**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

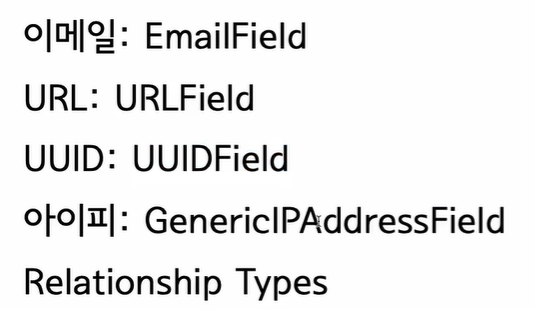
inspectdb 명령은 초안으로 사용하고 validators를 통해 타이트하게 지정할 것

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 makemigrations [앱이름]

$ python manage.py migrate [앱이름]

**8. static 파일관리**

static: html, css, image 등의 정적인 파일

**99. 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