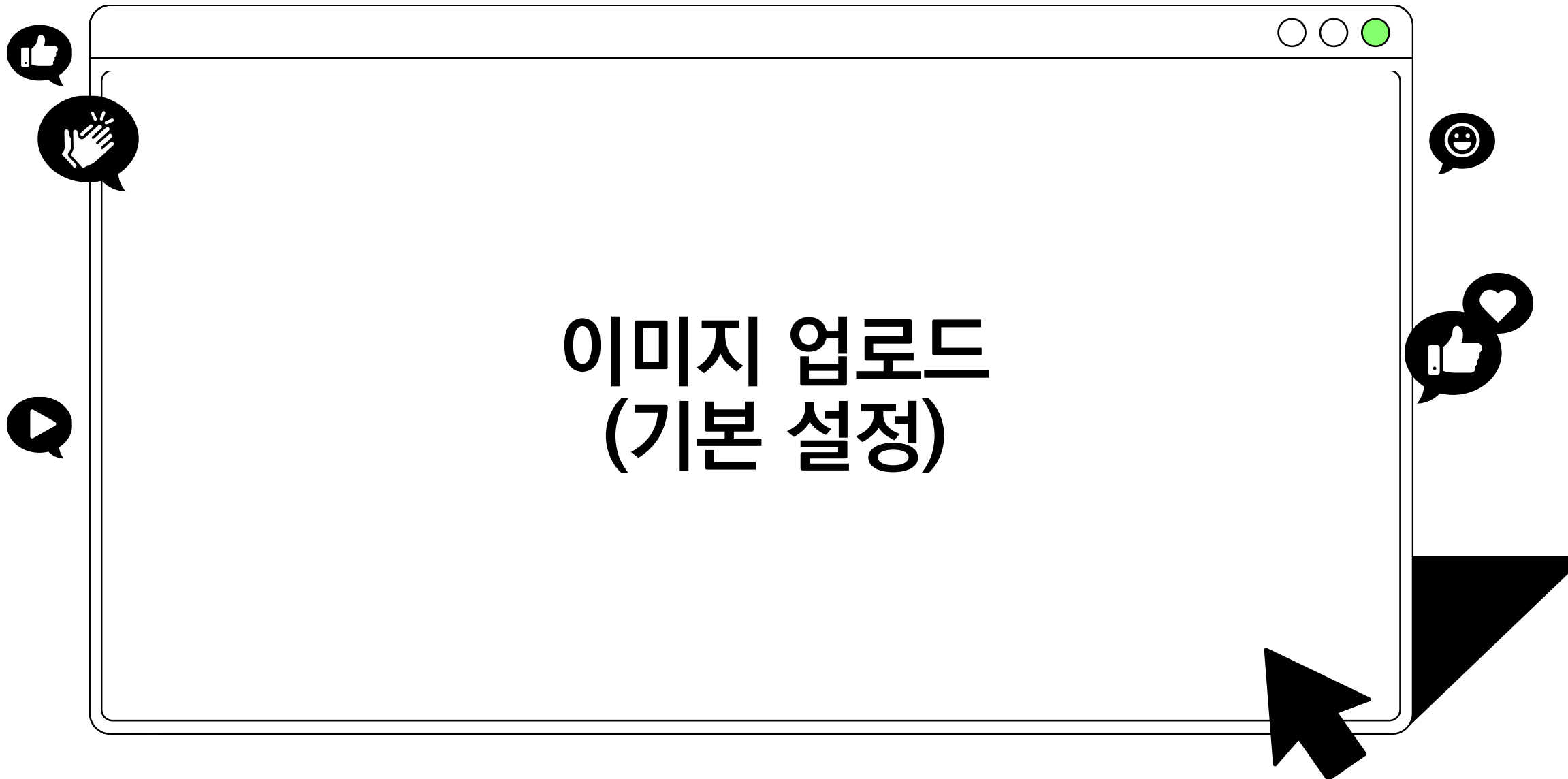


# 이미지 업로드 (기본 설정)



## 미디어 파일

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

## Media 관련 필드

- 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](#) 라이브러리가 필요

## Media 관련 필드

- FileField
  - 파일 업로드에 사용하는 모델 필드
  - 2개의 선택 인자를 가지고 있음
    - upload\_to
    - storage

## 모델 설정

- upload\_to argument
  - 문자열 경로 지정 방식

```
# 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/')
```

- 함수 호출

```
# models.py

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

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

## URL 설정

- settings.py에 MEDIA\_ROOT, MEDIA\_URL 설정
- upload\_to 속성을 정의하여 업로드 된 파일에 사용 할 MEDIA\_ROOT의 하위 경로를 지정
- 업로드 된 파일의 경로는 django가 제공하는 'url' 속성을 통해 얻을 수 있음

```

```

## URL 설정

- 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

```
# settings.py

MEDIA_ROOT = BASE_DIR / 'media'
```

## URL 설정

- 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

```
# settings.py  
  
MEDIA_URL = '/media/'
```



## URL 설정

- 개발 단계에서 사용자가 업로드 한 파일 제공하기

```
# 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 업로드 한 파일이 우리 프로젝트에 업로드 되지만,  
실제로 사용자에게 제공하기 위해서는 업로드 된 파일의 URL이 필요함



이미지 업로드  
(CREATE)

## 모델 설정

- ImageField
  - 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=20)
    content = models.TextField()
    # saved to 'MEDIA_ROOT/images'
    image = models.ImageField(blank=True, upload_to='images/')
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

## 모델 설정

- 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)
    - 필드에 blank=True가 있으면 form 유효성 검사에서 빈 값을 입력할 수 있음

## 모델 설정

- Model field option – “null”
  - 기본 값 : False
  - True면 django는 빈 값을 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 중복되며, Django는 NULL이 아닌 빈 문자열을 사용하는 것이 규칙

## 모델 설정

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

## 모델 설정

- 마이그레이션 실행

(단, ImageField를 사용하기 위해서는 Pillow 라이브러리 설치 필요)

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

```
$ pip install Pillow
```

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

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

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

## HTML 설정

- 게시글 작성 form enctype 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">  
</form>
```

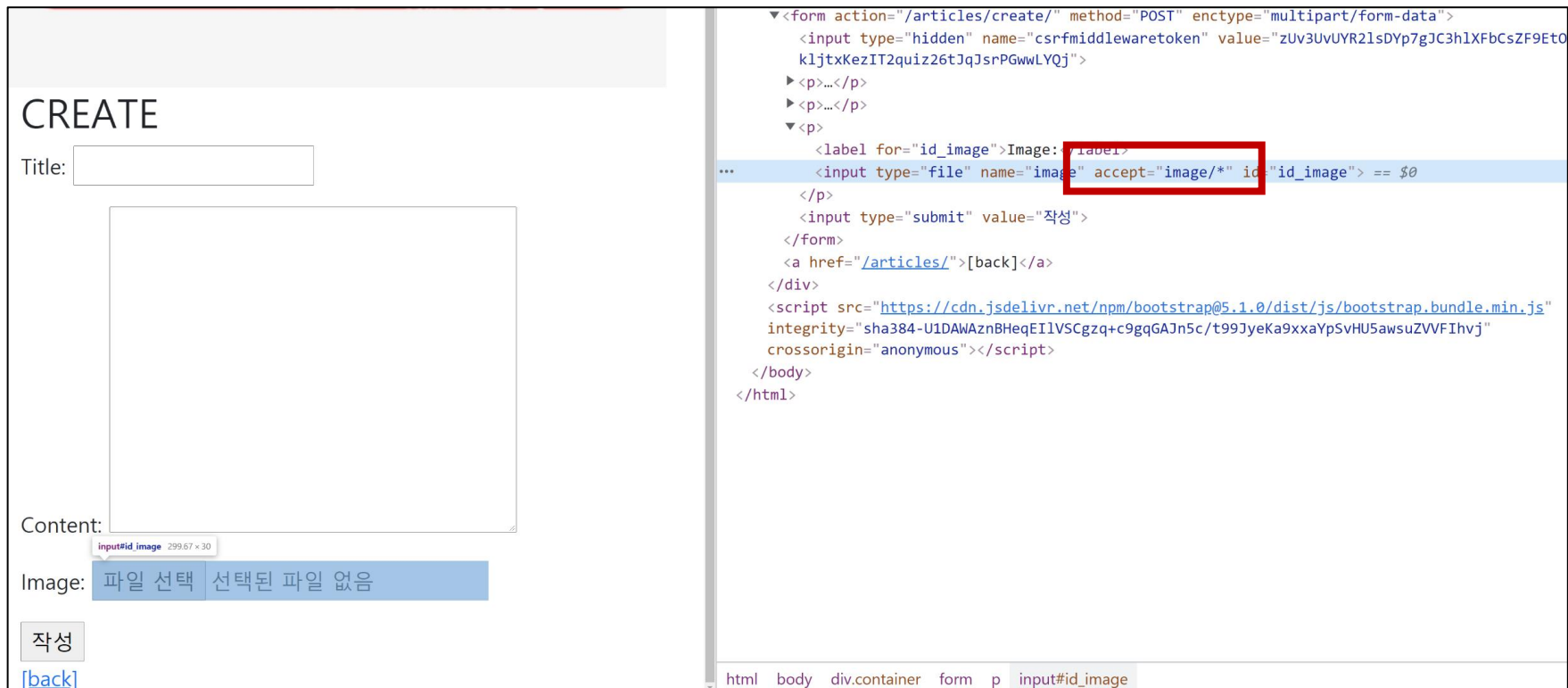


## HTML 설정

- form 요소 - enctype(인코딩) 속성
  - multipart/form-data
    - 파일/이미지 업로드 시에 반드시 사용해야 함 (전송되는 데이터의 형식을 지정)
    - <input type="file">을 사용할 경우에 사용
  - application/x-www-form-urlencoded
    - (기본값) 모든 문자 인코딩
  - text/plain
    - 인코딩을 하지 않은 문자 상태로 전송
    - 공백은 '+' 기호로 변환하지만, 특수 문자는 인코딩 하지 않음

## HTML 설정

- input 요소의 accept 속성 확인



The screenshot displays a web application interface on the left and its corresponding HTML source code on the right. The interface, titled "CREATE", includes a "Title:" text input, a large "Content:" text area, and an "Image:" section with a file selection button labeled "파일 선택" and a status "선택된 파일 없음". A "작성" button is at the bottom, along with a "[back]" link.

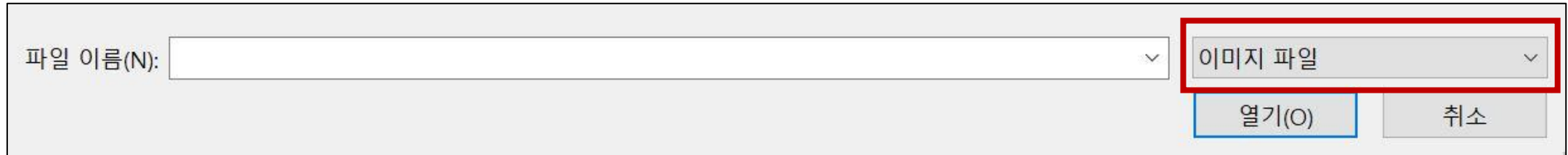
The HTML source code on the right shows a form with the following structure:

```
<form action="/articles/create/" method="POST" enctype="multipart/form-data">
  <input type="hidden" name="csrfmiddlewaretoken" value="zUv3UvUYR2lsDYp7gJC3h1XFbCsZF9Et0kljtxKezIT2quiz26tJqJsrPGwWLYQj">
  <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" value="작성">
</form>
<a href="/articles/">[back]</a>
</div>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/js/bootstrap.bundle.min.js" integrity="sha384-U1DAWAznBHeqElIVScgzq+c9gqGAJn5c/t99JyeKa9xxaYpSvHU5awsuZVFihvj" crossorigin="anonymous"></script>
</body>
</html>
```

In the code, the `accept="image/*"` attribute of the `<input type="file">` is highlighted with a red box. The browser's developer tools at the bottom show the element path: `html > body > div.container > form > p > input#id_image`.

## HTML 설정

- input 요소 - accept 속성



A screenshot of a web form for file upload. It features a text input field labeled '파일 이름(N):' followed by a dropdown arrow. To the right of this is another dropdown menu with the text '이미지 파일' and a dropdown arrow. This second dropdown is highlighted with a red rectangular border. Below the dropdowns are two buttons: '열기(O)' (Open) and '취소' (Cancel).

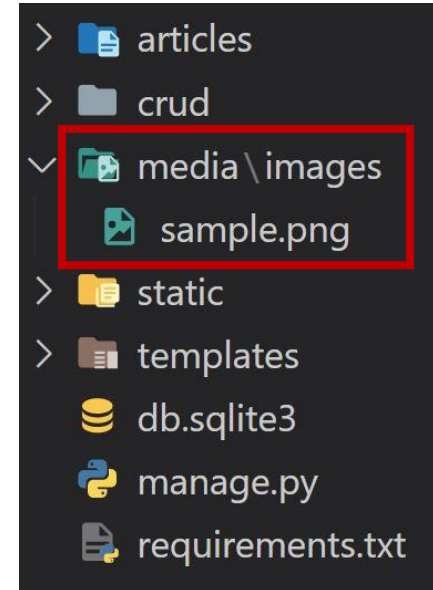
## View 설정

- 업로드 한 파일은 request.FILES 객체로 전달됨

```
# 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)
```

- DB 및 파일 트리 확인
- 실제 파일 위치
  - MEDIA\_ROOT/images/



SQL ▼					
< 1 / 1 > 1 - 2 of 2					
id	title	content	created_at	updated_at	image
1	제목제목	내용내용	2021-03-18 09:53:06.995661	2021-03-18 09:53:06.995661	images/sample_img.jpg
2	하하	호호	2021-03-18 09:53:47.281045	2021-03-18 09:53:47.281045	



이미지 업로드  
(READ)


## img 태그 활용

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

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

{% extends 'base.html' %}

{% block content %}
    <h2 class='text-center'>DETAIL</h2>
    <h3>{{ article.pk }} 번 글</h3>
    
    <hr>
    ...
{% endblock %}
```




DETAIL

6 번째 글

Elements Console Sources Network Performance Memory

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    
    <div class="container">
      
      <h2>DETAIL</h2>
      <h3>6 번째 글</h3>
      ...
       == $0
    <hr>
    <p>제목 :
    <p>내용 :
    <p>작성시각
    <p>수정시각
    <hr>
    <a href=
    > <form act
    <a href=
    </div>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/js/boots
```



Rendered size: 367 × 235 px  
Rendered aspect ratio: 367:235  
File size: 31.8 kB  
Current source: <http://127.0.0.1:8000/media/images/sample.png>



이미지 업로드  
(UPDATE)

## 이미지 수정하기

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

```
<!-- 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 %}
```

## 이미지 수정하기

```
# 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)
```

# 이미지 Resizing



## Django-imagekit

- 실제 원본 이미지를 서버에 그대로 업로드 하는 것은 서버의 부담이 큰 작업
- <img> 태그에서 직접 사이즈를 조정할 수도 있지만 (width 와 height), 업로드 될 때 이미지 자체를 resizing 하는 것을 사용해 볼 것
- [django-imagekit](#) 라이브러리 활용

## Django-imagekit

- 1. django-imagekit 설치
- 2. INSTALLED\_APPS에 추가

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

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

## Django-imagekit

- 이미지 크기 변경하기 (3/3)

```
# models.py

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

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

    def __str__(self):
        return self.title
```

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

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