.middleware.WhiteNoiseMiddleware',
]
```

8.Procfile 생성

```
###Procfile###
web: gunicorn config.wsgi
```

9.runtime환경 문서 생성

```
###runtime.txt###  
python-3.7.1
```

10.heroku login, .gitignore 생성

```
###.gitignore###  
*.pyc  
*~  
/venv  
__pycache__  
db.sqlite3  
.DS_Store
```

11.배포

```
###heroku 업로드###  
$ git init  
$ git add -A  
$ git commit -m "heroku upload"  
$ heroku create volatility-django  
$ git push heroku master  
####heroku 초기화###  
$ heroku run python manage.py migrate  
$ heroku run python manage.py createsuperuser
```

4.3.2 구현 간 애로사항

heroku는 linux 기반 web server이기 때문에 windows 환경에서 배포를 진행하려면 windows에 종속된 모듈들을 제거해야 합니다.

5. 프로젝트 마무리

프로젝트를 진행하면서 간단한 CRUD를 만드는데 생각하는 대로 쉽게 되지 않는구나 느끼고 많이 부족하며 앞으로 배워야 할 것이 많이 있구나, 아직 우물 안 개구리라고 느낀 프로젝트입니다. AJAX, HTTP, 배포, python 문법 등등 공부를 더 진행하고 나서 다음 프로젝트를 진행할까 합니다.

6. 참고

Dev-Ops :

DIGITAL BOOKS 배프의 오지랖 파이선 웹 프로그래밍 -저자 배프

DFD Image:

<https://www.smartdraw.com/data-flow-diagram/>

<http://cjmyun.tripod.com/Knowledgebase/DFD.htm>

work break down Image :

<https://online.visual-paradigm.com/>

DB Image :

<http://aquerytool.com/>