

PJT 06

PJT 06 라이브

목차

- Handling HTTP requests
- Media files
 - Image Upload
 - Image Resizing

Handling HTTP requests

| Handling HTTP requests

- Django에서 HTTP 요청을 처리하는 방법
 1. Django shortcut functions
 2. View decorators

1. Django shortcuts functions (1/2)

- django.shortcuts 패키지는 개발에 도움 될 수 있는 여러 함수와 클래스를 제공
- shortcuts function 종류
 - render()
 - redirect()
 - get_object_or_404()
 - get_list_or_404()

1. Django shortcuts functions (2/2)

- 만약 `render()`가 없었다면..

```
from django.http import HttpResponse
from django.template import loader

def index(request):
    articles = Article.objects.order_by('-pk')
    template = loader.get_template('articles/index.html')
    context = {
        'articles': articles,
    }
    return HttpResponse(template.render(context, request))
```



```
from django.shortcuts import render

def index(request):
    articles = Article.objects.order_by('-pk')
    context = {
        'articles': articles,
    }
    return render(request, 'articles/index.html', context)
```

| `get_object_or_404()`

- 모델 manager인 objects에서 `get()`을 호출하지만,
해당 객체가 없을 경우 `DoesNotExist` 예외 대신 `Http 404`를 raise
- `get()`에 경우 조건에 맞는 데이터가 없을 경우에 예외를 발생 시킴
 - 코드 실행 단계에서 발생한 예외 및 에러에 대해서 브라우저는 http status code 500으로 인식함
- 상황에 따라 적절한 예외처리를 하고

클라이언트에게 올바른 에러 상황을 전달하는 것 또한 개발의 중요한 요소 중 하나

| get_object_or_404() 사용 전

```
[20/Mar/2021 19:56:04] "GET /articles/100/ HTTP/1.1" 500 78822
```

존재하지 않는 게시물 조회 시

| get_object_or_404() 사용 후

```
# views.py

from django.shortcuts import get_object_or_404

def detail(request, pk):
    article = get_object_or_404(Article, pk=pk)

def update(request, pk):
    article = get_object_or_404(Article, pk=pk)

def delete(request, pk):
    article = get_object_or_404(Article, pk=pk)
```

```
[20/Mar/2021 19:56:49] "GET /articles/100/ HTTP/1.1" 404 1739
```

존재하지 않는 게시물 조회 시

| [참고] `get_object_or_404()` 를 사용하지 않는다면..

```
from django.http import Http404

def detail(request, pk):
    try:
        article = Article.objects.get(pk=pk)
    except Article.DoesNotExist:
        raise Http404('No Article matches the given query.')
    context = {
        'article': article,
    }
    return render(request, 'articles/detail.html', context)
```

2. Django View decorators (1/6)

- Django는 다양한 HTTP 기능을 지원하기 위해 view 함수에 적용할 수 있는 여러 데코레이터를 제공
- [참고] Decorator (데코레이터)
 - 어떤 함수에 기능을 추가하고 싶을 때, 해당 함수를 수정하지 않고 기능을 연장 해주는 함수
 - 즉, 원본 함수를 수정하지 않으면서 추가 기능만을 구현할 때 사용

| 2. Django View decorators (2/6)

- Allowed HTTP methods
 - 요청 메서드에 따라 view 함수에 대한 액세스를 제한
 - 요청이 조건을 충족시키지 못하면 `HttpResponseNotAllowed`을 return
(405 Method Not Allowed)
 - `require_http_methods()`, `require_POST()`, `require_safe()`,
`require_GET()`

| 2. Django View decorators (3/6)

1. `require_http_methods()`

- view 함수가 특정한 method 요청에 대해서만 허용하도록 하는 데코레이터

2. `require_POST()`

- view 함수가 POST method 요청만 승인하도록 하는 데코레이터

3. `require_safe()`

- view 함수가 GET 및 HEAD method만 허용하도록 요구하는 데코레이터
- django는 `require_GET` 대신 `require_safe`를 사용하는 것을 권장

2. Django View decorators (4/6)

- View decorator 작성

```
# views.py

from django.views.decorators.http import require_http_methods, require_POST, require_safe

@require_safe
def index(request):
    pass

@require_http_methods(['GET', 'POST'])
def create(request):
    pass

@require_safe
def detail(request, pk):
    pass

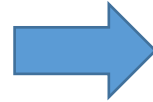
@require_POST
def delete(request, pk):
    pass

@require_http_methods(['GET', 'POST'])
def update(request, pk):
    pass
```

2. Django View decorators (5/6)

- delete view 함수 코드 변경

```
def delete(request, pk):  
    article = Article.objects.get(pk=pk)  
    if request.method == 'POST':  
        article.delete()  
        return redirect('articles:index')  
    return redirect('articles:detail', article.pk)
```



```
@require_POST  
def delete(request, pk):  
    article = Article.objects.get(pk=pk)  
    article.delete()  
    return redirect('articles:index')
```

* 불필요해진 코드 삭제

| 2. Django View decorators (6/6)

- URL 로 delete 요청 후 405 http status code 확인

```
Method Not Allowed (GET): /articles/3/delete/  
[04/Jan/2021 04:52:10] "GET /articles/3/delete/ HTTP/1.1" 405 0
```

결론

- HTTP 요청에 따라 적절한 예외처리 혹은 데코레이터를 사용해 서버를 보호하고 클라이언트에게 정확한 상황을 제공하는 것의 중요성

Media files (이미지 업로드)

| Media file

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

| Model field (1/2)

- ImageField()
 - 이미지 업로드에 사용하는 모델 필드
 - FileField를 상속받는 서브 클래스이기 때문에 FileField의 모든 속성 및 메서드를 사용 가능하며, 더해서 사용자에게 의해 업로드 된 객체가 유효한 이미지인지 검사함
 - ImageField 인스턴스는 최대 길이가 100자인 문자열로 DB에 생성되며, max_length 인자를 사용하여 최대 길이를 변경 할 수 있음
- [주의] 사용하려면 반드시 [Pillow](https://docs.djangoproject.com/en/3.2/ref/models/fields/#django.db.models.ImageField) 라이브러리가 필요

| Model field (2/2)

- `FileField()`
 - 파일 업로드에 사용하는 모델 필드
 - 2개의 선택 인자를 가지고 있음
 1. `upload_to`
 2. `storage`

| ImageField 작성

- `upload_to='images/'`
 - 실제 이미지가 저장되는 경로를 지정
- `blank=True`
 - 이미지 필드에 빈 값(빈 문자열)이 허용되도록 설정 (이미지를 선택적으로 업로드 할 수 있도록)

```
# articles/models.py

class Article(models.Model):
    title = models.CharField(max_length=10)
    content = models.TextField()
    # saved to 'MEDIA_ROOT/images/'
    image = models.ImageField(upload_to='images/', blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

| 'upload_to' argument

- 업로드 디렉토리와 파일 이름을 설정하는 2가지 방법을 제공
 1. 문자열 값이나 경로 지정
 2. 함수 호출

| 'upload_to' argument – 1. 문자열 경로 지정 방식

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

```
# 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/')
```

[참고] time 모듈의 strftime()

- `time.strftime(format[, t])`
- 날짜 및 시간 객체를 문자열 표현으로 변환하는 데 사용됨
- 하나 이상의 형식화된 코드 입력을 받아 문자열 표현을 반환

Directive	Meaning	Notes
%a	Locale's abbreviated weekday name.	
%A	Locale's full weekday name.	
%b	Locale's abbreviated month name.	
%B	Locale's full month name.	
%c	Locale's appropriate date and time representation.	
%d	Day of the month as a decimal number [01,31].	
%H	Hour (24-hour clock) as a decimal number [00,23].	
%I	Hour (12-hour clock) as a decimal number [01,12].	
%j	Day of the year as a decimal number [001,366].	
%m	Month as a decimal number [01,12].	
%M	Minute as a decimal number [00,59].	

```
from time import gmtime, strftime

strftime('%a, %d %b %Y %H:%M:%S', gmtime())

# 'Thu, 28 Jun 2001 14:17:15'
```

| 'upload_to' argument – 2. 함수 호출 방식 (1/2)

- 반드시 2개의 인자(instance, filename)를 사용 함

1. instance

- FileField가 정의된 모델의 인스턴스
- 대부분 이 객체는 아직 데이터베이스에 저장되지 않았으므로 PK 값이 아직 없을 수 있음

2. filename

- 기존 파일에 제공된 파일 이름

| upload_to argument – 2. 함수 호출 방식 (2/2)

```
# models.py 예시

def articles_image_path(instance, filename):
    # 'MEDIA_ROOT/image_<pk>/' 경로에 '<filename>' 이름으로 업로드
    return f'image_{instance.pk}/{filename}'

class Article(models.Model):
    image = models.ImageField(upload_to=articles_image_path)
```

| Model field option – “blank”

- 기본 값 : False
- True인 경우 필드를 비워 둘 수 있음
 - DB에는 ' '(빈 문자열)이 저장됨
- 유효성 검사에서 사용 됨 (`is_valid`)
 - 모델 필드에 `blank=True`를 작성하면 form 유효성 검사에서 빈 값을 입력할 수 있음

| Model field option – “null”

- 기본 값 : False
- True인 경우 django는 빈 값에 대해 DB에 NULL로 저장
- 주의 사항
 - CharField, TextField와 같은 문자열 기반 필드에는 사용하는 것을 피해야 함
 - 문자열 기반 필드에 True로 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 중복되는 것이며, Django는 NULL이 아닌 빈 문자열을 사용하는 것이 규칙

| blank & null 비교 (1/2)

- blank
 - Validation-related
 - null
 - Database-related
- 문자열 기반 및 비문자열 기반 필드 모두에 대해 null option은 DB에만 영향을 미치므로, form에서 빈 값을 허용하려면 **blank=True**를 설정해야 함

blank & null 비교 (2/2)

```
# 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)
```


| 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**' 속성을 통해 얻을 수 있음

```

```

| MEDIA_ROOT

- 사용자가 업로드 한 파일(미디어 파일)들을 보관할 디렉토리의 절대 경로
- django는 성능을 위해 업로드 파일은 데이터베이스에 저장하지 않음
 - 실제 데이터베이스에 저장되는 것은 **파일의 경로**
- [주의] MEDIA_ROOT는 STATIC_ROOT와 반드시 다른 경로로 지정해야 함

```
# settings.py
```

```
MEDIA_ROOT = BASE_DIR / 'media'
```

| MEDIA_URL

- MEDIA_ROOT에서 제공되는 미디어를 처리하는 URL
- 업로드 된 파일의 주소(URL)를 만들어 주는 역할
 - 웹 서버 사용자가 사용하는 public URL
- 비어 있지 않은 값으로 설정 한다면 반드시 slash(/)로 끝나야 함
- [주의] MEDIA_URL은 STATIC_URL과 반드시 다른 경로로 지정해야 함

```
# settings.py
```

```
MEDIA_URL = '/media/'
```

개발 단계에서 사용자가 업로드 한 파일 제공하기

- `settings.MEDIA_URL`
 - 업로드 된 파일의 URL
- `settings.MEDIA_ROOT`
 - `MEDIA_URL`을 통해 참조하는 파일의 실제 위치

```
# 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)
```

* 사용자가 업로드 한 파일이 프로젝트에 업로드 되더라도, 실제로 사용자에게 제공하기 위해서는 업로드 된 파일의 URL이 필요함

```
$ python manage.py makemigrations
SystemCheckError: System check identified some issues:

ERRORS:
articles.Article.image: (fields.E210) Cannot use ImageField because Pillow is not installed.
    HINT: Get Pillow at https://pypi.org/project/Pillow/ or run command "python -m pip install Pillow".
```

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

```
$ pip install Pillow
```

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

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

```
$ pip freeze > requirements.txt
```

Pillow 라이브러리 설치 및 마이그레이션 진행

이미지 업로드 (CREATE)

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

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

게시글 작성 form에 enctype 속성 지정

| form 태그 - enctype(인코딩) 속성

1. multipart/form-data

- 파일/이미지 업로드 시에 반드시 사용해야 함 (전송되는 데이터의 형식을 지정)
- `<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

CREATE

Title:

Content:

Image: 파일 선택 선택된 파일 없음

제출

[back](#)

Elements Console Sources Network Performance Memory Application

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <div class="container">
      <h1>CREATE</h1>
      <hr>
      <form action="/articles/create/" method="POST">
        <input type="hidden" name="csrfmiddlewaretoken" value="ntNGjGXZ6wnfQuSrKEGt7cwFTp
        iPzfNBQhi5UAesQbk3">
        <p>...</p>
        <p>...</p>
        <p>
          <label for="id_image">Image:</label>
          ...
          <input type="file" name="image" accept="image/*" id="id_image"> == $0
        </p>
        <input type="submit">
      </form>
      <a href="/articles/">back</a>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.m
    a384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENB00LRn5q+8nbTov4+1p" crossorigin="
    </body>
  </html>
```

input 요소의 accept 속성 확인

| input 요소 - accept 속성 (1/2)

- 입력 허용할 파일 유형을 나타내는 문자열
- 쉼표로 구분된 “고유 파일 유형 지정자” (unique file type specifiers)

- 파일을 검증 하는 것은 아님

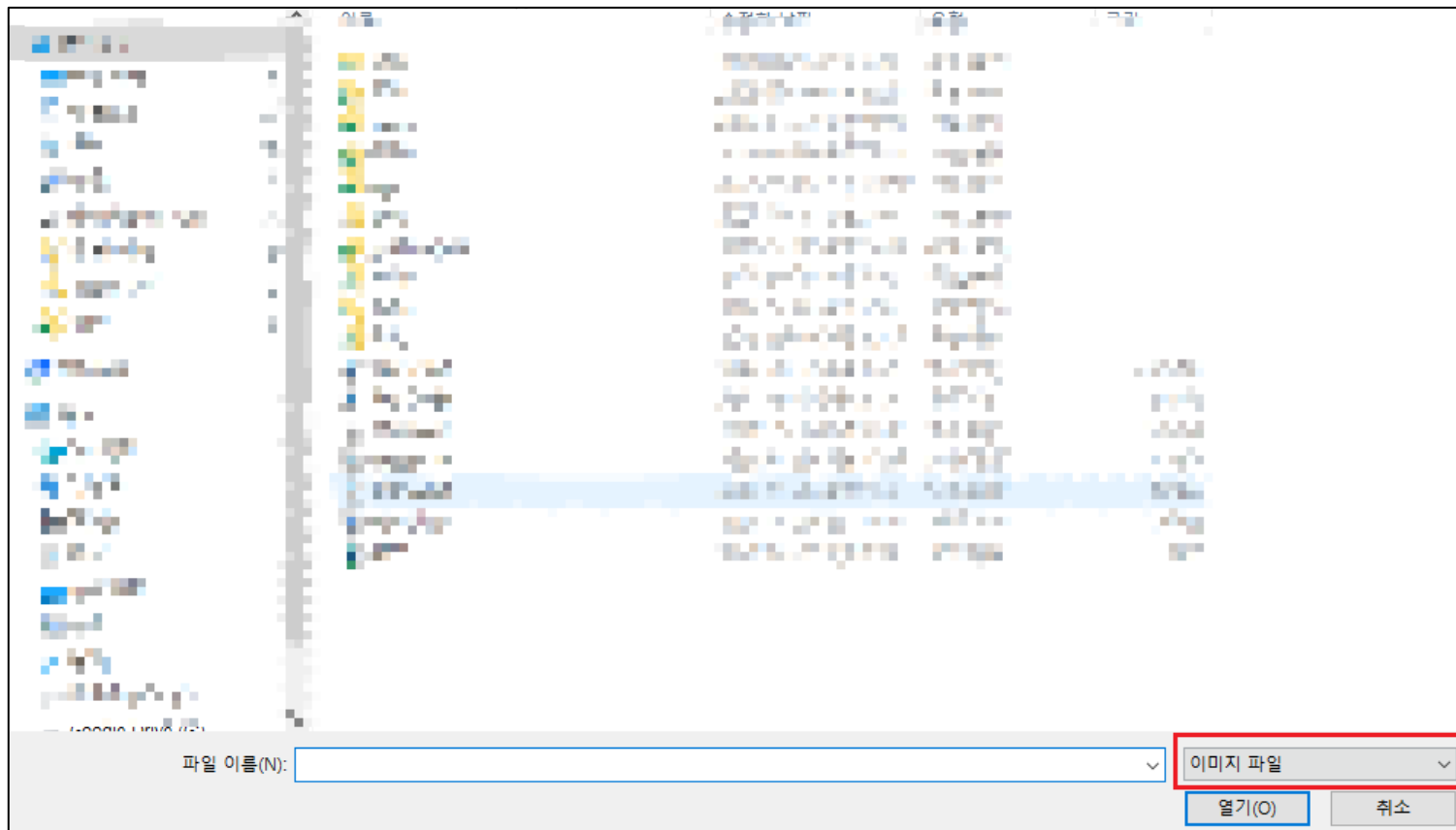
(accept 속성 값을 image라고 하더라도 비디오나 오디오 및 다른 형식 파일을 제출할 수 있음)

- 고유 파일 유형 지정자

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

input 요소 - accept 속성 (2/2)

- 파일 업로드 시 허용할 파일 형식에 대해 자동으로 필터링



views.py 수정

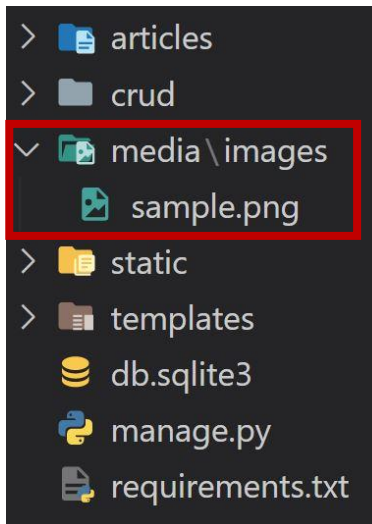
- 업로드 한 파일은 **request.FILES** 객체로 전달됨

```
# views.py

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

DB 및 파일 트리 확인

- 실제 파일 위치
 - MEDIA_ROOT/images/



- DB에 저장되는 것은 이미지 파일 자체가 아닌 **파일의 경로**

SQL ▾					
< 1 / 1 > 1 - 1 of 1					
id	title	content	created_at	updated_at	image
1	제목	내용	2024-01-01 12:00:00	2024-01-01 12:00:00	images/sample.png

이미지 업로드 (READ)

| 이미지 경로 불러오기

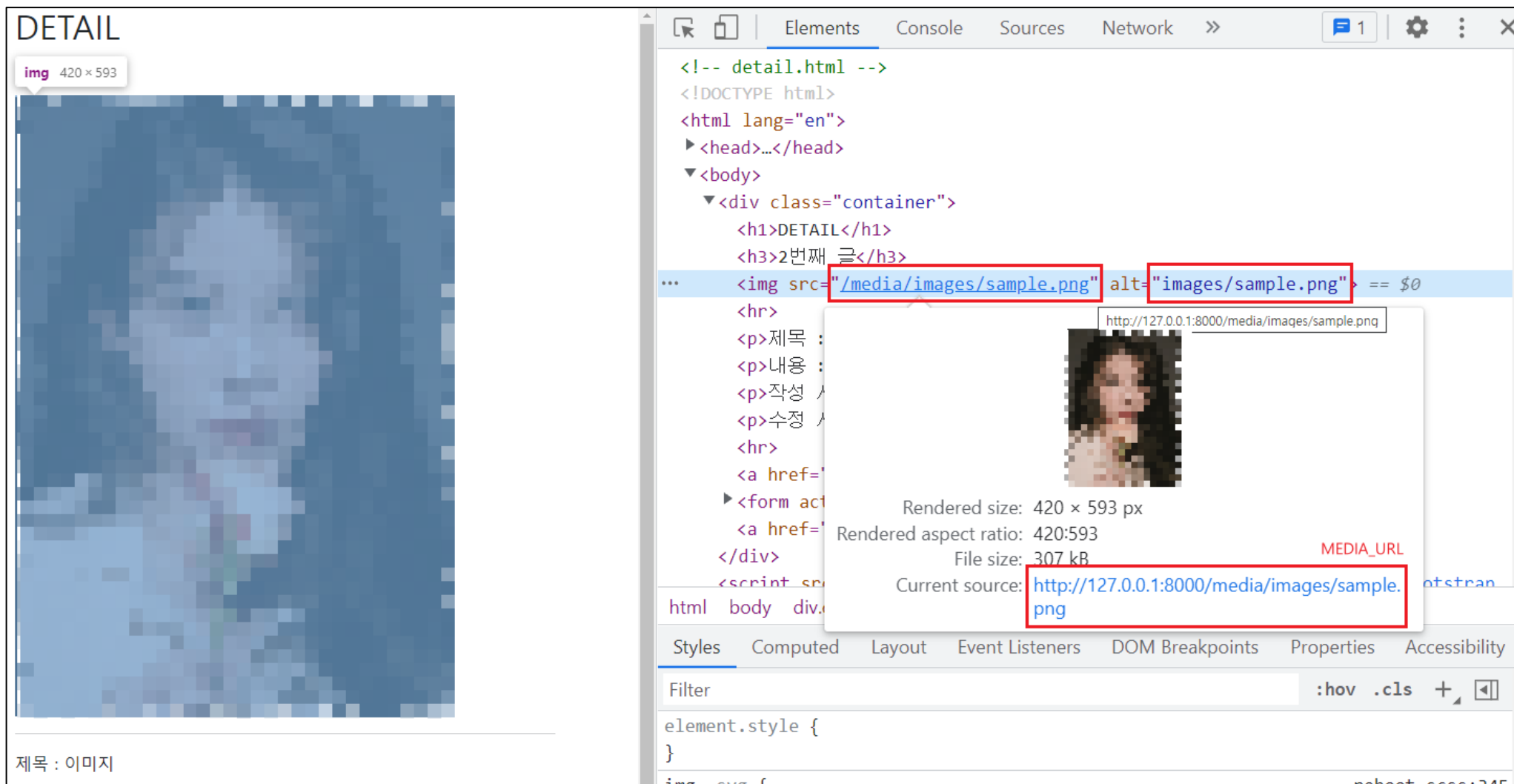
- article.image.url
 - 업로드 파일의 경로
- article.image
 - 업로드 파일의 파일 이름

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

{% extends 'base.html' %}

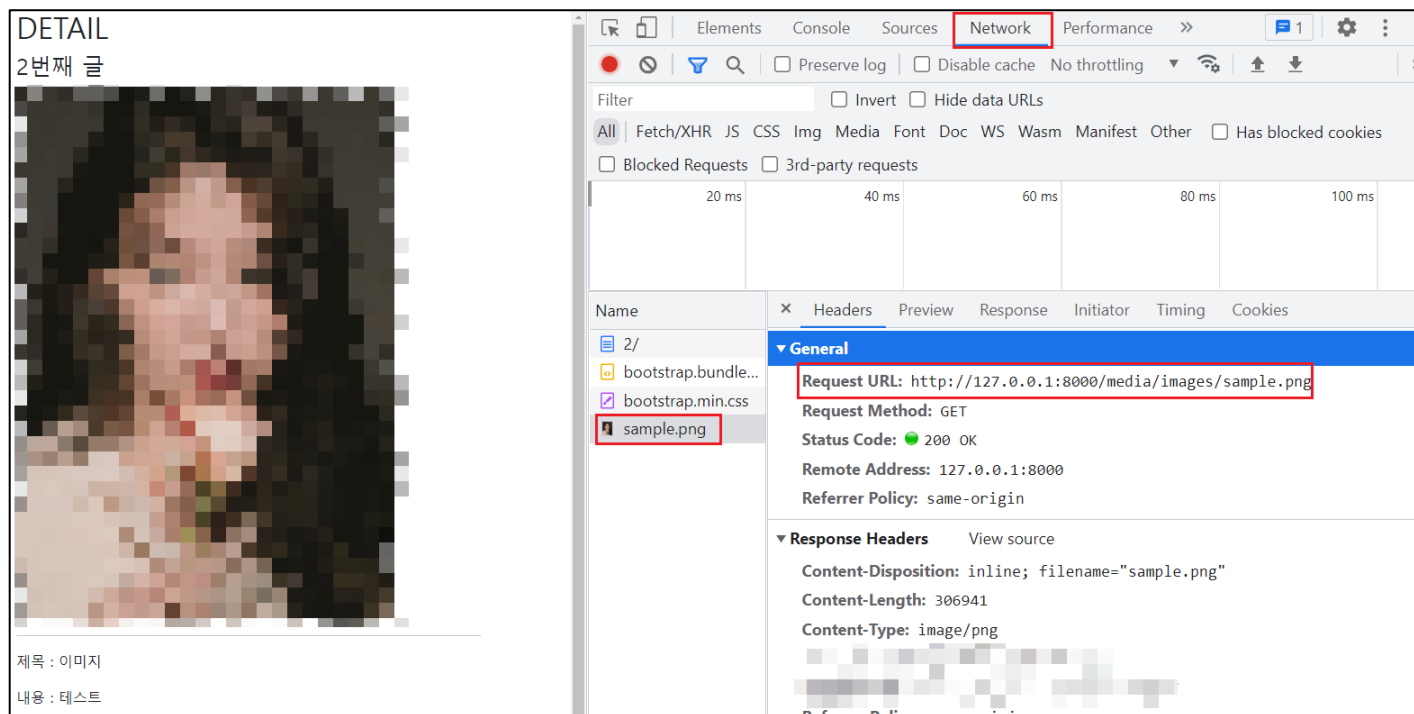
{% block content %}
    <h1>DETAIL</h1>
    <h3>{{ article.pk }}번째 글</h3>
    
    <hr>
```


이미지 출력 및 MEDIA_URL 값 확인하기



STATIC_URL과 MEDIA_URL

- static, media 파일 모두 서버에 요청해서 조회하는 것
 - 서버에 요청하기 위해서는 url이 필요



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

이미지 업로드 (UPDATE)

| 이미지 수정하기 (1/4)

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

이미지 수정하기 (2/4)

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

{% extends 'base.html' %}

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

이미지 수정하기 (3/4)

```
# views.py

def update(request, pk):
    article = Article.objects.get(pk=pk)
    if request.method == 'POST':
        form = ArticleForm(request.POST, request.FILES, instance=article)
        if form.is_valid():
            article = 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)
```

이미지 수정하기 (4/4)

- 이미지가 없이 작성된 게시글은 detail 페이지가 출력되지 않는 문제 해결
- image가 없는 게시글의 경우 출력할 이미지 경로가 없기 때문

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

{% extends 'base.html' %}

{% block content %}
  <h1>DETAIL</h1>
  <h3>{{ article.pk }}번째 글</h3>
  {% if article.image %}
    
  {% endif %}
  <hr>
```

이미지 Resizing

| 이미지 크기 변경하기 (1/2)

- 실제 원본 이미지를 서버에 그대로 업로드 하는 것은 서버의 부담이 큰 작업
- `` 태그에서 직접 사이즈를 조정할 수도 있지만 (width와 height 속성 사용), 업로드 될 때 이미지 자체를 resizing 하는 것을 고려하기
- [django-imagekit](https://github.com/matthewwithanm/django-imagekit/) 라이브러리 활용

| 이미지 크기 변경하기 (2/2)

1. django-imagekit 설치
2. INSTALLED_APPS에 추가

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

```
# settings.py  
  
INSTALLED_APP = [  
    ...  
    'imagekit',  
    ...  
]
```

원본 이미지를 재가공하여 저장 (원본X, 썸네일O) (1/2)

- django-imagekit 라이브러리 문서를 참고하여 작성

```
# models.py
```

```
from imagekit.processors import Thumbnail
from imagekit.models import ProcessedImageField
from django.db import models
```

```
class Article(models.Model):
    title = models.CharField(max_length=10)
    content = models.TextField()
    # image = models.ImageField(upload_to='images/', blank=True)
    image = ProcessedImageField(
        blank=True,
        upload_to='thumbnails/',
        processors=[Thumbnail(200,300)],
        format='JPEG',
        options={'quality': 90},
    )
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

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


ProcessedImageField()의 parameter로 작성된 값들은
변경하더라도 다시 makemigrations를 해줄 필요없이
즉시 반영 됨



원본 이미지를 재가공하여 저장 (원본X, 썸네일O) (2/2)

- 이미지 업로드 후 결과 확인

DETAIL

img 200 x 300



제목 : 리사이징
내용 : 테스트
작성 시각 : 
수정 시각 : 

[수정](#)
[삭제](#)
[back](#)

Elements Console Sources Network Performance

```
<!-- detail.html -->
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <div class="container">
      <h1>DETAIL</h1>
      <h3>3번째 글</h3>
      ...
      
      <hr>
      <p>제목 :
      <p>내용 :
      <p>작성 /
      <p>수정 /
      <hr>
      <a href=
      <form act
      <a href=
      </div>
      <script s
      html body div
```

Rendered size: 200 x 300 px
Rendered aspect ratio: 2:3
File size: 14.4 kB
Current source: http://127.0.0.1:8000/media/thumbnails/sample.jpg

원본 이미지를 재가공하여 저장 (원본O, 썸네일O) (1/3)

- django-imagekit 라이브러리 문서를 참고하여 작성

```
# models.py

from imagekit.processors import Thumbnail
from imagekit.models import ProcessedImageField, ImageSpecField
from django.db import models

class Article(models.Model):
    title = models.CharField(max_length=10)
    content = models.TextField()
    image = models.ImageField(upload_to='images/', blank=True)
    image_thumbnail = ImageSpecField(
        source='image', # 원본 ImageField 명
        processors=[Thumbnail(200,300)],
        format='JPEG',
        options={'quality': 90},
    )
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

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

| 원본 이미지를 재가공하여 저장 (원본O, 썸네일O) (2/3)

- 추가된 ImageSpecField() 필드 사용

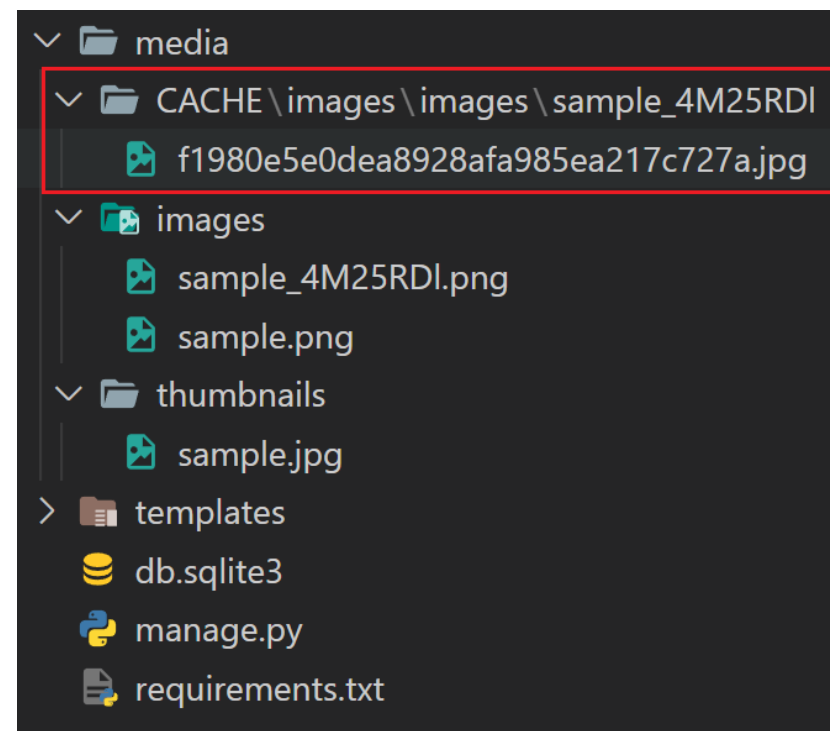
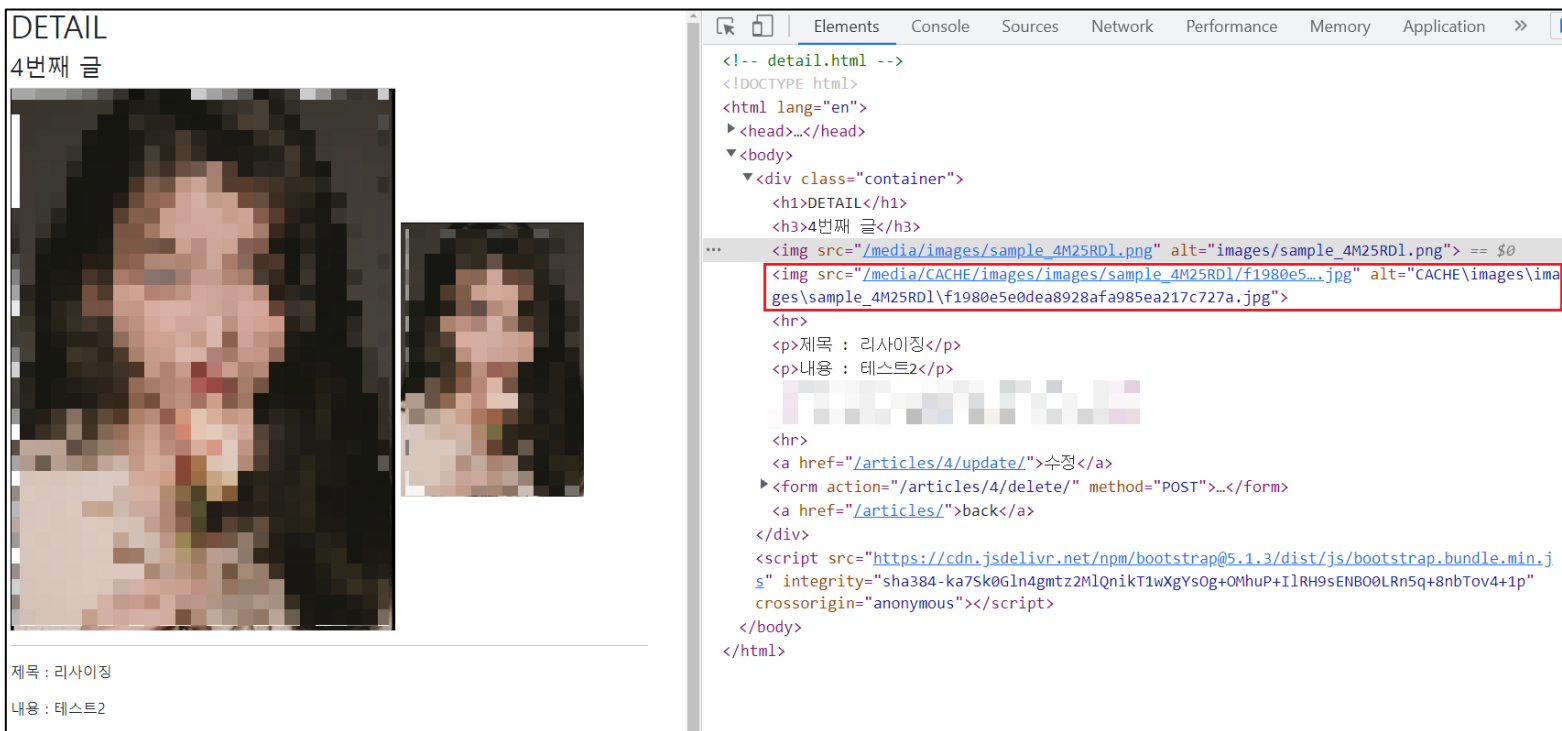
```
<!-- detail.html -->

{% extends 'base.html' %}

{% block content %}
    <h1>DETAIL</h1>
    <h3>{{ article.pk }}번째 글</h3>
    {% if article.image %}
        
        
    {% endif %}
    <hr>
```

원본 이미지를 재가공하여 저장 (원본O, 썸네일O) (3/3)

- 이미지 업로드 후 결과 확인



*ProcessedImageField()와 달리
업로드 후 이미지를 출력해야 썸네일용 이미지가 생성됨

마무리

마무리

- Handling HTTP requests
- Media files
 - Image Upload
 - Image Resizing