1. 아나콘다 프롬프트 내에서의 가상환경 및 프로젝트 생성

$ conda create --name djangoproject: 가상환경 생성

$ conda activate djangoproject: 가상환경 진입

$ conda install django: django 설치

$ django-admin startproject myproject: 프로젝트 myproject 생성

2. 프로젝트 초기화 작업 및 개발 서버 구동

$ python manage.py migrate

$ python manage.py createsuperuser: 슈퍼 계정 생성

user: ksh03003

pw: a

$ python manage.py runserver: 서버 실행

3. vscode anaconda 가상환경 사용

ctrl+shift+p > python: Select Interpreter 인터프리터 및 가상환경 선택

vscode 터미널창 command propmt 열기

참조:

<https://velog.io/@jeongm/vscode%EC%97%90%EC%84%9C-anaconda-%EA%B0%80%EC%83%81%ED%99%98%EA%B2%BD-%EC%82%AC%EC%9A%A9%ED%95%98%EA%B8%B0>

4. Django app 생성

1) app 생성 후 기본 과정

$ python manage.py startapp blog1 : 블로그앱 생성

settings.py > installed\_apps에 blog1 추가

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'blog1',

]

blog1/urls.py 생성

urlpatterns = [

]

urls.py>urlspatterns blog1 path 추가

urlpatterns = [

    path('admin/', admin.site.urls),

    path("blog1", include('blog1.urls'))

]

2) 모델링 및 view, template 생성

blog1/models.py 모델 생성

class Post(models.Model):

    title = models.TextField()

    content =  models.TextField()

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

$ python manage.py makemigrations blog1

$ python manage.py migrate blog1

blog1>admin.py 등록

from .models import Post

admin.site.register(Post)

blog1>views.py view 추가

def post\_list(request):

    # Query set

    qs = Post.objects.all()

    return render(request, 'blog1/post\_list.html',{'post\_list': qs})

blog1/templates html post\_list.html 추가

<h1>Post List</h1>

<ul>

    {% for post in post\_list %}

    <h2>{{post.title}}</h2>

    {{post.content}}

    {% endfor %}

</ul>

blog1/urls.py 생성 후 urlspatterns 추가

urlpatterns = [

    path('',views.post\_list, name = 'post\_list')]

5. Django model 모델링

1) 모델링 방법

모델 클래스 작성 ->

$ make migrations: 마이그레이션 파일 생성 ->

$ migrate: 마이그레이션 파일 DB 적용 ->

$ python manage.py sqlmigrate [app이름] [migration 이름]: sql 확인

class Post(models.Model):

    title = models.TextField()

    content =  models.TextField()

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

2) 모델링 유의사항

모델 클래스명은 단수형으로 지정 (Posts가 아닌 Post)

DB 테이블명: 앱이름\_모델명 (ex: blog\_comment, blog\_comment…)

매핑되는 모델은 DB 테이블과 일치해야 함

데이터베이스를 먼저 설계하고 모델을 생성할 것

$ inspectdb : 외부 db로부터 가져올 때 최종본이 아닌 초안으로 사용할 것

설계한 데이터베이스 구조에 따라 필드 타입을 타이트하게 지정해주는 것을 권장함

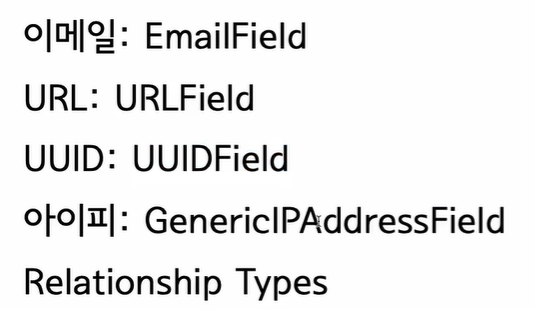
inspectdb 명령은 초안으로 사용하고 validators를 통해 타이트하게 지정할 것

blank, null 지정은 최소화 할 것

직접 유효성 검사 로직을 만들지 말고 기존의 구성된 것을 사용할 것

3) 모델 필드 타입 및 옵션





blank: validation 시 empty 허용

null: null 허용 여부

default: 기본값 지정

validators: 필드에 따른 고유한 validators 등록 (ex: 이메일)

6. Django admin

1) 간단한 admin 선언

admin.site.register(Post)

2) 다음과 같이 선언하여 admin에서 원하는 필드만 볼수도, 검색할 컬럼을 설정할수도,

검색 필터를 만들수도, 사용자 정의로 된 다른 필드를 생성할 수도 있다.

@admin.register(Post)

class PostAdmin(admin.ModelAdmin):

    # 원하는 필드

    list\_display = ['id','message','created\_at', 'updated\_at','is\_public','messeage\_length']

    # 링크 설정

    list\_display\_links = ['message']

    # 검색 필드

    search\_fields=['message']

    # 검색 필터

    list\_filter=['is\_public']

    # 사용자 정의 필드

    def messeage\_length(self, post):

        return f'{len(post.message)} 글자'

7. media 파일 관리

media: file 또는 imagefield를 통해 저장한 모든 파일

1) 파일 처리 순서

httprequest.FILES를 통해 파일 전달 -> 뷰, 폼을 통해 유효성 검사 수행

-> FileField/imageFiled에 경로 저장 -> MEDIA\_ROOT 경로에 파일 저장

2) file field 모델링

settings.py: Media 파일 관련값 설정

MEDIA\_URL = '/media/'   # 각 media 파일에 대한 url prefix

MEDIA\_ROOT = os.path.join(BASE\_DIR,'media') # 파일 필드를 통한 저장시, 파일을 저장할 root 경로

app이름/urls.py: 파일 불러올 때의 urls.py 설정

from django.conf import settings

from django.conf.urls.static import static

if settings.DEBUG:

    urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

$ pip install pillow: 라이브러리 설치

app이름/admin.py: admin에서 사진 미리보기 및 크기 지정

    from django.utils.safestring import mark\_safe

    # 사진 미리보기

    def photo\_tag(self, post):

        if post.photo:

            return mark\_safe( f'<img src="{post.photo.url}" style = "width:75px;" />')

        return None

프로젝트명/utils.py: 파일 생성 후 uuid를 통해 경로와 고유한 파일명을 반환하는 모듈 생성

import os

from uuid import uuid4

from django.utils import timezone

# uuid를 통한 파일명 정하기

def uuid\_name\_upload\_to(instance, filename):

    app\_label = instance.\_\_class\_\_.\_meta.app\_label  # 앱 별로

    cls\_name = instance.\_\_class\_\_.\_\_name\_\_.lower()  # 모델 별로

    ymd\_path = timezone.now().strftime('%Y/%m/%d')  # 업로드 하는 년/월/일 뱌럴

    uuid\_name = uuid4().hex

    extension = os.path.splitext(filename)[-1].lower()  # 확장자 추출 및 소문자 변환

    return '/'.join([

        app\_label,

        cls\_name,

        ymd\_path,

        uuid\_name[:2],

        uuid\_name + extension,

    ])

models.py: Imagefield 추가 및 업로드 경로에 uuid 모듈 적용

    from myproject.utils import uuid\_name\_upload\_to

    # uuid를 통한 중복방지 파일명 지정

    photo = models.ImageField(blank = True, upload\_to=uuid\_name\_upload\_to, max\_length=255)

$ python manage.py makemigrations [앱이름]

$ python manage.py migrate [앱이름]

**8. static 파일관리**

static: html, css, image 등의 정적인 파일

**9. Django – jupyer notebook 연동**

시각화, 데이터 분석시 jupyter 사용 권장

anaconda prompt 진입

$ conda activate djangoproject: 프로젝트의 가상환경 진입

$ pip install jupyter notebook: 가상환경내 jupyter 설치

$ jupyter notebook 실행

1) setting

import os

os.environ['DJANGO\_SETTINGS\_MODULE'] = 'myproject.settings'

os.environ['DJANGO\_ALLOW\_ASYNC\_UNSAFE']='true'

import django

django.setup()

2) 모델 조회

from instagram.models import Post

qs = Post.objects.all()

qs

**10. 모델 조회(select)**

1) 전체 조회: qs = Post.objects.all()

2) 범위 지정 및 아이디 순으로 정렬 후 범위 처리: qs = Post.objects.all().order\_by('-id')[:2]

3) 결과 딕셔너리 출력:

for post in qs:

print(post.message)

print(post.\_\_dict\_\_)

for post in qs:

print ("id: {id}, message: {message} (작성일자: {created\_at})".format(\*\*post.\_\_dict\_\_))

4) 조건문 (filter, exclude)

filter안에 들어가는 다양한 오퍼레이터를 알아두는 것이 좋다

두 개 이상의 필터는 and 조건으로 적용됨

qs = Post.objects.filter(message\_\_icontains = '테스트').order\_by('-id')

5) 조건문 (get, first, last)

post.get : 값이 없을 경우 예외 반환

(값이 없다면DoesNotExist / 두 개 이상의 값이 나오면 MultipleObjectsReturned 반환)

post.fitst : 모델객체 없으면 None

post.last : 모델객체 없으면 None

6) Q: 필터 조건을 or 로 주고 싶을 때 사용

from django.db.models import Q

# and 조건

cond = Q(id\_\_gte=2)&Q(message\_\_icontains="테스트")

# or 조건

cond = Q(id\_\_gte=2)|Q(message\_\_icontains="테스트")

qs = Post.objects.filter(cond)

# 결과

for post in qs:

print(post.\_\_dict\_\_)

7) 다양한 오퍼레이터

필드명\_\_lt : 필드명 < 조건

필드명\_\_lte: 필드명 <= 조건

필드명\_\_gt: 필드명>조건

필드명\_\_gte: 필드명>=조건

필드명\_\_startswith : 필드명 like "조건값%"

필드명\_\_istartswith : 필드명 like "조건값%" (대소문자 구별 x)

기타 endwith, contains...

**11. Django debug tool bar**

디버그에 사용되는 도구

SQL의 소요 시간, request 값등을 쉽게 사용할 수 있음.

<body> 태그가 있어야 디버그 툴바가 나타남

참고: <https://django-debug-toolbar.readthedocs.io/en/latest/installation.html>

Django-qeury count를 통해 sql 실행 내역을 개발 서버 콘솔로 출력 가능

**12. Foreign key**

1) 1:N models.ForeignKey

ex) 1명의 유저가 쓰는 다수의 포스팅, 1명의 유저가 쓰는 다수의 댓글

- N에서 관계 명시

- models.ForeignKey(대상모델, on\_delete)

- on\_delete:

cascade: 참조 하는 모델의 리코드도 삭제

그 밖에 PROTECT, SET\_NULL, SET\_DEFAULT, SET, DO\_NOTHING 등이 있음

다만 reverse\_name의 경우 앱이름은 고려하지 않고 모델명만 고려하기 때문에

이름 충돌 발생할 수 있음.

이 때 related\_name을 통해 한쪽의 reverse\_name을 포기하거나 reversename 변경하는것을 권장

2) 1:1 models.OneToOneField

ex) 1명의 유저에는 1개의 프로필

- models.ForeignKey(대상모델, on\_delete)

3) M:N models.ManyToManyField

ex) 1개의 게시글 다수의 태그, 1개의 태그에는 다수의 게시글

- models.ManyToManyField(대상모델, blank 여부)

- M,N 어느쪽에서 선언할 수 있으나 사용하는 쪽에서 선언하는 것, 또는 참조가 덜 되는 쪽에서 선언하는 것이 적합하다.

ex) 태그의 경우 Post, Article 등 다양한 곳에서 사용되므로 Post, Article 에서 선언하는 것이 좋음.

tag\_set = models. ManyToManyField(‘Tag’, blank= True)

manytomany 관계의 경우 테이블 두개로는 정의하기 힘들기 때문에 중간 테이블이 생성된다.

**13. migration을 통한 DB 스키마 관리**

순서: 모델변경 > makemigrations > 마이그레이션 파일 확인 > migrate > DB 적용

모델 필드에서 변경 내역이 발생 시에 수행

적용된 마이그레이션 파일은 삭제 금지. 히스토리를 쌓아가는 원리이므로

Django 를 통해서 데이터베이스 스키마의 히스토리 관리

[앱이름]을 생략하면 전체를 대상으로 진행됨. 그러나 [앱이름] 지정을 권고

같은 Migration 파일이라 하더라도 DB에 따라 다른 SQL이 생성될수도

$ python manage.py makemigrations [앱이름] : 마이그레이션 파일 생성

$ python manage.py migrate [앱이름] : 지정 DB에 마이그레이션 적용

$ python manage.py showmirgrations [앱이름] : 마이그레이션 적용 현황 출력. 빈칸은 적용안된 것

$ python manage.py sqlmigrate [앱이름]: 지정 마이그레이션 sql 출력

필수 필드가 추가된다면 마이그레이션을 하면 기존 레코드에 어떤값을 채워넣을지 물음

(default값을 지정하는게 가장 낫긴 함)

- 1. 값을 입력 2. 명령수행중단

팀원 각자가 마이그레이션 파일 생성시 충돌 발생 위험!

한명이 전담해서 생성하는 것을 권장.

생성한 파일을 버전 관리에 넣고 다른 팀원들은 migrate만 수행

13. User

import get\_user\_model

User = get\_user\_model() : 현재 존재하는 모든 유저 가져오기

- 1:1 관계인 profile 가져오기

user.profile:

Profile.objects.get(user=user)

**98. 기타 cmd 명령어**

$ conda env lsit: 가상환경 리스트 확인

$ start .: 현재 디렉토리 열기

$ pip list: requirement.txt 작성

$ pip install -r requirement.txt: requirements 설치

**99. 기타 이슈**

1. html 자동완성: 인터프리터 html 로 변경 후 ! + tab

2. vscode자동완성

<https://wonpaper.tistory.com/339>

3. 자동 정렬:

100. 단축키

ctrl + p: 프로젝트 내 파일 검색