2020년도 소프트웨어 설계 및 실험 프로젝트

중간 보고서 - 11조 정찬휘

현재까지의 구현내용

장고는 MTV패턴을 사용해 개발하고 있고 템플릿은 별도로 만들지 않고, 모델과 뷰만 만들어 API 콜을 이용해 주고받을 생각이다.

models.py > clip > tag_save

```python
from django.db import models
from django.conf import settings
from django.shortcuts import reverse
import re

class Tag(models.Model):
    name = models.CharField(max_length=140, unique=True)

    def __str__(self):
        return self.name
#해쉬태그 모델, 최장길이는 140자
class clip(models.Model):
    #글 모델
    contents = models.TextField(null=True) # 본문내용
    tag_set = models.ManyToManyField('Tag', blank=True) # 해쉬태그 모음

    id = models.AutoField(auto_created=True, primary_key=True, serialize=False) # 고유글 넘버
    title = models.CharField(max_length=1000) # 글 제목 1000자 제한
    summary = models.TextField(null=True) # 요약문
    tag_field = models.TextField(null=True) # 해쉬태그 표시용


    class Meta:
        ordering = ['-updated']

    def tag_save(self):
        tags = re.findall(r'#(\w+)\b', self.contents)
        #태그를 모델에서 처리하는 함수
        if not tags:
            return

        for t in tags:
            tag, tag_created = Tag.objects.get_or_create(name=t)
            self.tag_set.add(tag)  # NOTE: ManyToManyField 에 인스턴스 추가
```

저장형식을 규정하는 모델 부분 코드.

```python
from django.shortcuts import render
from django.template.context_processors import request
from django.contrib import messages
from .models import clip,Tag #모델중에 clip과 tag를 갖고온다
from django.http import HttpResponse
from django.shortcuts import redirect
from django.db.models import Count
from .forms import ClipPostForm #clip을 폼의 형식을 이용해 입력을 받는다.
from django.core.paginator import Paginator,EmptyPage
from .upload_fuc import get_summary #get_summary 함수에서 아마존과의 통신, textrank 처리가 이뤄질 예정
```

뷰에서 TextRank 알고리즘과 연결시켜 데이터를 가공하고 저장할 것을 염두에 두고 짰음.

```python
def clip_list(request, tag=None):
    tag_all = Tag.objects.annotate(num_post=Count('clip'))

    q = request.GET.get('q','')
    if q:
        tag=q
    if tag:
        clips = clip.objects.filter(tag_set__name__iexact=tag).prefetch_related('tag_set')
        # 처음은 태그를통한 검색으로 이 함수를 작동시켰는지 확인
    else:
        clips = clip.objects.prefetch_related('tag_set').all()
        #검색으로 찾지 않았다면 전체 문서를 다 불러옴

    if request.method == 'POST':
        #POST유형으로 요청한다
        tag = request.POST.get('tag')
        tag_clean = ''.join(e for e in tag if e.isalnum())  # 특수문자 삭제
        return redirect('clip:clip_search', tag_clean)
    paginator = Paginator(clips,5)#페이지네이터를 이용해 글 5개씩 끊어줌
    page = request.GET.get('page')
    clips = paginator.get_page(page)
    return render(request,'clip/list.html', {'clips':clips, 'tag':tag, tag_all:tag_all, })
    #정해진 탬플릿으로 이를 보내고 클립, 태그를 연동시켜줌
```

모든 클립(글)을 긁어오거나 해쉬태그를 이용해서 검색을 지원하는 뷰부분 코드이다

```python
def clip_new(request):
    #새 클립 생성에 필요한 함수
    if request.method == 'POST':
        form = ClipPostForm(request.POST)
        #정해진 폼을 통해서 입력을 받는다.
        if form.is_valid():
            #form에 대해 타당성검사를 한다.
            clip = form.save(commit=False)
            #클립폼을 먼저 저장한다. 장고는 폼이라는 형식을 통해 입력을 받게된다.
            #입력의 정형화를 위해 유리한 방법이다.
            clip.title = str(get_summary(clip.contents)[0])
            clip.summary = get_summary(clip.contents)
            clip.tag_field = get_tag(clip.contents)
            #이 과정에서 제목, 요약문, 태그필드는 문장들을 get_summary함수로 보내서 함수에서 바꿔서 저장한다.
            clip.save()
            clip.tag_save()

            return redirect('clip:clip_list')
            #끝났다면 cliplist함수를 다시 실행시켜서 원래의 리스트를 다시 본다
    else:
        form = ClipPostForm()
    return render(request, 'clip/upload.html', {'form': form})
```

이 부분에서 알고리즘과 연동이 일어날 코드인데, 새로운 클립을 생성하는 코드이다.

admin.py > ...

```python
from django.contrib import admin
from .models import clip
# Register your models here.
class clipAdmin(admin.ModelAdmin):

    list_display = ['contents','id','title' ]
    #관리자페이지에서 볼 수 있는 것은 글내용, 글번호, 글제목으로 제한한다.
    ordering = ['-updated']
    #이를 업데이트된 순서로 나열한다.

admin.site.register(clip, clipAdmin)
```

CRUD 중 Update와 Delete를 관리할 관리자 시점에서의 관리페이지 규정이다.

앞으로의 개발계획

1. 장고가 갖고 있는 API콜을 부르고 출력하는 모듈을 이용해 이를 모바일과 통신 구현
2. TextRank 알고리즘을 서버에 올려 처리해 저장하는 일련의 과정을 구현