

Django CRUD - UD

# 7번째 세션

---

NEXT X LIKELION 김범진

# 오늘 할 일

update & delete



CREATE



READ



UPDATE



DELETE

---

C

R

U

D

# 오늘의 실습

독후감 블로그

## home.html

작성한 모든 독후감을 보여줍니다.

### 독후감 블로그

- [오타와 함께 사라지다](#)
- [누가 내 코드에 똥 썼어?](#)
- [다빈치 코딩](#)
- [장고에서 살아남기](#)

[글 쓰러가기](#)

## new.html

새로운 독후감을 작성합니다.

제목

제목을 입력해주세요

내용을 입력해주세요

내용

작성하기

# 오늘의 실습

독후감 블로그

## detail.html

작성한 특정 독후감과 관련된 상세 내용을 보여줍니다.

### 책 제목

누가 내 코드에 똥 썼어?

### 책 내용

범인은 이 git 안에 있다  
[홈으로](#) [수정하기](#) [삭제하기](#)

## edit.html

작성한 특정 독후감을 수정합니다.

제목

범인은 이 git 안에 있다

내용



(보안이 우수하고 유지보수가 편리한) 웹사이트를 신속하게 개발하도록 도움을 주는 파이썬 기반 웹 프레임워크

- 로그인/로그아웃
- 데이터베이스
- 보안

=> 이미 구현되어 있는 기능이 많아 잘 사용하면 된다!

# Django 복습

코드 입력

```
mkdir session7
cd session7
pipenv shell
pipenv install django
django-admin startproject {project}
cd {project}
python manage.py startapp {app}
```

작업할 폴더 생성 & 이동

가상환경 생성

가상환경 내에 Django 패키지 설치

장고 프로젝트 생성 & 이동

프로젝트 폴더 내에 장고 앱 생성

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app',
]
```

```
TIME_ZONE = 'Asia/Seoul'
```

```
python manage.py runserver
```

로컬 서버에서 장고 서버 실행

# Django 복습

코드 입력

django

View [release notes](#) for Django 3.2



The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.



**Django Documentation**  
Topics, references, & how-to's



**Tutorial: A Polling App**  
Get started with Django

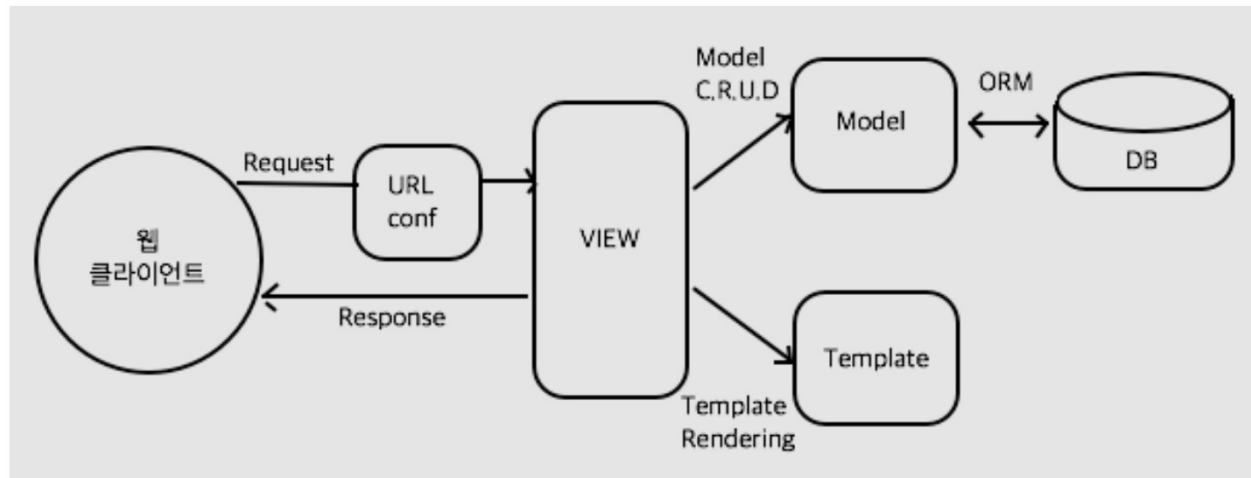


**Django Community**  
Connect, get help, or contribute

# MTV 패턴

장고의 설계 방식

## 장고의 동작 과정: MTV 패턴



**MTV 패턴이란?**

**모델을 만들고 => 템플릿을 만들고 => 뷰를 만드는 Django의 설계 패턴**



# MTV 패턴

장고의 설계 방식

## Model – 데이터베이스 설계

- 데이터베이스에 저장되는 데이터의 영역

## Template – 화면 UI 설계

- 사용자에게 보여지는 영역, 화면
- HTML

## View – 프로그램 로직 설계

- 요청에 따라 Model에서 필요한 데이터를 가져와 처리
- 처리 결과를 Template에 전달

```
# project/app/models.py  
  
from django.db import models  
  
# Create your models here.  
  
class Post(models.Model):  
    title = models.CharField(max_length=200)  
    content = models.TextField()  
  
    def __str__(self):  
        return self.title
```

python manage.py makemigrations

migrations만들기

python manage.py migrate

DB에 반영

python manage.py createsuperuser

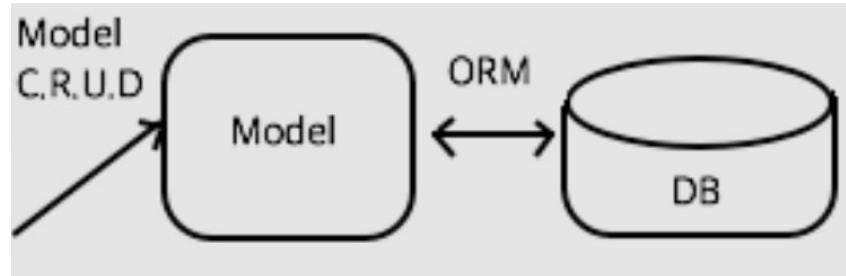
관리자 계정 생성

python manage.py runserver

로컬 서버에서 장고 서버 실행

# Django의 DB처리 방식

ORM



Django의 DB처리는 **ORM(Object Relational Mapping)**기법을 사용한다.  
객체(Object)의 관계(Relational)를 연결(Mapping)해준다.

Model이란, 테이블을 정의하는 장고의 클래스를 의미하며, models.py 파일에 테이블 관련 사항들을 정의한다.

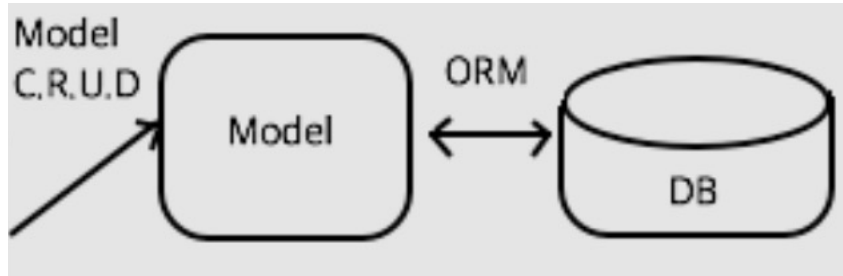
ORM 방식을 기반으로 테이블을 클래스로 매핑해서  
테이블에 대한 CRUD 기능을 클래스 객체에 대해 수행하면,  
장고가 내부적으로 SQL처리하여 DB에 반영해준다.

테이블 == 클래스 (django.db.models.Model클래스 상속)

테이블의 컬럼 == 클래스의 속성 (models의 필드 사용)

# Django의 DB처리 방식

migration



models.py에서 DB 변경사항이 생기면 반영해야된다.

=> 테이블 및 필드를 생성, 삭제, 변경 등 DB에 대한 변경 사항을 알려주는 정보인 **마이그레이션**을 도입

`python manage.py makemigrations (app)` (app)은 생략 가능.  
애플리케이션 디렉터리 하위에 migrations/을 만든다.

`python manage.py migrate (app)`  
migrations/정보를 추출하여 DB에 반영한다. (= 테이블 생성)

`python manage.py showmigrations (app)`  
마이그레이션별 적용 여부를 알 수 있다.

**결론!** `models.py`에 변경사항이 생길 때 마다,  
`python manage.py makemigrations`와 `python manage.py migrate`를 해주시면 됩니다.

models.py 작성 후,  
admin페이지에서 볼 수 있도록 admin.py에 등록해줍니다.

```
# project/app/admin.py
```

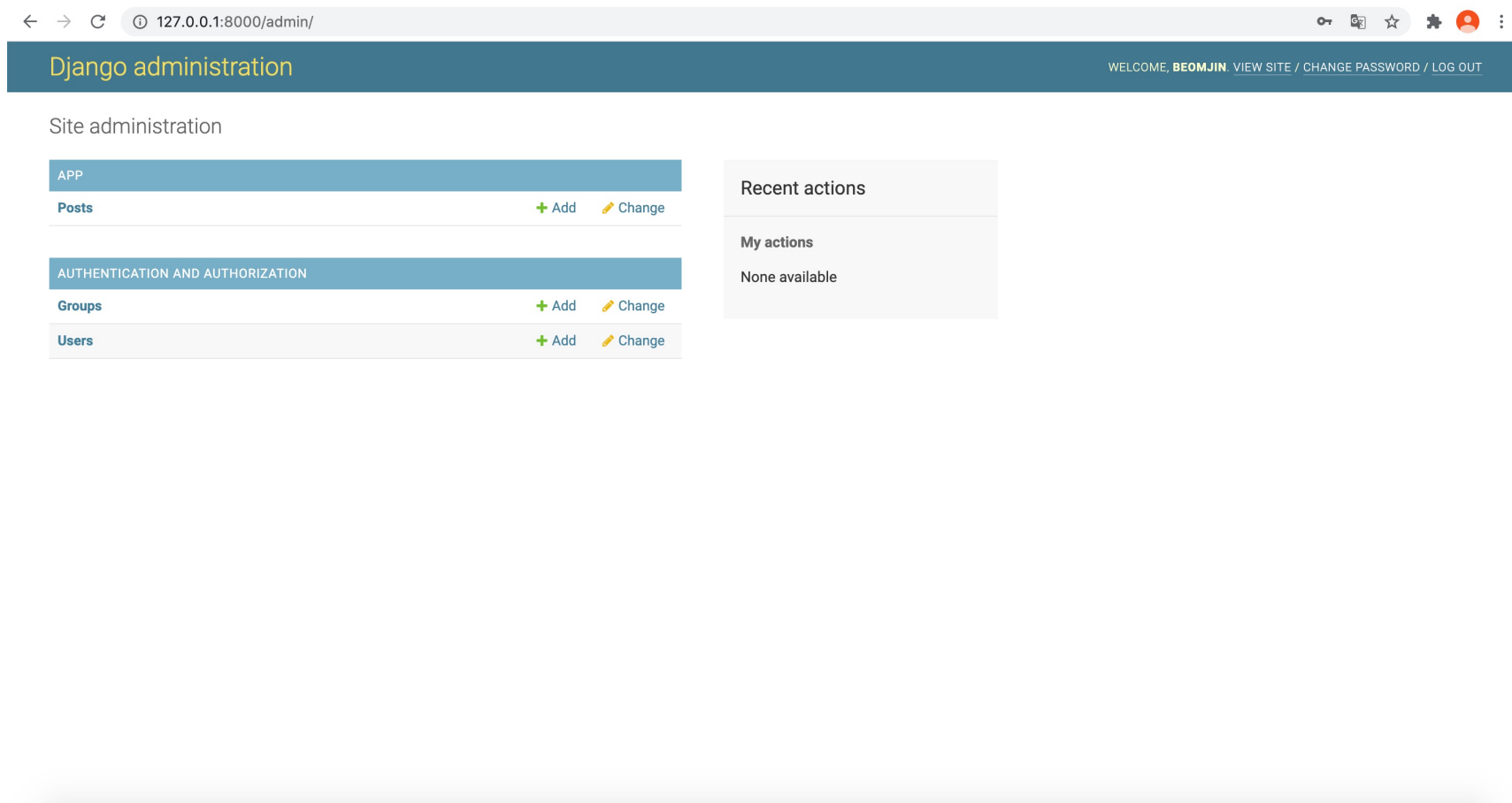
```
from django.contrib import admin
from .models import Post

# Register your models here.
admin.site.register(Post)
```

장고의 admin페이지는 많은 기능을 제공합니다!  
(간단한 데이터베이스 CRUD, admin.ModelAdmin상속을 통한 여러 커스터마이징 등)

**admin 페이지에서 직접 Post를 추가해보세요!**

# admin 페이지



# admin 페이지

The screenshot displays the Django administration interface in a web browser. The address bar shows the URL `127.0.0.1:8000/admin/app/post/add/`. The page header includes the Django administration logo and a welcome message for user **BEOMJIN**, with links for [VIEW SITE](#), [CHANGE PASSWORD](#), and [LOG OUT](#). The breadcrumb trail indicates the path: `Home > App > Posts > Add post`.

The left sidebar contains a menu with the following items:

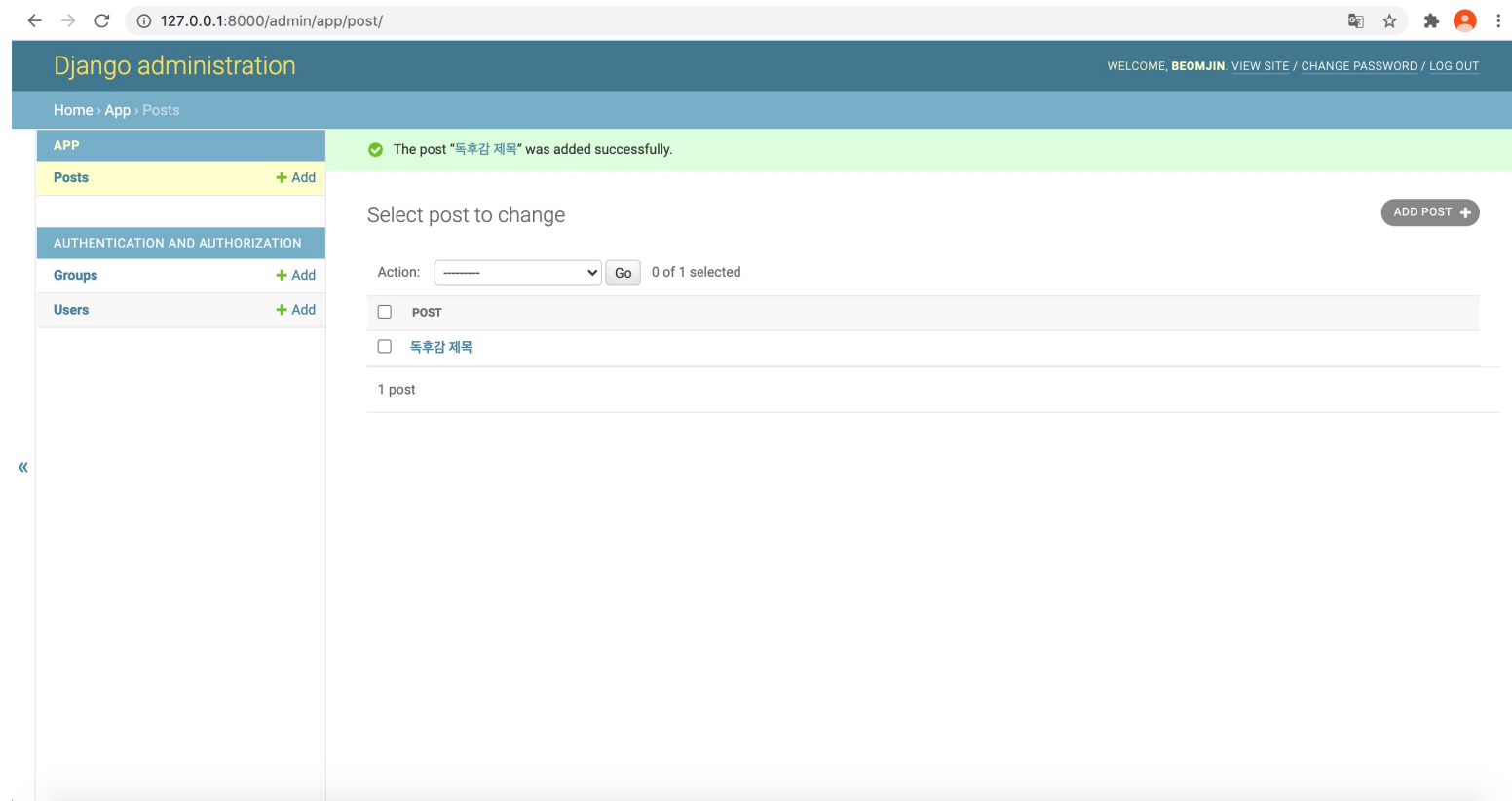
- APP**
  - Posts** (highlighted in yellow) with a [+ Add](#) link.
- AUTHENTICATION AND AUTHORIZATION**
  - Groups** with a [+ Add](#) link.
  - Users** with a [+ Add](#) link.

The main content area is titled "Add post" and contains the following form fields:

- Title:** A text input field containing the text "독후감 제목".
- Content:** A large text area containing the text "독후감 내용".

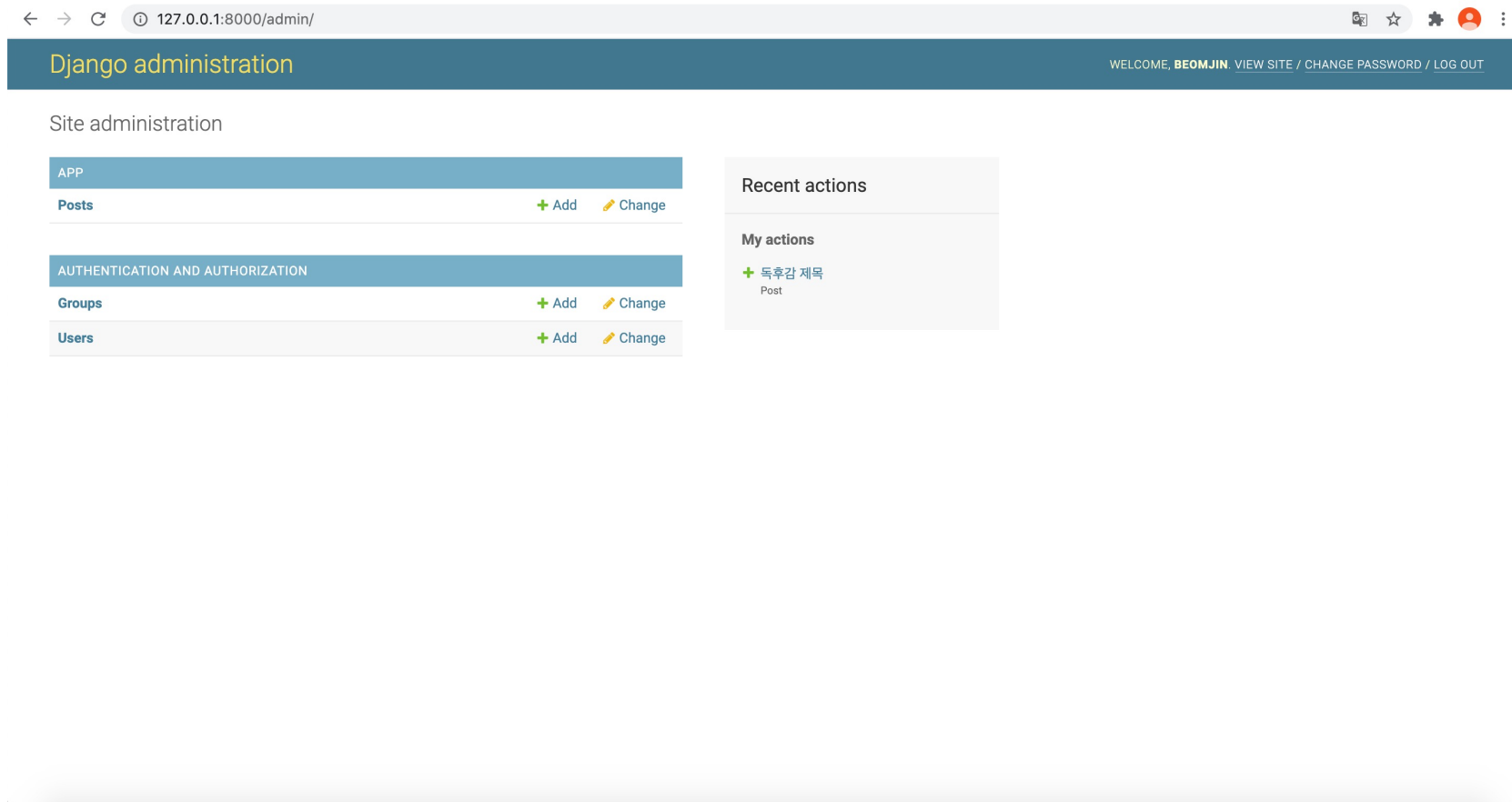
At the bottom right of the form, there are three buttons: "Save and add another", "Save and continue editing", and "SAVE".

# admin 페이지





# admin 페이지



T로 들어가기 전에, urls.py에서 templates과 views를 매핑해줍니다.

```
# project/project/urls.py
```

```
from django.contrib import admin
from django.urls import path
from app import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name="home")
]
```

app/ 에 templates 폴더를 만들고, home.html을 작성해줍니다.

```
# project/app/templates/home.html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>독후감 블로그</h1>
  {{ posts }}
</body>
</html>
```

views.py에 home함수를 추가해줍니다.

```
# project/app/views.py
```

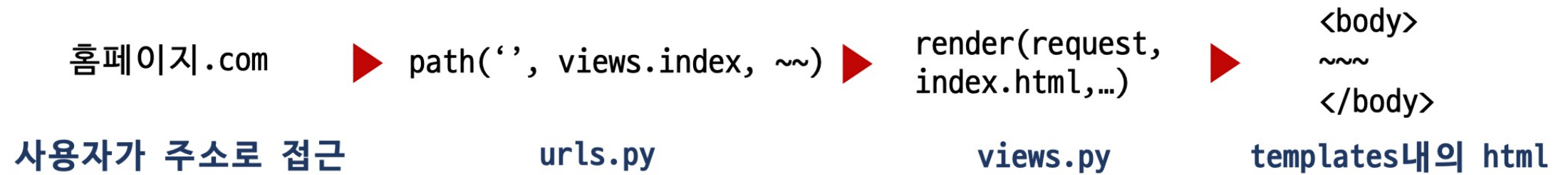
```
from django.shortcuts import render
from .models import Post

# Create your views here.
def home(request):
    posts = Post.objects.all()

    return render(request, 'home.html', { 'posts' : posts })
```

# Django 작동 원리

home



# Django 작동 원리

home

사용자가 'localhost:8000'으로 접속

```
# project/project/urls.py
```

```
from django.contrib import admin
from django.urls import path
from app import views
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name="home")
]
```

url이 ""인 요청을 인식하면,

views.home을 실행한다.

url 패턴의 이름은 "home"이다.  
(html에서 url template tag로 사용)

# Django 작동 원리

home

```
# project/app/views.py
```

```
from django.shortcuts import render  
from .models import Post
```

```
# Create your views here.
```

```
def home(request):
```

```
    posts = Post.objects.all()
```

```
    return render(request, 'home.html', { 'posts' : posts })
```

Post 모델을 전부 가져와서,

posts라는 변수에 저장한다.

render 함수를 통해, 위에서 저장한 posts 변수를, posts라는 이름으로, home.html에 전달하고 해당 html을 불러온다.

posts는 파이썬 변수.

하지만, html은 파이썬 변수를 인식할 수 없다.

## Django Template Language

파이썬 변수 및 문법을 HTML에서 사용할 수 있도록  
Django에서 제공하는 언어



# Django 작동 원리

## Django Template Language

<https://docs.djangoproject.com/en/3.0/ref/templates/builtins/#for>

<https://docs.djangoproject.com/en/3.1/ref/templates/language/>

Contents 구글에 'django template syntax' 검색

- [The Django template language](#)
  - [Templates](#)
  - [Variables](#)
  - [Filters](#)
  - [Tags](#)
  - [Comments](#)
  - [Template inheritance](#)
  - [Automatic HTML escaping](#)
    - [How to turn it off](#)
      - [For individual variables](#)
      - [For template blocks](#)
    - [Notes](#)
    - [String literals and automatic escaping](#)
  - [Accessing method calls](#)
  - [Custom tag and filter libraries](#)
    - [Custom libraries and template inheritance](#)

Contents 구글에 'django template tags' 검색

- [Built-in template tags and filters](#)
  - [Built-in tag reference](#)
    - [autoescape](#)
    - [block](#)
    - [comment](#)
    - [csrf\\_token](#)
    - [cycle](#)
    - [debug](#)
    - [extends](#)
    - [filter](#)
    - [firstof](#)
    - [for](#)
    - [for ... empty](#)
    - [if](#)
      - [Boolean operators](#)
        - [== operator](#)
        - [!= operator](#)
        - [< operator](#)
        - [> operator](#)
        - [<= operator](#)

# Django 작동 원리

Django Template Language

자주 사용하는 것만 기억해두고,  
필요할 때마다 그때그때 검색해서 사용하면 됩니다.

<code>{% if (조건문) %}</code>	<code>{% for~ in ~ %}</code>	<code>{% url 'index' %}</code>	<code>{{객체.어트리뷰트}}</code>
~~~~	~~~~		
<code>{% endif %}</code>	<code>{% endfor %}</code>		

기본 문법은 {% %}로 감싸주고, 모델 객체(글)/변수와 관련된 것은 {{}}로 감싸준다!

# Django 작동 원리

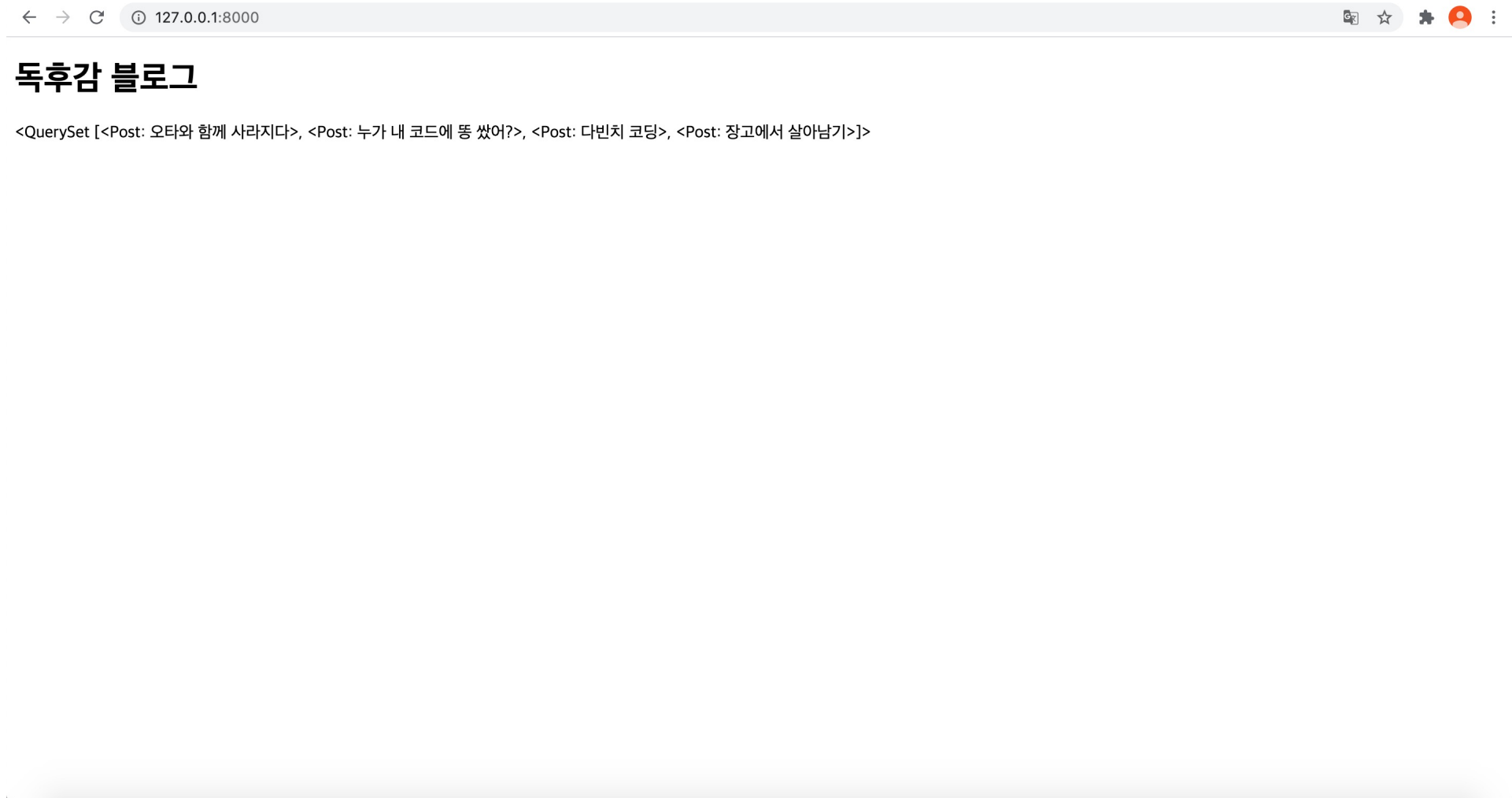
## QuerySet

```
# project/app/templates/home.html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>독후감 블로그</h1>
  {{ posts }}
</body>
</html>
```

# Django 작동 원리

## QuerySet



# Django 작동 원리

## QuerySet

### QuerySet

데이터베이스에서 전달받은 모델의 객체 목록(list) = 모델들의 집합

```
# project/app/views.py
```

```
from django.shortcuts import render
from .models import Post

# Create your views here.
def home(request):
    posts = Post.objects.all()

    return render(request, 'home.html', { 'posts' : posts })
```

이때 반환되는 것이 QuerySet

# Django 작동 원리

## QuerySet

### QuerySet Method

데이터 베이스에서 가져온 모델들을 사용자 입맛에 맞게 바꿔주는 일종의 함수  
Ex) id 순으로 정렬, 특정 필드값이 큰 순으로 정렬, 작성일자 순으로 정렬 등

<https://docs.djangoproject.com/en/3.0/ref/models/queries/>

#### ◦ QuerySet API

##### ▪ Methods that return new QuerySets

- [filter\(\)](#)
- [exclude\(\)](#)
- [annotate\(\)](#)
- [order\\_by\(\)](#)
- [reverse\(\)](#)
- [distinct\(\)](#)
- [values\(\)](#)
- [values\\_list\(\)](#)
- [dates\(\)](#)
- [datetimes\(\)](#)
- [none\(\)](#)
- [all\(\)](#)
- [union\(\)](#)
- [intersection\(\)](#)
- [difference\(\)](#)
- [select\\_related\(\)](#)

##### ▪ Operators that return new QuerySets

- [AND\(\)](#)
- [OR\(\)](#)

##### ▪ Methods that do not return QuerySets

- [get\(\)](#)
- [create\(\)](#)
- [get\\_or\\_create\(\)](#)
- [update\\_or\\_create\(\)](#)
- [bulk\\_create\(\)](#)
- [bulk\\_update\(\)](#)
- [count\(\)](#)
- [in\\_bulk\(\)](#)
- [iterator\(\)](#)
  - [With server-side cursors](#)
  - [Without server-side cursors](#)
- [latest\(\)](#)
- [earliest\(\)](#)
- [first\(\)](#)

- [explain\(\)](#)

##### ▪ Field lookups

- [exact](#)
- [iexact](#)
- [contains](#)
- [icontains](#)
- [in](#)
- [gt](#)
- [gte](#)
- [lt](#)
- [lte](#)
- [startswith](#)
- [istartswith](#)
- [endswith](#)
- [iendswith](#)
- [range](#)
- [date](#)

# Django 작동 원리

## QuerySet

**외우실 필요 전혀 없습니다!**  
**필요할 때마다 그때그때 검색해서 사용하면 됩니다.**

“Django에서 model을 template에 보여줄 때, QuerySet을 사용하는구나” 정도만 아셔도 충분합니다!

교안에서 기본적인 CRUD를 구현할 때는, `get()`, `filter()`, `create()`, `update()`, `delete()` 만 사용하니 너무 부담가지시지 않으셔도 됩니다.

# CRUD

create

## 독후감 블로그


- 오타와 함께 사라지다
- 누가 내 코드에 똥 썼어?
- 다빈치 코딩
- 장고에서 살아남기

[글 쓰러가기](#)

제목

내용을 입력해주세요

내용



저희가 원하는 것은, home에서 글 쓰러가기를 누르면, 글을 작성할 수 있는 페이지(new)로 이동하고, 해당 페이지에서 새로운 글을 작성할 수 있는 기능입니다.

이것 역시 **MTV** 패턴에 따라

**Model**을 만들고 => **Template**를 만들고 => **View**를 만들어줍니다.

Model은 이미 만들었으니, Template부터 만들어줍니다.



# CRUD

create

MTV

project/project/urls.py

```
from django.contrib import admin
from django.urls import path
from app import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name="home"),
    path('new/', views.new, name="new"),
]
```

# CRUD

create

project/app/templates/new.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <form action="" method="post">
    {% csrf_token %}
    <div>
      <label for="title">제목</label>
      <input type="text" id="title" name="title" placeholder="제목을 입력해주세요.">
    </div>
    <div>
      <label for="content">내용</label>
      <textarea name="content" id="content" cols="30" rows="10" placeholder="내용을 입력해주세요."></textarea>
    </div>
    <button type="submit">작성하기</button>
  </form>
</body>
</html>
```

# CRUD

create

project/app/templates/home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>독후감 블로그</h1>
  <div>
    <ul>
      {% for post in posts %}
      <li>{{ post.title }}</li>
      {% endfor %}
    </ul>
  </div>
  <a href="{% url 'new' %}">글 쓰러가기</a>
</body>
</html>
```

project/app/views.py

```
def new(request):  
    if request.method == 'POST':  
        Post.objects.create(  
            title = request.POST['title'],  
            content = request.POST['content']  
        )  
  
    return render(request, 'new.html')
```

home에서 “글 쓰기가기” 를 클릭하셔서

localhost:8000/new 또는 128.0.0.1:8000/new에 접속 후,

제목과 내용을 입력하고 form을 제출하고

/admin에서 모델이 잘 만들어졌는지 확인해보세요!

# CRUD

read

## 독후감 블로그

- [오타와 함께 사라지다](#)
- [누가 내 코드에 똥 썼어?](#)
- [다빈치 코딩](#)
- [장고에서 살아남기](#)

[글 쓰러가기](#)

## 책 제목

누가 내 코드에 똥 썼어?

## 책 내용

범인은 이 git 안에 있다  
[홈으로](#)

저희가 원하는 것은, home에서 특정 독후감의 제목을 누르면 해당하는 독후감의 상세 내용을 보여주는 페이지(detail)로 이동하는 기능입니다.  
(+ 새 독후감 작성 직후에도 해당 독후감의 detail로 이동)

이것 역시 **MTV** 패턴에 따라

**Model**을 만들고 => **Template**를 만들고 => **View**를 만들어줍니다.

Model은 이미 만들었으니, Template부터 만들어줍니다.

# CRUD

read

MTV

project/project/urls.py

```
from django.contrib import admin
from django.urls import path
from app import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name="home"),
    path('new/', views.new, name="new"),
    path('detail/<int:post_pk>', views.detail, name="detail"),
]
```

# CRUD

read

MTV

project/app/templates/detail.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <div>
    <h2>책 제목</h2>
    <p>{{ post.title }}</p>
  </div>
  <div>
    <h2>책 내용</h2>
    <p>{{ post.content }}</p>
  </div>
  <a href="{% url 'home' %}">홈으로</a>
</body>
</html>
```

# CRUD

read

project/app/templates/home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>독후감 블로그</h1>
  <div>
    <ul>
      {% for post in posts %}
      <li>
        <a href="{% url 'detail' post.pk %}">{{ post.title }}</a>
      </li>
      {% endfor %}
    </ul>
  </div>
  <a href="{% url 'new' %}">글 쓰러가기</a>
</body>
</html>
```



# CRUD

read

project/app/views.py

```
from django.shortcuts import render, redirect
```

```
def new(request):  
    if request.method == 'POST':  
        new_post = Post.objects.create(  
            title = request.POST['title'],  
            content = request.POST['content']  
        )  
        return redirect('detail', new_post.pk)
```

```
    return render(request, 'new.html')
```

```
def detail(request, post_pk):  
    post = Post.objects.get(pk=post_pk)
```

```
    return render(request, 'detail.html', {'post': post})
```

home에서 글의 제목을 클릭해서

localhost:8000/detail/(id)에 접속 후,

id에 해당하는 pk를 가진 Post가 보이는지 확인해보세요!

+새 글을 작성한 후에  
detail 페이지로 가지는지도 확인해보세요!

MTV

# 오늘 할 일

update & delete



CREATE



READ



UPDATE



DELETE

C

R

U

D

# CRUD

update

## 책 제목

누가 내 코드에 똥 썼어?

## 책 내용

범인은 이 git 안에 있다  
[홈으로 수정하기](#)

제목 누가 내 코드에 똥 썼어?

범인은 이 git 안에 있다

내용

수정하기

저희가 원하는 것은, detail에서 '수정하기'를 누르면 글을 수정할 수 있는 페이지(edit)로 이동하고, 글을 수정할 수 있는 기능입니다.

이것 역시 **MTV** 패턴에 따라  
**Model**을 만들고 => **Template**를 만들고 => **View**를 만들어줍니다.

Model은 이미 만들었으니, Template부터 만들어줍니다.

# CRUD

update

MTV

project/project/urls.py

```
from django.contrib import admin
from django.urls import path
from app import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name="home"),
    path('new/', views.new, name="new"),
    path('detail/<int:post_pk>', views.detail, name="detail"),
    path('edit/<int:post_pk>', views.edit, name="edit"),
]
```

# CRUD

update

project/app/templates/edit.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <form action="" method="post">
    {% csrf_token %}
    <div>
      <label for="title">제목</label>
      <input type="text" id="title" name="title" value="{{ post.title }}">
    </div>
    <div>
      <label for="content">내용</label>
      <textarea name="content" id="content" cols="30" rows="10" >{{ post.content }}</textarea>
    </div>
    <button type="submit">수정하기</button>
  </form>
</body>
</html>
```

project/app/templates/detail.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <div>
    <h2>책 제목</h2>
    <p>{{ post.title }}</p>
  </div>
  <div>
    <h2>책 내용</h2>
    <p>{{ post.content }}</p>
  </div>
  <a href="{% url 'home' %}">홈으로</a>
  <a href="{% url 'edit' post.pk %}">수정하기</a>
</body>
</html>
```

detail에서 “수정하기”를 클릭하셔서

localhost:8000/edit/(id)에 접속 후,

pk를 통해 받아온 모델이 잘 수정 되는지 확인해보세요!

project/app/views.py

```
def edit(request, post_pk):
    post = Post.objects.get(pk=post_pk)

    if request.method == 'POST':
        Post.objects.filter(pk=post_pk).update(
            title = request.POST['title'],
            content = request.POST['content']
        )
        return redirect('detail', post_pk)

    return render(request, 'edit.html', {'post': post})
```

**주의!** update()는 QuerySet 타입에서만 사용이 가능합니다.

filter()는 여러개의 객체를 포함하는 QuerySet을 반환해줍니다. (all()도 QuerySet을 반환함)  
그러나, get()은 객체(여기선 Post) 하나만 반환해줍니다.

따라서 update()는 all()이나 filter() 후에 사용해야합니다.

project/app/views.py

```
def edit(request, post_pk):  
    post = Post.objects.get(pk=post_pk)  
  
    if request.method == 'POST':  
        updated_post = Post.objects.filter(pk=post_pk).update(  
            title = request.POST['title'],  
            content = request.POST['content']  
        )  
        return redirect('detail', post_pk)  
  
    return render(request, 'edit.html', {'post': post})
```

**주의2!** 이렇게 해도 수정은 되긴 하지만, 유의해야할 점이 있습니다.

update()는 수정한 post를 return하지 않고, update한 post의 개수를 return합니다.  
따라서 여기서 updated\_post의 값은 1입니다. post 모델이 아닙니다.  
그러므로 redirect()에 전달하는 pk값으로 updated\_post.pk가 아니라  
edit()의 인자로 받아온 **post\_pk**를 사용해야합니다.



# CRUD

delete

## 책 제목

누가 내 코드에 똥 썼어?

## 책 내용

범인은 이 git 안에 있다  
홈으로 수정하기 삭제하기

저희가 원하는 것은, detail에서 '삭제하기' 를 눌러 글을 삭제하는 것입니다.  
(delete는 별도의 페이지가 필요 없습니다.)

이것 역시 **MTV** 패턴에 따라  
**Model**을 만들고 => **Template**를 만들고 => **View**를 만들어줍니다.

Model은 이미 만들었으니, Template부터 만들어줍니다.

# CRUD

## delete

project/project/urls.py

```
from django.contrib import admin
from django.urls import path
from app import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name="home"),
    path('new/', views.new, name="new"),
    path('detail/<int:post_pk>', views.detail, name="detail"),
    path('edit/<int:post_pk>', views.edit, name="edit"),
    path('delete/<int:post_pk>', views.delete, name="delete"),
]
```

# CRUD

delete

MTV

project/app/templates/detail.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <div>
    <h2>책 제목</h2>
    <p>{{ post.title }}</p>
  </div>
  <div>
    <h2>책 내용</h2>
    <p>{{ post.content }}</p>
  </div>
  <a href="{% url 'home' %}">홈으로</a>
  <a href="{% url 'edit' post.pk %}">수정하기</a>
  <a href="{% url 'delete' post.pk %}">삭제하기</a>
</body>
</html>
```

# CRUD

## delete

project/app/views.py

```
def delete(request, post_pk):  
    post = Post.objects.get(pk=post_pk)  
    post.delete()  
  
    return redirect('home')
```

detail 페이지에서 ‘삭제하기’를 눌러

pk를 통해 받아온 모델이 삭제되는지 확인해보세요!

# 과제

## To-Do-List 게시판 만들기

### 기능

- 할 일들을 전부 볼 수 있다.(제목 or 제목+내용 or 내용)
- 할 일을 생성할 수 있다.
- 각 할 일들의 제목, 세부사항, 마감 기한을 볼 수 있다.
- 각 할 일들을 수정할 수 있다.
- 각 할 일들을 삭제할 수 있다.

### 필수 사항

- 마감 기한이 적게 남은 순으로 할 일들이 보이게 정렬해주세요.
- CSS로 예쁘게 꾸며주세요.
- 오늘 배운 home.html, detail.html, edit.html, new.html이 구현되어야 합니다.
- 오늘 만든 Django 프로젝트가 아니라, 새로운 Django 프로젝트를 생성 해주셔야합니다.

### 선택 사항

- 마감 기한으로부터 오늘까지 남은 날짜를 할 일 옆에 보여주세요.