

Django CRUD - UD

# 7번째 세션

---

NEXT X LIKELION 김범진

# 수강신청 알리미

앞광고

<https://ku-sugang.notion.site/ku-sugang/9f2735e373e54e1aa5aed825405aa94b/>

수강신청 알리미

2월 23일 - 24일은  
신/편입생 수강신청 기간입니다.

2.23(수) 10:00 - 2.24(목) 12:00

과목명, 학수번호, 교수명으로 검색

즐거찾기한 강의

과목명/학수번호를 클릭해 복사해보세요

수강신청

BUSS203-01  
경영수학  
박광태  
66/67

GELI001-6K  
자유정의진리 I  
강동원  
28/30

공지사항

2022.02.17  
빈자리 알림 기능 사용법

홈 즐겨찾기 마이페이지

수강신청 즐겨찾기한 강의 9 / 10개

수강신청 수강희망

BUSS203-01  
경영수학  
박광태  
66/67

GELI001-4E  
자유정의진리 I  
박성태  
29/30

GELI001-AI  
자유정의진리 I  
이창희  
28/30

GELI001-AH  
자유정의진리 I  
김복희  
28/30

GELI001-CH  
자유정의진리 I  
김민수  
66/66

GELI001-BI  
자유정의진리 I  
남중권  
66/66

홈 즐겨찾기 마이페이지

수강신청 알리미팀은, 고려대학교 학우분들의 쉽고 편한 수강신청을 목표로 항상 노력하고 있습니다. 수강신청 알리미는, 고려대학교 대표 커뮤니티 '고파스'와 협업 관계에 있으며, 고려대학교 디지털정보처의 공식 승인을 받아 진행되고 있음을 알려드립니다.

- 수강신청/수강희망 과목 검색 : 과목명, 교수명, 학수번호 등으로 검색 가능
- 수강신청/수강희망 경쟁률 확인 : 검색 페이지에서 2021년도 2학기부터 현재까지의 경쟁률 확인 가능
- 즐거찾기 : 여러 과목의 마감현황을 한번에 보기
- 고파스로 로그인 : 안정화와 보안을 위해 고파스 소셜 로그인을 통해 고려대학교 학생 인증
- 빈자리 알림 : 즐겨찾기에 등록한 과목 중 빈자리가 생기면 고파스 앱 푸시 알림 제공
- 커뮤니티 : 수강신청 관련 공지사항 및 콘텐츠 제공

✓2020년도 2학기 600 여명 유입

✓2021년도 1학기 2,500 여명 유입, 조회 수 25,452 회

✓2021년도 2학기 13,097 명 유입, 조회 수 218,371 회, 빈자리 알림 4,492 건

✓2022년도 1학기 28,675 명 유입, 조회 수 711,146 회, 빈자리 알림 57,712 건

# 오늘 할 일

update & delete



CREATE



READ



UPDATE



DELETE

---

C

R

U

D

# 오늘의 실습

독후감 블로그

**http://127.0.0.1:8000/**

작성한 모든 독후감을 보여줍니다.

## 독후감 블로그

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

[글 쓰러가기](#)

**http://127.0.0.1:8000/new/**

새로운 독후감을 작성합니다.

제목

제목을 입력해주세요

내용을 입력해주세요

내용

작성하기

# 오늘의 실습

독후감 블로그

**http://127.0.0.1:8000/detail/  
<int:post\_pk>/**

작성한 특정 독후감과 관련된 상세 내용을 보여줍니다.

## 책 제목

누가 내 코드에 똥 썼어?

## 책 내용

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

**http://127.0.0.1:8000/edit/  
<int:post\_pk>/**

작성한 특정 독후감을 수정합니다.

제목

내용

# Django 복습

Django란?



(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

가상환경

## 파이썬 가상 환경은 PC에 독립된 파이썬을 추가로 설치하는 것과 유사하다!

파이썬 런타임, 인터프리터, 패키지 매니저(pip), 라이브러리 저장소 등을 별도의 환경에서 사용할 수 있게 된다.

which python 명령어로 현재 실행중인 파이썬 경로 확인 가능!

```
(session7) beomjin ~ /session/session7  
▶ which python  
/Users/beomjin/.local/share/virtualenvs/session7-wKxfxp-x/bin/python
```

```
beomjin ~ /session/session7  
▶ which python  
/Users/beomjin/.pyenv/shims/python
```

# Django 복습

가상환경

**주의!** pipenv shell은 명령어를 입력한 경로를 기반으로 가상환경 이름을 만들기 때문에 다른 경로에서 명령어를 입력하면, 다른 가상환경으로 인식한다.  
웬만하면 프로젝트 루트 디렉터리(pipfile 있는 곳)에서만 pipenv shell을 하자!

**주의!** 패키지나 라이브러리를 다운받을 때는 pipenv install을 사용해야지  
Pipfile과 Pipfile.lock에 기록된다!  
pip install을 사용하면 가상환경 내의 파이썬에 설치는 되지만,  
Pipfile과 Pipfile.lock에 기록되지 않는다.  
웬만하면 pipenv install을 하자!



# 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',
]
```

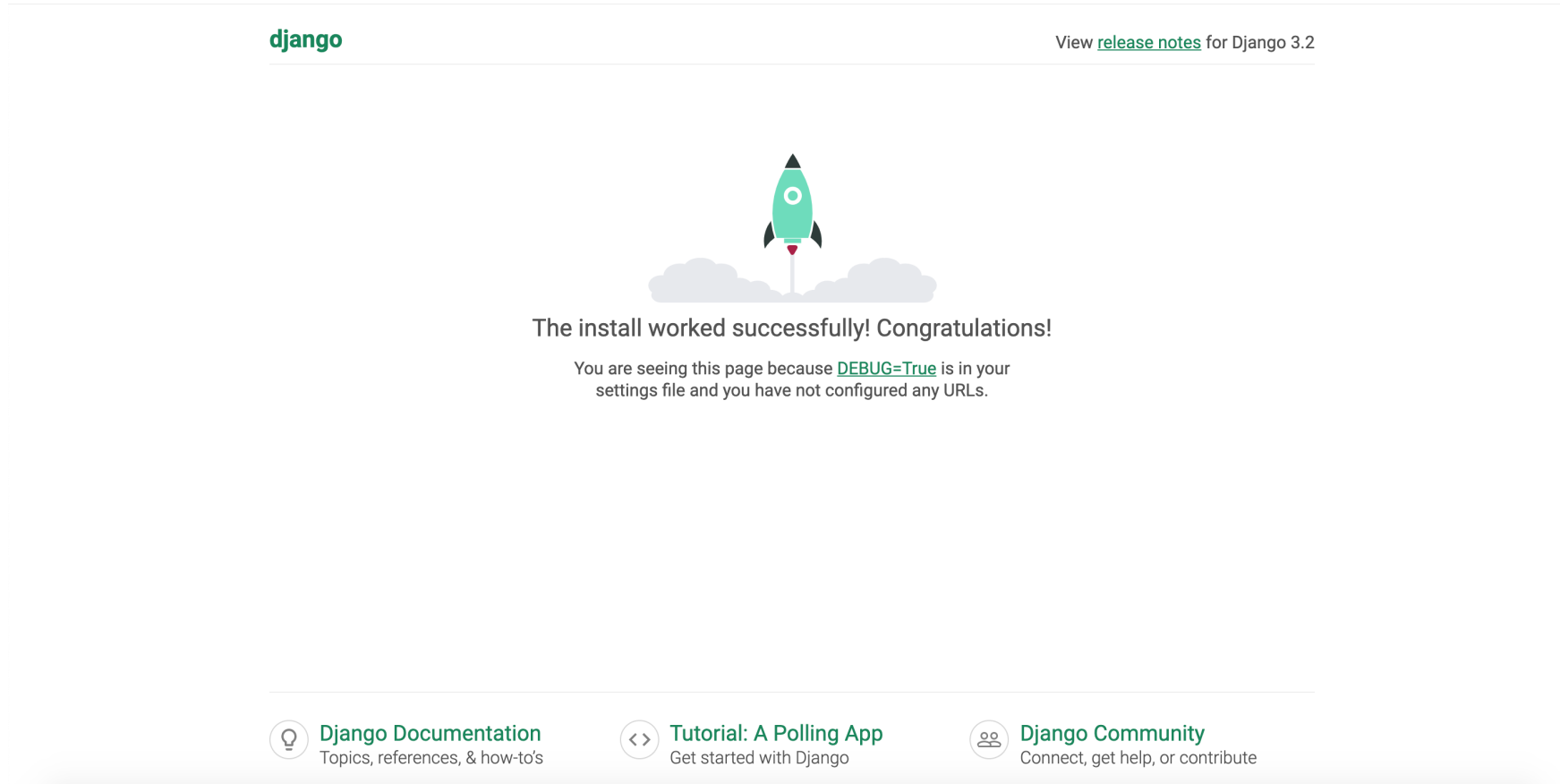
```
TIME_ZONE = 'Asia/Seoul'
```

```
python manage.py runserver
```

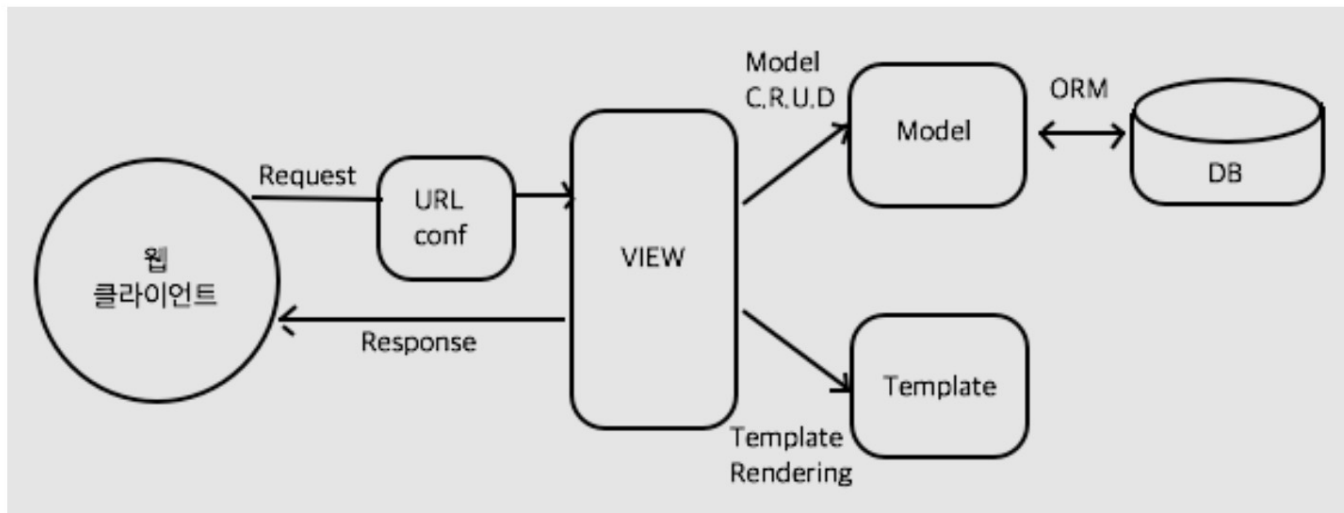
로컬 서버에서 장고 서버 실행

# Django 복습

코드 입력



## 장고의 동작 과정: MTV 패턴



MTV 패턴이란?

**Model, Template, View** 로 이뤄진 Django의 설계 패턴!

## Model – 데이터베이스 설계

- 데이터베이스에 저장되는 데이터의 영역

## Template – 화면 UI 설계

- 사용자에게 보여지는 영역, 화면
- HTML

## View – 프로그램 로직 설계

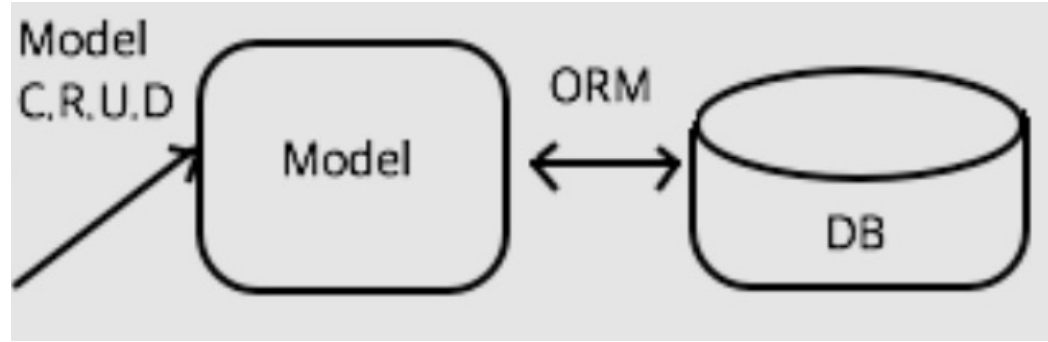
- 요청에 따라 Model에서 필요한 데이터를 가져와 처리
- 처리 결과를 Template에 전달

## MTV 패턴의 장점

- 관심사에 따라 코드가 분리되어 가독성이 높아진다.
- Model, View, Template 간의 독립성을 유지할 수 있다.
- 모듈 간의 결합도 느슨해져 변경 사항의 범위가 줄어든다.
- 화면 디자이너, DB 개발자, 응용 SW 개발자 간의 협업이 쉬워진다.

## Django의 DB처리 방식

ORM



Django의 DB처리는 **ORM(Object Relational Mapping)기법**을 사용한다.  
객체(Object)와 관계(Relational)를 연결(Mapping)해준다.

**Model**이란, 테이블을 정의하는 장고의 클래스를 의미하며, **models.py** 파일에 테이블 관련 사항들을 정의한다.

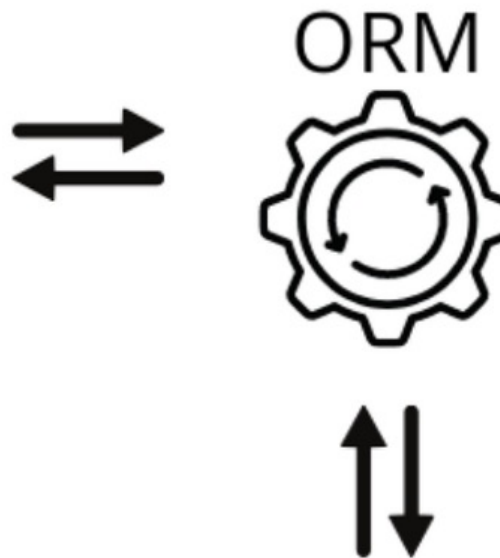
테이블	==	클래스 (django.db.models.Model클래스 상속)
테이블의 컬럼	==	클래스의 속성 (models의 필드 사용)

# Django의 DB처리 방식

ORM

심화!

```
class Recipe(models.Model):  
    name = models.CharField(max_length=150)  
    image = models.ImageField(upload_to="images/")
```



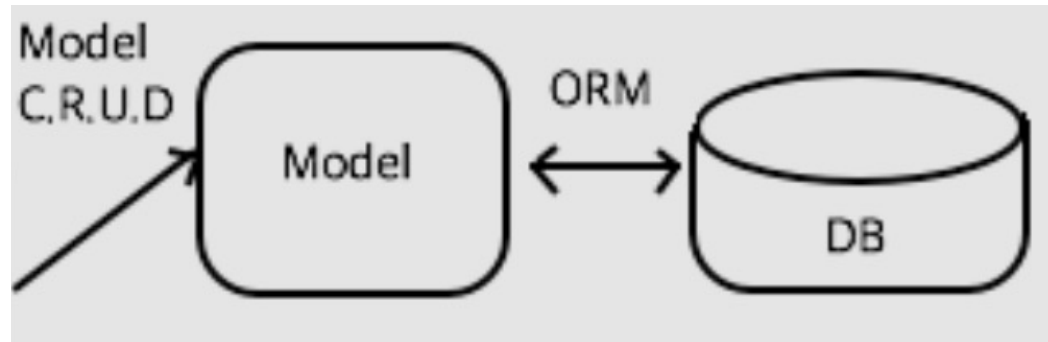
Database



```
CREATE TABLE recipe (  
    id int NOT NULL PRIMARY KEY,  
    name VARCHAR(150),  
    image BLOB,  
);
```

# Django의 DB처리 방식

## Migration



Model에 변경 사항이 생기면 DB도 변경되므로 Model의 변경 내역을 DB에게 알려줘야 한다.  
이때 Model의 변경 내역을 DB 스키마에 적용시키는 방법이 **Migration**이다.

`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.



결론은,

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

```
# 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

python manage.py migrate

python manage.py createsuperuser

python manage.py runserver

migrations만들기

DB에 반영

관리자 계정 생성

로컬 서버에서 장고 서버 실행

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

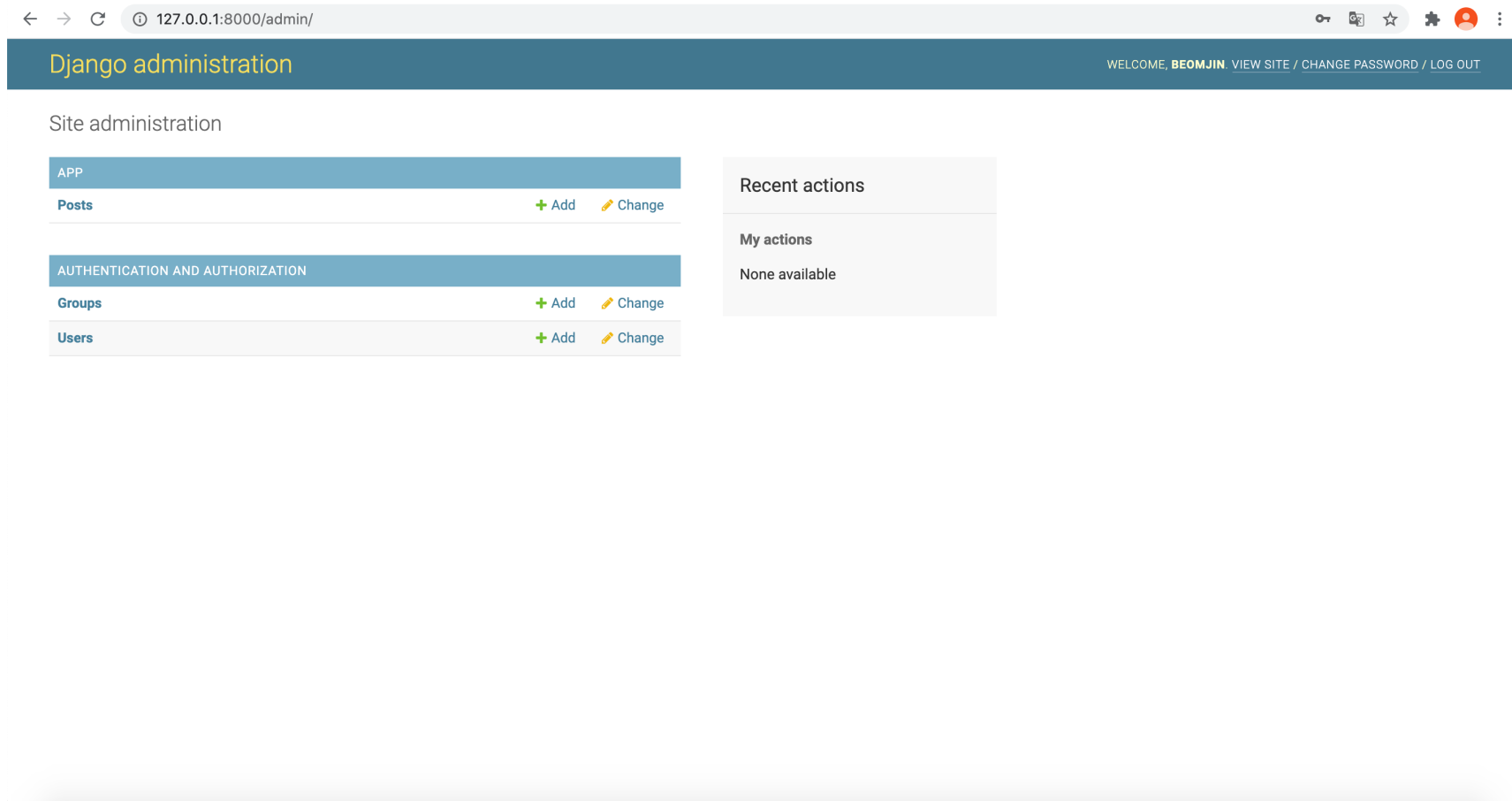
```
# project/app/admin.py
from django.contrib import admin
from .models import Post

# 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 title is "Django administration". The top navigation bar includes the text "WELCOME, BEOMJIN" and links for "VIEW SITE", "CHANGE PASSWORD", and "LOG OUT". The breadcrumb trail is "Home > App > Posts > Add post".

The left sidebar contains a menu with the following items:

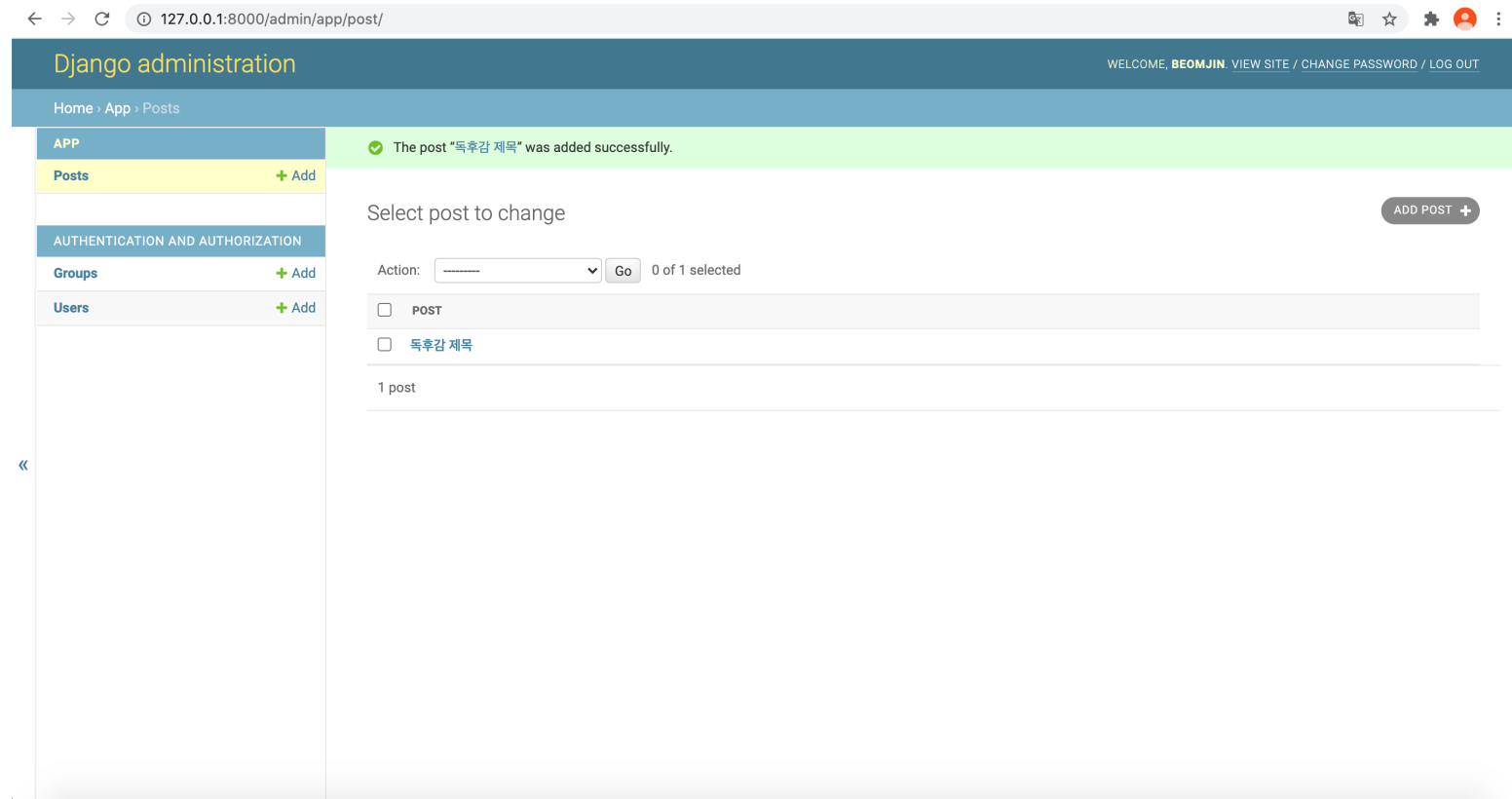
- APP
  - Posts [+ Add](#)
- AUTHENTICATION AND AUTHORIZATION
  - Groups [+ Add](#)
  - Users [+ Add](#)

The main content area is titled "Add post". It contains two form fields:

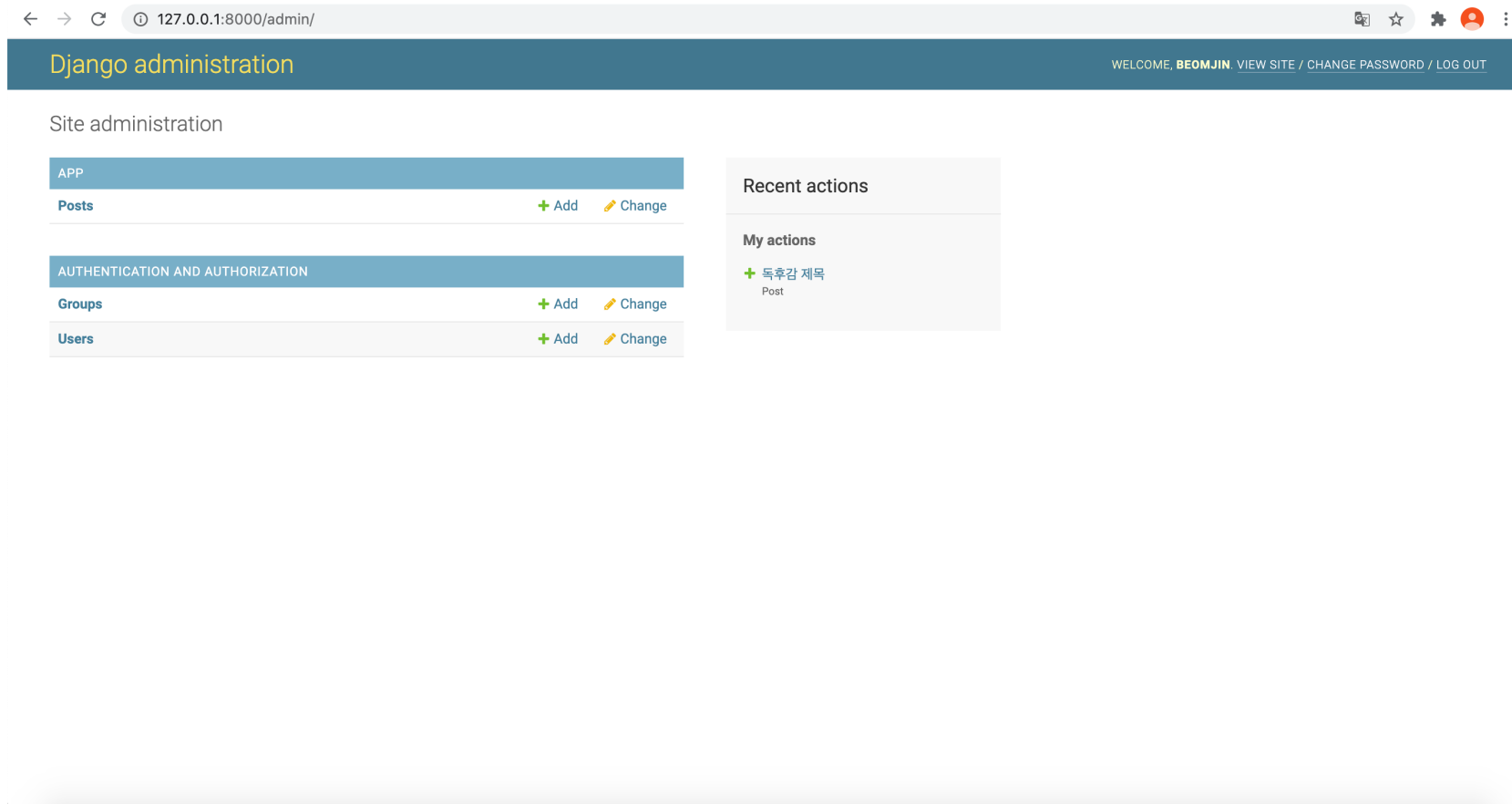
- Title:** A text input field containing the text "독후감 제목".
- Content:** A large text area containing the text "독후감 내용".

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 url과 View를 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.

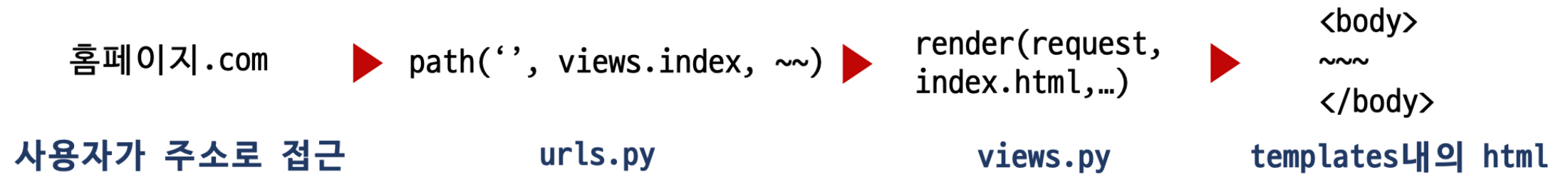
```
# project/app/views.py
from django.shortcuts import render
from .models import Post

# 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 `http://localhost:8000/`으로 접속하면,

```
# 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 인식하고, `views.home`을 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.

하지만 posts는 파이썬 변수.  
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/ko/4.0/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](#)

<https://docs.djangoproject.com/ko/4.0/howto/custom-template-tags/>

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

심화!

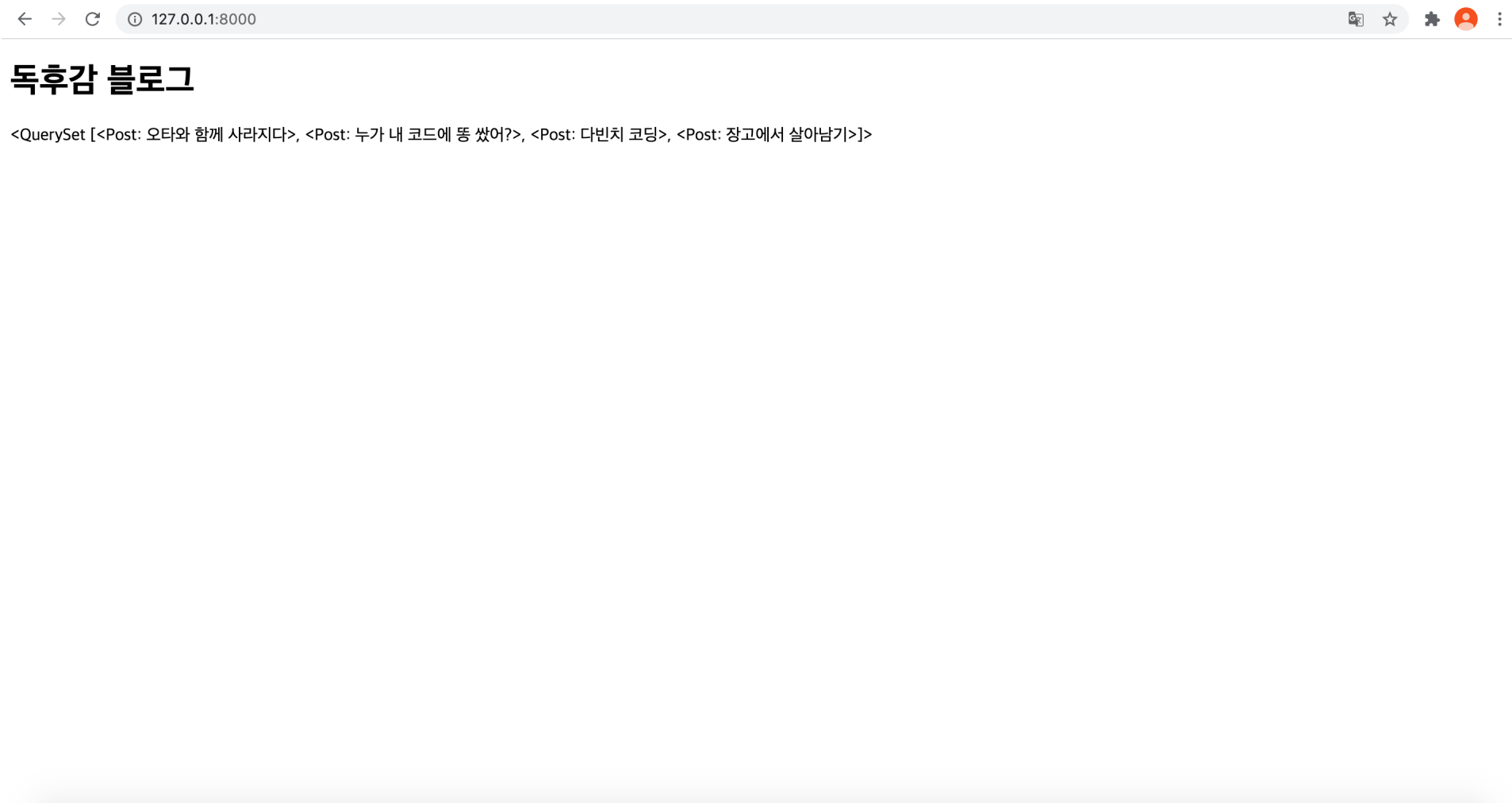
```
# project/app/templates/home.html
```

```
<!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

심화!



## 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

심화!

### QuerySet Method

데이터 베이스에서 가져온 모델들을 사용자 입맛에 맞게 바꿔주는 일종의 함수

Ex) 모든 객체 가져오기, 특정 객체 가져오기, 개수 세기, 필터링, 조건에 맞춰 정렬 등

<https://docs.djangoproject.com/ko/4.0/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`
- `icontains`
- `contains`
- `in`
- `gt`
- `gte`
- `lt`
- `lte`
- `startswith`
- `istartswith`
- `endswith`
- `iendswith`
- `range`
- `date`

**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.

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" name="title" id="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 쓰러가기” 를 클릭하셔서  
http://localhost:8000/new/ 또는 http://128.0.0.1:8000/new/에 접속 후,  
제목과 내용을 입력하고 form을 제출하고  
Admin 페이지에서 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.

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>
```

project/app/views.py

home에서 글의 제목을 클릭해서  
http://localhost:8000/detail/(id)/에 접속 후,  
id에 해당하는 pk를 가진 Post가 보이는지 확인해보세요!

```
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})
```

+새 글을 작성한 후에  
detail 페이지로 가지는지도 확인해보세요!

# 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.

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  
http://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.

update

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_post`의 값은 1입니다. post 모델이 아닙니다.  
그러므로 `redirect()`에 전달하는 pk값으로 `updated_post.pk`가 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.

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>
```

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 해주셔야 합니다.

### 선택 사항

- D-day를 할 일 옆에 보여주세요.