1.pystargram 기획.

서버에 접근할 URL

- 개별 사진 보기 : /photos/<사진 ID="">/
- 사진에 좋아요 누르기 : /photos/<사진 ID="">/like/
- 사진에 댓글 달기 : /photos/<사진 ID="">/comment/
    - 사진에 달린 댓글 가져오기 : /photos/<사진 ID="">/get_comments/
    - 사진에 달린 댓글 지우기 : /photos/<사진 ID="">/comment/<댓글 ID="">/delete/
- 사진에 태그 달기 : /photos/<사진 ID="">/tag/

이렇게 URL에 직접 만들어 줄 수 있음.하지만 2.0이후로는 그렇게 사용하지 않음 규칙을 따라 URL 작성. 페이지 구성도가 중요함. 시작하기전에 먼저 잘 만들어놔야 나중에 헷갈리지 않음.

2.기획하기.

---

virtualenv 사용하기.

`` python -m venv myvenv ``

`` myvenv\Scripts\activate.bat ``

** django 설치하기**

`` pip install django ``

`` python -m django --version `` (장고버전확인하기.)

Postman - REST Client는 HTTP 기반으로 동작하는 API를 편리하게 호출하는 클라이언트(client)입니다.postman은 개발에 용이한 클라이언트 인터페이스를 제공하는 도구.

python

들여 쓰기가 규칙 준수.

3.Photo 앱과 모델 만들기

1.django Project와App

python package는 python module을 묶어놓은 단위 반드시 초기화 모듈인 **init**.py 가 필요. djago는 Django Project단위로 만듬.

** Pystargram프로젝트로 만든다는 건Pystargram이라는 Python패키지를 만들고, Pystargram에 들어가는 기능은 Python모듈로 만든다는뜻**

`` django-admin startproject pystargram ``

pystagram/

├── manage.py

└── pystagram

```
    ├── __init__.py
```

```
    ├── settings.py
```

```
    ├── urls.py
```

```
    └── wsgi.py
```

프로젝트 서버가 구동되는지 확인하기.

`` python manage.py runserver ``

데이터베이스와 동기화하기.

``python manage.py migrate ``

``python manage.py createsuperuser `` **최고 권한 이용자 만들기**

이렇게 수행한 데이터베이스 작업은 db.sqlite3에 저장.

Django App 초기화.

`` python manage.py startapp photos ``

photos packge가 Django App이다.

admin.py는 관리자 영역에서 App을 다루는 코드를 담는 모듈이다.

models.py는 모델을 정의하는 모듈로서 모델은 데이터를 구성하는 항목자체(field)와 데이터를 다루는 행위를 포함 객체로 표현한다.

views.py는 특정주소에 접근하면 화면에 내용을 표시하는 Python함수를 호출하는 내용을 담고 있다.

MVC패턴에서는view가 표현물 Django에서는 Temlpate이 표현물 Django에서 view는 데이터(model) 를 표현(Template) 하는 연결자이자 안내자 MVC패턴으로 보면 Controller와 유사하다.

models.py를 고쳐보자.

class Photo(models.Model):

```
    image = models.ImageField()
```

```
    filtered_image = models.ImageField()
```

```
    content = models.TextField(max_length=500)
```

```
    created_at = models.DateTimeField(auto_now_add=True)
```

데이터베이스에 반영(migration)

setting.py INSTALLED_APPS 에 적용후.

`` python manage.py makemigration `` 에러가 나온다.

image와 filtered_image 모델 필드는 ImageField인데 처리하는 도구가 필요함.

`` pip install pillow `` 라는 도구가 필요

한후에 다시 ``python mange.py makemigrations ``

`` python manage.py migrate ``

하면 0001_initial.py에 저장후 실제로 반영.

4.Photo 모델로 Admin 영역에서 데이터 다루기.

Admin에서 Photo모델에 데이터 넣기.

`` from django.contrib import admin ``

`` from .models import Photo ``

`` admin.site.register(Photo) ``

blank 필드옵션 : 빈칸을 의미 한다. 본문에 내용이 없어도 에러 나지 않음.

파일 업로드 경로지정.

`` image = models.ImageField(upload_to='uploads/%Y/%m/%d/orig') ``

`` filtered_image = models.ImageField(upload_to='uploads/%Y/%m/%d/filtered') ``

연도 월 일 시간으로 구분되어질 수 있도록 이미지파일을 관리 할 수 있음.

첨부파일 삭제하기.

Photo모델의 객체를 지워보자. :객체 자체는 지워지지만 연결된 파일들은 삭제되지 않는다. Django의 모델 기능은 모델 객체가 삭제되어도 그 모델 객체의 파일 필드에 연결된 파일을 지우지 않는다.

delete라는 인스턴스 메서들 호출하여 모델 객체를 지울 수 있음

class Photo(models.Model):

```
# 중략
```

```
def delete(self, *args, **kwargs):
    self.image.delete()
    self.filtered_image.delete()
    super(Photo, self).delete(*args, **kwargs)
```

*args와 **kwargs 는 함수가 넘겨받는 인자를 미리 알지 못하는 경우에 함수가 넘겨받는 인자를 담는 객체이다

python class의 인스턴스 메서드 안에서 속성에 접근하려면 self.속성이름 으로 접근. self는 delete 인스턴스 메서드에서 첫번째 인자로 넘겨 받는다 따라서 `` self.filtered_image.delete(*args, **kwargs) `` 뒤에 코드가 없으면 첨부된 업로드 파일만 삭제되고 모델 객체는 삭제되지 않는다. 지우는건 Model클래스에 있는 delete메소드이기 때문이다.

5. URL에 VIEW함수 연결해서 사진 출력하기.

이용자가 URL로 접근하여 뭔가를 요청하면 그 URL에 대한 정보를 urls.py로 대표되는 URL로 접근하여 뭔가를 요청하면 그 정보를 실행함.

1.BaseHandler 클래스가 URL로 요청(request) 받음

2.RegexURLResolver로 URL을 보냄

3.RegexURLResolver가 URL에 연결된 View를 찾아서 callback 함수와 인자 등을 BaseHandler로 반환

4.BaseHandler에서 이 함수를 실행하여 결과값인 출력물을 받음.

5.출력

이렇게 사용하기 위에서 위에 내용이 필요함

1.Model이나 View에 기능을 구현

2.이용자가 서버에 있는 자원에 접근하는 경로인 URL을 URL Dispatch 처리 모듈인 urls.py에 등록하고 그 URL에 구현부를 연결

-views.py

-urls.py

` from django.conf.urls import url ` ` from django.contrib import admin ` ` from photos.views import hello `

` urlpatterns = [

```
    url(r'^hello/$', hello),
    url(r'^admin/', admin.site.urls),
```

] `

정규표현식

- ^ : ^ 문자 뒤에 나열된 문자열로 시작
- [0-9] : 0부터 9까지 범위에 속하는 문자
- ㅇ : 앞에 지정한 문자열 패턴이 한 번 이상 반복
- () : 패턴 부분을 묶어냄(grouping)
- ?P : 묶어낸 패턴 부분에 이름을 pk로 붙임.
- **:** 문자 앞에 나열된 문자열로 끝

from .models import Photo

Photo 모델로 객체 찾기

` def detail(request, pk):

```
    photo = Photo.objects.get(pk=pk)
```

` `

```
    messages = (
        '<p>{pk}번 사진 보여줄게요</p>'.format(pk=photo.pk),
        '<p>주소는 {url}</p>'.format(url=photo.image.url),
    )
```

`

`

```
    return HttpResponse('\n'.join(messages))
```

`

photo모델의 objects 객체의 get 메서드를 이용해 뷰 함수의 인자pk에 해당하는 사진 데이터를 가져와서 photo변수에 담음. 모델에 있는 image 필드애 접근해 url 속성을 이용해 지정된 사진 출력

django는 이용자가 업로드한 파일을 MEDIA_URL과 MEDIA_ROOT라는 설정값을 참조하여 제공

원래는 urls.py에도 이와 관련된 내용을 등록해야한다.

django는 이런걸 처리해주는 기능이 있다.

` django.conf.urls.static # 여기에 있는 함수static을 사용 `

` urlpatterns += static('upload_files', document_root=settings.MEDIA_ROOT) `

이런식으로 사용하면됨.

render 와 HttpResponse

HttpResponse는 django view가 HTTP handler로 보내는 출력물의 가장 기본형택인 객체를 만드는 클래스. 따라서 템플릿을 같은걸 처리하는 기능이 없음 따라서 템플릿을 따로 처리하여 그려낸 출력물을 문자열 긋채로 받아서 출력해야함. 따라서 이런 처리에 필요한 코드는 꽤 반복되므로 반복되는 부분을 별도함수로 만들어서 편하게 템플릿으로 그려낸 출력물을 HttpResponse 로 보내는 함수가 바로 render이다. render로 반환하는 최종값도 결국은 HttpResponse 클래스로 만든 객체이다.