Managing Static Files

Managing Static Files

Static files

- Static file (정적 파일)
- Static file 구성
- Django template tag
- The staticfiles app
  - 1) STATIC_ROOT
    - [참고] collectstatic
  - 2) STATIC_URL
  - 3) STATICFILES_DIRS
- 정적 파일 사용하기 - 1) 기본경로
- 정적파일 사용하기 - 2) 추가 경로
- 부트스트랩 CSS, JS 파일 Static 파일로 추가해보기 (hw)

Media files

- Media file
- Model field
  - 1) ImageField
  - 2) FileField
    - `upload_to` argument
- ImageField (or FileField)를 사용하기 위한 몇 가지 단계
- MEDIA_ROOT
- MEDIA_URL
- 개발 단계에서 사용자가 업로드 한 파일 제공하기

Image Upload

- < CREATE >
  - 1. ImageField 작성
    - `upload_to='images/'`
    - `blank=True`
  - Model field option - "blank"
  - Model field option - "null"
    - blank & null 비교
  - 2) form 요소 - enctype(인코딩) 속성
    - multipart/form-data 적용
  - input - accept 속성
  - 3) Views.py 수정
    - DB 및 파일 트리 확인
- < READ >
  - 1) 이미지 경로 불러오기
    - MEDIA_URL 확인하기
  - STATIC_URL과 MEDIA_URL
- < UPDATE >
  - 1) 이미지 수정하기
    - 수정 1) enctype 추가
    - 수정 2) views.py 수정
    - 수정 3) 이미지 안넣고 싶을 때

Image Resizing
이미지 크기 변경하기 (1/3)
이미지 크기 변경하기 (2/3) - django-imagekit 라이브러리 활용
이미지 크기 변경하기 (3/3)
1) 원본 이미지를 재가공하여 저장하기 (원본x, 썸네일o)
2) 원본도 저장, 썸네일도 저장
참고
캐시

Static files

Static file (정적 파일)

- 응답할 때 별도의 처리 없이 파일 내용을 그대로 보여주면 되는 파일
  - 즉, 사용자의 요청에 따라 내용이 바뀌는 것이 아니라, 요청한 것을 그대로 보여주는 파일
- 예를 들어, 웹 사이트는 일반적으로 **이미지, 자바 스크립트 또는 CSS와 같은 미리 준비된 추가 파일(움직이지 않는)을 제공해야 함**
- Django에서는 이러한 파일들을 **static file(정적 파일)**이라 함

Static file 구성

1. `django.contrib.staticfiles`가 `INSTALLED_APPS`에 포함되어 있는지 확인 ~~(이미 되어 있음)~~
2. `settings.py`에서 `STATIC_URL`을 정의 ~~(이미 되어 있음)~~

```
# settings.py

STATIC_URL = '/static/'
```

3. 템플릿에서 `static` **템플릿 태그**를 사용하여 지정된 상대경로에 대한 URL을 빌드

```
{% load static %} <!-- → static 태그 사용할 수 있다(static 쓰겠다)는 의미 -->

<img src="{% static 'my_app/example.jpg' %}" alt="My image">
```

4. 앱의 static 폴더에 정적 파일을 저장
   - 예) `my_app/static/my_app/example.jpg`
   - (templates 와 경로 구성 방식이 같다.)

Django template tag

- **load**
  - 사용자 정의 템플릿 태그 세트를 로드
  - 로드하는 라이브러리, 패키지에 등록된 모든 태그와 필터를 로드
- **static**
  - `STATIC_ROOT`에 저장된 정적 파일에 연결
  - ★ static 파일을 사용하기 위해서 `load`태그 꼭 있어야 한다.

```
{% load static %}

<img src="{% static 'my_app/example.jpg' %}" alt="My image">
```

The staticfiles app

1) STATIC_ROOT

- **collectstatic이 배포를 위해 정적 파일을 수집하는 디렉토리의 절대 경로**
- django 프로젝트에서 사용하는 모든 정적 파일을 한 곳에 모아 넣는 경로
- 개발 과정에서 settings.py의 DEBUG 값이 `True`로 설정되어 있으면, 해당 값은 작용되지 않음
  - 직접 작성하지 않으면 django 프로젝트에서는 settings.py에 작성되어 있지 않음
- 실 서비스 환경(배포 환경)에서 django의 모든 정적 파일을 다른 웹 서버가 직접 제공하기 위함
  - (실 서비스 환경(배포 환경)은 django 서버 아님 !)

[참고] collectstatic

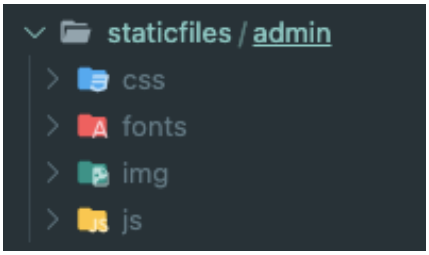
- `STATIC_ROOT`에 정적 파일을 수집
  - `STATIC_ROOT` 작성

```
# settings.py

STATIC_ROOT = BASE_DIR / 'staticfiles'
```

  - `collectstatic` 명령어 사용

```
$ python manage.py collectstatic
```

  - 실행 결과



- 위 과정들을 진행하면, 해당 경로 안에 현재 장고가 사용하는 모든 staticfile들을 모두 모아준다.
- 이게 우리가 알게모르게 사용하고 있던 정적 파일들이다.
  예를 들어 처음 로켓페이지 뜨는거나, admin 홈페이지 들어갔을때 디자인이나 구조들은 이런 정적파일들에 의해 출력이 되고 있는 것이었다. 그래서 배포할때 따로 모아서 배포한다.
- 아마존 웹 서버로 배포한다고 예를 들었을 때,
  아마존 웹 서버에서 이 장고 프로젝트가 가지고 있는 모든 정적파일들(이런 물리적인 파일들)을 그대로 받아서 사용할 수 있는 것이다.
  즉, 배포할때 쓰는 개념이다 ~

2) STATIC_URL

```
# settings.py

STATIC_URL = '/static/'
```

- **`STATIC_ROOT`에 있는 정적 파일을 참조할 때 사용할 URL** ~~(이미 되어 있음)~~
  - (참고) 개발 단계에서는 실제 정적 파일들이 저장되어 있는 각 `app/static/` 폴더 (기본 경로) 및 `STATICFILES_DIR`에 정의된 추가 경로들을 탐색함
    ~~(전에 templates를 settings에 설정해줬던 것 처럼 !)~~
- ⚠ **주의사항**⚠
  - **실제 파일이나 디렉토리가 아니며, URL로만 존재**
  - 비어있지 않은 값으로 설정한다면 **반드시 slash(/)로 끝나야 함**

3) STATICFILES_DIRS

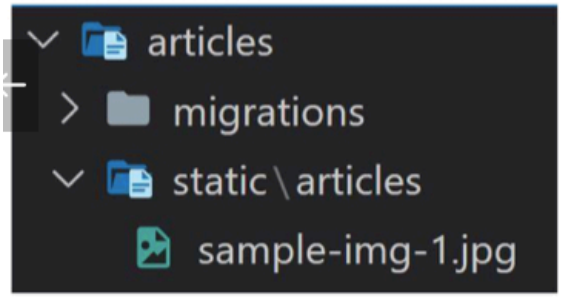
```
# settings.py

STATICFILES_DIRS = [
    BASE_DIR / 'static',
]
```

- `app/static/` 디렉토리 경로를 사용하는 것(기본 경로) 외에, **추가적인 정적 파일 경로 목록을 정의하는 리스트**
- 추가 파일 디렉토리에 대한 **전체 경로를 포함**하는 문자열 목록으로 작성되어야 함

✔ `STATICFIELS_DIRS` 은 추가경로'들'을 가져오는거라서 리스트 !
`MEDIA_ROOT`의 경우에는 절대경로를 하나 설정할거기 때문에 리스트 X

정적 파일 사용하기 - 1) 기본경로

- `static`이 app 안에 있을 경우.



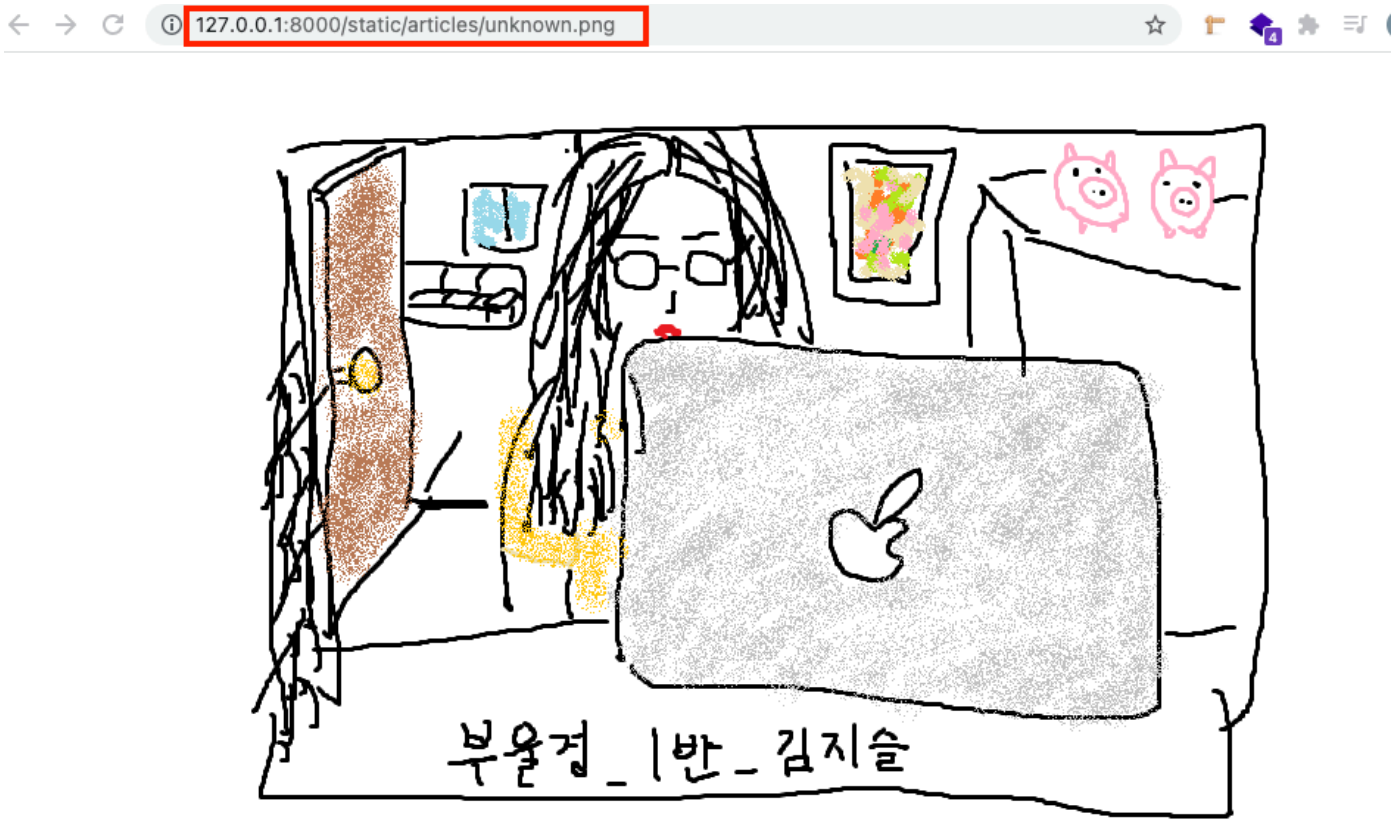
정적 파일 경로
app/static/app/

```
<!-- articles/detail.html -->

{% extends 'base.html' %}
{% load static %}

{% block content %}
  <img src="{% static 'articles/sample-img-1.jpg' %}" alt="sample-img">
  <h2>DETAIL</h2>
  ...
{% endblock content %}
```

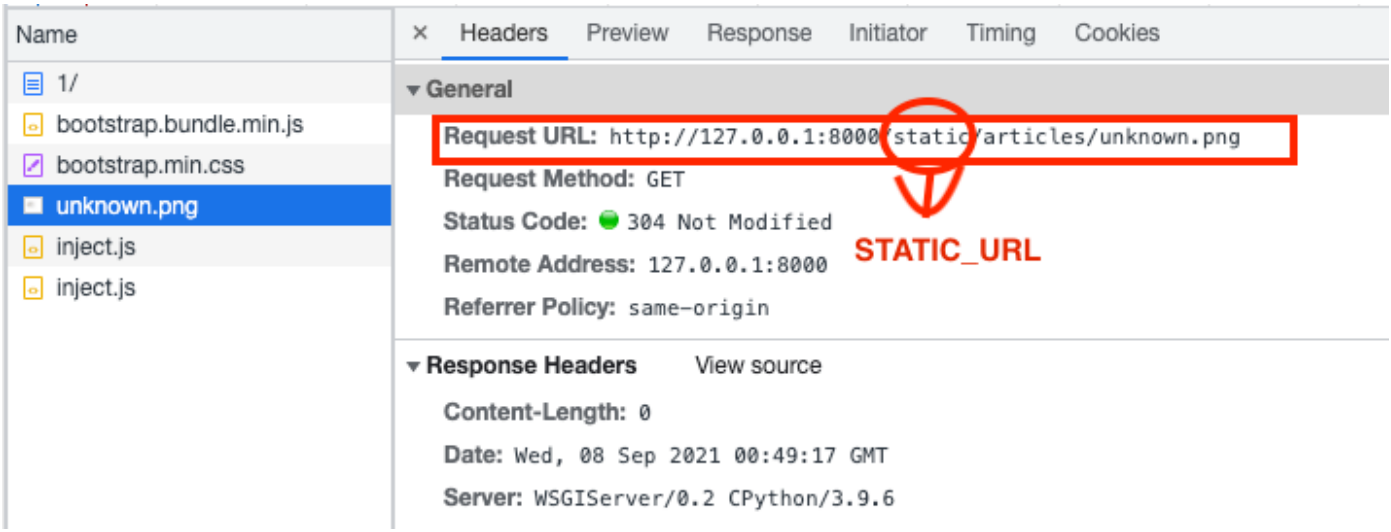
template에서 경로 참조

```
<div class="container">
  <img src="/static/articles/unknown.png" alt="sample image"> == $0
  <h2>DETAIL</h2>
  <h3>1 번째 글</h3>
```

- 위에서의 `img` 태그의 `src` 속성은 이 `img` 태그가 실제로 물리적으로 있는 위치에 있는 이미지를 출력하기 위해서 인것이고,



- 웹에서 보기 위해서는 이미지의 **url (주소)**가 필요하다.



✔ 질문) 이미지 파일이 로컬에 있는거 그냥 보여주는거 아니야?

- 웹 상에서는 요청을 보내서 응답 받아야 하는데 우리가 detail 페이지에 대해 하나의 문서를 받는 것처럼,
  거기에 포함되어 있는 이미지도 결국 그 **이미지에 대한 응답을 받으려면 그 이미지 고유의 주소가 있어야 한다.**

  ☞ 그 주소를 `STATIC_URL`이 만들어준다.

✔ 정리) 파일의 경로는 템플릿에서 `img` 태그의 `src` 속성이 쓰는거고, 실제로 우리가 웹상에서 이미지를 보기 위해서는(장고가 이미지를 제공하기 위해서는) 이미지를 제공할 **url**이 필요하다.

정적파일 사용하기 - 2) 추가 경로

- `static`이 최상위 폴더에 있을 때

정적 파일 위치 및
추가 경로 작성

template에서 경로 참조

부트스트랩 CSS, JS 파일 Static 파일로 추가해보기 (hw)

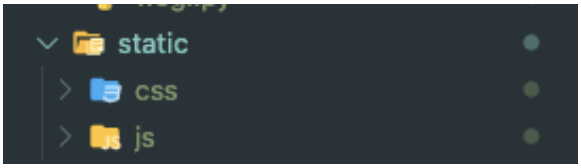
1. 추가 정적 파일 경로 설정

   `settings.py`에 `STATICFILES_DIRS`추가

   ```
   # settings.py

   STATICFILES_DIRS = [
       BASE_DIR / 'static'
   ]
   ```

2. 프로젝트 및 앱 디렉토리와 동일한 위치에 정적 파일을 담을 폴더(`static`) 생성

3. bootstrap에서 다운받은 css, js 파일 `static` 파일에 넣기



4. `static` 태그 활용하여 `base.html`에 css, js 적용하기

```
{% load static %}

<link rel="stylesheet" href="{% static 'css/bootstrap.css' %}">
<script src="{% static 'js/bootstrap.js' %}"></script>
```

위의 코드를 넣어준다.

Media files

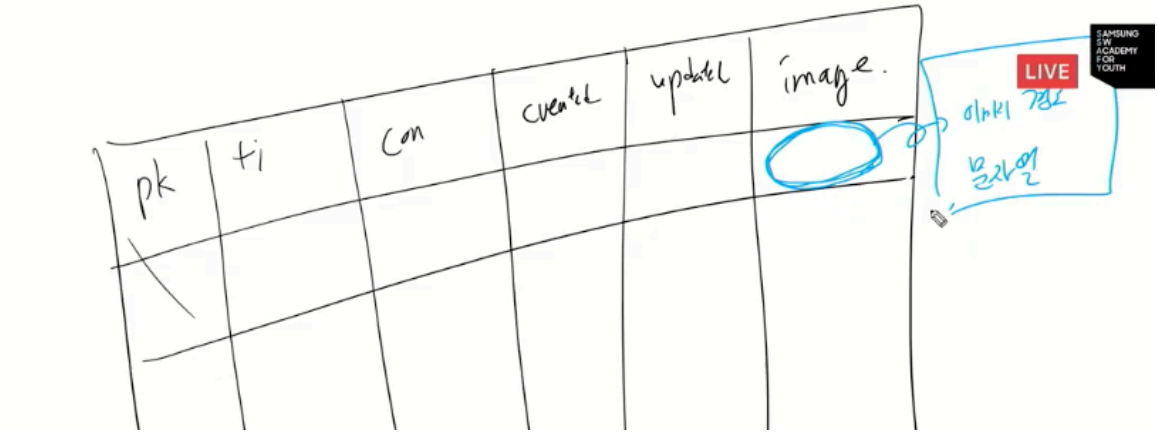
Media file

- 미디어 파일
- 사용자가 웹에서 업로드하는 정적 파일 (user-uploaded)
- 유저가 업로드 한 모든 정적 파일

Model field

1) ImageField

- **이미지 업로드에 사용하는 모델 필드**
- `FileField`를 상속받는 서브 클래스이기 때문에 `FileField`의 모든 속성 및 메서드를 사용 가능하며,
  더해서 사용자에 의해 업로드 된 객체가 유효한 이미지인지 검사함
  - (유효한 이미지인지 검사한다는게 별도로 코드를 작성하는 것이 아니라, 이미지 필드를 사용하면 자연스럽게 따라오는 기능)
  - `FileField`에서 상속받으니 `FileField`에서 벗어날 수 없다.
- **`ImageField` 인스턴스는** 최대 길이가 100자인 **문자열로 DB에 생성**되며,
  `max_length` 인자를 사용하여 최대 길이를 변경 할 수 있음
  - **이미지 파일 자체를 DB에 저장하는 것이 아니라, 해당 업로드된 이미지 파일의 경로가 담긴 문자열을 저장한다. ★**



- **[주의] 사용하려면 반드시 Pillow 라이브러리가 필요**

: Python Imaging Library

```
$ pip install Pillow
```

2) FileField

- **파일 업로드에 사용하는 모델 필드**
- 2개의 선택 인자를 가지고 있음
  1. upload_to
  2. ~~storage~~ (3.1버전부터)

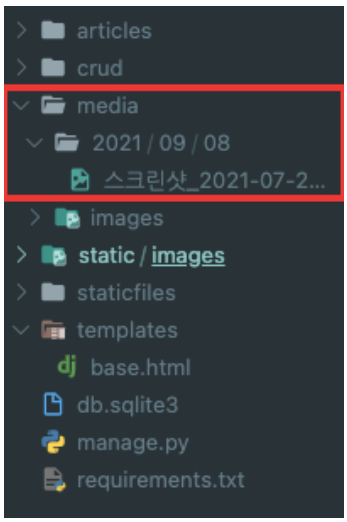
`upload_to` argument

- 업로드 디렉토리와 파일 이름을 설정하는 2가지 방법을 제공

1. 문자열 값이나 경로 지정

```python
# models.py

class MyModel(models.Model):
    # MEDIA_ROOT/uploads/ 경로로 파일 업로드
    upload = models.FileField(upload_to='uploads/')
    # or
    # MEDIA_ROOT/uploads/2021/01/01/ 경로로 파일 업로드
    upload = models.FileField(upload_to='uploads/%Y/%m/%d/')
```

- 파이썬의 `strtime()` 형식이 포함될 수 있으며, 이는 파일 업로드 날짜/시간으로 대체 됨



- 시작점은 `MEDIA_ROOT` 이후이다. (생략되어있는거! settings에 설정해줬잖아)
  - `upload_to` 를 설정하지 않는다면 `MEDIA_ROOT` 에 설정한 폴더에 저장된다.

✔ 참고) 이미 설계도 한번 만들어진 모델에 중간에 코드 넣어도 어차피 나중에 작성한거라, 테이블 중간에 들어가는게 아니라 마지막으로 들어간다.

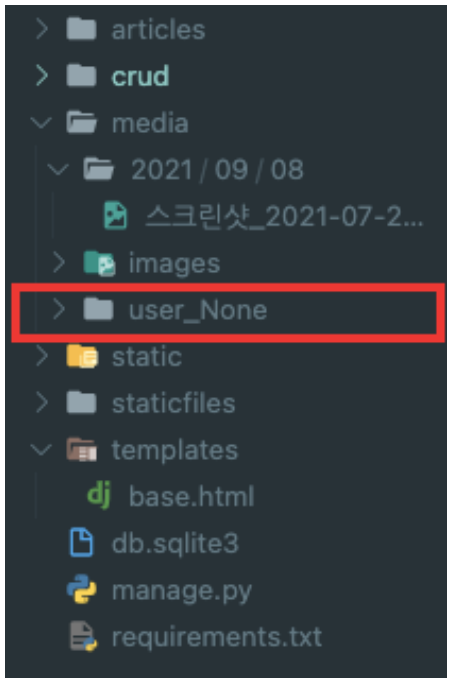
2. 함수 호출

- 함수를 이용하면 더 복잡하고 구조적인 경로 만들 수 있다.
- **반드시 2개의 인자(instance, filename)를 사용 함**
  1. **instance**
  - **저장되는 객체**
    - `FileField`가 정의된 모델의 인스턴스
  - 대부분 이 객체는 아직 데이터베이스에 저장되지 않았으므로 `PK` 값이 아직 없을 수 있음
    - 그럼 `None`으로 만들어진다.
  2. **filename**
  - 기존 파일에 제공된 **파일 이름**

```
# models.py
                          # 인자 순서 상관 X
def articles_image_path(instance, filename):
    # MEDIA_ROOT/user_<pk>/ 경로로 <filename> 이름으로 업로드
    # 하나의 경로를 만드는거임
    return f'user_{instance.pk}/{filename}'

class Article(models.Model):  # upload_to에 위 함수의 return값을 넣는다.
    image = models.ImageField(upload_to=articles_image_path)
```

✔ 참고) `instance.user.pk` : 이 객체를 만든 유저의 pk

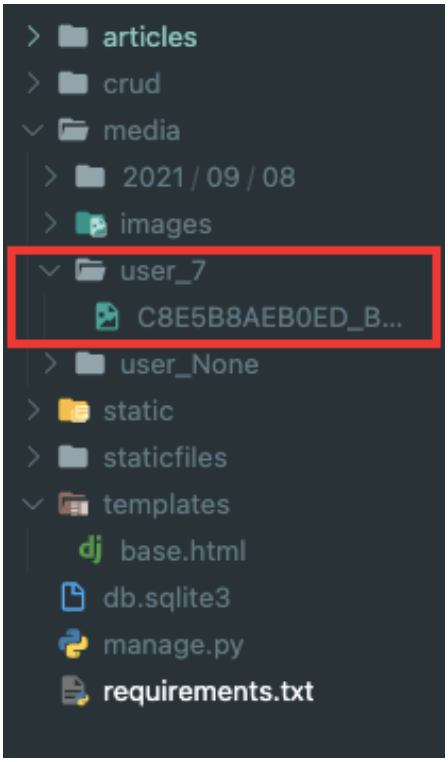
이 코드를 썼다면 ??????? ↓↓↓↓↓



- 폴더 이름이 `user_None` 인 것을 보니, `image` 객체가 만들어지는 시점에 `instance.pk` 값이 없었다(`None`) 는 것을 알 수 있다.

- 왜냐면 저장 되기 전이기 때문

- 그럼 해당 글을 수정(이미지파일도 수정) 해보자



  - `user_7` 이 생성되었다.

    ➡ 이제 `instance`는 `pk` 값을 가진 이후이다. (7번째) 그러니 수정할 때의 시점은 `None`이 아닌 것이다.
    (따라서 `pk`값을 사용하면 `None` 값이 나오니까, `pk`보다는 구분할 수 있는 다른 값들을 사용해서 해보기)

ImageField (or FileField)를 사용하기 위한 몇 가지 단계

1. settings.py에 `MEDIA_ROOT`, `MEDIA_URL` 설정
2. `upload_to` 속성을 정의하여 업로드 된 파일에 사용 할 `MEDIA_ROOT`의 하위 경로를 지정
3. 업로드 된 파일의 경로는 django가 제공하는 **`url` 속성**을 통해 얻을 수 있다.

```
<img src="{{ article.image.url }}" alt="{{ article.image }}">
```

  - `image`라는 애도 속성을 가지고 있다. (`image.url`) ~~(article.title 처럼)~~
  - `article.image`에는 파일이름과 데이터(?)가 들어있는데,

    경로(`article.image.url`)는 추가 작업이 필요하다.
    `upload_to`에 정의해놓은 " MEDIA_URL에 정해놓은 경로 + images 경로 + 파일이름 "

➡ **즉, `FileField`나 `ImageField`를 통해 업로드된 것들을 제공해주기 위해서는 경로를 알아야한다.**

그래서 장고는 `.url` 속성을 제공해준다.
왜 ? **데이터베이스에서는 저장되어 있는 값이 객체 그대로가 아니라 경로가 저장되어있기 때문에 !!!**

MEDIA_ROOT

```
# settings.py

MEDIA_ROOT = BASE_DIR / 'media'
```

- **사용자가 업로드 한 파일(미디어 파일)들을 보관할 디렉토리의 절대 경로**
  - 사용자가 업로드한 파일이 다 저기 경로로 모임
  - 미디어 파일이 올라가는 파일의 시작점은 `MEDIA_ROOT`.
    `MEDIA_ROOT`의 경로가 만들어 져야 그 경로가 데이터베이스에 저장이 된다.
  - 실제로 저장되는건 이미지 자체가 아니라 파일의 경로가 저장되니까 (성능을 위해)
- django는 성능을 위해 업로드 파일은 데이터페이스에 저장하지 않음
  - **실제 데이터베이스에 저장되는 것은 파일의 경로 ★**
- **[주의] `MEDIA_ROOT` 는 `STATIC_ROOT`와 반드시 다른 경로로 지정해야 함**

✔ ⚠ 폴더 생성해야 하는거 아님!!. 사용자가 업로드하면 자동으로 폴더가 만들어지면서 업로드 된다.

MEDIA_URL

```
# settings.py

MEDIA_URL = '/media/'
```

- **`MEDIA_ROOT`에서 제공되는 미디어를 처리하는 URL**
- 업로드 된 파일의 주소(URL)를 만들어 주는 역할
  - **웹 서버 사용자가 사용하는 public URL**
  - [상세설명] `STATIC_URL` 처럼 사용자가 업로드한 이미지파일을 사용자에게 제공하려면 미디어 파일들에 대한 고유의 url이 있어야 한다. 그걸 처리하는게 `MEDIA_URL`
- 일반적으로 '/media/' 라고 한다. 굳이 다른것으로 바꾸지 않는편
- ⚠ **주의** ⚠
  - 비어 있지 않은 값으로 설정 한다면 **반드시 slash(/)로 끝나야 함**
  - **`MEDIA_URL` 은 `STATIC_URL` 과 반드시 다른 경로로 지정해야 함**

개발 단계에서 사용자가 업로드 한 파일 제공하기

```python
# crud/urls.py

from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static


urlpatterns = [
    path('admin/', admin.site.urls),
    path('articles/', include('articles.urls')),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)


# 업로드 된 파일의 URL == settings.MEDIA_URL
# 위 URL을 통해 참조하는 파일의 실제 위치 == settings.MEDIA_ROOT
```

- 사용자가 업로드한 파일이 우리 프로젝트에 업로드 되지만,
  실제로 사용자에게 제공하기 위해서는 **업로드 된 파일의 URL**이 필요함
- 공식문서) 정적 파일 관리 - 개발 중, 사용자가 업로드한 파일 제공

```python
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    # ... the rest of your URLconf goes here ...
] + static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)
```

  - (공식 문서에서 가져온 코드)
  - `MEDIA_URL`과 `MEDIA_ROOT`는 이전에 꼭 작성되어 있어야 한다고 합니다.

```python
urlpattenrs = [

] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

    - `settings.MEDIA_URL` : 제공되는 url (요청이 들어오는 곳 (`articles/`))
    - `document_root=settings.MEDIA_ROOT` : url이 제공될때 사용하는 실제 파일의 경로 (실제 경로로 처리하는 곳 (`article/upload_to/img.png`))

----- 요기까지는 파일 받을 준비 였습니다-----

Image Upload

< CREATE >

1. ImageField 작성

`upload_to='images/'`

- **실제 이미지가 저장되는 경로를 지정**
- 사용하면 `MEDIA_ROOT/images/abc.jpg` 가 된다.

`blank=True`

- **이미지 필드에 빈 값(빈 문자열)이 허용되도록 설정 (이미지를 선택적으로 업로드 할 수 있도록)**
- 사전에 작성된 게시물들은 이미지가 없다. (그럼 이미지 필드 추가할때 과거의 아이들은 이미지 필드 없는데 넣어야 하지 않을까 ? 라고 이야기 해준다.)
  또한 게시판은 사진 무조건 올려야 하는거 아니니, 사진 없어도 빈 값으로도 저장될 수 있어야 한다. 그걸 위해 사용하는것

```
# articles/models.py

class Article(models.Model):
    # saved to 'MEDIA_ROOT/images'
    image = models.ImageField(upload_to='images/', blank=True)
```

Model field option - "blank"

- 기본 값 : `False`
- **`True`인 경우 필드를 비워둘 수 있음**
  - **DB에는 `''`(빈 문자열)이 저장됨**
- 유효성 검사에서 사용 됨 (`is_valid()`)
  - 필드에 `blank=True`가 있으면 form 유효성 검사에서 빈 값을 입력할 수 있음

Model field option - "null"

- 기본 값 : `False`
- **`True`면 django는 빈 값을 DB에 NULL로 저장**

  ✔ DB는 기본적으로 Not Null이라는 옵션을 기본적으로 가지고 있다.
  그럼 빈 값은 어떻게 처리해 ? ☞ 빈 문자열로 처리하도록
- 주의사항
  - `CharField`, `TextField`, `ImageField`(★경로가 저장되니까)와 같은 **문자열 기반 필드에는 사용하는 것을 피해야 함 ★★★**
  - 문자열 기반 필드에 `True`로 설정 시 '데이터 없음(no data)'에
    "빈 문자열(1)"과 "NULL(2)"의 2가지 가능한 값이 있음을 의미하게 됨

- 대부분의 경우 "데이터 없음"에 대해 두 개의 가능한 값을 갖는 것은 중복되며,
  Django는 NULL이 아닌 빈 문자열을 사용하는 것이 규칙
  - 데이터 없음을 두가지로 표현하면, 데이터 없음을 표현하는 정확한 의미표현 할 수 없다.
  - 따라서 문자열 기반 필드에는 `null=True`를 안쓰는게 맞다.

blank & null 비교

- **blank**
  - Validation-related (유효성과 관련)
- **null**
  - Database-related (DB와 관련)

➡ 문자열 기반 및 비문자열 기반 필드 모두에 대해 null option은 DB에만 영향을 미치므로,
`form`에서 빈 값을 허용하려면 **`blank=True`**를 설정해야 함

적용해보자 ! ! !

```python
# models.py

class Person(models.Model):
    name = models.CharField(max_length=10)

    # null=True 금지
    bio = models.TextField(max_length=50, blank=True)

    # null, blank 모두 설정 가능 -> 문자열 기반 필드가 아니기 때문
    birth_date = models.DateField(null=True, blank=True)
```

```
$ python manage.py makemigrations

$ pip install Pillow

$ python manage.py makemigrations
$ python manage.py migrate

$ pip freeze > requirements.txt
```

마이그레이션 실행
(단, ImageField를 사용하기 위해서는 Pillow 라이브러리 설치 필요)

CREATE

Title:

Content:

Image: 파일 선택 4EC764E8-C9A...D9409B83.jpeg

작성

[back]

```
▼<form action="/articles/create/" method="POST">
  <input type="hidden" name="csrfmiddlewaretoken" value="P84XMzog6vxGTIa
  PADBEMPbCT81iN7t7e9gfc63fq03gzfGlk07uTI0sOvdeWR6f">
  ▶<p>…</p>
  ▶<p>…</p>
  ▼<p>
    <label for="id_image">Image:</label>
    <input type="file" name="image" accept="image/*" id="id_image">
  </p>
  <input type="submit" value="작성">
</form>
```

id	title	content	created_at	updated_at	image
1	안녕	테스트	2021-09-08 00:43:26.659450	2021-09-08 00:43:26.659500	
2	여행가고싶어요	여행가고시팔구	2021-09-08 02:05:04.586414	2021-09-08 02:05:04.586634	

- 엥 근데 아직 비어있네 ????????? 뭐 설정해줘야할까 ?? ↓↓↓

2) form 요소 - enctype(인코딩) 속성

1. **multipart/form-data**
    - **파일 이미지 업로드 시에 반드시 사용**
    - **`<input type="file">`을 사용할 경우에 사용**
2. ~~application/x-www-form-urlencoded~~
    - (기본값) 모든 문자 인코딩
3. ~~text/plain~~
    - 인코딩을 하지 않은 문자 상태로 전송
    - 공백은 '+' 기호로 변환하지만, 특수 문자는 인코딩 하지 않음

multipart/form-data 적용

1. **`form` 태그 `enctype` 속성에 multipart/form-data 넣어주기**

```
<!-- articles/create.html -->

<form action="{% url 'articles:create' %}" method="POST" enctype="multipart/form-data">
  {% csrf_token %}
  {{ form.as_p }}
  <input type="submit" value="작성">
</form>
```

- `enctype` 속성은 method 특성이 `POST`인 경우에만 작동한다.
- MDN - form태그 enctype 속성

2. **`input` 태그 `accept` 속성 지정**

```
<!-- articles/create.html -->

<form action="{% url 'articles:create' %}" method="POST" enctype="multipart/form-data">
  {% csrf_token %}
  {{ form.as_p }}
  <input type="submit" value="작성" accept="image/*">
</form>
```

- `accept=""` 하면 지정된 형식만 받는다.

input - accept 속성

- **입력 허용할 파일 유형을 나타내는 문자열**
- 쉼표로 구분된 "고유 파일 유형 지정자" (unique file type specifiers)
- ⚠ **파일 검증을 하는 것은 아님** ⚠
  (이미지만 accept 해 놓더라도 비디오나 오디오 파일을 제출할 수 있음)
    - 사용자의 경험을 올려주는 것이라고 생각하면 된다. 사용자가 편하도록! 사용자의 경험 측면에서 도움을 주는 것.
- **고유 파일 유형 지정자**

: `<input type="file">` 에서 선택할 수 있는 파일의 종류를 설명하는 문자열

- **파일 업로드 시 허용할 파일 형식에 대해 자동으로 필터링**

> - 이미지파일로 필터링 되는 것처럼. (열기 눌렀을 때 이미지 파일들만 보이도록)

  - 근데 우리가 바꿀 수 있다. 드롭박스 클릭해서 !! 그래서 충분히 다른 파일들 올릴 수 있다.
    ⚠ **이미지 파일만 올릴 수 있는 것이 아님** ⚠

```
▼<form action="/articles/create/" method="POST">
    <input type="hidden" name="csrfmiddlewaretoken" value="P84XMzog6vxGTIa
    PADBEMPbCT81iN7t7e9gfc63fq03gzfGlk07uTI0sOvdeWR6f">
  ▶<p>…</p>
  ▶<p>…</p>
  ▼<p>
      <label for="id_image">Image:</label>
      <input type="file" name="image" accept="image/*" id="id_image">
    </p>
    <input type="submit" value="작성">
  </form>
```

✔ 공식 문서

- MDN - input 태그 accept 속성
- accpet 특성은 파일 입력 칸이 허용할 파일 유형을 나타내는 문자열로, 쉼표로 구분한 **고유 파일 유형 지정자**의 목록입니다.
- `image/*` 는 모든 이미지 파일을 의미한다. 확장자 상관없이 !!!
  - `*`은 사용자 지정 형식을 의미한다.
  - 표준 이미지 형식 뿐만 아니라 PDF 파일도 받을 수 있어야 한다면?

```
<input type="file" accept="image/*,.pdf">
```

그래도 안돼 이미지가 안떠 !!!! 왜 ????

이미지는 `request.POST` 에 들어있지 않으니까 !

Request Information을 보면 `POST`가 아닌 `FILES`에 있는 것을 알 수 있다.

수정하러 가자 ↓↓↓↓

3) Views.py 수정

- **업로드한 파일은 `request.FILES` 객체로 전달됨**

```python
# views.py

@require_http_methods(['GET', 'POST'])
def create(request):
    if request.method == 'POST':
        form = ArticleForm(request.POST, request.FILES)
        # form = ArticleForm(data=request.POST, files=request.FILES)
        if form.is_valid():
            article.save()
            return redirect('articles:detail', article.pk)
    else:
        form = ArticleForm()
    context = {
        'form': form,
    }
    return render(request, 'articles/create.html', context)
```

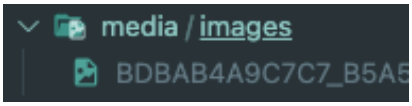
  - Model Form(`ArticleForm`)에 들어가는 두번째 인자의 키워드가 `files`이다. 그래서 순서가 맞기 때문에 `files=` 생략 가능하다.

- 여기까지 하면 웹에서는 아직 안보이지만, DB에는 경로가 저장되었다.

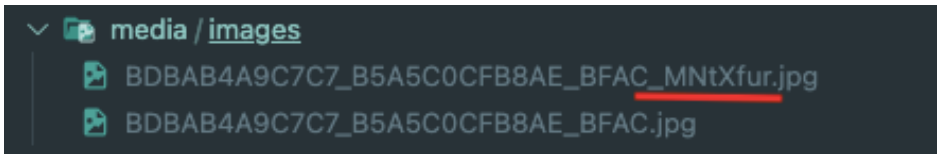
id	title	content	created_at	updated_at	image
1	안녕	테스트	2021-09-08 00:43:26.659450	2021-09-08 00:43:26.659500	
2	여행가고싶어요	여행가고시팓구	2021-09-08 02:05:04.586414	2021-09-08 02:05:04.586634	
3	gdg	gdgd	2021-09-08 05:36:48.165789	2021-09-08 05:36:48.165842	
4	gdg	gdgd	2021-09-08 05:40:32.020332	2021-09-08 05:40:32.020461	images/BDBAB4A9C7C7_B5A5C0CFB8AE_BFAC.jpg

  ✔ 경로의 기준 ? 앞에 `MEDIA_ROOT/` 가 생략되어 있는거임. 그 이후부터의 경로가 올라가있는고임

- 그리고 media 폴더가 생겼고, 하위에 `/images/어쩌구.jpg` 생긴 것을 알 수 있다.



✔ 만약 같은 이미지 파일을 올린다면 ??



- 장고가 덮어씌워지지 않도록 이름 뒤에 임의의 난수형태의 문자열값을 붙여준다.

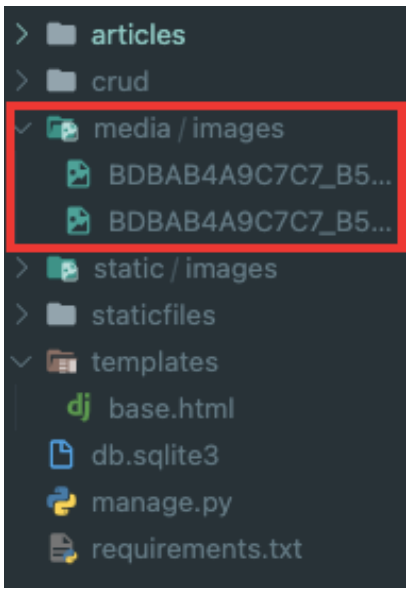
✔ 참고) 누군가의 질문

```
if request.method == 'POST':
    form = ArticleForm(request.POST, request.FILES)
```

- ??? : 위 코드에서 `request.FILES` 는 POST 요청과 GET 요청 모두 통과하는 걸까요?
- ➡ 조건문을 보자. `request.FILES`가 POST로 왔냐 라는 코드 아니다.
  `request.FILES`는 파일만 담겨온거임 `request.method`는 속성!
  둘 각각 다른 정보를 담고 있다. 파이썬이라는걸 생각하자

DB 및 파일 트리 확인

- 실제 파일 위치
  - MEDIA_ROOT/images/



- ★ DB에 저장되는 것은 **파일의 경로** ★

id	title	content	created_at	updated_at	image
1	안녕	테스트	2021-09-08 00:43:26.659450	2021-09-08 00:43:26.659500	
2	여행가고싶어요	여행가고시팓구	2021-09-08 02:05:04.586414	2021-09-08 02:05:04.586634	
3	gdg	gdgd	2021-09-08 05:36:48.165789	2021-09-08 05:36:48.165842	
4	gdg	gdgd	2021-09-08 05:40:32.020332	2021-09-08 05:40:32.020461	images/BDBAB4A9C7C7_B5A5C0CFB8AE_BFAC.jpg

< READ >

1) 이미지 경로 불러오기

- `article.image.url` == 업로드 파일의 경로
- `article.image` == 업로드 파일의 파일 이름

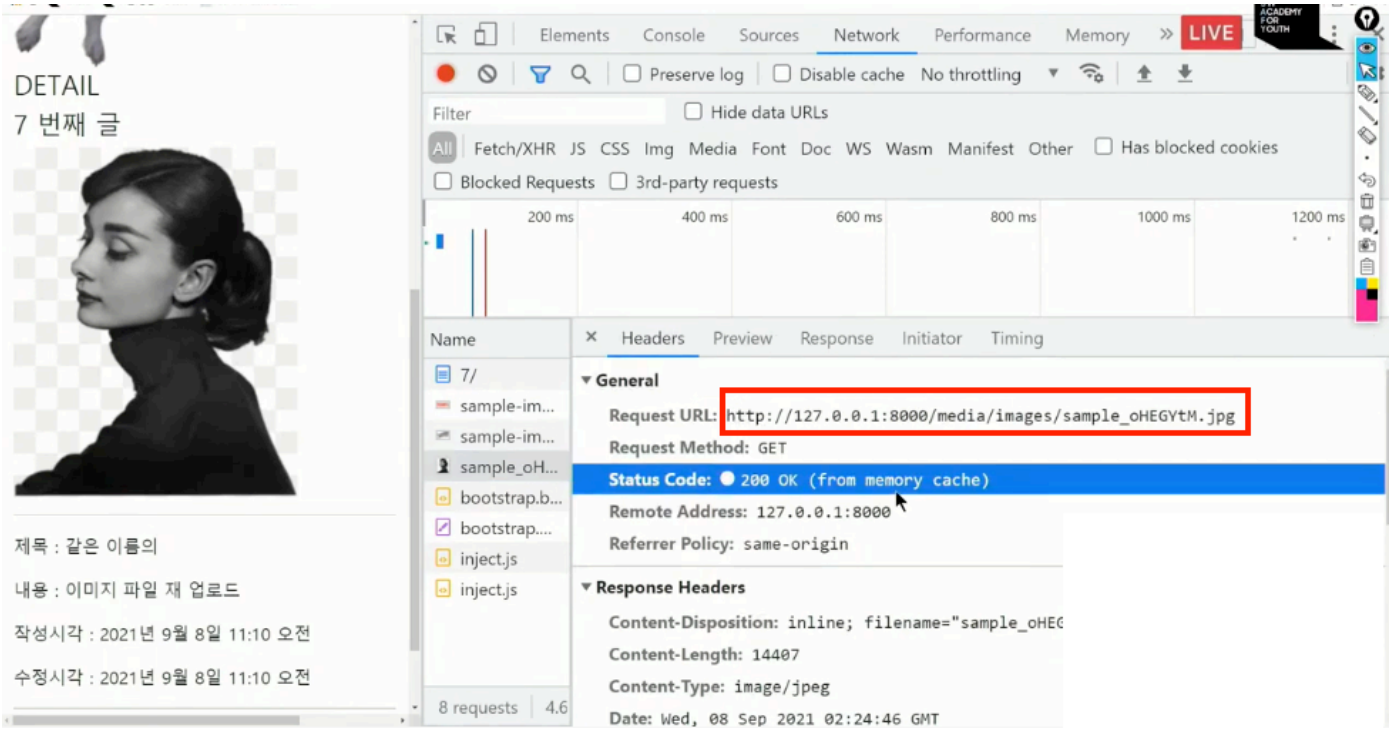
```
<!-- detail.html -->

<img src="{{ article.image.url }}" alt="{{ article.image }}">
```

MEDIA_URL 확인하기

- 이미지 url(current source)은 바로 `MEDIA_URL`이 만들어준 주소이다.
  사용자가 업로드 한 이미지를 제공하고 있다.

STATIC_URL과 MEDIA_URL

- static, media 결국 서버에 요청해서 조회하는 것
  - 요청을 하려면 요청을 받는 주소가 있어야겠지 ?
  - 이미지 하나하나 주소가 있어야 우리가 조회할 수 있다.
- 서버에 요청하기 위한 url을 urls.py가 아닌 settings에 먼저 작성 후, urlpatterns에 추가하는 방식



- 개발자 도구 - Network 탭에서 이미지를 조회하기 위해 요청을 보내는 주소(Request URL) 확인

< UPDATE >

1) 이미지 수정하기

- 이미지는 바이너리 데이터(하나의 덩어리)이기 때문에 텍스트처럼 일부만 수정 하는 것은 불가능
- 때문에 **새로운 사진으로 덮어 씌우는 방식을 사용**

수정 1) enctype 추가

```
<!-- articles/update.html -->

{% extends 'base.html' %}

{% block content %}
  <h1>UPDATE</h1>
  <form action="{% url 'articles:update' article.pk %}" method="POST" enctype="multipart/form-data">
    {% csrf_token %}
    {{ form.as_p }}
    <button>수정</button>
  </form>
  <hr>
  <a href="{% url 'articles:index' %}">[back]</a>
{% endblock content %}
```

수정 2) views.py 수정

```python
# view.py

@require_http_methods(['GET', 'POST'])
def update(request, pk):
    article = get_object_or_404(Article, pk=pk)
    if request.method == 'POST':
        form = ArticleForm(request.POST, request.FILES, instance=article)
        if form.is_valid():
            form.save()
            return redirect('articles:detail', article.pk)
    else:
        form = ArticleForm(instance=article)
    context = {
        'article': article,
        'form': form,
    }
    return render(request, 'articles/update.html', context)
```

- create 함수에서와 같이 두번째 인자의 키워드가 `files`이다. 그래서 순서가 맞기 때문에 키워드 생략 가능하다.

수정 3) 이미지 안넣고 싶을 때

이미지 업로드 기능 작성하기 전의 게시물을 들어가면 이미지가 없으므로 컬럼이 비어있다. 그럼 아래처럼 에러가 난다. (ValueError)

그리고 게시물 작성할때 이미지 안넣고 싶으면 어떻게 해 ? 아래에서 해결해보자

ValueError at /articles/3/

The 'image' attribute has no file associated with it.

Request Method:	GET
Request URL:	http://127.0.0.1:8000/articles/3/
Django Version:	3.2.7
Exception Type:	ValueError
Exception Value:	The 'image' attribute has no file associated with it.
Exception Location:	/Users/wizdom/.pyenv/versions/3.9.6/lib/python3.9/site-packages/django/db/models/fields/files.py, line 40, in _require_file
Python Executable:	/Users/wizdom/.pyenv/versions/3.9.6/bin/python
Python Version:	3.9.6
Python Path:	['/Users/wizdom/Desktop/ssafy/무제 폴더/03_django_staticfiles', '/Users/wizdom/.pyenv/versions/3.9.6/lib/python39.zip', '/Users/wizdom/.pyenv/versions/3.9.6/lib/python3.9', '/Users/wizdom/.pyenv/versions/3.9.6/lib/python3.9/lib-dynload', '/Users/wizdom/.pyenv/versions/3.9.6/lib/python3.9/site-packages']
Server time:	Wed, 08 Sep 2021 18:36:14 +0900

- **detail 페이지를 출력하지 못하는 문제 해결 방법**
  (image가 없는 게시글의 경우 출력할 이미지가 없기 때문)
  1. 가장 기본적인 방법.
     - **조건문으로 `article`에 `image` 객체가 있을 때에만(값이 존재한다면) 출력**. 아니면 빈칸

```
<!-- detail.html -->

{% extends 'base.html' %}

{% block content %}
  <h1>DETAIL</h1>
  <h2>{{ article.pk }} 번 글</h2>
  {% if article.image %}
    <img src="{{ article.image.url }}" alt="{{ article.image }}">
  {% endif %}
```

     - 아니면 **else문으로 기본 이미지 넣어주는 방법**도 있다.
       - 이건 미리 준비해 놓는 파일이니 static file 이다 !!

```
{% if article.image %}
   <img src="{{ article.image.url }}" alt="{{ article.img }}">
{% else %}
  <img src="{% static 'images/asd.jpeg' %}" alt="default image">
{% endif %}
```

Image Resizing

이미지 크기 변경하기 (1/3)

- 실제 원본 이미지를 서버에 그대로 업로드 하는 것은 서버의 부담이 큰 작업
- `<img>` 태그에서 직접 사이즈를 조정할 수도 있지만 (width와 height),
  업로드 될 때 이미지 자체를 resizing 하는 것을 사용해 볼 것 (resizing해서 저장된다.)
  - `<img>` 태그를 이용하는건 근본적인 해결이 아님. 왜냐하면 원본은 media 폴더에 그대로 있으니까
    원본을 자른게 아니라, 출력을 작게 만들뿐인고임
- [django-imagekit](#) 라이브러리 활용
  - 썸네일, 흑백사진 처리 등 다양한 기능이 있다.
  - 위 공식문서 잘 활용해서 만들어보기 !

이미지 크기 변경하기 (2/3) - django-imagekit 라이브러리 활용

1. **django-imagekit** 설치

   (그전에 `PIL` 또는 `Pillow` 가 설치되어 있어야한다. (그런데 `ImageField` 사용하고 있다면 이미 설치했겠지?))

   ```
   $ pip install django-imagekit
   $ pip freeze > requirements.txt
   ```

2. `INSTALLED_APPS` 에 추가

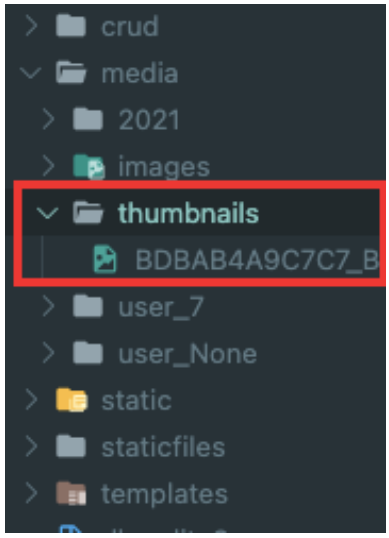
   ```
   # settings.py

   INSTALLED_APP = [
       ...
       'imagekit',
       ...
   ]
   ```

이미지 크기 변경하기 (3/3)

1) 원본 이미지를 재가공하여 저장하기 (원본x, 썸네일o)

```
# articles/models.py

from django.db import models
from imagekit.models import ProcessedImageField
from imagekit.processors import ResizeToFill


# Create your models here.
class Article(models.Model):
    ...
```

```
    image = ProcessedImageField(
        upload_to='thumbnails/',
        processors=[ResizeToFill(100, 50)],
        format='JPEG',
        options={'quality': 60}
    )
    ...
```



- `media/thumbnails`에 **resizing된 이미지가 저장되어 있다.** 원본 저장 아님 !!
- 업로드 되는 시점에 `ResizeToFill()` 이 동작되었다.
  퀄리티도 낮춰 설정해주어서 화질도 낮아짐.

- 위의 코드에서 `ProcessedImageField()`의 parameter로 작성된 값들은 변경이 되더라도 **다시 makemigrations를 해줄 필요없이 즉시 반영 됨**
  왜냐면 모델 자체 설계도가 변화하는 요소들이 아니기 때문

2) 원본도 저장, 썸네일도 저장

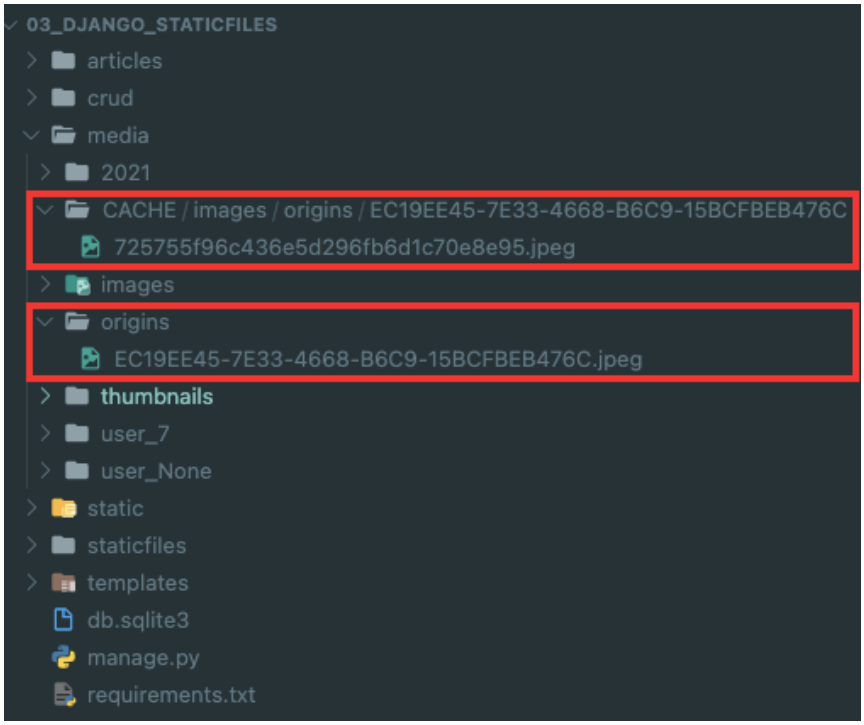
원본이 하나 있고, 그거에 대해 소스를 연결하고 있는 것 하나

```
# articles/models.py

from django.db import models
from imagekit.models import ImageSpecField
from imagekit.processors import ResizeToFill


# Create your models here.
class Article(models.Model):
    ...
    # 원본 O, 썸네일 O
    image = models.ImageField(upload_to='origins/', blank=True)
```

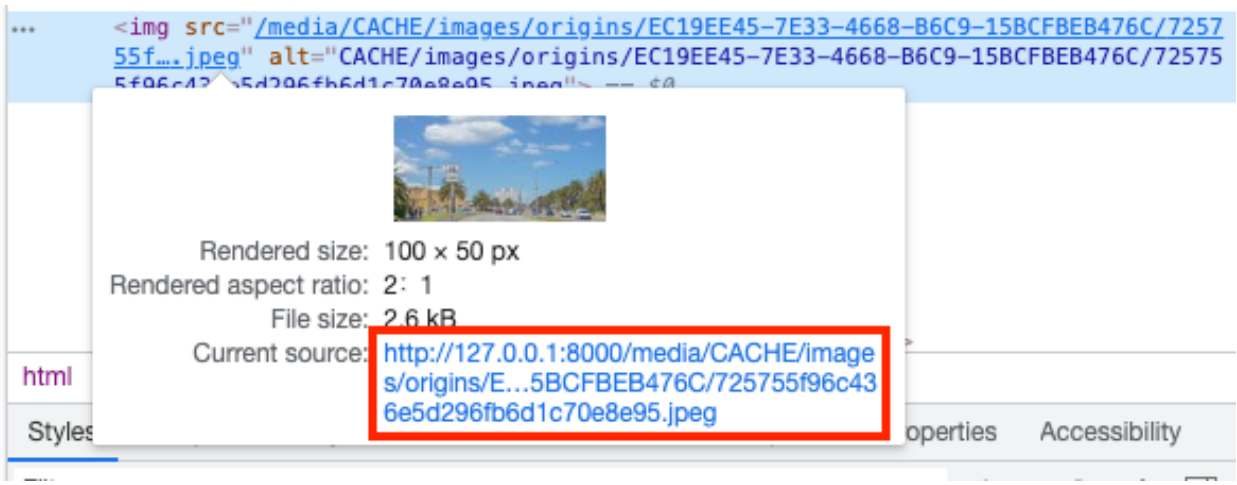
```
    image_thumbnail = ImageSpecField(
        source='image', # source에 원본이미지에 컬럼 이름을 쓴다.
        processors=[ResizeToFill(100, 50)],
        format='JPEG',
        options={'quality': 90},
    )
    ...
```

```
<!-- detail.html -->

<img src="{{ article.image_thumbnail.url }}" alt="{{ article.image_thumbnail }}">
```

- `detail.html` 의 `img` 태그 수정하기
- 이 방식쓰면 비율도 안깨진다



- 이걸 한번 호출(detail로 이동)하는 순간 캐시라고 하는 파일(썸네일 이미지)이 하나 생겼다.
- **이 구조는 처음에 업로드할때는 원본만 저장한다,**
  썸네일은 조회하기 전까지는 안만든다. 사용을 해야 만들어짐 (필요할때만 생성할 수 있도록)(효율성 고려)



✔ models.py 조작하고나서 잊지말자

```
$ python manage.py makemigrations
$ python manage.py migrate
```

참고

캐시

: 데이터나 값을 미리 복사해놓는 임시 장소

- 왜 쓸까?
    - 한 페이지에 이미지 세개 있다고 가정하자. 이 페이지 띄울때 마다 용량이 큰 이미지 세개를 매번 다 요청해서 가져와야 할까 ? 브라우저가 ? 그럼 데이터 소모값이 너무 크지 않을까 ?
    - 그럼 어떻게 처리하냐면 ↓

아래 사진(원래 페이지에서 새로고침함)에서 size보면 다른애들은 용량을 가지고 있는데, img는 memory cache 사이즈를 가지고 있다.
이 용량만큼 데이터를 쓴 것이다. 근데 이미지를 대상으로는 안쓴 것이다.

왜냐면 브라우저에 캐시 메모리 어딘가에 이미지들을 저장해두고 가져다 쓴것이다. (다시 요청보내서 용량만큼 쓴게 아니라) 한번 들어갔을때 사이즈 큰것들을 판단을해서 자기 브라우저 메모리에 넣어둔것이다.

Name	Status	Type	Initiator	Size	Time
9/	200	document	Other	2.0 kB	97 ms
logo.png	200	png	(index)	(memory cache)	0 ms
unknown.png	200	png	(index)	(memory cache)	0 ms
bootstrap.min.css	200	stylesheet	(index)	(disk cache)	5 ms
725755f96c436e5d296fb6d1c70e8e9…	304	jpeg	(index)	209 B	27 ms
bootstrap.bundle.min.js	200	script	(index)	(memory cache)	1 ms
inject.js	200	script	content.js:36	1.3 kB	6 ms
inject.js	200	script	content.js:36	1.3 kB	9 ms

캐시 비우고 새로고침하면 처음 받아야하는거니까 사이즈뜬다. 근데 새로고침하면 위와 같아진다.

Name	Status	Type	Initiator	Size	Time
9/	200	document	Other	2.0 kB	17.85 s
bootstrap.min.css	200	stylesheet	(index)	25.8 kB	62 ms
logo.png	200	png	(index)	6.2 kB	10 ms
bootstrap.bundle.min.js	200	script	(index)	23.8 kB	91 ms
unknown.png	200	png	(index)	139 kB	11 ms
725755f96c436e5d296fb6d1c70e8e95…	200	jpeg	(index)	3.0 kB	21 ms
favicon.ico	404	text/html	Other	2.6 kB	35 ms
inject.js	200	script	content.js:36	1.3 kB	6 ms
inject.js	200	script	content.js:36	1.3 kB	7 ms