



고용노동부



한국산업인력공단



국가인적자원개발컨소시엄

# Python Web Programming with Django

백명숙 ( [ms.baek@hucloud.co.kr](mailto:ms.baek@hucloud.co.kr) )



■ 웹 프레임워크 중 하나인 파이썬 Django 강좌 입니다. 간결하면서도 중요한 내용들을 쉽게 풀어 설명하면서, 실습을 통해 Django프레임워크를 사용하며 전반적인 웹서비스 개발의 흐름도 자연스럽게 익힐 수 있는 과정입니다.

- 파이썬 Django 프레임워크 사용법을 배우게 됩니다.
- 웹 서비스 예제를 만들어 보며 자연스럽게 기술을 체득할 수 있습니다.
- 웹 개발의 전반적인 흐름을 배우게 됩니다.
- 예제를 진행하며 중요한 개념을 반복적으로 사용하게 됩니다



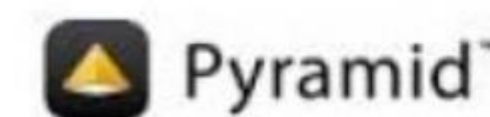
# Django (Python Full stack Web Framework)

---

- Django(/dʒæŋɡoʊ/ jang-goh/쟁고/장고)는 Python으로 만들어진 무료 오픈소스 웹 애플리케이션 프레임워크(web application framework)

쉽고 빠르게 웹사이트를 개발할 수 있도록 돕는 구성요소로 이루어진 웹 프레임워크

- 백엔드를 담당하는 파이썬 풀스택 웹프레임워크
- Lawrence Journal-World 신문사에서 2003년부터 개발하여, 2005년에 세상에 공개
- 2008년에 1.0 릴리즈 (Django Roadmap)
- 기타리스트 Django Reinhardt 이름을 따서, Django (쟁고, 장고)
- Django 와 비슷한 웹프레임워크들
  - Flask : a micro framework for Python based on Werkzeug.
  - Pyramid : a small, fast, down-to-earth,
  - Bottle : a fast and simple micro framework for small web-applications



# Python 설치 및 버전 확인

---

- Python 3.7 이상
- <https://www.python.org/downloads/>
- 윈도우에서 데이터 분석이나 머신러닝을 같이 하실려면,  
Anaconda Python 64 비트 추천
- <https://www.anaconda.com/download/#windows>
- 쉘 > python --version  
    쉘 > pip --version



주의 : 맥/리눅스에서는 python 명령으로 파이썬2가 실행될 수 있습니다. 그럴 때에는 python3 명령을 입력해 보세요. 환경변수 PATH에 대한 명확한 이해가 필요합니다.

# Django 설치 및 버전 확인

---

- Pypi에서 Django 확인해 보기
- <https://pypi.python.org/pypi/Django>
- Django 에서의 LTS(Long Term Support) 버전  
대개 3년 동안 업데이트를 지원 (Django Roadmap)
- 공식 소스코드 저장소 : <http://github.com/django/django>
- 설치 : 파이썬 패키지 매니저인 pip를 이용
- 쉘 > pip install django
- 쉘 > django-admin --version



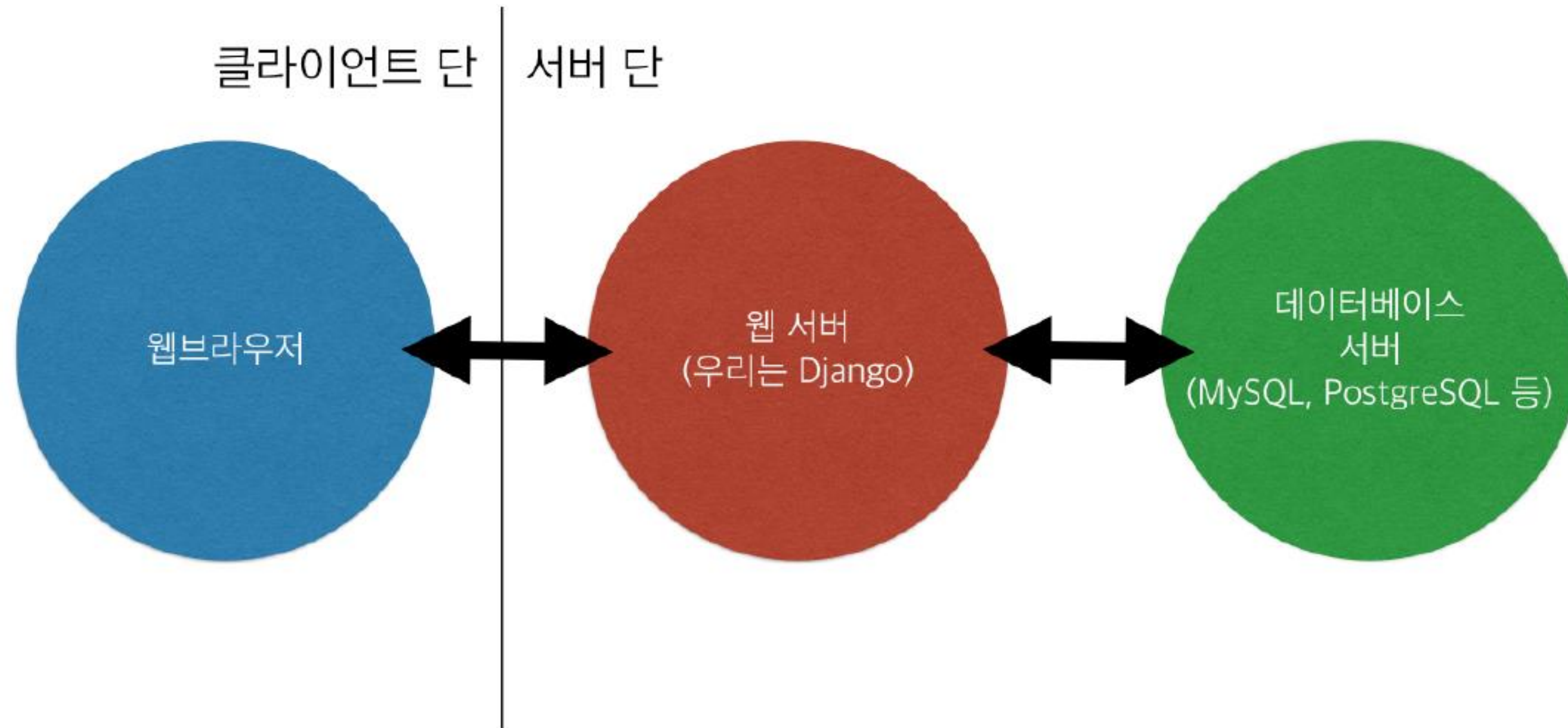
# Code Editor 설치

---

- JetBrains pycharm  
<https://www.jetbrains.com/pycharm/download/#section=windows>
- Microsoft Visual Studio Code  
<https://code.visualstudio.com/docs/?dv=win>
- GitHub 에서 만든 Atom  
<https://atom.io/>
- Sublime Text 3  
<https://www.sublimetext.com/3>

# 웹 어플리케이션 기본 구조

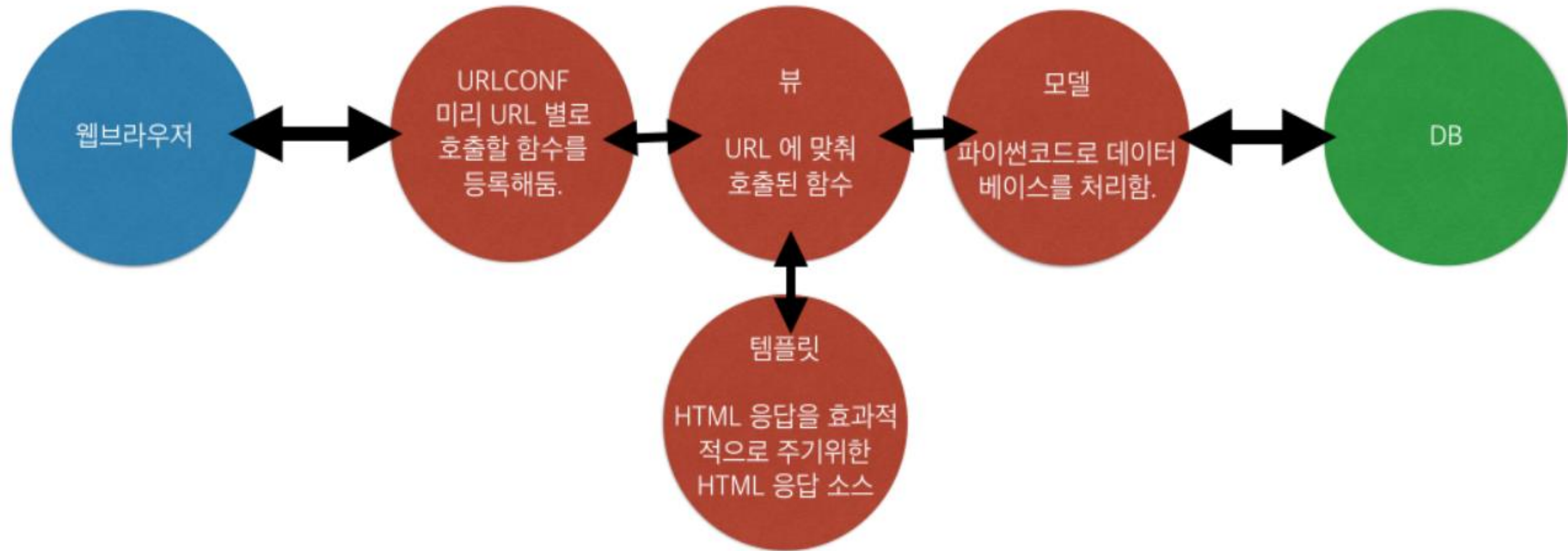
---





# Django 기본 구조

---





# MTV 개발 방식

---

- MTV(Model Template View) 패턴

- 1) Model : 테이블을 정의한다.

- 2) Template : 사용자가 보게 될 화면의 모습을 정의한다.

- 3) View : 애플리케이션의 제어 흐름 및 처리 로직을 정의한다.

모델은 `model.py` 파일에, 템플릿은 `templates` 디렉토리 하위의 `*.html` 파일에, 뷰는 `views.py` 파일에 작성하도록 처음부터 뼈대를 만들어 줍니다.

모델, 템플릿, 뷰 모듈 간에 독립성을 유지할 수 있고, 소프트웨어 개발의 중요한 원칙인 느슨한 결합(Loose Coupling) 설계의 원칙에도 부합된다.

Django에서 프로젝트를 생성하기 위해 `startproject` 및 `startapp` 명령을 실행하면 자동으로 프로젝트 뼈대(skeleton)에 해당하는 디렉토리와 파일들을 만들어 줍니다.

# MTV 개발 방식

- Django's Architecture

Django's Structure ( <https://djangobook.com/mdj2-django-structure/> )

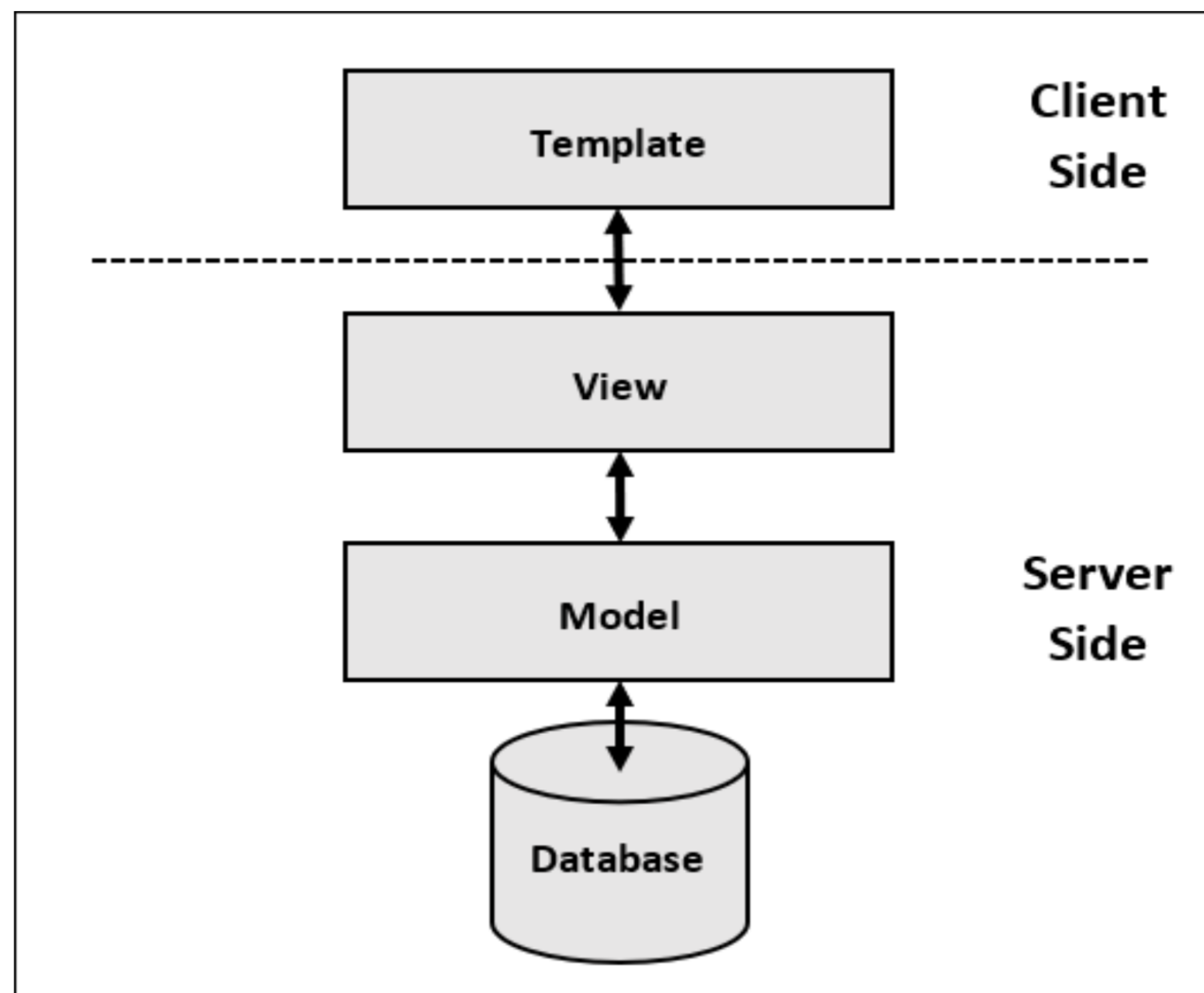


Figure 3.2: A slightly different view of Django's MTV "stack".

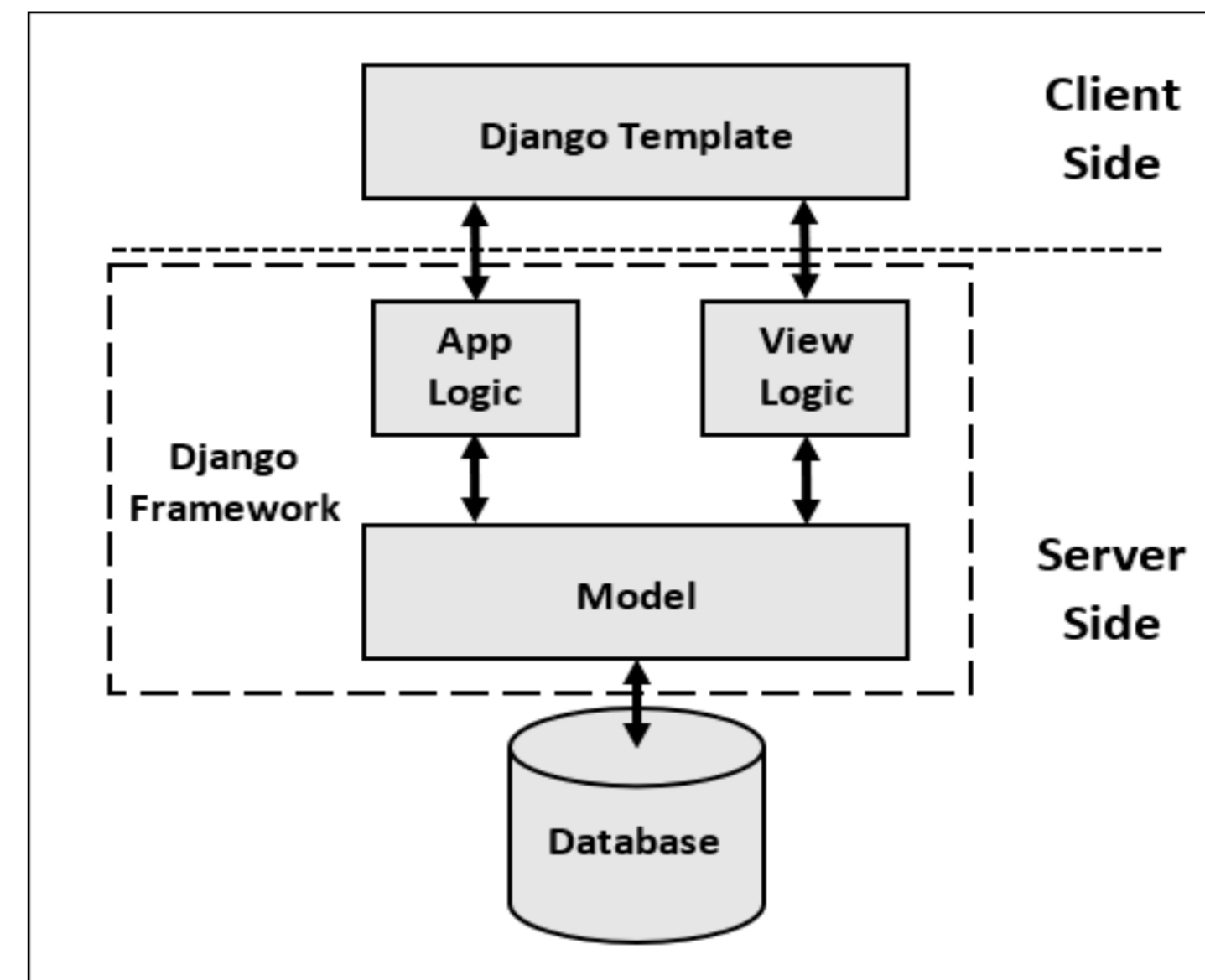


Figure 3.3: A more holistic view of Django's architecture.

# blog/urls.py

---

```
from django.conf.urls import url
from blog import views

urlpatterns = [
    # http://localhost:8000/blog/ 주소로 접근하면
    url(r'^blog/$', views.post_list),
    # http://localhost:8000/blog/new/ 주소로 접근하면
    url(r'^blog/new/$', views.post_new),
    # http://localhost:8000/blog/10/ 주소로 접근하면
    url(r'^blog/10/$', views.post_detail),
    # http://localhost:8000/blog/숫자/edit/ 주소로 접근하면,
    url(r'^blog/(?P<id>\d+)/edit/$', views.post_edit),
]
```

# blog/views.py

---

```
from django.shortcuts import get_object_or_404, render
from blog.models import Post

def post_list(request):
    qs = Post.objects.all() # DB로부터 Post목록을 Fetch할 예정
    return render(request, 'blog/post_list.html', {'post_list': post_list})

def post_detail(request, id):
    post = get_object_or_404(Post, id=id) # DB로부터 지정Post를 Fetch
    return render(request, 'blog/post_detail.html', {'post': post})

def post_edit(request, id):
    # TODO: 구현 예정
    pass
```



# models.py

---

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

# blog/templates/blog/post\_list.html

---

```
<table>
  <tbody>
    {% for post in post_list %}
      <tr>
        <td>{{ post.id }}</td>
        <td><a href="{% url 'blog:post_detail' post.id %}">{{ post.title }}</a></td>
        <td>{{ post.updated_at }}</td>
      </tr>
    {% endfor %}
  </tbody>
</table>
```

# Django 프로젝트 생성

---

mypython 폴더 아래에 django\_src 하위 폴더를 생성한다.

# Django 프로젝트 생성

mypython> **django-admin startproject mydjango** django\_src

=> django/conf/project\_template 구성으로 생성 됨 #src

▲ DJANGO\_SRC

▲ mydjango

🐍 \_\_init\_\_.py

🐍 settings.py

🐍 urls.py

🐍 wsgi.py

🐍 manage.py

- manage.py : 웹사이트 관리를 도와주는 역할을 하는 파일
- settings.py : 웹사이트 설정이 있는 파일
- urls.py : urlresolver가 사용하는 요청 패턴 목록을 포함하고 있는 파일

# Django 프로젝트 설정 변경

---

# settings.py의 LANGUAGE\_CODE와 TIME\_ZONE 변경하기

mydjango/settings.py

```
LANGUAGE_CODE = 'ko'
```

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

# settings.py에 정적 파일 경로를 추가함 STATIC\_URL항목 바로 아래에 STATIC\_ROOT을 추가

mydjango/settings.py

```
STATIC_URL = '/static/'
```

```
STATIC_ROOT = os.path.join(BASE_DIR, STATIC_URL)
```



# Django 프로젝트 DB 생성과 Server 시작

---

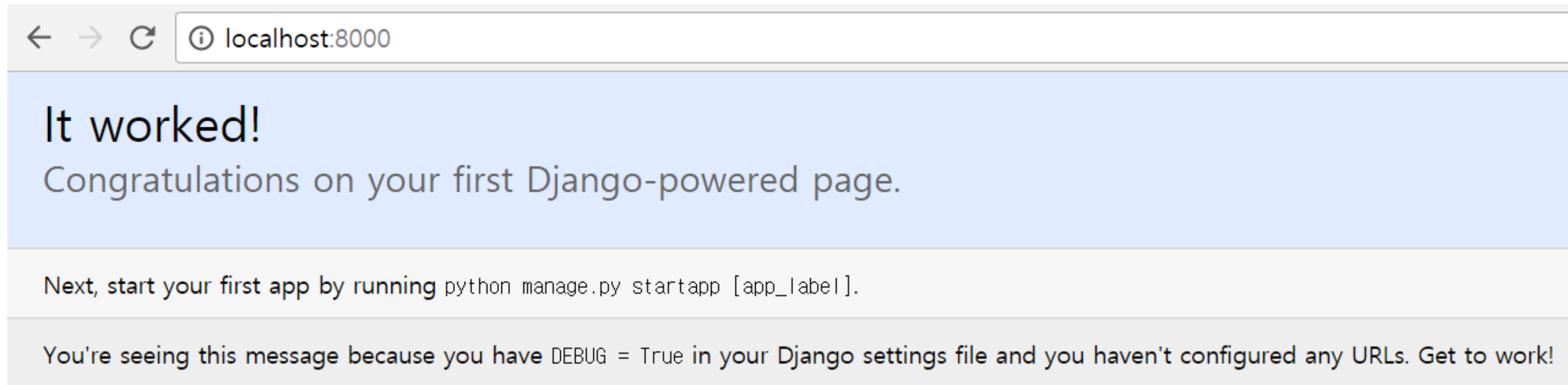
# 데이터베이스를 생성

```
django_src> python manage.py migrate
```

=> db.sqlite3 파일이 생성됨

# Server 시작

```
django_src> python manage.py runserver
```



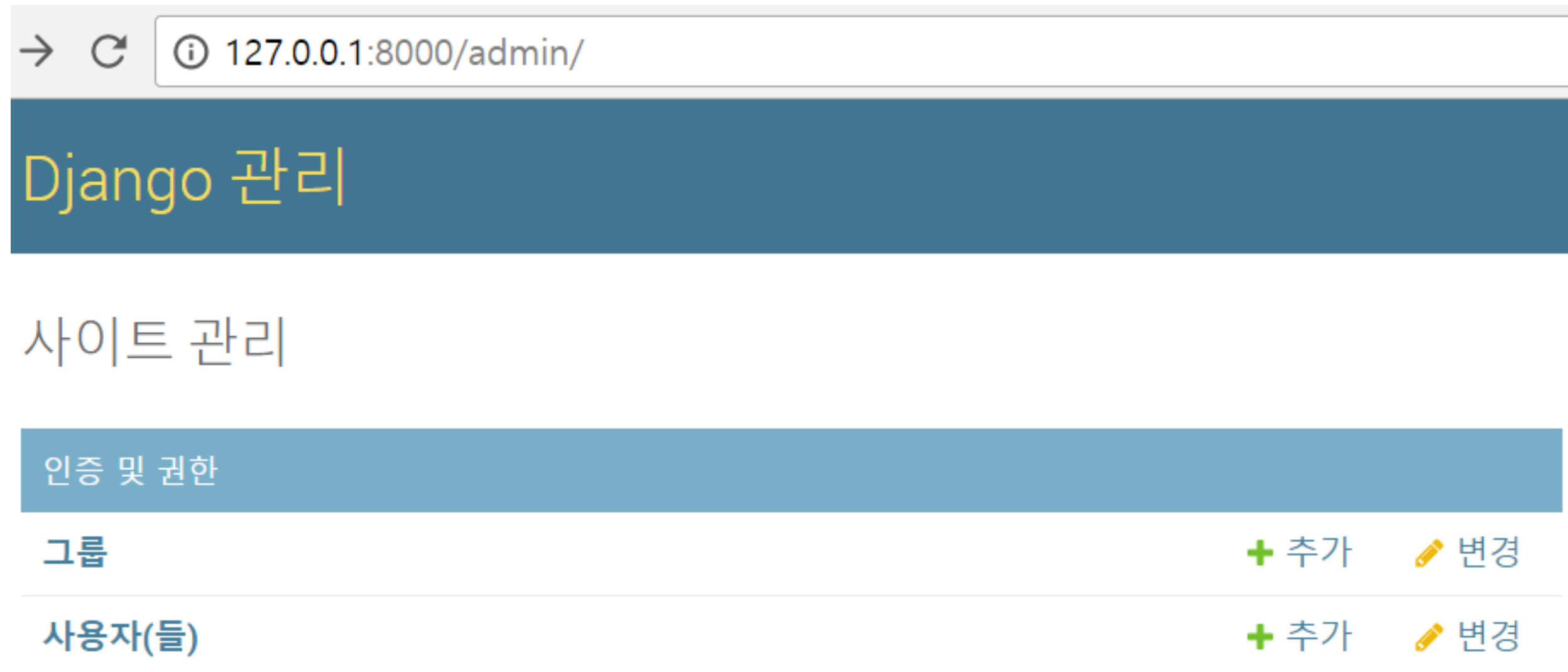
# Superuser 생성 및 관리자 화면

# Superuser 계정 생성

django\_src> `python manage.py createsuperuser`

=> password를 체크 하므로 너무 짧거나 간단한 글자로 입력하면 안됩니다.

- `http://localhost:8000/admin/` 으로 접속

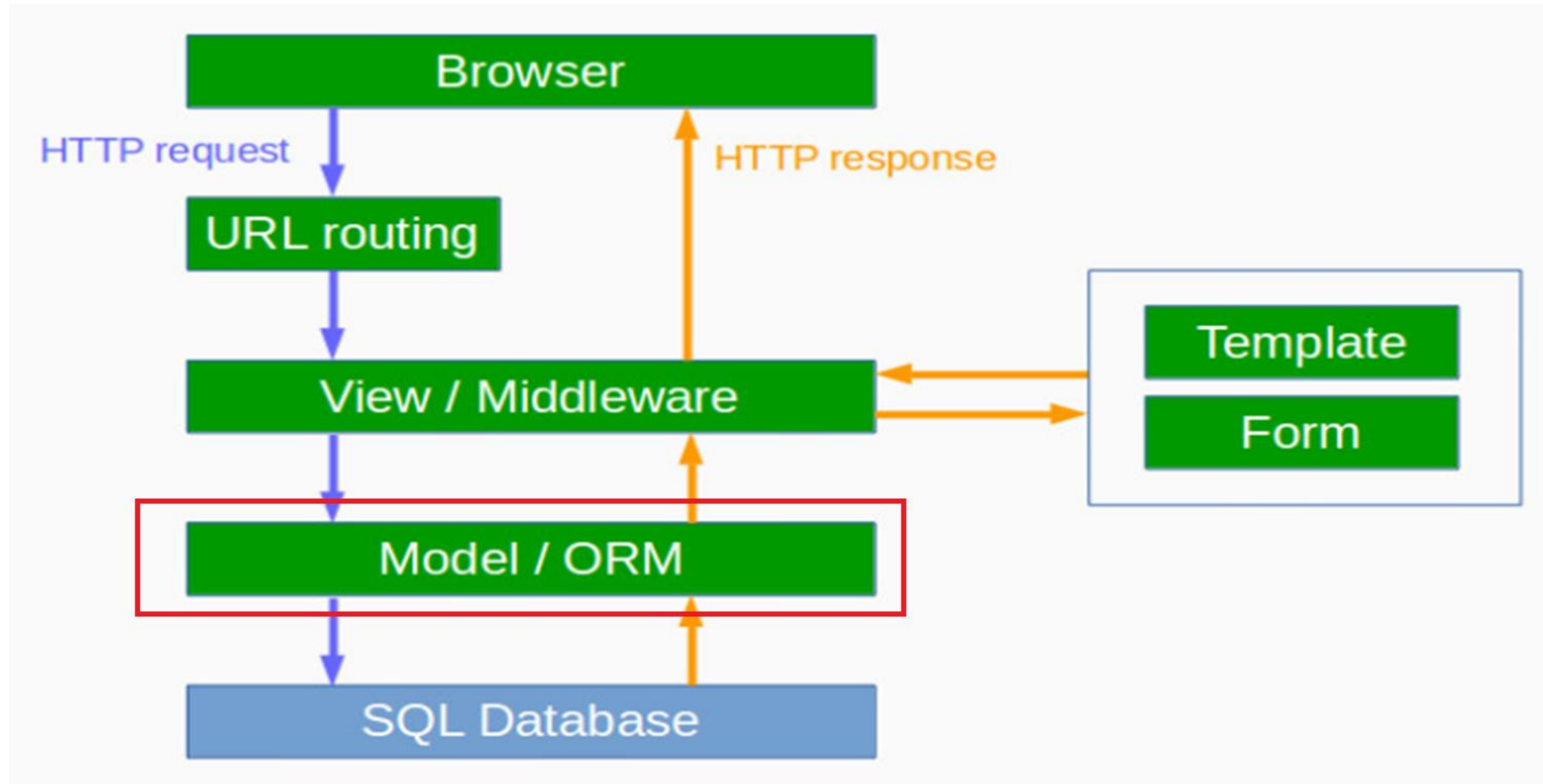


# Blog App 작성

: Model클래스와 테이블

# Django Architecture – Model

---





# Django Model

---

## 1) Django 내장 ORM(Object Relational Mapping)

( <https://docs.djangoproject.com/en/3.0/topics/db/models/> )

2) SQL을 직접 작성하지 않아도, Django Model을 통해 데이터베이스로의 접근 가능 (조회/추가/수정/삭제)

3) <Python 클래스> 와 <데이터베이스 테이블> 을 매핑

- Model : DB 테이블과 매핑
- Model Instance : DB 테이블의 1 Row
- blog앱 Post모델 : blog\_post 데이터베이스 테이블과 매핑
- blog앱 Comment모델 : blog\_comment 데이터베이스 테이블과 매핑

# Django Model

---

- 1) 커스텀 모델 정의 (`blog/models.py`)
- 2) 데이터베이스 테이블 구조/타입을 먼저 설계를 한 다음에 모델 정의  
모델 클래스명은 단수형 (`Posts`가 아니라, `Post`)

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

# 지원하는 모델필드 타입 #ref

---

## 1) Field Types :

**AutoField, BigInteger, BinaryField, BooleanField, CharField, DateField, DateTimeField, DecimalField, DurationField, EmailField, FileField, ImageField, IntegerField, GenericIPAddressField, PositiveIntegerField, PositiveSmallIntegerField, SlugField, TextField, URLField, UUIDField 등**

## 2) Relationship Types :

**ForeignKey, ManyToManyField, OneToOneField**

# 데이터 타입 매핑

---

1) 파이썬 데이터타입과 데이터베이스 데이터타입을 매핑 :

`AutoField (int), BinaryField (bytes), BooleanField (bool),  
NullBooleanField (None, bool), CharField/TextField/  
EmailField/GenericIPAddressField/SlugField/URLField(str)`

2) 같은 파이썬 데이터 타입에 매핑 되더라도, "데이터 형식" 에 따라 여러  
`Model Field Types` 로 나뉨.



# 필드 옵션

---

## 1) 자주 쓰는 필드 옵션 :

`null` (DB 옵션) : DB 필드에 `NULL` 허용 여부 (디폴트 : `False`)

`unique` (DB 옵션) : 유일성 여부

`blank` : 입력값 유효성(`validation`)검사할 때 `empty`값 허용여부(디폴트:`False`)

`default` : 디폴트 값 지정. 값이 지정되지 않았을 때 사용

`choices` (form widget 용) : `select box` 소스로 사용

`validators` : 입력값 유효성 검사를 수행할 함수를 다수 지정

각 필드마다 고유한 `validators` 들이 이미 등록되어 있기도 함.

ex) 이메일만 받기, 최대길이 제한, 최소길이 제한, 최대값 제한, 최소값 제한, etc.

`verbose_name` : 필드 레이블. 지정되지 않으면 필드명이 쓰여짐.

`help_text` (form widget 용) : 필드 입력 도움말

# 첫번째 앱 <blog> 앱 생성

---

1) App 디렉토리 생성

```
django_src> python manage.py startapp blog
```

django/conf/app\_template 구성으로 App 디렉토리가 생성되어진다. #src

2) App을 프로젝트에 등록 : 아래와 같이 mydjango/settings.py을 편집하여,  
INSTALLED\_APPS 항목 끝에 blog App 이름을 추가한다.

# mydjango/settings.py 파일

```
INSTALLED_APPS = [
```

```
# 생략
```

```
'blog',
```

```
]
```

# blog앱 글(Post) Model 만들기

---

1) Post(게시글)의 속성들

title(제목)

text(내용)

author(글쓴이)

created\_date(작성일)

published\_date(게시일)

2) Model 객체는 blog/models.py 파일에 선언하여 모델을 만듭니다.

이 Model을 저장하면 그 내용이 데이터베이스에 저장되는 것입니다.

# blog 앱 글 (Post) Model 만들기

blog/models.py

```
from django.db import models
from django.utils import timezone

class Post(models.Model):
    author = models.ForeignKey('auth.User', on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    text = models.TextField()
    created_date = models.DateTimeField(default=timezone.now)
    published_date = models.DateTimeField(blank=True, null=True)

    def publish(self):
        self.published_date = timezone.now()
        self.save()

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

# DB에 테이블 만들기

---

1) 마이그레이션 파일(migration file) 생성하기

```
django_src> python manage.py makemigrations blog
```

Migrations for 'blog':

```
blog\migrations\0001_initial.py
```

```
- Create model Post
```

2) 실제 데이터베이스에 Model 추가를 반영하기

```
django_src> python manage.py migrate blog
```

Operations to perform:

```
Apply all migrations: blog
```

Running migrations:

```
Applying blog.0001_initial... OK
```

# Django Admin(관리자)

1) 관리자 페이지에서 만든 모델을 보기 위해 Post 모델을 등록

blog/admin.py

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

admin.site.register(Post)
```

2) 관리자 화면에서 확인하기

<http://localhost:8000/admin/> 으로 접속

사이트 관리

BLOG

Posts

+ 추가    ✎ 변경



# Migrations #ref

---

- django-south(<https://south.readthedocs.io/en/latest/>) 프로젝트가 킥스타터 펀딩 (£17,952, 507 Backers)을 통해, Django 1.7에 포함
  - 모델 변경내역 히스토리 관리
  - 모델의 **변경내역**을 Database Schema (데이터베이스 데이터 구조)로 반영시키는 효율적인 방법을 제공
- 관련 명령

셸> python manage.py makemigrations <app-name> # 마이그레이션 파일 생성

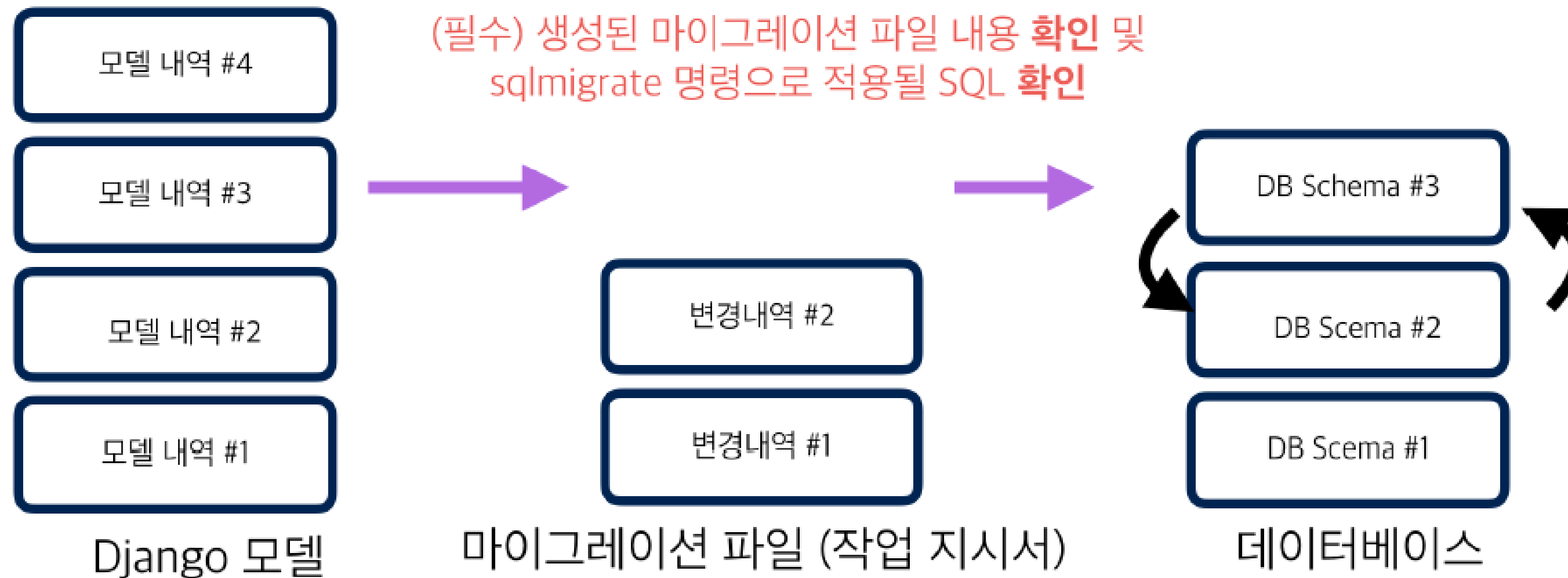
셸> python manage.py migrate <app-name> # 마이그레이션 적용

셸> python manage.py showmigrations <app-name> # 마이그레이션 적용 현황

셸> python manage.py sqlmigrate <app-name> <migration-name> # 지정 마이그레이션의 SQL 내역

# migration 파일 생성 및 적용

1. 마이그레이션 파일 (초안) 생성하기 : `makemigrations` 명령
2. 해당 마이그레이션 파일을 DB에 반영하기 : `migrate` 명령



# Migrate (Forward/Backward)

---

셸> `python manage.py migrate <app-name>`

미적용 <마이그레이션 파일> 부터 <최근 마이그레이션 파일> 까지  
"Forward 마이그레이션" 이 순차적으로 수행

셸> `python manage.py migrate <app-name> <마이그레이션 파일명>`

<지정된 마이그레이션 파일> 이 <현재 적용된 마이그레이션> 보다  
이후이면, "Forward 마이그레이션" 이 순차적으로 수행  
이전이면, "Backward 마이그레이션" 이 순차적으로 수행 (**롤백**)

# Migrate (Forward/Backward)

---

전체 파일명을 지정 하지 않더라도, 유일한 1개의 파일명을 판독 가능하면,  
파일명 일부로도 지정 가능

```
blog/migrations/0001_initial.py
```

```
blog/migrations/0002_create_field.py
```

```
blog/migrations/0002_update_field.py
```

```
python manage.py migrate blog 000 # FAIL (다수 파일에 해당)
```

```
python manage.py migrate blog 100 # FAIL (해당되는 파일이 없음)
```

```
python manage.py migrate blog 0001 # OK
```

```
python manage.py migrate blog 0002 # FAIL (다수 파일에 해당)
```

```
python manage.py migrate blog 0002_c # OK
```

```
python manage.py migrate blog 0002_create # OK
```

```
python manage.py migrate blog 0002_update # OK
```

```
python manage.py migrate blog zero # blog앱의 모든 마이그레이션을 취소
```

# 필수 입력 필드를 추가

- 기존 모델 클래스에 필수 필드를 추가하여 makemigrations 수행
- 필수 입력 필드를 추가하므로, 기존 Row들에 필드를 추가할 때, 어떤 값으로 채워 넣을 지 묻습니다.

선택1) 지금 값을 입력

선택2) 모델 클래스를 수정하여 디폴트 값을 제공

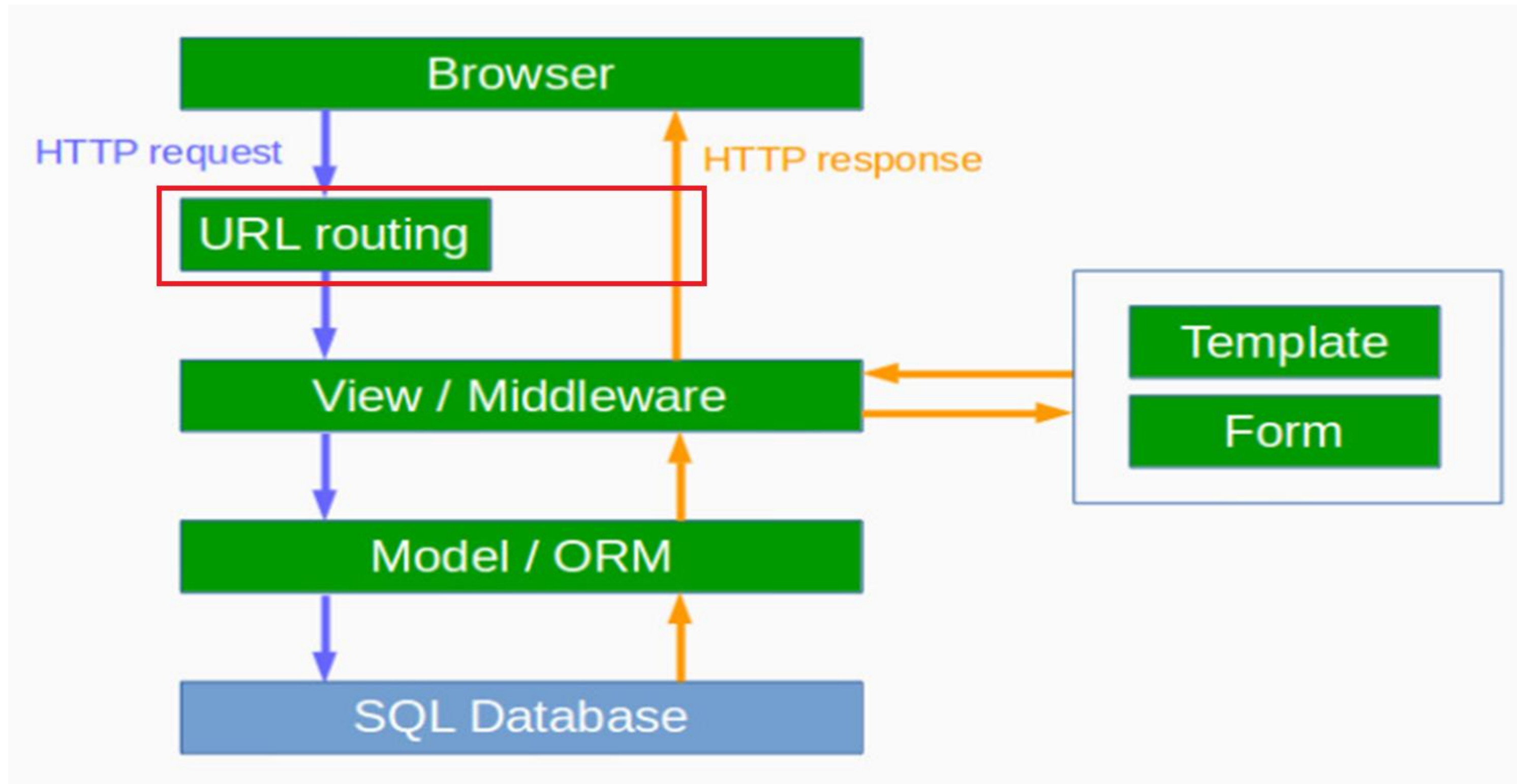
```
$ python manage.py makemigrations blog
You are trying to add a non-nullable field 'author' to post without a default; we can't do that (
atabase needs something to populate existing rows).
Please select a fix:
  1) Provide a one-off default now (will be set on all existing rows)
  2) Quit, and let me add a default in models.py
Select an option: 1
Please enter the default value now, as valid Python
The datetime and django.utils.timezone modules are available, so you can do e.g. timezone.now()
>>> 'anonymous'
Migrations for 'blog':
  0002_post_author.py:
    - Add field author to post
```

# Blog App 작성

: URL Routing

# Django Architecture : URL Routing

---



# Django URLConf (configuration)란?

---

- 1) 프로젝트/settings.py 에 최상위 URLConf 모듈을 지정
- URLConf는 장고에서 URL과 일치하는 view를 찾기 위한 패턴들의 집합이다.
  - 특정 URL과 View 매핑 List
  - Django 서버로 Http 요청이 들어올 때마다, URLConf 매핑 List를 처음부터 끝까지 순차적으로 찾으며 검색합니다.

mydjango/urls.py

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'', include('blog.urls')),
]
```



# 정규 표현식(Regex)

---

## 1) 정규 표현식이란?

문자열의 패턴, 규칙, Rule을 정의하는 방법

## 2) 파이썬3 정규 표현식 라이브러리

<https://docs.python.org/3/library/re.html>

## 3) 정규 표현식 예시

- 최대 3자리 숫자 : "[0-9]{1,3}" 혹은 "\d{1,3}"
- 휴대폰번호 : "010[1-9]\d{7}"
- 한글이름 2글자 혹은 3글자 : "[ㄱ-힣]{2,3}"
- 성이 "이" 인 이름 : "이[ㄱ-힣]{1,2}"

# 정규 표현식(Regex)

---

## 4) 다양한 1글자 패턴

- 숫자 1글자 : "[0123456789]" 또는 "[0-9]" 또는 "\d"
- 알파벳 소문자 1글자 : "[abcdefghijklmnopqrstuvwxyz]" 혹은 "[a-z]"
- 알파벳 대문자 1글자 : "[ABCDEFGHIJKLMNOPQRSTUVWXYZ]" 혹은 "[A-Z]"
- 알파벳 대/소문자 1글자 : "[a-zA-Z]"
- 16진수 1글자 : "[0-9a-fA-F]"
- 문자열의 시작을 지정 : "^"
- 문자열의 끝을 지정 : "\$"
- 한글 1글자 : "[ㄱ-힣]"

# 정규 표현식(Regex)

---

## 5) 반복횟수 지정

- `"\d?"` : 숫자 0회 또는 1회
- `"\d*"` : 숫자 0회 이상
- `"\d+"` : 숫자 1회 이상
- `"\d{m}"` : 숫자 m글자
- `"\d{m,n}"` : 숫자 m글자 이상, n글자 이하
- 주의 : 정규 표현식은 띄워 쓰기 하나에도 민감합니다.

# 정규표현식(Regex)

---

## 6) 휴대폰 번호 example

```
import re

def validate_phone_number(number):
    if not re.match(r'^01[016789][1-9]\d{6,7}$', number):
        return False
    # 후에 Form Validator에서는 forms.ValidationError예외를 발생시킴
    return True

print(validate_phone_number('01012341234')) # True
print(validate_phone_number('010123412')) # False
print(validate_phone_number('01012341234a')) # False
```

# 정규표현식(Regex)

---

## 7) url 매핑 example

http://www.mysite.com/post/12345/라는 요청이 있을 때 12345는 글번호 정규표현식으로 url 패턴을 만들어 숫자 값과 매칭되게 할 수 있음

`^post/(\d+)/$.`

`^post/` : url이(오른쪽부터) `post/`로 시작합니다.

`(\d+)` : 숫자(한 개 이상)가 있습니다.

`/` : `/`뒤에 문자가 있습니다.

`$` : url 마지막이 `/`로 끝납니다.

# Django URLConf (configuration) 설정하기

---

## 1) main urls.py

: admin/로 시작하는 모든 URL을 view와 매핑하여 찾아냅니다.

: http://127.0.0.1:8000/' 요청이 오면 views.post\_list를 보여준다.

mydjango/urls.py

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

urlpatterns = [
    path('admin/', admin.site.urls),
    url(r'^$', views.post_list),
]
```

# Django URLConf (configuration) 설정하기

---

2) blog/urls.py

: blog/urls.py를 작성하여 blog와 관련된 url들을 따로 정의함

**mydjango/urls.py**

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('blog.urls')),
]
```

**blog/urls.py**

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.post_list, name='post_list'),
]
```

# Django URLConf (configuration) 설정하기

---

2) blog/urls.py

: 'http://127.0.0.1:8000/' 요청이 오면 views.post\_list를 보여준다.

blog/urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.post_list, name='post_list'),
]
```

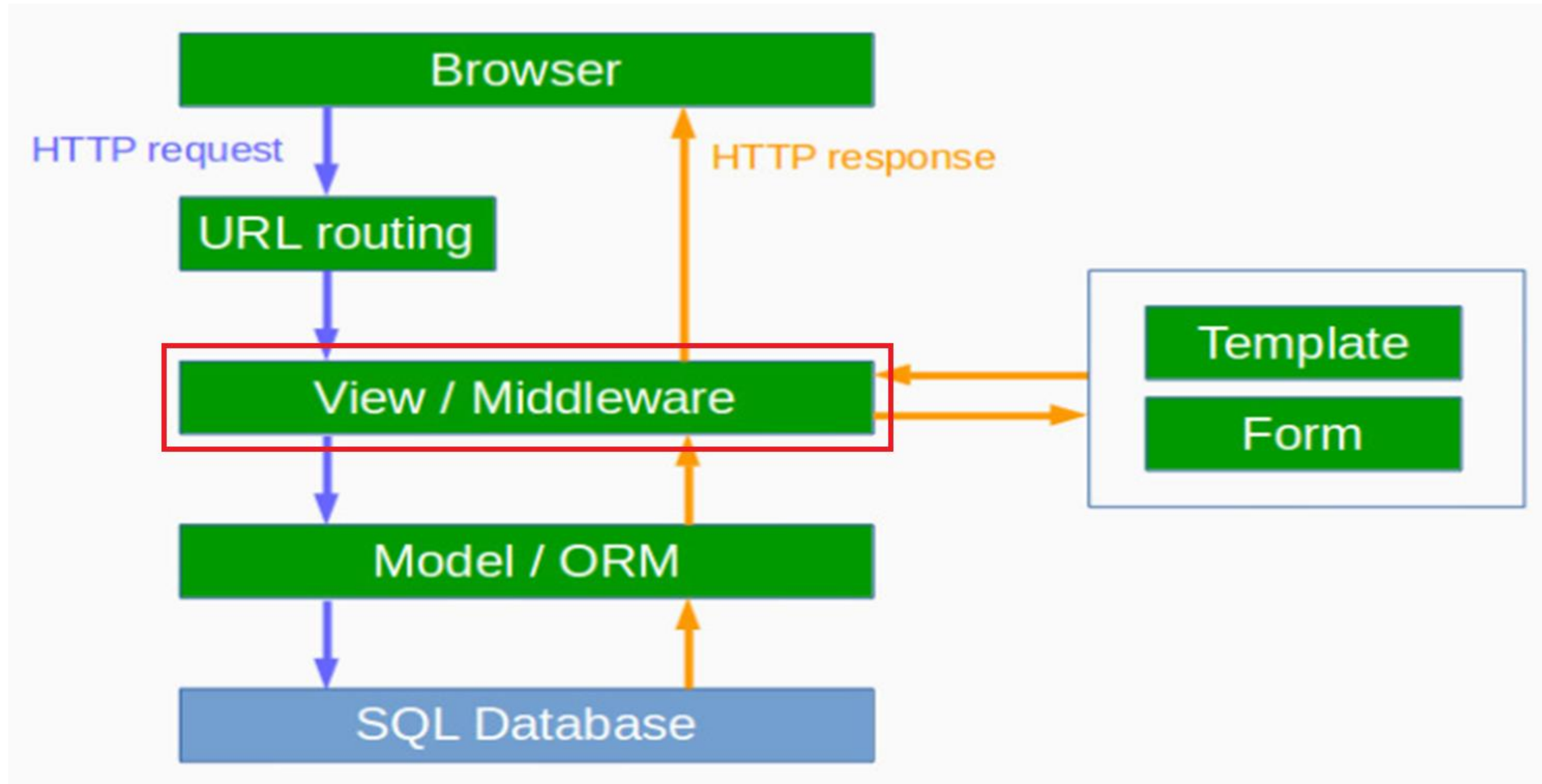


# blog App 작성

: View

# Django Architecture : View

---



# View

---

## 1) View의 역할

- View는 애플리케이션의 "로직"을 포함하며 모델에서 필요한 정보를 받아 와서 템플릿에 전달하는 역할을 한다.
- View는 Model과 Template을 연결하는 역할을 한다.
- URLConf에 매핑된 Callable Object  
첫번째 인자로 HttpRequest 인스턴스를 받습니다.  
반드시 HttpResponse 인스턴스를 리턴 해야 합니다.

<https://docs.djangoproject.com/en/1.11/ref/request-response/>

# View

---

## 2) blog/views.py

: post\_list라는 함수(def) 만들어 요청(request)을 받아서 직접 문자열로 HTML형식 응답(response)하기

blog/views.py

```
from django.http import HttpResponse

def post_list(request):
    name = '장고'
    return HttpResponse('' <h1>Hello Django</h1>
                        <p>{name}</p>'' .format(name=name))
```

# View

---

## 3) blog/views.py 수정

: post\_list라는 함수(def) 만들어 요청(request)을 넘겨받아 render 메서드를 호출합니다.

: 함수는 호출하여 받은(return) blog/post\_list.html 템플릿을 보여줍니다.

blog/views.py

```
from django.shortcuts import render

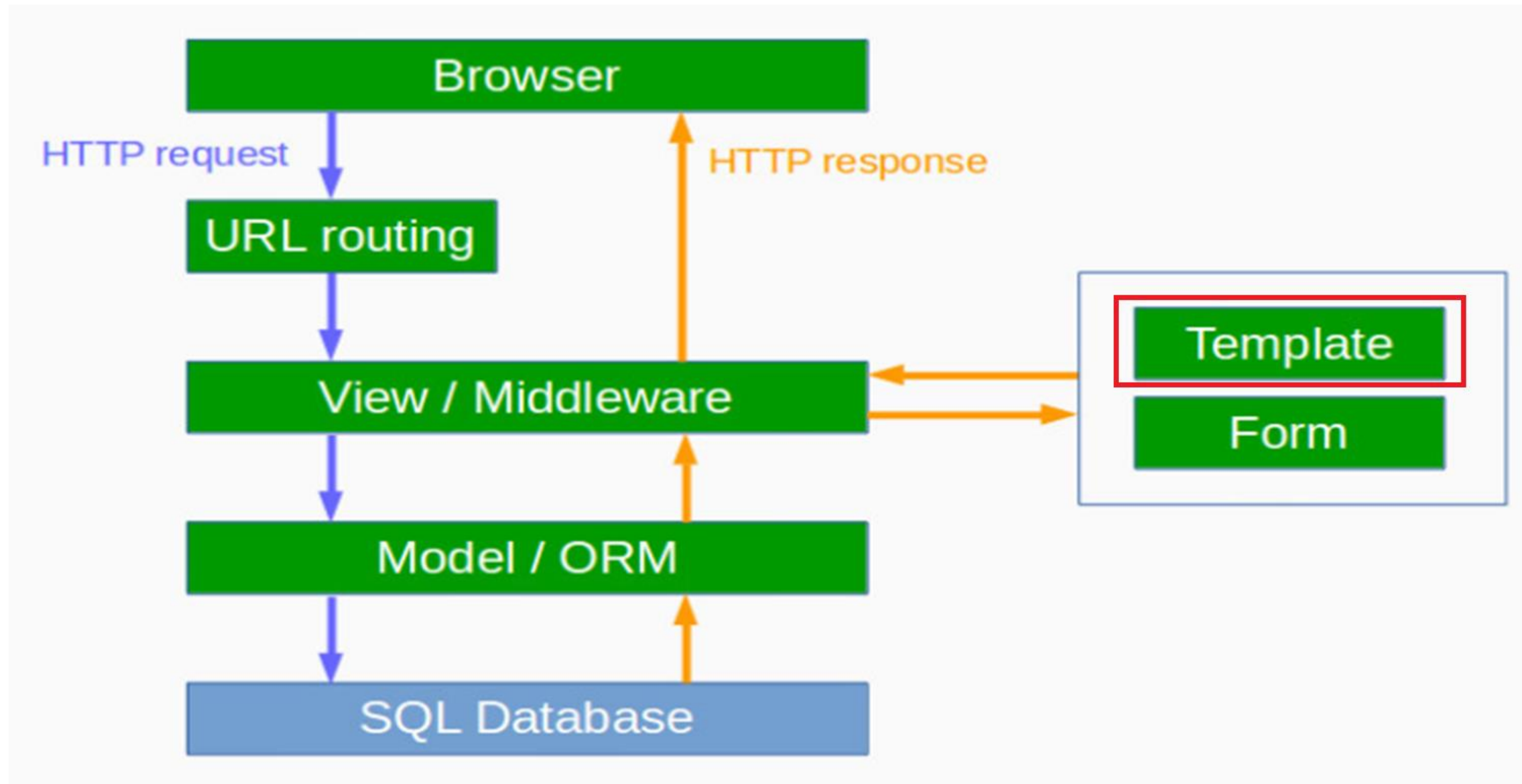
def post_list(request):
    return render(request, 'blog/post_list.html')
```

# blog App 작성

## : Template

# Django Architecture : Template

---



# Template

## 1) Template의 역할

- Template은 정보를 일정한 형태로 표시하기 위해 재사용 가능한 파일을 말함
- Django의 template 양식은 HTML을 사용합니다.
- 템플릿은 `blog/templates/blog` 디렉토리에 저장합니다.

`blog/templates/blog/post_list.html`

```
<html>
  <head>
    <title>Django blog</title>
  </head>
  <body>
    <p>Hi there!</p>
    <p>It works!</p>
  </body>
</html>
```

```
▲ DJANGO_SRC
  ▲ blog
    ▸ __pycache__
    ▸ migrations
    ▲ templates
      ▲ blog
        <> post_list.html
      __init__.py
      admin.py
      apps.py
      models.py
      tests.py
      urls.py
      views.py
```



# Template

## 2) post\_list.html 템플릿 수정

blog/templates/blog/post\_list.html

```
<html>
  <head>
    <title>Django blog</title>
  </head>
  <body>
    <div>
      <h1><a href="">Django's Blog</a></h1>
    </div>

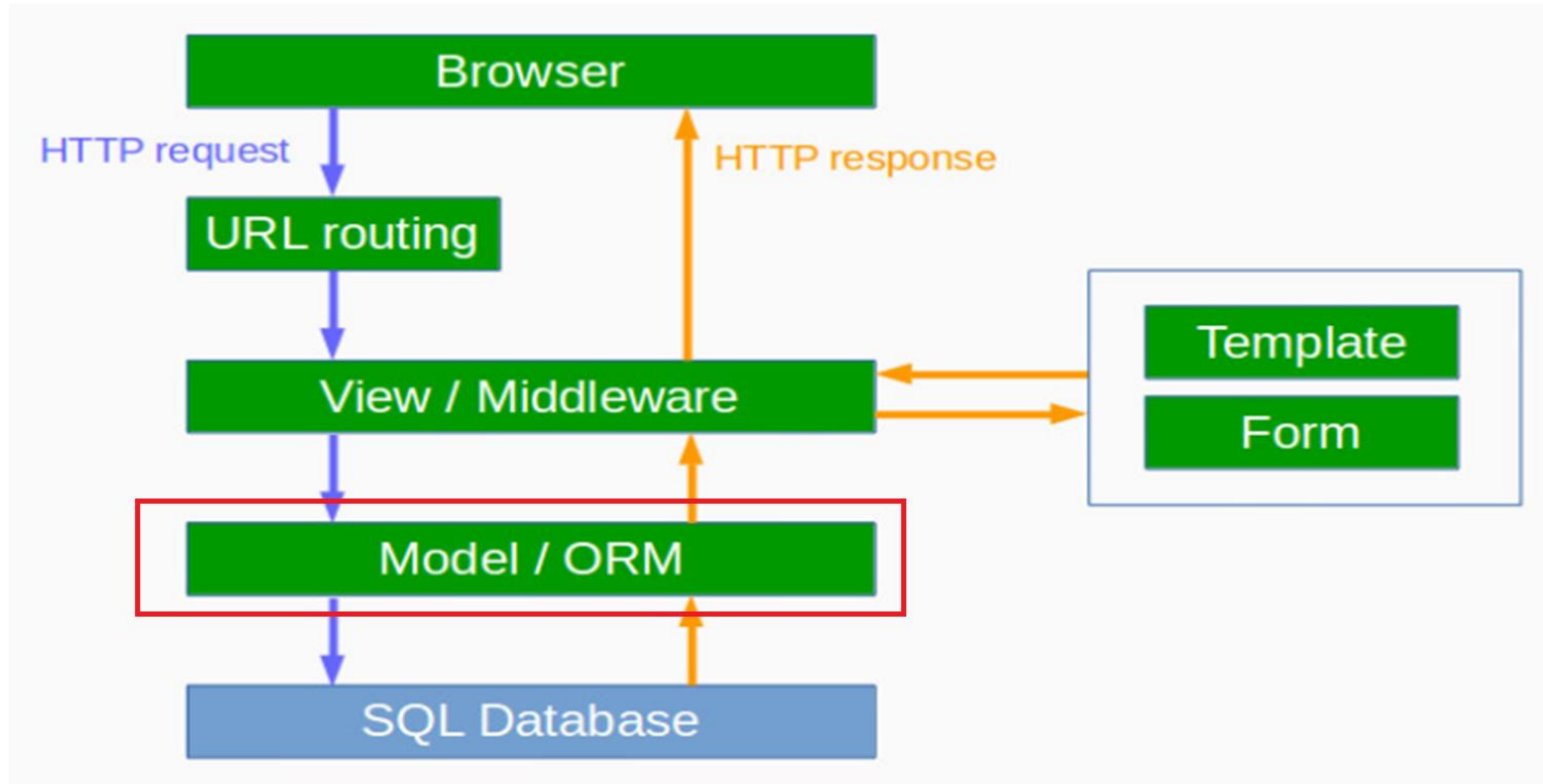
    <div>
      <p>published: 14.06.2014, 12:14</p>
      <h2><a href="">My first post</a></h2>
      <p>Aenean eu leo quam. Pellentesque ornare sem lacinia quam venenatis vestibulum. Donec id elit non mi porta
gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit
amet risus.</p>
    </div>

    <div>
      <p>published: 14.06.2014, 12:14</p>
      <h2><a href="">My second post</a></h2>
      <p>Aenean eu leo quam. Pellentesque ornare sem lacinia quam venenatis vestibulum. Donec id elit non mi porta
gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut f.</p>
    </div>
  </body>
</html>
```

# blog App 작성

: ORM과 쿼리셋(QuerySets)

# Django의 Architecture : ORM과 Queryset



# QuerySet

---

## 1) QuerySet이란?

- 쿼리셋(QuerySet)은 DB로부터 데이터를 읽고, 필터링을 하거나, 정렬을 할 수 있습니다.
- 쿼리셋을 사용하기 위해 먼저 `python shell`을 실행한다.

# 인터랙티브 콘솔(Interactive Console) 실행

Django\_src> `python manage.py shell`

```
(InteractiveConsole)
```

```
>>>
```

# QuerySet

---

## 1) 모든 객체 조회하기

모든 객체를 조회하기 위해 `all()` 함수를 사용합니다.

```
>>> Post.objects.all()
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'Post' is not defined
```

```
>>> from blog.models import Post
```

```
>>> Post.objects.all()
<QuerySet [<Post: my post title>, <Post: another post title>]>
```

# QuerySet

---

## 2) 객체 생성하기

객체를 저장하기 위해 `create()` 함수를 사용합니다.

작성자(author)로서 `User`(사용자) 모델의 인스턴스를 가져와 전달 해줘야 합니다.

```
>>> from django.contrib.auth.models import User
```

```
>>> User.objects.all()  
<QuerySet [<User: ola>]>
```

```
>>> me = User.objects.get(username='ola')
```

```
>>> Post.objects.create(author=me, title='Sample title', text='Test')
```

```
>>> Post.objects.all()  
<QuerySet [<Post: my post title>, <Post: another post title>]>
```

# QuerySet

---

## 3-1) 필터링 하기

원하는 조건으로 데이터를 필터링 한다. `filter()` 괄호 안에 원하는 조건을 넣어 주면 됩니다.

# 특정 사용자가 작성한 글을 찾고자 할 때

```
>>> Post.objects.filter(author=me)
[<Post: Sample title>, <Post: Post number 2>, <Post: My 3rd post!>, <Post: 4th title of post>]
```

# 글의 제목(title)에 'title' 이라는 글자가 들어간 글을 찾고자 할 때

```
>>> Post.objects.filter(title__contains='title')
[<Post: Sample title>, <Post: 4th title of post>]
```

# 게시일(published\_date)로 과거에 작성한 글을 필터링하여 목록을 가져올 때

```
>>> from django.utils import timezone
>>> Post.objects.filter(published_date__lte=timezone.now())
[]
```

# QuerySet

---

## 3-2) 필터링 하기

# 게시(publish)하려는 Post의 인스턴스를 가져온다.

```
>>> post = Post.objects.get(title="Sample title")
```

# 가져온 Post 인스턴스를 publish() 메서드를 이용하여 게시한다.

```
>>> post.publish()
```

# 게시일(published\_date)로 과거에 작성한 글을 필터링하여 목록을 다시 가져온다.

```
>>> Post.objects.filter(published_date__lte=timezone.now())  
[<Post: Sample title>]
```



# QuerySet

---

## 4) 정렬 하기

# 작성일(created\_date) 기준으로 오름차순으로 정렬하기

```
>>> Post.objects.order_by('created_date')
[<Post: Sample title>, <Post: Post number 2>, <Post: My 3rd post!>, <Post: 4th title of post>]
```

# 작성일(created\_date) 기준으로 내림차순으로 정렬하기 : - 을 붙이면 내림차순 정렬

```
>>> Post.objects.order_by('-created_date')
[<Post: 4th title of post>, <Post: My 3rd post!>, <Post: Post number 2>, <Post: Sample title>]
```

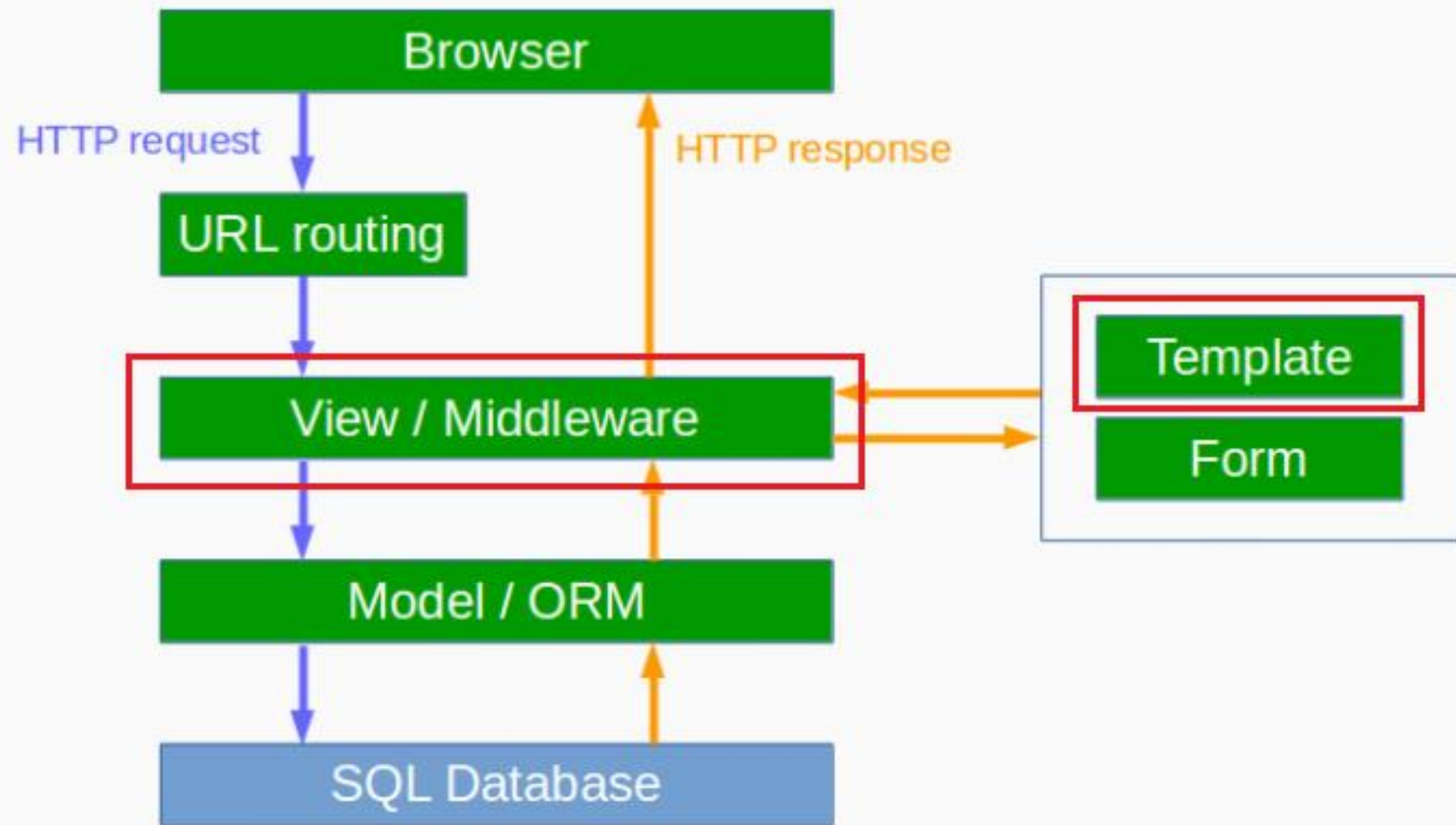
# 쿼리셋들을 함께 연결(chaining) 하기

```
>>> Post.objects.filter(published_date__lte=timezone.now()).order_by('published_date')
```

# blog App 작성

: 템플릿에서 동적 데이터 처리

# Django의 Architecture : 동적(dynamic) 데이터 처리하기



# View에서 동적(dynamic) 데이터 생성하기

## 1) View의 수정

- View는 DB에 저장되는 Model에서 정보를 가져올 때 쿼리셋(QuerySet)을 사용하며, 템플릿에 전달하는 역할을 한다.

# 게시일(published\_date) 기준으로 과거에 작성한 글을 필터링하여 정렬하여 글 목록 가져오기

blog/views.py

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

def post_list(request):
    posts = Post.objects.filter(published_date__lte=timezone.now()).\
        order_by('published_date')
    return render(request, 'blog/post_list.html', {'posts': posts})
```

# Template에서 동적(dynamic) 데이터 사용하기

## 2-1) Template의 수정

- Template에서는 view에서 저장한 posts 변수를 받아와서 HTML에 출력한다.
- 변수의 값을 출력하려면 중괄호를 사용한다.
- {% for %} 와 {% endfor %} 사이에서 목록의 모든 객체를 반복하여 출력함.

blog/templates/blog/post\_list.html

```
<div>
  <h1><a href="/">Django's Blog</a></h1>
</div>

{% for post in posts %}
  {{ post }}
{% endfor %}
```

# Template에서 동적 데이터 사용하기

## 2-2) Template의 수정

- `|linebreaksbr` 같이 파이프 문자(`|`)를 사용하여, 블로그 글 텍스트에서 행이 바뀌면 문단으로 변환하여 출력한다.

blog/templates/blog/post\_list.html

```
<div>
    <h1><a href="/">Django's Blog</a></h1>
</div>

{% for post in posts %}
    <div>
        <p>published: {{ post.published_date }}</p>
        <h1><a href="">{{ post.title }}</a></h1>
        <p>{{ post.text|linebreaksbr }}</p>
    </div>
{% endfor %}
```

# Template Engine

---

- Django Template Engine : Django 기본 지원 템플릿 엔진
- Django Template Engine Syntax

```
{% extends "base.html" %}
```

```
{% for row in rows %}
```

```
    <tr>
```

```
        {% for name in row %}
```

```
            <td>{{ name }}</td>
```

```
        {% endfor %}
```

```
    </tr>
```

```
{% endfor %}
```

# Template Engine 문법

---

## 1) Variables

- `{{ first_name }}`
- `{{ mydict.key }}` : dict의 key에 attr 처럼 접근
- `{{ myobj.attr }}`
- `{{ myobj.func }}` : 함수 호출도 attr 처럼 접근. 인자 있는 함수 호출 불가
- `{{ mylist.0 }}` : 인덱스 접근도 attr 처럼 접근

## 2) Django Template Tag

- `{% %}` 1개 쓰이기도 하며, 2개 이상이 조합되기도 함.
- 빌트인 Tag가 지원되며, 장고앱 별로 커스텀 Tag 추가 가능

block, comment, csrf\_token, extends, for, for ... empty, if, ifchanged, include, load, lorem, now, url, verbatim, with 등



# Template Engine 문법

---

## 3) block tag

- 템플릿 상속에서 사용
- 자식 템플릿이 오버라이딩 할 block 영역을 정의
- 자식 템플릿은 부모가 정의한 block에 한해서 재 정의만 가능. 그 외는 모두 무시됩니다.

```
{% block block-name %}
```

block 내에 내용을 쓰실 수 있습니다.

```
{% endblock %}
```

## 4) Comment Tag : 템플릿 주석

```
{% comment "Optional note" %}
```

이 부분은 렌더링 되지 않습니다.

```
{% endcomment %}
```

# Template Engine 문법

---

## 5) csrf\_token tag

- Cross Site Request Forgeries를 막기 위해 CSRF Middleware가 제공
- 이는 HTML Form의 POST요청에서 CSRF토큰을 체크하며, 이때 CSRF토큰이 필요
- csrf\_token tag를 통해 CSRF토큰을 발급받을 수 있습니다.

```
<form method="POST" action="">  
    {% csrf_token %}  
    <input type="text" name="author" />  
    <textarea name="message"></textarea>  
    <input type="submit" />  
</form>
```

# Template Engine 문법

---

## 6) extends tag

- 자식 템플릿에서 부모 템플릿 상속을 명시
- extends tag는 항상 템플릿의 처음에 위치해야 합니다.
- 상속받은 자식 템플릿은 부모 템플릿에서 정의한 block만 재정의할 수 있습니다.

```
{% extends "base.html" %}
```

## 7) for tag

- 지정 객체를 순회하며 파이썬의 for문과 동일

```
{% for athlete in athlete_list %}
```

```
    <li>{{ athlete.name }}</li>
```

```
{% endfor %}
```

# Template Engine 문법

---

## 6) extends tag

- 자식 템플릿에서 부모 템플릿 상속을 명시
- extends tag는 항상 템플릿의 처음에 위치해야 합니다.
- 상속받은 자식 템플릿은 부모 템플릿에서 정의한 block만 재정의할 수 있습니다.

```
{% extends "base.html" %}
```

## 7) for tag

- 지정 객체를 순회하며 파이썬의 for문과 동일

```
{% for athlete in athlete_list %}
```

```
    <li>{{ athlete.name }}</li>
```

```
{% endfor %}
```

# Template Engine 문법

---

7) for ... empty tag

- for tag 내에서 지정 object를 찾을 수 없거나, 비었을 때 empty block이 수행

```
{% for athlete in athlete_list %}
```

```
    <li>{{ athlete.name }}</li>
```

```
{% empty %}
```

```
    <li>Sorry, no athletes in this list.</li>
```

```
{% endfor %}
```

# Template Engine 문법

---

8) if tag

- 파이썬의 if문과 동일

```
{% if athlete_list %}
```

```
    Number of athletes: {{ athlete_list|length }}
```

```
{% elif athlete_in_locker_room_list %}
```

```
    Athletes should be out of the locker room soon!
```

```
{% else %}
```

```
    No athletes.
```

```
{% endif %}
```

# Template Engine 문법

---

## 9) url tag

- URL Reverse를 수행한 URL문자열을 출력
- 인자 처리는 `django.shortcuts.resolve_url` 함수와 유사하게 처리하나, **`get_absolute_url`** 처리는 하지 않음.

```
{% url "some-url-name-1" %}
```

```
{% url "some-url-name-2" arg arg2 %}
```

```
{% url "some-url-name-2" arg arg2 as the_url %}
```

Django Templates Tag Documentation

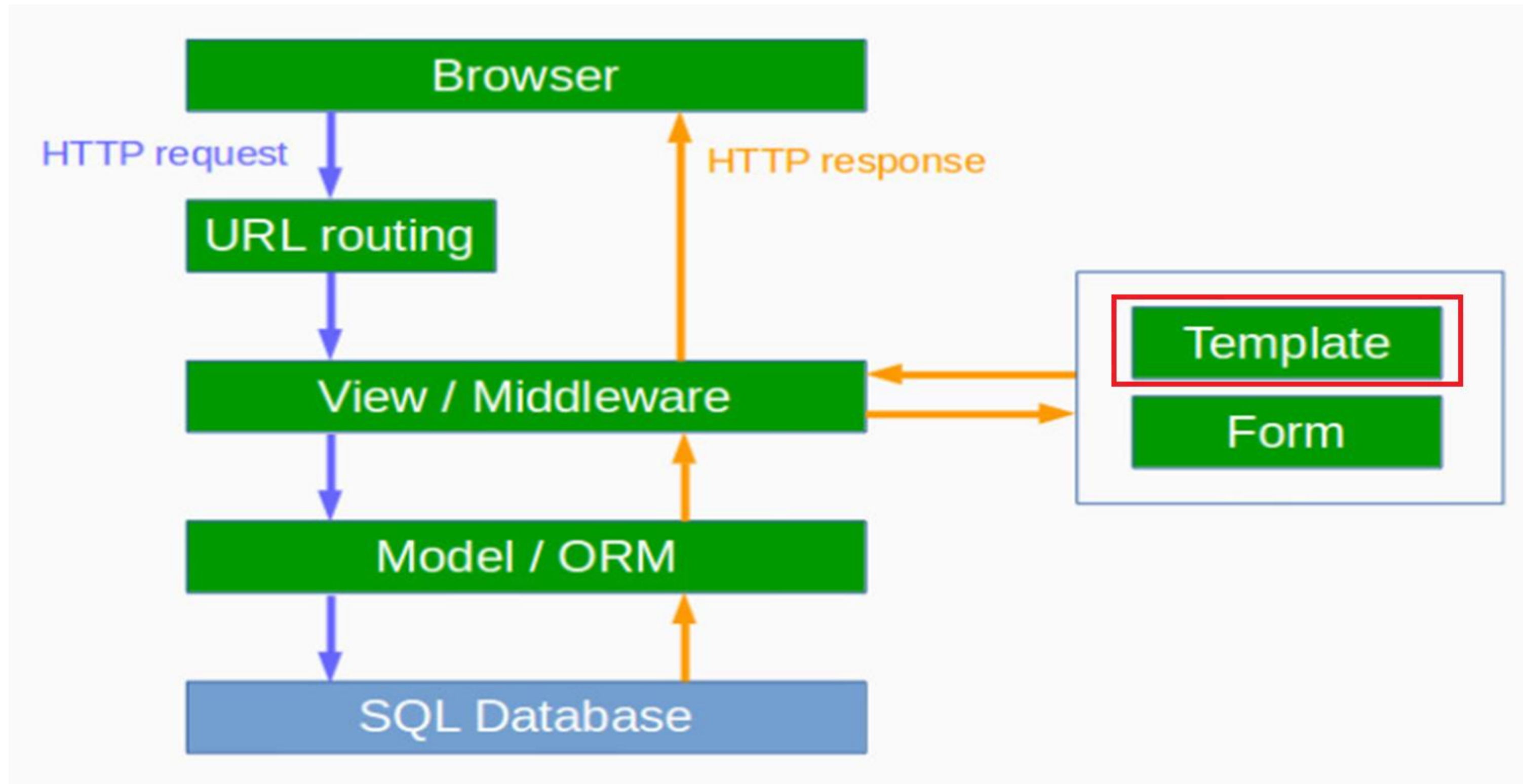
<https://docs.djangoproject.com/en/3.0/ref/templates/builtins/>

# blog App 작성

: 템플릿에 CSS 적용하기



# Django의 Architecture : Template에 css 적용



# 정적(static) 파일 처리하기

## 1) CSS 파일 작성

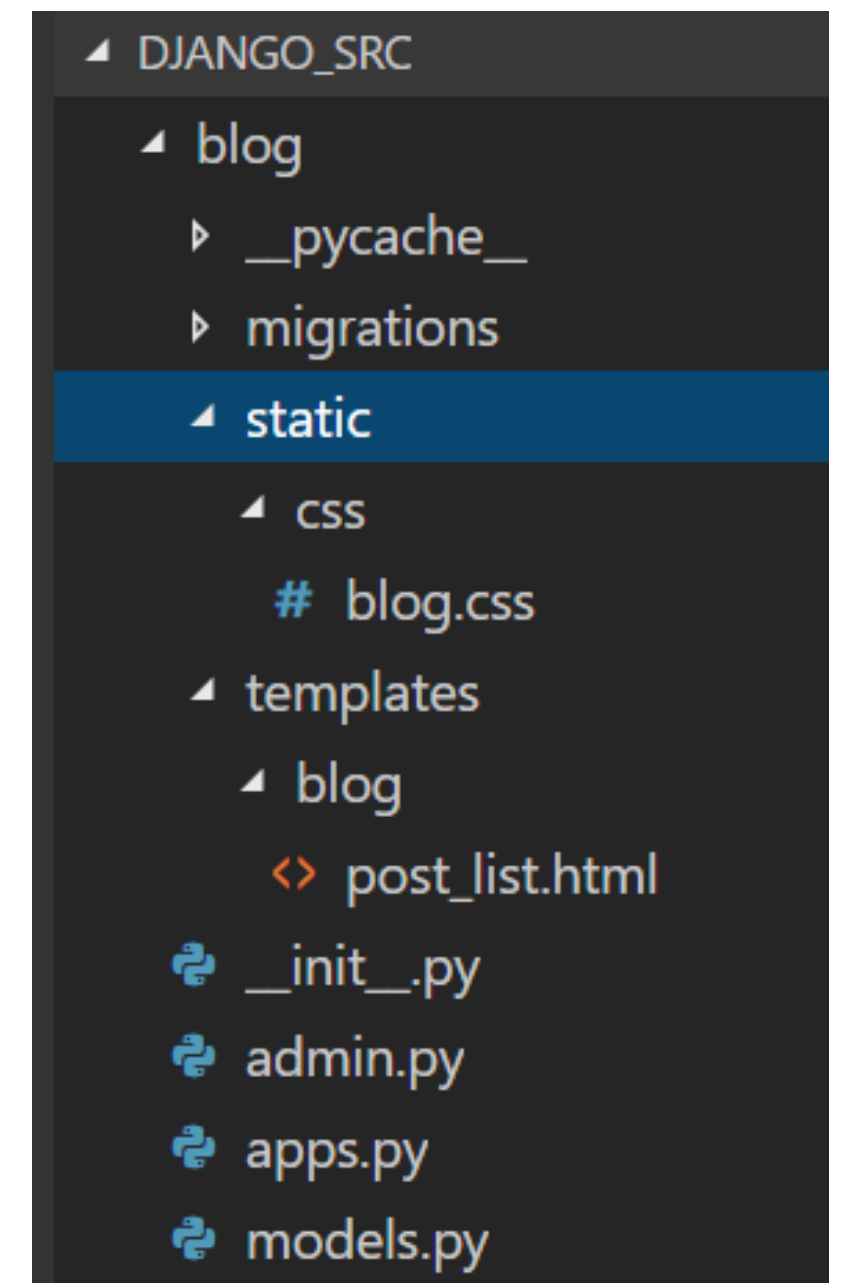
- static 디렉토리 안에 css 디렉토리를 만들고 blog.css라는 파일을 작성.

blog/static/css/blog.css

```
h1 a {
    color: #FCA205;
    font-family: 'Lobster';
}
body { padding-left: 15px;}
```

blog/templates/blog/post\_list.html

```
{% load static %}
<html>
  <head>
    <title>Django's blog</title>
    <link rel="stylesheet" href="{% static 'css/blog.css' %}">
  </head>
```



# 정적(static) 파일 처리하기

## 2) 부트스트랩(Bootstrap) 적용하기

- 부트스트랩을 설치하려면 .html 파일 내 <head>에 아래의 링크를 넣어야 합니다.

blog/templates/blog/post\_list.html

```
{% load static %}
<html>
  <head>
    <title>Django's blog</title>
    <link rel="stylesheet"
href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
    <link rel="stylesheet"
href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-
theme.min.css">
    <link rel="stylesheet"
href="//fonts.googleapis.com/css?family=Lobster&subset=latin,latin-ext"
type="text/css">
    <link rel="stylesheet" href="{% static 'css/blog.css' %}">
  </head>
```

# 정적(static) 파일 처리하기

## 3) 완성된 blog.css

### blog/static/css/blog.css

```
.page-header {
  background-color: #ff9400;
  margin-top: 0;
  padding: 20px 20px 20px 40px;
}
.page-header h1, .page-header h1 a, .page-header h1 a:visited, .page-header h1 a:active {
  color: #ffffff;
  font-size: 36pt;
  text-decoration: none;
}
.content { margin-left: 40px; }
h1, h2, h3, h4 { font-family: 'Lobster', cursive; }
.date { color: #828282; }
.save { float: right; }
.post-form textarea, .post-form input { width: 100%; }
.top-menu, .top-menu:hover, .top-menu:visited {
  color: #ffffff;
  float: right;
  font-size: 26pt;
  margin-right: 20px;
}
.post { margin-bottom: 70px; }
.post h1 a, .post h1 a:visited { color: #000000; }
```

# 정적(static) 파일 처리하기

## 4) 완성된 post\_list.html

blog/templates/blog/post\_list.html

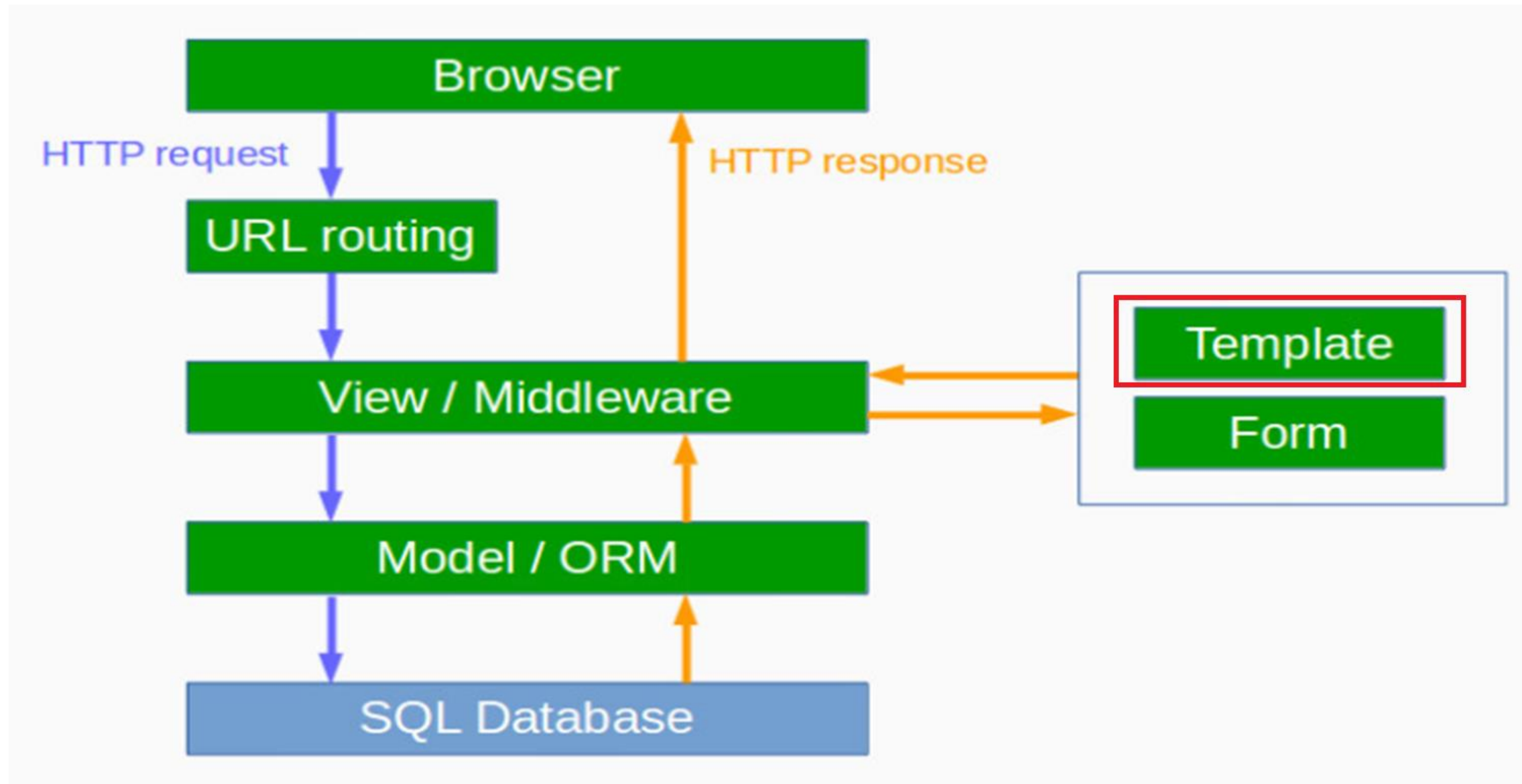
```
<body>
<div class="page-header">
  <h1><a href="/">Django Girls Blog</a></h1>
</div>
<div class="content container">
  <div class="row">
    <div class="col-md-8">
      {% for post in posts %}
        <div class="post">
          <div class="date">
            <p>published: {{ post.published_date }}</p>
          </div>
          <h1><a href="">{{ post.title }}</a></h1>
          <p>{{ post.text|linebreaksbr }}</p>
        </div>
      {% endfor %}
    </div>
  </div>
</div>
</body>
</html>
```

# blog App 작성

: 템플릿 상속하기(Template Inheritance)

# Django Architecture : Template Inheritance

---



# Template 상속(Inheritance) 하기

---

## 1) Template 상속(Inheritance)이란?

- Template 상속을 사용하면 동일한 정보/레이아웃을 사용 하고자 할 때, 모든 파일마다 같은 내용을 반복해서 입력 할 필요가 없게 됩니다.
- 또한 수정할 부분이 생겼을 때도, 각각 모든 파일을 수정 할 필요 없이 한번만 수정 하면 됩니다.

## 2) 기본 템플릿 html 생성하기

- 기본 템플릿은 웹사이트 내 모든 페이지에 확장되어 사용되는 가장 기본적인 템플릿입니다.
- `blog/templates/blog/`에 `base.html` 파일을 생성한다.
- `post_list.html`에 있는 모든 내용을 `base.html`에 아래 내용을 복사해 붙여 넣는다.



# 기본 템플릿 작성하기

## 3) 기본 템플릿 (base.html)

{% for post in posts %}{% endfor %} 사이에 있는 코드를 제거하고  
{% block content %}{% endblock %} 으로 변경한다.

blog/templates/blog/base.html

```
<body>
  <div class="page-header">
    <h1><a href="/">Django Girls Blog</a></h1>
  </div>
  <div class="content container">
    <div class="row">
      <div class="col-md-8">
        {% block content %}
        {% endblock %}
      </div>
    </div>
  </div>
</body>
```

# 기본 템플릿 작성하기

## 4) post\_list.html 수정하기

- ① {% block content %}와 {% endblock %} 사이에 {% for post in posts %}부터 {% endfor %} 코드를 넣는다.
- ② 두 템플릿을 연결하기 위해 {% extends 'blog/base.html' %} 코드를 추가한다.

blog/templates/blog/post\_list.html

```
{% extends 'blog/base.html' %}

{% block content %}
    {% for post in posts %}
        <div class="post">
            <div class="date">
                {{ post.published_date }}
            </div>
            <h1><a href="">{{ post.title }}</a></h1>
            <p>{{ post.text|linebreaksbr }}</p>
        </div>
    {% endfor %}
{% endblock %}
```

# blog App 작성

: Post Detail (글 상세) 페이지 작성하기

# Post Detail ( 글 상세) 페이지 작성하기

## 1) urls.py에 url 추가

- ① ^은 "시작"을 뜻합니다.
- ② post/란 URL이 post 문자를 포함해야 한다는 것을 말합니다
- ③ (?P<pk>\d+) 정규표현식은 장고가 pk 변수에 값을 넣어 view로 전송하겠다는 뜻입니다.  
\d은 숫자만 올 수 있다는 것을 말합니다. +는 하나 또는 그 이상의 숫자가 올 수 있습니다
- ④ /은 다음에 / 가 한 번 더 와야 한다는 의미입니다.
- ⑤ \$는 "마지막"을 말합니다. 그 뒤로 더는 문자가 오면 안 됩니다.

따라서 `http://127.0.0.1:8000/post/5/`라고 입력하면, `post_detail` view를 찾아 매개변수 `pk`가 5인 값을 찾아 view로 전달합니다.

blog/urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.post_list, name='post_list'),
    path('post/<int:pk>/', views.post_detail, name='post_detail'),
]
```

# Post Detail ( 글 상세) 페이지 작성하기

## 2) post\_list.html에 Post detail 페이지 링크 추가

- ① {% %}는 장고 템플릿 태그이며, post\_detail은 url에서 정의한 view name이다.
- ② post.pk는 Post 모델의 primary key(기본키)를 의미합니다.

blog/templates/blog/post\_list.html

```
{% extends 'blog/base.html' %}

{% block content %}
    {% for post in posts %}
        <div class="post">
            <div class="date">
                {{ post.published_date }}
            </div>
            <h1><a href="{% url 'post_detail' pk=post.pk %}">{{ post.title }}</a></h1>
            <p>{{ post.text|linebreaksbr }}</p>
        </div>
    {% endfor %}
{% endblock %}
```

# Post Detail (글 상세) 페이지 작성하기

## 3) views.py에 post\_detail() 함수 추가

- ① `def post_detail(request, pk):`라고 정의하며, `urls(pk)`과 동일하게 이름을 사용해야 합니다.
- ② 블로그 게시물 한 개만 보려면 `Post.objects.get(pk=pk)` 쿼리셋을 작성해야 하는데 만약 해당 primary key(pk)의 Post를 찾지 못하면 오류가 날 수 있으므로 Django에서는 이를 해결하기 위해 `get_object_or_404`라는 특별한 기능을 제공한다.

blog/views.py

```
from django.shortcuts import render, get_object_or_404
from .models import Post

def post_detail(request, pk):
    post = get_object_or_404(Post, pk=pk)
    return render(request, 'blog/post_detail.html', {'post': post})
```

# Post Detail ( 글 상세) 페이지 작성하기

## 4) post\_detail.html 페이지 추가

- ① base.html을 확장하고, content블록에서 블로그 글의 게시일, 제목과 내용을 보이게 한다.
- ② {% if ... %} ... {% endif %}라는 템플릿 태그에서는 post의 게시일(published\_date)이 있는지, 없는지를 확인합니다.

blog/templates/blog/post\_detail.html

```
{% extends 'blog/base.html' %}

{% block content %}
    <div class="post">
        {% if post.published_date %}
            <div class="date">
                {{ post.published_date }}
            </div>
        {% endif %}
        <h1>{{ post.title }}</h1>
        <p>{{ post.text|linebreaksbr }}</p>
    </div>
{% endblock %}
```

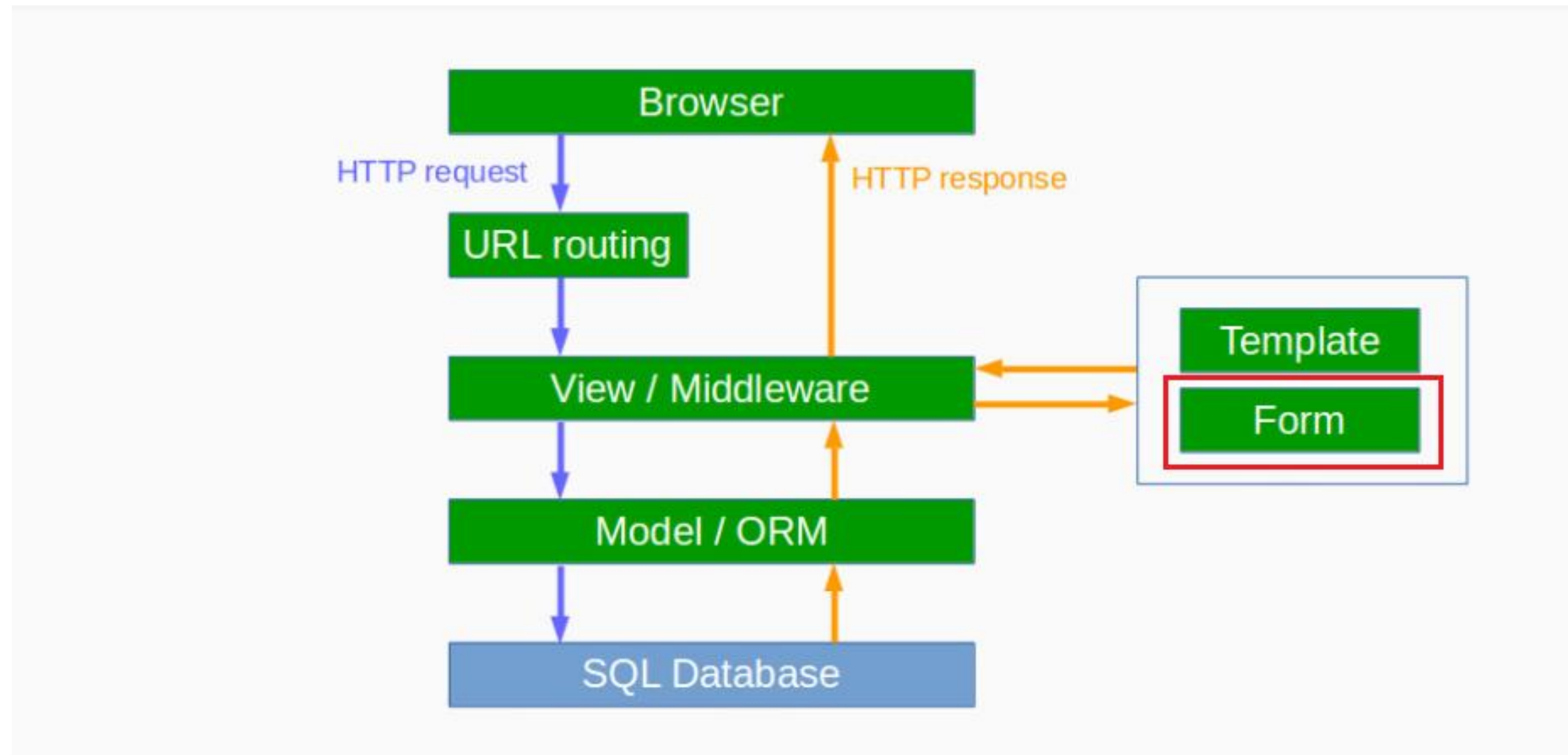
# blog App 작성

: Django Form (글 추가)



# Django의 Architecture : Form

---



# Django Form

---

## 1) Form이란?

- Model클래스와 유사하게 Form클래스를 정의
- 주요 역할 : 커스텀 Form클래스를 통해
  - ❖ 입력 폼 HTML 생성 : `.as_table()`, `.as_p()`, `.as_ul()` 기본제공
  - ❖ 입력 폼 값 검증(validation) 및 값 변환

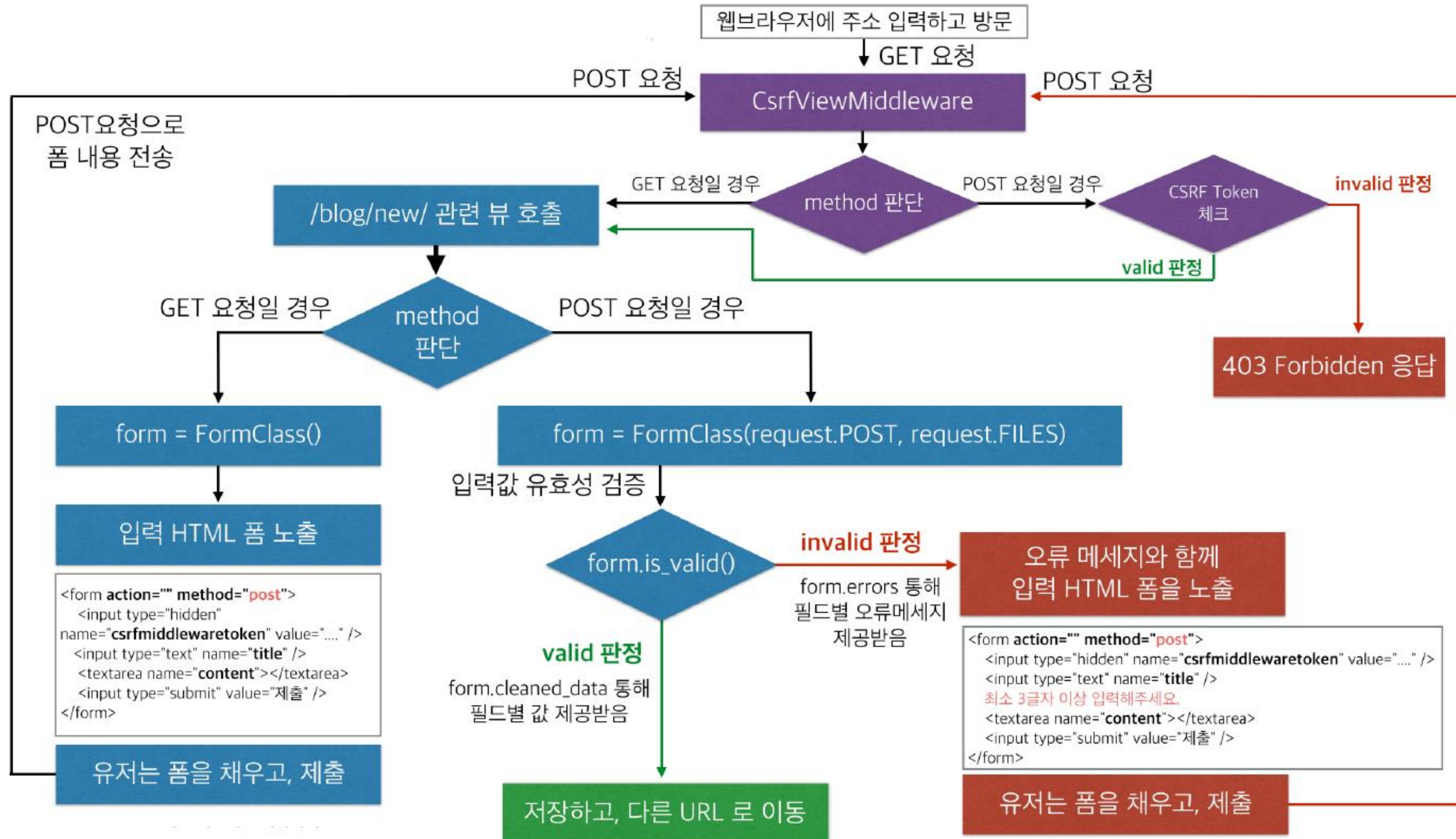
## 2) Form 처리 : HTTP Method에 따라

- 폼 처리 시에 같은 URL(즉 같은 뷰)에서 GET/POST로 나눠 처리
- GET방식으로 요청 : 입력 폼을 보여줍니다.
- POST방식으로 요청 : 데이터를 입력 받아 유효성 검증 과정을 거칩니다.

검증 성공 시 : 해당 데이터를 저장하고 SUCCESS URL로 이동

검증 실패 시 : 오류 메시지와 함께 입력 폼을 다시 보여줍니다.

# Django Form



# Django Form

---

1)step1: Form 클래스 정의

```
from django import forms

class PostForm(forms.Form):
    title = forms.CharField()
    text = forms.CharField(widget=forms.Textarea)
```

2)step2: 필드 유효성 검사 함수 추가 적용

```
from django import forms

def min_length_3_validator(value):
    if len(value) < 3:
        raise forms.ValidationError('3글자 이상 입력해주세요.')

from django import forms

class PostForm(forms.Form):
    title = forms.CharField(validators=[min_length_3_validator])
    text = forms.CharField(widget=form.Textarea)
```

# Django Form

---

3)step3: View 함수 내에서 Form 인스턴스 생성

GET요청을 통해 View 함수가 호출이 될 때, GET/POST 요청을 구분해서 Form 인스턴스 생성

```
# myapp/views.py
```

```
from .forms import PostForm
```

```
if request.method == 'POST':
```

```
    # POST 요청일 때
```

```
    form = PostForm(request.POST, request.FILES)
```

```
else:
```

```
    # GET 요청일 때
```

```
    form = PostForm()
```

# Django Form

---

4)step4: POST요청에 한해 입력 값 유효성 검증

```
# myapp/views.py
```

```
if request.method == 'POST':
```

```
    # POST인자는 request.POST와 request.FILES를 제공받음.
```

```
    form = PostForm(request.POST, request.FILES)
```

```
    # 인자로 받은 값에 대해서, 유효성 검증 수행
```

```
    if form.is_valid(): # 검증이 성공하면, True 리턴
```

```
        # 검증에 성공한 값들을 dict타입으로 제공받아서 이 값을 DB에 저장하기
```

```
        form.cleaned_data
```

```
        post = Post(**form.cleaned_data) # DB에 저장하기
```

```
        post.save()
```

```
        return redirect('/success_url/')
```

```
    else: # 검증에 실패하면, form.errors와 form.각필드.errors 에 오류정보를 저장
```

```
        form.errors
```

```
else:
```

```
    form = PostForm()
```

```
return render(request, 'myapp/form.html', {'form': form})
```

# Django Form

---

5)step5: 템플릿을 통해 HTML폼 생성

유효성 검증에서 실패했을 때 Form 인스턴스를 통해 HTML폼 출력하고,오류 메시지도 있다면 같이 출력

<table>

```
<form action="" method="post">
```

```
{% csrf_token %}
```

```
<table>{{ form.as_table }}</table>
```

```
<input type="submit" />
```

```
</form>
```

</table>

# Django ModelForm 클래스

---

## 1) Model Form이란?

- 지정된 Model로부터 필드 정보를 읽어 들여, form fields 를 세팅

```
class PostForm(forms.ModelForm):
```

```
    class Meta:
```

```
        model = Post
```

```
        fields = '__all__' # 전체필드지정 혹은 list로 읽어올 필드명 지정
```

- 내부적으로 model instance를 유지
- 유효성 검증에 통과한 값들로, 지정 model instance로의 저장 (save)을 지원 (Create or Update)



# Django Form vs ModelForm

---

```
from django import forms
from .models import Post

class PostForm(forms.Form):
    title = forms.CharField()
    text = forms.CharField(widget=forms.Textarea)
# 생성되는 Form Field는 PostForm과 거의 동일

class PostModelForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ['title', 'text']
```

# Post New ( 글 추가 ) 페이지 작성하기

---

## 1) forms.py 추가

- ① ModelForm을 생성해 자동으로 Model에 결과물을 저장할 수 있다.
- ② Form을 하나 만들어서 Post 모델에 적용한다.
- ③ blog 디렉토리 안에 forms.py라는 파일을 작성한다.

forms.ModelForm은 django에 이 폼이 ModelForm이라는 것을 알려주는 구문이다.

class Meta 구문은 Form을 만들기 위해서 어떤 model이 쓰여야 하는지 django에 알려주는 구문  
이 폼에 필드는 title과 text만 보여지게 된다. author는 로그인 하고 있는 사람이고,  
created\_date는 글이 등록되는 시간이다.

### blog/forms.py

```
from django import forms
from .models import Post
class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ('title', 'text',)
```

# Post New ( 글 추가 ) 페이지 작성하기

## 2) urls.py에 Post New(글 추가) url 추가

- ① ^은 "시작"을 뜻합니다.
- ② post/new란 URL이 post 문자를 포함해야 한다는 것을 말합니다
- ③ /은 다음에 / 가 한 번 더 와야 한다는 의미입니다.
- ④ \$는 "마지막"을 말합니다. 그 뒤로 더는 문자가 오면 안 됩니다.

blog/urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.post_list, name='post_list'),
    path('post/<int:pk>/', views.post_detail, name='post_detail'),
    path('post/new/', views.post_new, name='post_new'),
]
```

# Post New ( 글 추가 ) 페이지 작성하기

---

## 3) base.html에 Post New(글 추가) 페이지 링크 추가

- ① blog/templates/blog/base.html 파일을 열어서, page-header 라는 div class에 등록 폼 link를 하나 추가한다.
- ② 새로운 view는 post\_new입니다.
- ③ 부트스트랩 테마에 있는 glyphicon glyphicon-plus 클래스로 더하기 기호가 보이게 됩니다.

blog/templates/blog/base.html

```
<div class="page-header">
  <a href="{% url 'post_new' %}" class="top-menu">
    <span class="glyphicon glyphicon-plus"></span></a>
  <h1><a href="/">Django's Blog</a></h1>
</div>
```

# Post New ( 글 추가 ) 페이지 작성하기

---

## 4) views.py에 post\_new() 함수 추가

① 새 Post 폼을 추가하기 위해 PostForm() 함수를 호출하도록 하여 템플릿에 넘깁니다.

blog/views.py

```
from .forms import PostForm

def post_new(request):
    form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})
```

# Post New ( 글 추가 ) 페이지 작성하기

## 5) post\_edit.html 페이지 추가

- ① {{ form.as\_p }}를 HTML 태그로 폼을 감싸세요. <form method="POST">...</form>
- ② <form ...>을 열어 {% csrf\_token %}를 추가하세요. 이 작업은 폼 보안을 위해 중요합니다.

HTML Form의 POST요청에서 CSRF 토큰을 체크하며, 이때 CSRF토큰이 필요합니다.

csrf\_token tag를 통해 CSRF 토큰을 발급 받을 수 있습니다.

blog/templates/blog/post\_edit.html

```
{% extends 'blog/base.html' %}

{% block content %}
    <h1>New post</h1>
    <form method="POST" class="post-form">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Save</button>
    </form>
{% endblock %}
```

# Post New ( 글 추가 ) 페이지 작성하기

---

## 6-1) Form 저장하기

등록 Form의 두 가지 상황

첫번째 : 처음 페이지에 접속 했을 때, 새 글을 쓸 수 있게 Form이 비어 있습니다.

이때의 Http method는 GET

두번째 : Form에 입력된 데이터를 view 페이지로 가지고 올 때입니다. 이때의 Http method는 POST

blog/views.py

```
from .forms import PostForm

def post_new(request):
    if request.method == "POST":
        form = PostForm(request.POST)
    else:
        form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})
```

# Post New ( 글 추가 ) 페이지 작성하기

---

## 6-2) Form 저장하기

폼에 들어있는 값들이 올바른 지를 확인하기 위해 `form.is_valid()`을 사용합니다.

작업을 두 단계로 나눈다.

첫번째 : `form.save()`로 폼을 저장하는 작업, `commit=False`란 데이터를 바로 Post 모델에 저장하지 않는다는 뜻입니다.

두번째 : `author`와 `published_date`를 추가하는 작업, `post.save()`는 변경사항을 유지하고 새 블로그 글이 만들어 집니다.

**blog/views.py**

```
if form.is_valid():
    post = form.save(commit=False)
    post.author = request.user
    post.published_date = timezone.now()
    post.save()
```



# Post New ( 글 추가 ) 페이지 작성하기

---

## 6-3) Form 저장하기

- ① 새 블로그 글을 작성한 다음에 post\_detail 페이지로 이동 합니다.
- ② post\_detail은 이동 해야 할 view의 name이고, post\_detail view는 pk=post.pk를 사용해서 view에게 값을 넘겨줍니다.
- ③ post는 새로 생성한 블로그 글입니다.

blog/views.py

```
from django.shortcuts import redirect  
  
return redirect('post_detail', pk=post.pk)
```

# Post New ( 글 추가 ) 페이지 작성하기

---

## 6-4) 완성된 post\_new 함수

blog/views.py

```
from django.shortcuts import redirect

def post_new(request):
    if request.method == "POST":
        form = PostForm(request.POST)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.published_date = timezone.now()
            post.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})
```

# blog App 작성

: Django Form (글 수정)

# Post Edit ( 글 수정 ) 페이지 작성하기

## 1) urls.py에 Post Edit(글 수정) url 추가

- ① ^은 "시작"을 뜻합니다.
- ② post/1/edit란 URL이 post 문자를 포함 해야 한다는 것을 말합니다
- ③ /은 다음에 / 가 한 번 더 와야 한다는 의미입니다.
- ④ \$는 "마지막"을 말합니다. 그 뒤로 더는 문자가 오면 안 됩니다.


blog/urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.post_list, name='post_list'),
    path('post/<int:pk>/', views.post_detail, name='post_detail'),
    path('post/new/', views.post_new, name='post_new'),
    path('post/<int:pk>/edit/', views.post_edit, name='post_edit'),
]
```

# Post Edit ( 글 수정 ) 페이지 작성하기

## 2) post\_detail.html에 Post Edit(글 수정) 페이지 링크 추가

- ① blog/templates/blog/post\_detail.html 파일을 열어서, link를 하나 추가한다.
- ② 새로운 view는 post\_edit입니다.
- ③ 부트스트랩 테마에 있는 glyphicon glyphicon-pencil 클래스로  보이게 됩니다.

blog/templates/blog/post\_detail.html

```
<div class="post">
    {% if post.published_date %}
        <div class="date">
            {{ post.published_date }}
        </div>
    {% endif %}
    <a class="btn btn-default" href="{% url 'post_edit' pk=post.pk %}">
    <span class="glyphicon glyphicon-pencil"></span></a>
    <h1>{{ post.title }}</h1>
    <p>{{ post.text|linebreaksbr }}</p>
</div>
```

# Post Edit ( 글 수정 ) 페이지 작성하기

## 3) views.py에 post\_edit() 함수 추가

작업을 두 단계로 나눈다.

첫 번째: url로부터 추가로 pk 매개변수를 받아서 처리합니다.

두 번째: get\_object\_or\_404(Post, pk=pk)를 호출하여 수정하고자 하는 글의 Post 모델 인스턴스(instance)로 가져온 데이터를 폼을 만들 때와 폼을 저장할 때 사용하게 됩니다.

### blog/views.py

```
@login_required
def post_edit(request, pk):
    post = get_object_or_404(Post, pk=pk)
    if request.method == "POST":
        form = PostForm(request.POST, instance=post)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.published_date = timezone.now()
            post.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = PostForm(instance=post)
    return render(request, 'blog/post_edit.html', {'form': form})
```

# blog App 작성

: Django Form (글 삭제)

# Post remove ( 글 삭제 ) 페이지 작성하기

---

1) urls.py에 Post remove(글 삭제) url 추가

blog/urls.py


```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.post_list, name='post_list'),
    path('post/<int:pk>/', views.post_detail, name='post_detail'),
    path('post/new/', views.post_new, name='post_new'),
    path('post/<int:pk>/edit/', views.post_edit, name='post_edit'),
    path('post/<int:pk>/remove/', views.post_remove, name='post_remove'),
]
```



# Post remove ( 글 삭제 ) 페이지 작성하기

## 2) post\_detail.html에 Post Remove(글 삭제) 페이지 링크 추가

- ① blog/templates/blog/post\_detail.html 파일을 열어서, link를 하나 추가한다.
- ② 새로운 view는 post\_remove입니다.
- ③ 부트스트랩 테마에 있는 glyphicon glyphicon-remove 클래스로  보이게 됩니다.

blog/templates/blog/post\_detail.html

```
<div class="post">
    {% if post.published_date %}
        <div class="date">
            {{ post.published_date }}
        </div>
    {% endif %}
    <a class="btn btn-default" href="{% url 'post_remove' pk=post.pk %}">
    <span class="glyphicon glyphicon-remove"></span></a>
    <h1>{{ post.title }}</h1>
    <p>{{ post.text|linebreaksbr }}</p>
</div>
```

# Post remove ( 글 삭제 ) 페이지 작성하기

---

## 3) views.py에 post\_remove() 함수 추가

작업을 두 단계로 나눈다.

첫 번째: url로부터 추가로 pk 매개변수를 받아서 처리 합니다.

두 번째: get\_object\_or\_404(Post, pk=pk)를 호출하여 삭제 하고자 하는 글의 Post 모델 인스턴스(instance)로 가져 와서 삭제 처리를 한다.

**blog/views.py**

```
@login_required
def post_remove(request, pk):
    post = get_object_or_404(Post, pk=pk)
    post.delete()
    return redirect('post_list')
```

# blog App 작성

: 로그인/로그아웃 처리하기

# 로그인 처리하기

## 1) @login\_required 데코레이터

- ① 로그인 사용자만 포스트를 접근 할 수 있도록 `post_new`, `post_edit`, `post_remove`의 View들을 보호하려고 한다면
- ② Django에서 제공하는 `django.contrib.auth.decorators` 모듈 안의 `login_required` 데코레이터를 사용하면 된다.
- ③ `login_required` 데코레이터는 로그인 페이지로 리다이렉션(Redirection) 된다.
  - \* 주의: 로그인 되어 있는 admin 페이지를 로그아웃 해야 함
  - \* <https://docs.djangoproject.com/en/2.0/topics/auth/default/#auth-web-requests>

blog/views.py

```
from django.contrib.auth.decorators import login_required

@login_required
def post_new(request):
    [...]
```

# 로그인 처리하기

---

## 2) urls.py 에 login url 추가

① blog/urls.py가 아니라 myjango/url.py에 로그인 url 추가

mydjango/urls.py

```
from django.contrib import admin
from django.urls import path, include
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('blog.urls')),
    path('accounts/login/', auth_views.LoginView.as_view(template_name="registration/login.html"), name="login"),
]
```

# 로그인 처리하기

## 3) 로그인 페이지 템플릿 추가

① blog/templates/registration 디렉토리를 생성하고, login.html 파일 작성

blog/templates/registration/login.html

```
{% extends "blog/base.html" %}

{% block content %}
    {% if form.errors %}
        <p>이름과 비밀번호가 일치하지 않습니다. 다시 시도해주세요.</p>
    {% endif %}

    <form method="post" action="{% url 'login' %}">
        {% csrf_token %}
        <table class="table table-bordered table-hover">
            <tr>
                <td>{{ form.username.label_tag }}</td><td>{{ form.username }}</td>
            </tr>
            <tr>
                <td>{{ form.password.label_tag }}</td><td>{{ form.password }}</td>
            </tr>
        </table>

        <input type="submit" value="login" class="btn btn-primary btn-lg" />
        <input type="hidden" name="next" value="{{ next }}" />
    </form>
{% endblock %}
```

# 로그인 처리하기

---

## 4) settings.py 에 설정 추가

- ① 로그인 하면 최상위 index 레벨에서 로그인이 된다.

mydjango/settings.py

```
LOGIN_REDIRECT_URL = '/'
```

# 로그인 여부 체크하기

## 5) base.html 수정

① 인증이 되었을 때는 추가/수정 버튼을 보여주고, 인증이 되지 않았을 때는 로그인 버튼을 보여줌

blog/templates/blog/base.html

```
<div class="page-header">
  {% if user.is_authenticated %}
    <a href="{% url 'post_new' %}" class="top-menu">
      <span class="glyphicon glyphicon-plus"></span></a>
  {% else %}
    <a href="{% url 'login' %}" class="top-menu">
      <span class="glyphicon glyphicon-lock"></span></a>
  {% endif %}
  <h1><a href="/">Django's Blog</a></h1>
</div>
```



# 로그인 여부 체크하기

## 6) post\_detail.html에 수정

- ① 로그인 사용자만 글을 수정, 삭제 할 수 있도록 체크하기
- ② {% if %} 태그를 추가해 관리자로 로그인한 사용자들 만 글 수정,삭제 링크가 보일 수 있게 만든다.

blog/templates/blog/post\_detail.html

```
{% if user.is_authenticated %}
    <a class="btn btn-default" href="{% url 'post_edit' pk=post.pk %}">
    <span class="glyphicon glyphicon-pencil"></span></a>
    <a class="btn btn-default" href="{% url 'post_remove' pk=post.pk %}">
    <span class="glyphicon glyphicon-remove"></span></a>
{% endif %}
```

# 로그아웃 처리하기

## 1) base.html 수정

- ① “Hello <사용자이름>” 구문을 추가하여 인증된 사용자라는 것을 알려주고, logout link를 추가함

blog/templates/blog/base.html

```
<div class="page-header">
    {% if user.is_authenticated %}
        <a href="{% url 'post_new' %}" class="top-menu">
            <span class="glyphicon glyphicon-plus"></span></a>
        <p class="top-menu">Hello {{ user.username }}<small>
            (<a href="{% url 'logout' %}?next={{request.path}}">Log
out</a>)</small></p>
        {% else %}
            <a href="{% url 'login' %}" class="top-menu">
                <span class="glyphicon glyphicon-lock"></span></a>
        {% endif %}
    <h1><a href="/">Django's Blog</a></h1>
</div>
```

# 로그아웃 처리하기

## 2) urls.py 에 logout url 추가

① blog/urls.py가 아니라 myjango/url.py에 로그아웃 url 추가

mydjango/urls.py

```
from django.contrib import admin
from django.urls import path, include
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('blog.urls')),
    path('accounts/login/', auth_views.LoginView.as_view(template_name="registration/login.html"), name="login"),
    path('accounts/logout/', auth_views.LogoutView.as_view(), {'next_page': None}, name='logout'),
]
```

# blog App 작성

: 댓글 모델(comment model) 작성

# 댓글(**Comment**) 작성 – **Comment Model** 속성들

---

## 1) Comment(댓글)의 속성들

post(Post Model를 참조하는 Foreign Key)

author(글쓴이)

text(내용)

created\_date(작성일)

approved\_comment(승인여부)

## 2) Comment 객체는 blog/models.py 파일에 선언하여 모델을 만듭니다.

blog/models.py 파일을 열고, 끝에 Comment class의 내용을 추가해주세요.

# 댓글(**Comment**) 작성 – **Comment Class** 작성

## 2) models.py 에 Comment class 추가

blog/models.py

```
from django.db import models
from django.utils import timezone

class Comment(models.Model):
    post = models.ForeignKey('blog.Post', on_delete=models.CASCADE,
related_name='comments')
    author = models.CharField(max_length=200)
    text = models.TextField()
    created_date = models.DateTimeField(default=timezone.now)
    approved_comment = models.BooleanField(default=False)

    def approve(self):
        self.approved_comment = True
        self.save()

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

# 댓글(Comment) 작성 – 테이블 생성 및 관리자 패널에 등록

---

3) 마이그레이션 파일(migration file) 생성하기

```
django_src> python manage.py makemigrations blog
```

4) 실제 데이터베이스에 Post, Comment Model 클래스를 반영하기

```
django_src> python manage.py migrate blog
```

5) 관리자 페이지에서 만든 모델을 보기 위해 Comment 모델을 등록

blog/admin.py에 아래 코드를 추가

blog/admin.py

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

```
admin.site.register(Post)
admin.site.register(Comment)
```

# 댓글(Comment) 작성 – 관리자 화면에서 확인

6) 관리자 화면에서 확인하기

<http://localhost:8000/admin/> 으로 접속

사이트 관리

BLOG		
Comments	+ 추가	✎ 변경
Posts	+ 추가	✎ 변경



# 댓글(Comment) 작성- Comment를 화면에 나타내기

7) blog/templates/blog/post\_detail.html 수정

Comment를 화면에 나타나게 하기 위해 {% endblock %} tag 전에 아래 코드를 추가

blog/templates/blog/post\_detail.html

```
<hr>
{% for comment in post.comments.all %}
    <div class="comment">
        <div class="date">{{ comment.created_date }}</div>
        <strong>{{ comment.author }}</strong>
        <p>{{ comment.text|linebreaks }}</p>
    </div>
{% empty %}
    <p>No comments here yet :(</p>
{% endfor %}
```

# 댓글(Comment) 작성- Comment를 화면에 나타내기

---

8) blog/css/blog.css 수정

아래의 css 코드 추가하기

blog/css/blog.css

```
.comment {  
    margin: 20px 0px 20px 20px;  
}
```

9) blog/templates/blog/post\_list.html 수정

Post list 페이지에서 각 post 별 댓글 갯수를 출력하기: 아래의 링크 추가

blog/templates/blog/post\_list.html

```
<a href="{% url 'post_detail' pk=post.pk %}">Comments:  
{% post.comments.count %}</a>
```

# 댓글(Comment) 작성- Comment를 등록하기

10) blog/forms.py 수정 - Comment form 추가

blog/forms.py 파일 끝에 아래 코드를 추가하기

blog/forms.py

```
from .models import Post, Comment

class CommentForm(forms.ModelForm):
    class Meta:
        model = Comment
        fields = ('author', 'text',)
```

11) blog/urls.py 수정 - Comment 등록 url 추가

blog/urls.py

```
from django.urls import path

path('post/<int:pk>/comment/', views.add_comment_to_post, name='add_comment_to_post'),
```

# 댓글(Comment) 작성- Comment를 등록하기

12) blog/views.py 수정 - add\_comment\_to\_post() 함수 추가

blog/views.py

```
from .forms import PostForm, CommentForm

def add_comment_to_post(request, pk):
    post = get_object_or_404(Post, pk=pk)
    if request.method == "POST":
        form = CommentForm(request.POST)
        if form.is_valid():
            comment = form.save(commit=False)
            comment.post = post
            comment.save()
            return redirect('post_detail', pk=post.pk)
    else:
        form = CommentForm()
    return render(request, 'blog/add_comment_to_post.html', {'form': form})
```

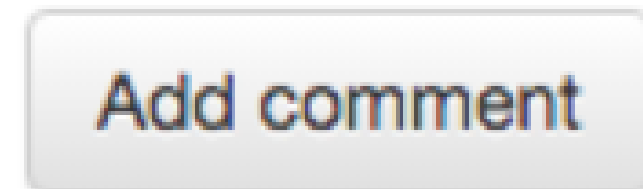
# 댓글(Comment) 작성- Comment를 등록하기

13) blog/templates/blog/post\_detail.html 수정 - Comment 등록 link 추가  
{% for comment in post.comments.all %} 전에 아래 코드를 추가

blog/templates/blog/post\_detail.html

```
<a class="btn btn-default" href="{% url 'add_comment_to_post' pk=post.pk %}">Add comment</a>
```

post detail 페이지에서, "Add Comment" 버튼을 확인할 수 있습니다.



No comments here yet :(

# 댓글(Comment) 작성- Comment를 등록하기

- 14) blog/templates/blog/add\_comment\_to\_post.html 추가  
: Comment 등록 할 수 있는 템플릿 추가

blog/templates/blog/add\_comment\_to\_post.html

```
{% extends 'blog/base.html' %}

{% block content %}
    <table>
    <h1>New comment</h1>
    <form method="POST" class="post-form">{% csrf_token %}
        <table class="table table-bordered table-hover">
            {{ form.as_table }}
        </table>
        <button type="submit" class="save btn btn-default">Send</button>
    </form>
    </table>
{% endblock %}
```

# 댓글(Comment) 관리 – 댓글 승인, 삭제 하기

- 1) 블로그 관리자가 댓글을 승인하거나 삭제할 수 있는 기능  
post detail 페이지에 댓글 삭제, 승인 버튼을 추가합니다.

blog/templates/blog/post\_detail.html

```
{% for comment in post.comments.all %}
    {% if user.is_authenticated or comment.approved_comment %}
    <div class="comment">
        <div class="date">
            {{ comment.created_date }}
            {% if not comment.approved_comment %}
                <a class="btn btn-default" href="{% url 'comment_remove' pk=comment.pk %}">
                    <span class="glyphicon glyphicon-remove"></span></a>
                <a class="btn btn-default" href="{% url 'comment_approve' pk=comment.pk %}">
                    <span class="glyphicon glyphicon-ok"></span></a>
            {% endif %}
        </div>
        <strong>{{ comment.author }}</strong>
        <p>{{ comment.text|linebreaks }}</p>
    </div>
    {% endif %}
{% empty %}
    <p>No comments here yet :(</p>
{% endfor %}
```

# 댓글(Comment) 관리 – 댓글 승인, 삭제 하기

---

2) blog/urls.py 수정 - Comment 승인, 삭제 url 추가

blog/urls.py

```
path('comment/<int:pk>/approve/', views.comment_approve, name='comment_approve'),  
path('comment/<int:pk>/remove/', views.comment_remove, name='comment_remove'),
```



# 댓글(Comment) 관리 – 댓글 승인, 삭제 하기

3) blog/views.py 수정 - comment\_approve(), comment\_remove() 함수 추가

blog/views.py

```
@login_required
def comment_approve(request, pk):
    comment = get_object_or_404(Comment, pk=pk)
    comment.approve()
    return redirect('post_detail', pk=comment.post.pk)

@login_required
def comment_remove(request, pk):
    comment = get_object_or_404(Comment, pk=pk)
    post_pk = comment.post.pk
    comment.delete()
    return redirect('post_detail', pk=post_pk)
```

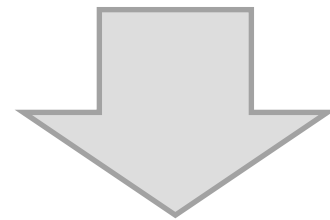
# 댓글(Comment) 관리 – 댓글 승인, 삭제 하기

4) blog/templates/blog/post\_list.html 수정

등록된 모든 댓글의 갯수가 대신에 **승인된 댓글의 갯수**가 노출 되도록 수정

blog/templates/blog/post\_list.html

```
<a href="{% url 'blog.views.post_detail' pk=post.pk %}">Comments:
{{ post.comments.count }}</a>
```



blog/templates/blog/post\_list.html

```
<a href="{% url 'blog.views.post_detail' pk=post.pk %}">Comments:
{{ post.approved_comments.count }}</a>
```

# 댓글(Comment) 관리 – 댓글 승인, 삭제 하기

---

5) blog/models.py 수정

Post 모델에 approved\_comments 메서드를 추가

blog/models.py

```
def approved_comments(self):  
    return self.comments.filter(approved_comment=True)
```

감사합니다.