

[Voaltilty]

Second Django Project

[FBV을 이용한 CRUD]

안영욱

2019. 11. 19

목 차

1. 프로젝트 개요	p.4
1.1 Excutive Summary	p.4
1.2 프로젝트 동기	p.4
1.2 프로젝트 목적	p.5
1.3 프로젝트 기대효과	p.5
2. 프로젝트 요구분석	p.5
2.1 요구분석	p.5
2.2 개발에 필요한 도구	p.6
2.2.1 S3 Image Server	p.6
2.2.2 Bootstrap	p.6
2.2.3 Heroku	p.6
3.프로젝트 설계	p.7
3.1 DB	p.7
3.2 Work Breakdown Structure	p.8
3.3 DFD	p.9
3.3.1 Post DFD	p.9
3.3.2 Comments DFD	p.10
3.3.3 HitCount DFD	p.10

4.구현	p.11
4.1 Front-end	p.11
4.1.1 구성	p.11
4.1.2 구현	p.11
4.1.2 구현 간 에러 사항	p.12
4.2 Back-end	p.13
4.2.1 구성	p.13
4.2.2 구현	p.14
4.2.3 구현 간 에러 사항	p.23
4.3 Dev-Ops	p.23
4.3.1 구성	p.24
4.3.2 구현	p.25
4.3.3 구현 간 에러 사항	p.29
5. 프로젝트 마무리	p.30
6. 참고	p.31

1. 프로젝트 개요

1.1 Executive Summary

dstagram이라는 현존하는 instagram을 모방하여 웹을 제작하던도중 스토리 기능을 어떻게 구현할까 고민했던 적 이 있다. 그 당시에는 간단한 CRUD 조차 만들지 못하고 그저 책만 따라가는 수준이였고, 역시나 허술하고 얇게 쌓여진 지식은 금방 밀천이 들어났다. 그 당시 고민거리는 어떻게 하면 내가 등록한 글을 작성기간 24시간 후 삭제 되게 할까? 였고, python의 datetime 모듈을 이용해 구현 하였지만, CBV에서 queryset을 어떻게 사용하나? 에 문제가 있었고 이 문제는 해결하려는 의지도 없이 포기해버렸었다. 그리고 처음부터 다시시작하자 는 생각으로 하나하나 정확히 이게 무엇인가 공부했던거 같다. 그 결과 CRUD는 어느정도 자신감이 생겼지만, 공부를 하면 할수록 해야 할 공부는 많으며 나의 지식은 정말 보잘 것 없다고 느꼈다.

1.2 프로젝트 동기

CBV을 이용한 게시판을 제작하던도중 나중에는 FBV만을 이용한 게시판을 만들어야겠다고 생각을 했고, 이번 프로젝트는 거기서 시작되었으며 FBV만 사용해서 구현하였다.

1.3 프로젝트 목적

FBV만을 이용한, 즉 함수형 View로만 구성된 웹사이트를 목표로 두었고, 이전 웹사이트에서 부족했던, 아쉬웠던 부분을 보충하고 사용하지 않았던 구현 방식을 구현하려 했다.

1.4 프로젝트 기대효과

이전에 FBV를 보면 두려움부터 앞섰던 것 같다. CBV는 정형화 되어 있지만 FBV는 개발자가 하나하나 구현해야 된다는 압박감 때문이었다. 하지만 하나둘 구현해보고 CBV와 같은 기능을 하는 FBV를 만들고 났을땐 ‘뭐야 크게 대단한건 없잖아?’ 라고 느끼게 되었다.

2. 프로젝트 요구분석

2.1 요구분석

우선, 이전의 웹사이트를 구현할 때 STATIC파일들을 로컬에 위치시켰었다. 그 결과 프론트단의 백그라운드 이미지들을 불러오는데 생각보다 오랜시간이 걸렸고 다음 프로젝트에선 S3서버에 STATIC파일들을 올려야겠다 구상을 하였었다.

또한 자주 사용하는 Fow.tv 사이트의 댓글관련 기능들을 클론코딩하기 위해 노력하였고, Disqus를 사용하는 대신 직접 개발한 댓글달기 기능을 사용해야겠다 생각했다.

이전 웹사이트에서는 없는 조회수 기능도 구현을 하려고 했으며 구글링 해본 결과 기존의 방법과는 다르게 시도했다.

2.2 개발에 필요한 도구

2.2.1 S3 Image Server

CRUD에 사진 올리기를 구현하진 않았지만 STATIC이미지들을 보다 빠르게 불러오기 위해 S3서버를 사용했다.

2.2.2 BootStrap

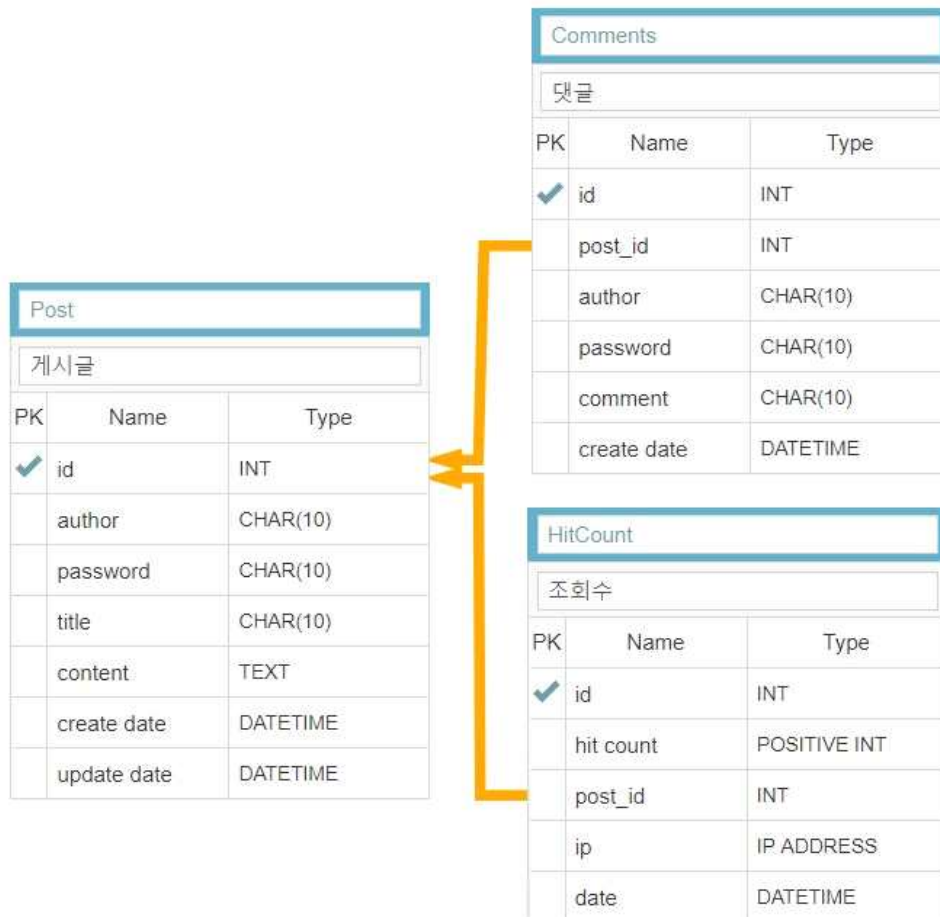
프론트단을 조금이나마 더 깔끔하고 이쁘게 꾸미기 위해 이번에도 BootStrap을 사용하였다.

2.2.3 heroku

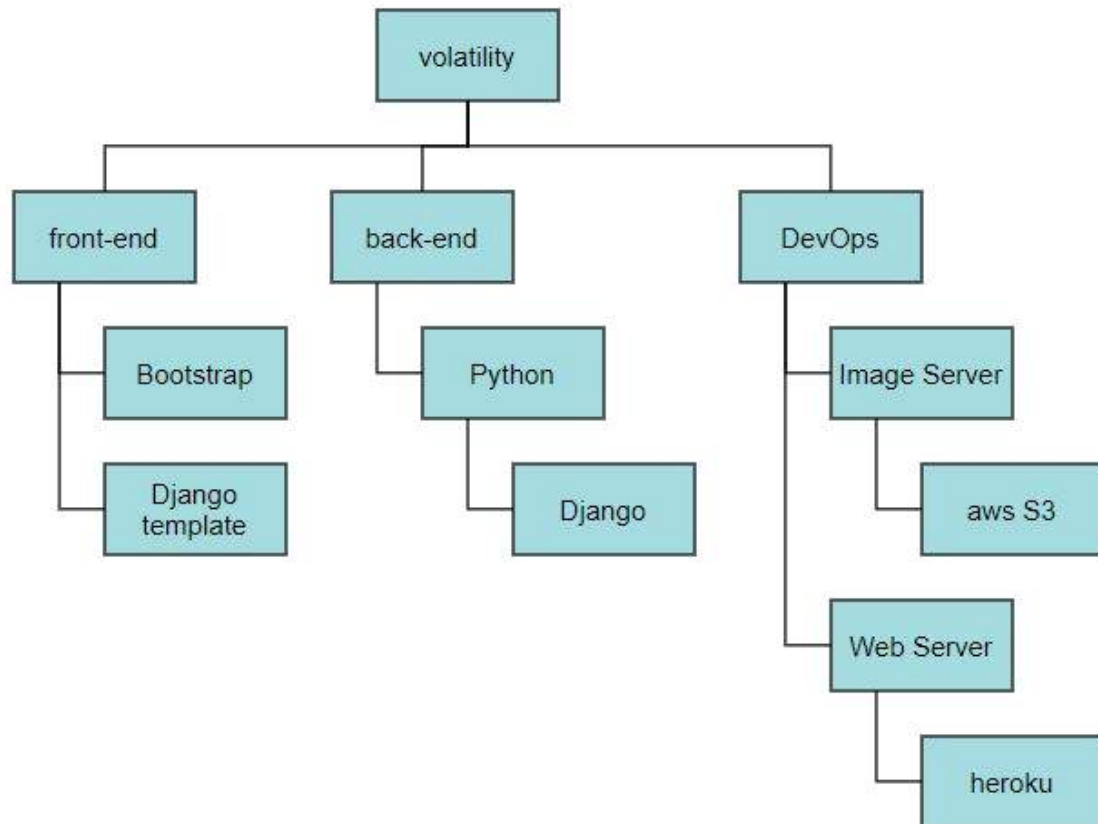
이전 웹사이트 배포때 실패한 heroku를 이용했다.
pythonanywhere에 비해 거의 모든 부분이 자동화 되어있는거 같다. 개발자가 따로 크게 구현하는 것 없이 Git에 push하듯 heroku repository에 push만 해주었다.

3.프로젝트 설계

3.1 DATABASE

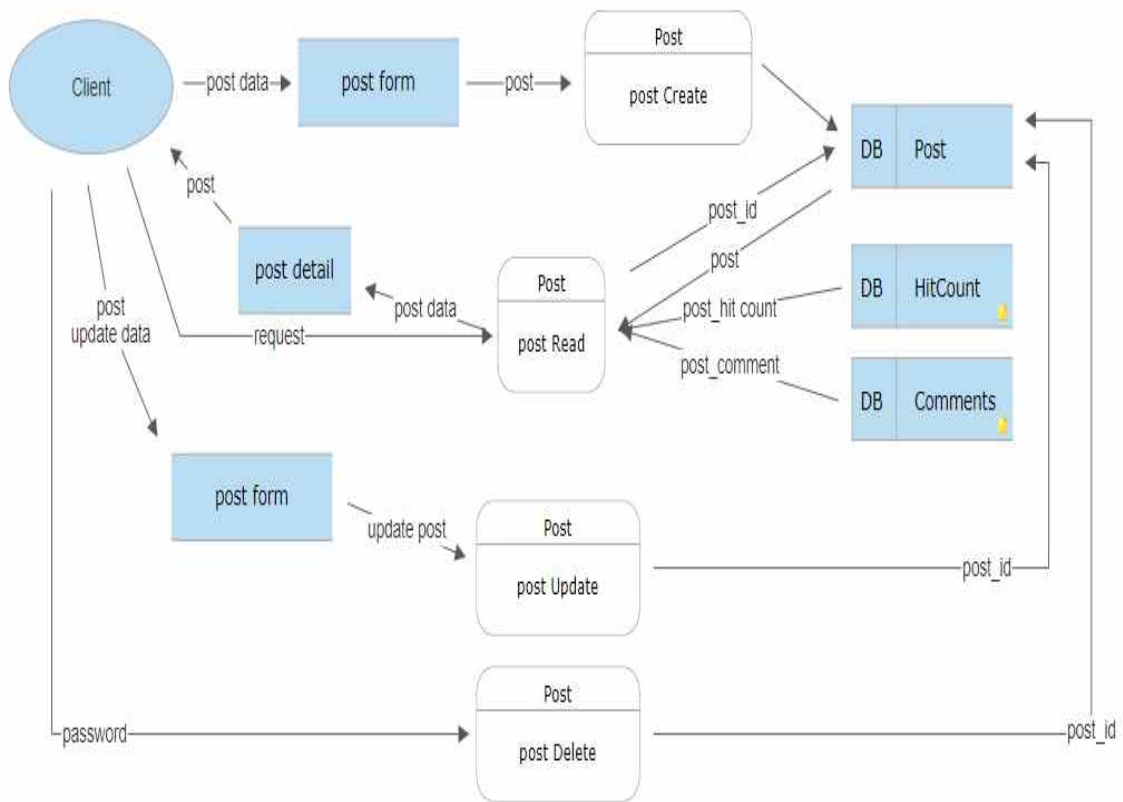


3.2 Work Breakdown Structure

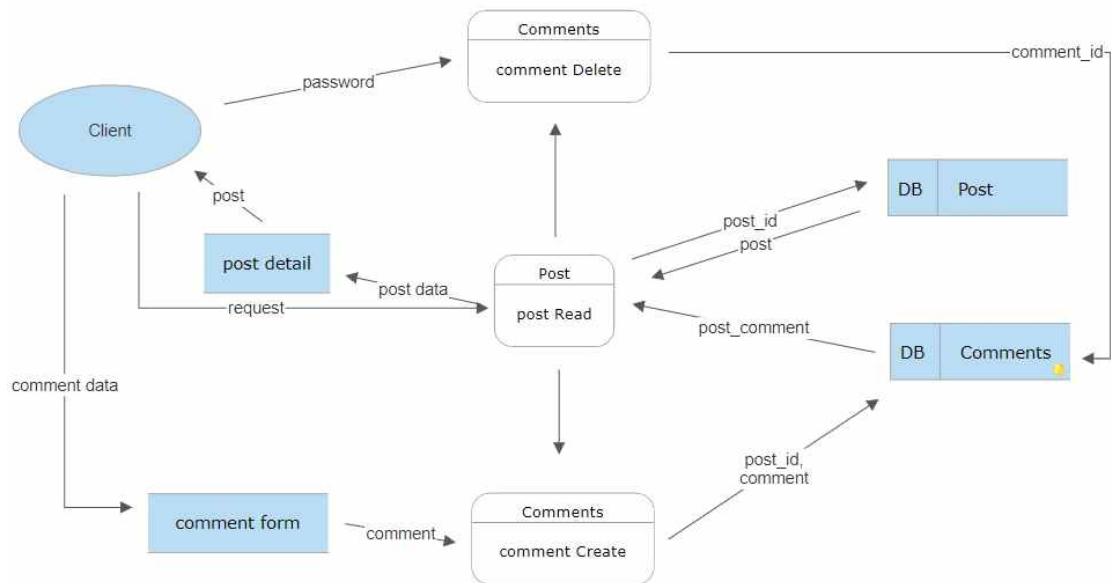


3.3 DFD

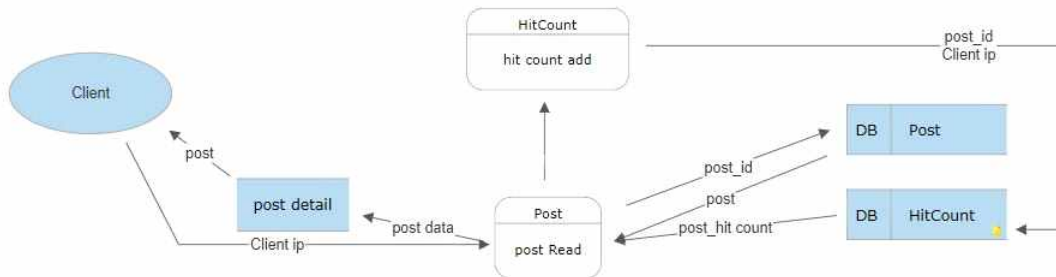
3.3.1 Post DFD



3.3.2 Comments DFD



3.3.3 HitCount DFD



4. 구현

4.1 front-end

4.1.1 구성

이번 프로젝트에서도 Bootstrap을 활용한 front-end를 구성하였다. 사용한 bootstrap은 startbootstrap.com 사이트의 agency 테마를 사용하였다.

Base.html을 홈페이지를 나타내는 home 기본으로 게시글 목록이 보여지는 index, 게시글 추가,삭제 하는 form과 delete, 게시글 상세보기인 detail html으로 구성하였다.

4.1.2 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);
}
```

front-end 부분은 특별히 바뀐점은 없으며 bootstrap을 이용하였기에 손댄 부분도 많이 없었다. 하지만 이전에 사용해보지 않은 messages 모듈을 사용하였다. view에서 messages 모듈을 통해 message를 보내면 front부분에서 messages가 있는지 체크후 있다면 alert하는 방식으로 사용했다.(ex:검색어가 있는지 없는지, 비밀번호가 맞는지 틀린지)

또한 fow.tv에서 비밀번호 입력을 prompt로 받듯 비슷하게 구현해 보았다. prompt로 입력받은 값을 getElementByld를 통해 form에 전달해주고 그 후에 form을 submit하게 하는 방식이다.

4.1.3 구현 간 에러 사항

처음 agency 샘플템플릿을 보았을 때 Modal을 사용하고 있는점이 매우 흥미로워 보였다. url 이동없이 새로운 사이트를 띄워주는 효과가 있어보였기 때문이었다. 그래서 index.html에서 modal을 이용해 detail.html의 내용까지 띄우려 했으나 back-end적으로 생각을 한다면 index view에 detail view의 기능도 구현 해야하기 때문에 처음 생각한 프로젝트 방안과는 달라서 다른 방법이 없나 찾아보았지만, 결론적으로 url이동 없이는 불가능하였다.

fow.tv 사이트에서는 댓글 삭제시 비밀번호 입력을 js의 prompt로 입력을 받았다. 같은 아이디어로 만들고자 js의 prompt로 입력받은 value를 django view로 가져오고 싶었지만 보통의 방법은 ajax를 사용하는 것이었고 아직 ajax에 대해

제대로 공부한 것이 없어서 다른 방법을 생각하던 도중 prompt의 값을 form에 getElementById방법으로 값을 넘겨주고 view단에서는 그 form을 가져오는 식으로 해결하였다. 뿐만아니라 자동새로고침 기능또한 만들고 싶었지만 역시 ajax가 필요했다.

이번 프로젝트 front-end를 진행하면서 ajax 공부의 필요성에 대해 절실히 느끼게 되었다.

4.2 Back-end

4.2.1 구성

앞서 말했듯이 이번 프로젝트는 FBV만 사용하기로 하였다. 계기로는 구글링 하며 공부하던 도중 어느분이 작성한 'CBV가 확장성에 좋고 다른 여러 가지 장점이 있는건 사실이지만, FBV도 코딩할줄 모르고 CBV를 사용한다면 코드가 개판이 될 수 있다' 글귀를 보았고 충격아닌 충격을 받아 간단한 CRUD이긴 하더라도 FBV로만 코딩해봐야겠다 생각했다.

저번 프로젝트에서 없던 기능을 구현했다라고 한다면 client의 ip를 가져오는 것을 들 수 있다. 조회수 기능을 구현하기 위해 구글링해본 결과 보통의 방법은 인터넷 쿠키를 이용한 방법이었다. 그에 대한 단점은 쿠키를 삭제후 다시 조회를 시도 할 경우 조회수는 또 오르게 되었다. 그렇기에 다른 방법이 무엇이 있을까 하고 생각하던도중 client의 ip를 이용해 조회수를 늘리는 것을 생각하게 되었다.

4.2.2 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
```

client의 ip를 가져오기 위해 META 데이터에서 HTTP_X_FORWARDED_FOR를 get해오고 그것을 ip표기 방식에 따라 구분을 지어주며 만약 해당 데이터가 없을시 META 데이터의 REMOTE_ADDR를 get 해온다.

```
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 해당 post의 조회수를 올렸는지 체크한다. 이때 get_or_create로 올리지 않았을 경우 새로운 데이터를 만들어 조회수를 올리고 이미 올렸을 경우에는 올린 데이터를 반환한다.

```
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()
```

index view 부분이다. 우선 게시글이 작성된 기준으로 24시간이 지나면 Read되지 않아야 하기 때문에 현재시간에서 -24시간 한 범위까지만 필터링한다. 또한 관리자의 게시글은 항상 보여지기 위해서 작성자가 'admin' 일 경우 또한 따로 필터링을 진행하였다.

```
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 한다. python에 맞게 형식도 바꿔준다고 함.
```

이번 사이트에서 검색또한 POST 방식으로 구현하였다. 여기서 눈여겨 볼 점은 form.cleaned_data[]인데, request.POST[] 방법보다 더 안전하며 python에 맞는 형식으로 가져온다고 구글링을 통해 알게되었다.

```

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)

```

type_check 함수는 검색하기 위한 키워드가 무엇인지 확인하는 함수이다. 검색하려는 사용자가 작성자,제목,내용 키워드로 검색을 하면 그 검색키워드에 맞는 내용으로 필터링 하기 위해 구성하였다. 또한 검색결과물이 없을 경우 messages모듈을 통해서 front단에 메시지를 전달한다.


```

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})

```

post Read 기능을 하는 view이다. 댓글과 조회수 관련은 post를 외래키로 가지고 있기 때문에 post_id로 필터링을 해주었고, 게시글을 작성할 때 작성자 이름을 '익명+랜덤숫자4글자'로 조합하기 위해 random_name 변수를 만들었다.

```
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})
```

post를 작성하는 view이다.

```
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})
```

post를 수정하는 view이다. 이미 작성된 데이터를 form에 미리 instance로 주고 있다.

```

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})

```

post를 삭제하는 뷰이다. PasswordCheck form을 이용하여 삭제를 요청한 사용자는 password를 입력하고 DB에 저장된 post의 비밀번호가 일치 할 경우 post를 삭제하도록 진행하며 비밀번호가 일치하지 않을 경우 messages모듈을 통해 메시지를 전달한다.

```
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)
```

댓글을 삭제하는 뷰이다. 따로 template은 가지지 않으며 detail.html에서 GET method값을 전달한다. 여기서 문제점은 이러한 비밀번호 검증을 위한 방식은 POST로 사용되어야 한다. GET방식은 url을 통해 입력받은 값을 전달하기에 보안상 매우 취약 할 수 밖에 없다. 보통 GET method는 검색기능이나 입력값이 보안될 필요가 없는 경우 사용하는거 같다.

```
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')
```

검색을 하기 위한 form이다. search_type을 통해 어떤 키워드로 검색 할지 입력받는다, 그에 따른 선택값들은 CHOICES로 정의한다.

```

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'}
            ),
        }

```

post를 작성하기 위해 준비한 form이다. create date와 update date는 입력받지 않으며 각각의 폼에 bootstrap에서 사용하는 class값을 주기위해 attrs값을 주고 있다.

```
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']
```

django에 이미 정의된 메소드로 clean_talbenamewith으로 내부함수를 선언 할 경우 author에 대해 추가적인 기능을 따로 만들겠다는 의미로 받는다고한다. 여기서는 작성하는 자가 관리자 닉네임으로 글을 작성하지 못하게 하려는 목적으로 만들었다.

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

    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']
```

댓글을 작성하기 위한 form이며, PostForm과 크게 다른 부분은 없다.

```
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 form이다.

비밀번호form이기에 widget으로 PasswordInput을 이용하였다.

4.2.3 구현 간 에러 사항

게시물이 작성된지 24시간 기준으로 READ되지 않게 하려고 사용한 모듈은 datetime모듈이었다, 하지만 django에서는 timezone 모듈을 사용하도록 하는데 그 이유는 datetime으로 현재시간 등등을 가져 올 경우 로컬 시간을 가져오는게 아닌 UTC 기준의 시간을 가져오므로 시간에 오차가 있을 수 있다. 즉 전 세계의 기준시간대와 로컬 시간대가 다를 경우 문제가 발생 할 수 있기에 디버깅 에러는 안나지만 경고가 발생한다.

또한 하나의 view에서 여러개의 같은 방식의 request method를 어떻게 넣을지도 에러 사항이었다.

조회수 기능을 구현 하는것도 많은 걱정이 있었다. 보통의 방법은 인터넷 쿠키를 이용하지만, 그의 단점으로는 쿠키를 삭제후 다시 조회 할 경우 조회수가 올라간다는 점이 있었다. 그래서 생각한 것이 사용자의 IP를 추출해서 해당 게시물에 사용자 IP가 있다면 조회수를 늘리지 않는 방법을 사용하였지만 이 역시 문제는 있었다. 모바일 기기를 사용한다면 지역,시간에 따라 통신사의 ip가 바뀔 수 있고. 통신사 통ip로 받으면

어쨌거나 걱정도 되었다. 또한 조회수가 많아진다면 그에 해당하는 DB내용도 많아지는것이기 때문에 후에 조회수양이 많다면 DB 로직에 문제가 발생하고 속도저하가 심각하진 않을까 걱정도 되었다. 좀더 검색을 해 본 결과 session을 이용하면 된다는 봤었으나, session에 대한 정확한 개념적 정리가 되지 않아서 처음 생각한 방식대로 구현하기로 했다.

후에 session이나 modal을 사용하는 class, hitcount를 사용하는 class에 대해서도 공부를 해야겠다.

4.3 Dev-Ops

4.3.1 구성

이전 웹사이트에서 개선해야겠다고 생각한 것은 static파일들을 로컬이 아닌 S3에 올려놓고 불러오는 방법을 해야겠다고 생각했다. 로컬 static 이미지 파일을 불러오는 시간이 생각보다 오래 걸렸기 때문이다.

또한 실패한 heroku 배포를 다시 시도하기로 했다. 애초에 heroku 배포는 어려운 배포가 아니며 해본결과 개발자가 git에 올리듯 heroku에 로드만 해준다면 별다른 추가 설정없이 쉽게 배포를 할 수 있는 웹서버이다.

4.3.2 구현

우선 S3를 이용하기 위해서는 S3 버킷을 만들어야 한다. 또한 만들고 나면 접근 할 수 있는 권한을 주기 위해 IAM에 사용자를 등록해야 한다. 등록을 마치면 마지막 단계에서 access key와 secret access키가 발급되는데, 보통 csv파일로 따로 저장을 하거나 해야한다. 이 정보들은 한 번 발급되면 재발급을 못하기에 IAM을 새로 만들어야 하는 번거로움이 발생 할 수 있다.

```
"""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'
```

django project settings.py에 정의해야 되는 부분이다. 어떤 IAM access key와 secret access key인지 알려주어야 하고. aws 위치도 설정해야한다. 설정 위치가 멀리 있을 경우 속도 저하가 발생 할 수 있다고 한다. 또한 static파일을 읽어오기만을 위한 이미지 서버이기 때문에 권한을 public-read로 설정했다.

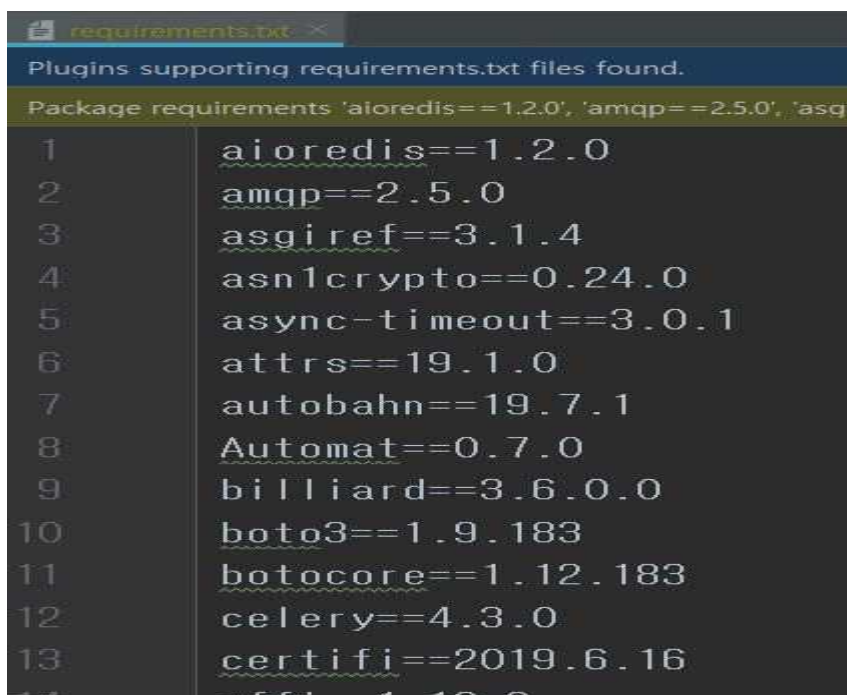
heroku에 배포를 하기 위해서는 heroku 계정이 있어야 하며 heroku가 설치되어 있어야 한다. 또한 git을 이용해서 heroku에 배포를 하기 때문에 git도 설치되어야 하고 git사용법을 조금이나마 알고 있어야 한다.

몇가지 모듈도 추가로 설치 해주어야 한다고 한다.

- 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

이 네 가지 추가 모듈을 다 설치 한 후 프로젝트에 설치된 모든 모듈 목록을 파일로 만들어야 한다.

pip freeze > requirements.txt을 하면 생성이 된다.



```
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 cffi==1.12.2
```

그리고 settings.py에

```
###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',
]
```

이것들을 새롭게 추가하거나 설정해주어야 한다.

그리고 Procfile을 만들어야 하는데 이것은 헤로쿠가 어떤 명령어로 우리가 업로드한 웹 서비스를 시작할 수 있는지 알려주는 파일이라고 한다.

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

어떤 파이썬을 사용할지 지정하는 파일도 만들어준다

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

여기까지 모든 설정이었다.

heroku에 로그인(heroku login)을 하고 git을 이용해 소스코드를 업로드 진행하기 위해 .gitignore파일을 만들었다.

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

이후에

```
###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
```

진행하여 배포가 끝났다.

4.3.3 구현 간 에러 사항

Dev-Ops에서는 ‘배프의 오지랖 파이썬 웹 프로그래밍’이라는 서적을 보면서 진행 하였기에 순서나 과정에 대해 큰 에러는 없었지만, 시스템 차이로 인한 에러는 존재했다.

특히 이전에도 실패한 경험이 있는 heroku에서 문제가 있었는데, heroku서버는 linux 기반으로 운영되는 웹서버이다. 그렇기에 windows에 종속되는 모듈들은 설치 할 수가 없다. 이전에 실패할 때도 문제가 무엇인지는 가늠하고 있었지만, 어떻게 해결할지 몰라 pythonanywhere로 진행을 하였었지만 이번에는 해결하고자 하였다. 답은 생각보다 단순했다. pywin32를 설치 할 수 없다고 자꾸 에러가 났었는데 이것이 windows에 종속되는 프로그램이다. 이름부터가 win32이다. stack flow에서 찾아본 결과 windows의 종속성을 없애면 된다고 하였다. 도저히 이해가지 않았다. 종속성. ‘windows의 종속성을 없애려면 어떻게 해야될까?’ 하고 고민하던중 그냥 무작정 삭제해보기로 하였다. ‘어차피 push도 안되느니 push되고 나서 에러를 경험해보자’ 라는 마인드였고 결과는 그냥 그렇게 에러가 해결되고 배포까지 되었다. 이전에는 몇날을 고민했었지만 막상 별 문제답지 않은 문제였다니 허탈하기도 했다.

해결 한 후 git에 프로젝트를 push하였다. 그랬더니 git자체에서 경고메세지가 날라왔다. 해석해보면 당신의 requirements.txt를 공개하는 것은 여러 가지 문제가 발생한다는거 같았다. 그래서 git에 배포할때는

requirements.txt뿐만 아니라 config/settings.py또한 제외시켰다.
settings.py에 S3세팅 정보와 access key, secret access key가 있어서 찜찜했기 때문이다.

5. 프로젝트 마무리

이번 프로젝트를 진행하면서 느낀점을 한 줄로 요약해보면
‘배울게 너무 많다’ 입니다. 간단한 CRUD 사이트를 만들면서도
여러 가지 방법이 있었고, 생각나는 방식으로 구현을 하려
할때마다 막히기 일쑤였습니다. 또한 front-end를 생각하지 않고
back-end만 구상을 한후 막연히 시작을 하니 front-end를
개발할 때는 점점 프로젝트가 산으로 가는게 아닌가? 싶기도
하고 간단한 CRUD조차도 생각대로 만들지 못한다는 것에
실망감도 없잖아 있었으며 의욕도 나지 않았을때도 있지만,
‘어떻게든 해보자’ 라는 식으로 맨땅에 헤딩한 것 같습니다.

back-end를 개발을 하면서도 특별한 기능도 없고 이전에
만들었던 사이트와 다른점은 그저 CBV를 FBV로 구현한 것
뿐인데 이게 맞을까 하는 의문도 들긴 했지만, front-end나
dev-ops쪽을 더 경험해본다는 생각으로 계속 진행한 것
같습니다. 개발자라면 front-end, back-end, dev-ops 구분되서
‘나는 back-end개발자니까 back-end만 해야지’ 생각보단
‘보다 다양한 경험을 해보자’ 라고 진행하였습니다.
다른방법으로 배포도 해보고, 다른 방식으로 form을 채워도 보고.

이 프로젝트를 진행하면서 앞으로 배워야 할 것 이라면 ajax, http, git, algorithm정도고 python또한 기초부터 다시 다져야겠다고 생각이 들게 되었습니다.

6. 참고

Dev-Ops :

DIGITAL BOOKS 배프의 오지랴 파이선 웹 프로그래밍 -저자 배프

DFD :

<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/>