Django 복습 & 심화

13번째 세션

NEXT X LIKELION 김범진

과제 공지

블로그 만들기

필수 기능

- 새로 Django 프로젝트를 생성하기 + HTML, CSS, Template 상속을 이용한 꾸미기
- 2개 이상의 APP(Blog, Accounts)을 사용하기 + 댓글, 회원가입, 로그인, 로그아웃 등 추가하기
- 게시글/댓글 작성 시 현재 로그인한 회원으로 작성자 등록하기 (로그아웃 상태에서 작성 불가)
- 게시글/댓글 수정 및 삭제는 현재 로그인한 회원과 글 작성자 일치할 때만 가능하도록 구현하기
- 게시글/댓글에 작성자 표시하기
- 회원 모델에 1개 이상의 필드 추가하여 User와 연결하기
- Admin 커스텀하기

선택 기능

- 게시글/댓글의 작성자를 클릭하면, 작성자의 글을 모아서 보여주는 페이지로 이동하기
- 게시글에 여러 정렬 기능 적용하기(작성자, 작성일자, 제목 등)
- User 상속 활용하기

목차

오늘의 주제

1. Django 복습
2. Django 심화 : APP 분리
3. Django 심화 : 상속

Django 복습

Django 소개

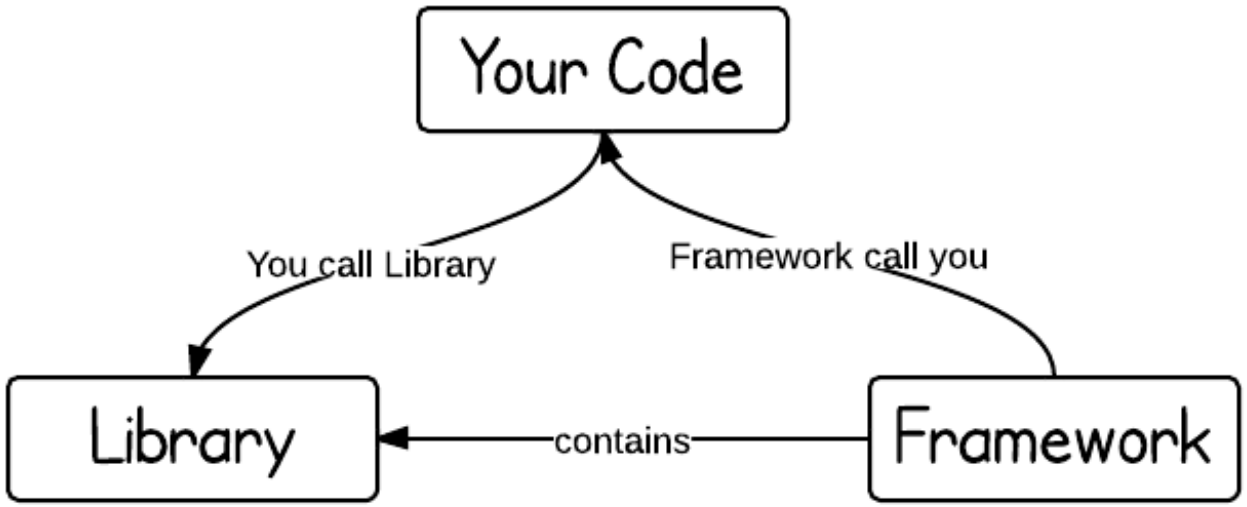
(보안이 우수하고 유지보수가 편리한) **웹사이트를 신속하게 개발하도록 도움을 주는 파이썬 기반 웹 프레임워크**

- 로그인/로그아웃
- 데이터베이스
- 보안

=> 이미 구현되어 있는 기능이 많아 잘 사용하면 된다!

Django 복습

프레임워크



API

응용 프로그램에서 운영체제나 프로그래밍 언어가 제공하는 기능을 제어할 수 있게 만든 인터페이스
(ex. 구글 소셜 로그인, 카카오 맵 등 Open API)

라이브러리

응용 프로그램 개발을 위해 필요한 기능을 모아 놓은 소프트웨어
(ex. requests, bs4)

프레임워크

응용프로그램이나 소프트웨어의 솔루션 개발을 수월하게 하기 위해 제공된 소프트웨어 환경
(ex. Django, Spring)

Django 복습

기존 방식

아직 따라하지 마세요!

```
mkdir session13                                  작업할 폴더 생성 & 이동
cd session13
pipenv shell                                     가상환경 생성
pipenv install django                            가상환경 내에 Django 패키지 설치
django-admin startproject {project}              장고 프로젝트 생성 & 이동
cd {project}
python manage.py startapp {app}                  프로젝트 폴더 내에 장고 앱 생성
```

settings.py 내의 INSTALLED_APPS에 '{app}'추가, TIME_ZONE 은 'Asia/Seoul'

```
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
```

MTV순으로 개발 시작

Django는 보통 하나의 Project와 여러 개의 App으로 구성됩니다.

Project
프로젝트의 환경설정
여러 app으로 구성되어 있다.

App
특정 동작을 하는 웹 어플리케이션
예) 회원 관리 시스템, 블로그, 투표, to do list

App을 여러 개로 구성함으로써 가독성과 유지보수에서 이점을 얻을 수 있습니다.
나중에 app과 project의 이름을 변경 및 삭제하는 경우 까다로우니 미리 구조를 작성하고 개발하는 게 좋습니다.

accounts
회원 관련 부분
회원가입, 회원탈퇴, 로그인, 로그아웃

blog
CRUD부분
글 작성, 글 읽기, 글 업데이트, 글 삭제
+댓글 작성, 댓글 읽기, 댓글 업데이트, 댓글 삭제

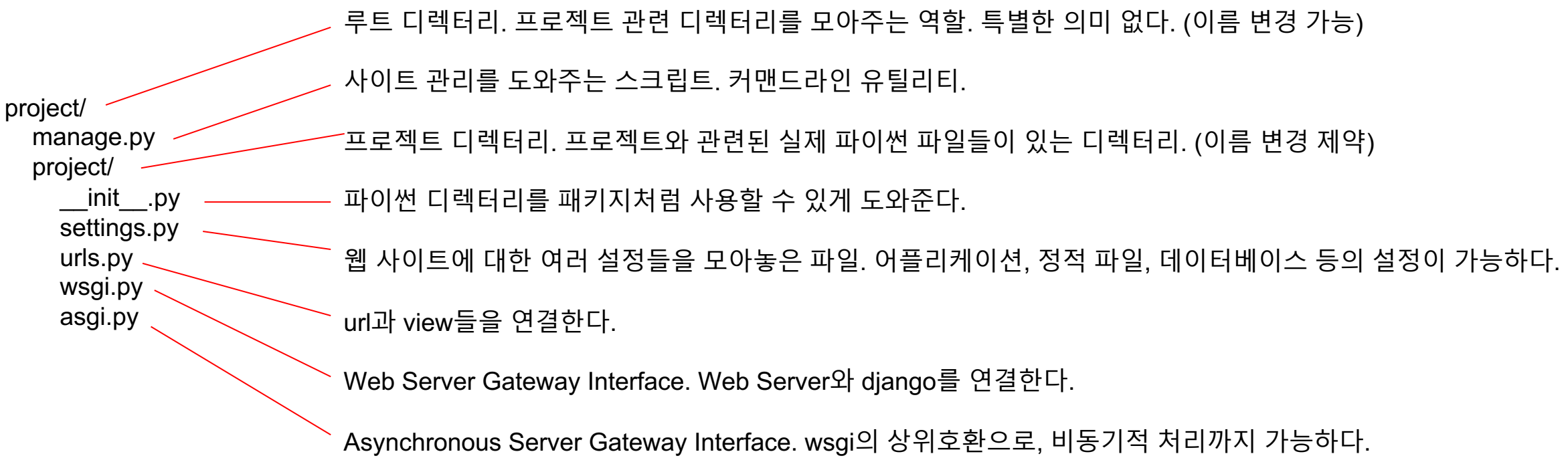
Django 심화

Project와 App을 나눠서 해봅시다!

```
mkdir session13                                  작업할 폴더 생성
cd session13                                     작업할 폴더 이동
pipenv shell                                     가상환경 생성
pipenv install django                            가상환경 내에 Django 패키지 설치
django-admin startproject {project}              장고 프로젝트 생성 & 이동
cd {project}
code .                                           vscode 열기

python manage.py showmigrations                  마이그레이션 확인
python manage.py migrate                         DB에 반영
python manage.py createsuperuser                 관리자 계정 생성
python manage.py runserver                       로컬 서버에서 장고 서버 실행
```

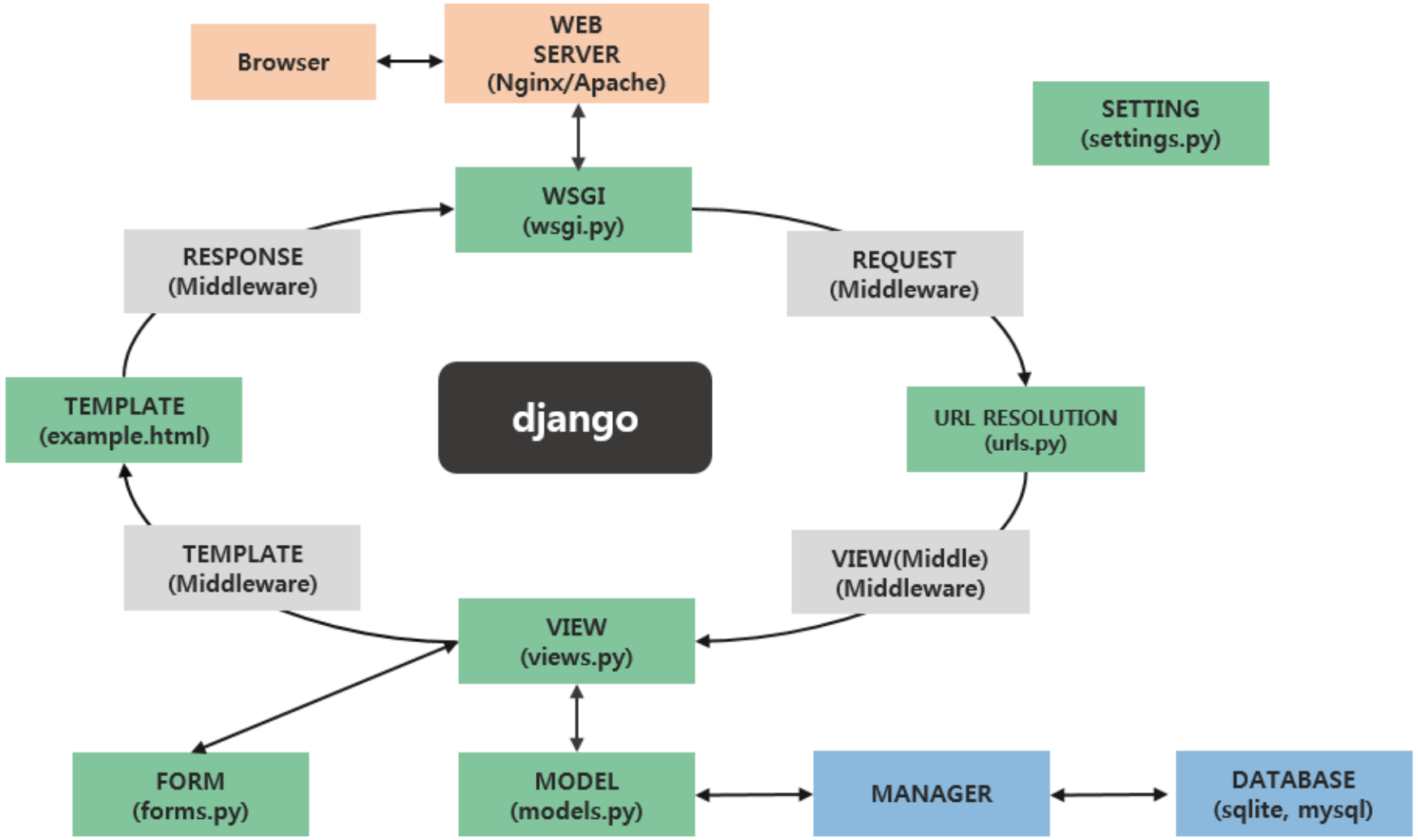
Django 심화

Project의 구조



Django 심화

Django의 동작 과정



Django 심화

App 생성 - blog

python manage.py startapp blog
프로젝트 폴더 내에 장고 앱 생성

settings.py 내의 INSTALLED_APPS에 ‘blog’추가, TIME_ZONE 은 'Asia/Seoul’

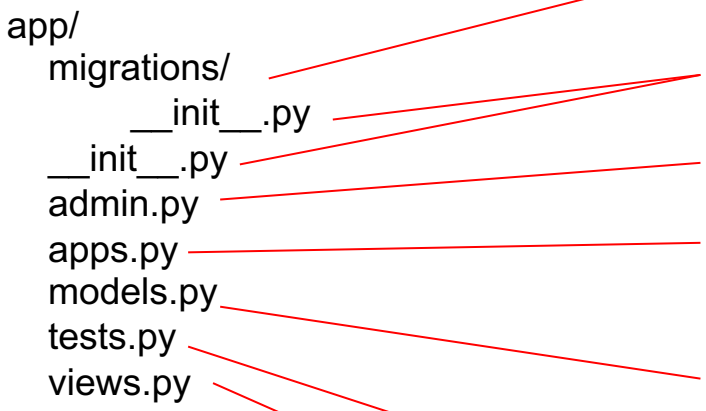
… Models.py 작업 …

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.

중요! models.py에 변경사항이 생길 때 마다,
python manage.py makemigrations와
python manage.py migrate를 해주셔야 합니다.

Django 심화

App의 구조



데이터베이스 반영을 위한 마이그레이션 파일들을 저장한다.

파이썬 디렉터리를 패키지처럼 사용할 수 있게 도와준다.

admin페이지에 대한 여러 설정을 할 수 있다. (커스터마이징)

어플리케이션에 대한 정보가 있는 곳. 설정 클래스를 작성할 수 있으며, 앱 이름, 레이블, 별칭, 경로 등을 설정할 수 있다.

테이블 관련 사항들을 정의한다. ORM 기법에 의해 테이블을 클래스로 정의하고 있으므로, 클래스의 특징인 변수와 메서드도 가질 수 있다.

테스트 코드를 작성할 수 있다.

웹 요청을 받아서 최종 응답 데이터를 웹 클라이언트로 반환하는 함수들이 있다.
(웹 요청을 분석하여 데이터베이스 처리 등 필요한 로직을 수행하고, 템플릿을 통해 화면에 표시할 데이터를 만든 후, 만들어진 최종 데이터를 웹 클라이언트에게 응답)

Django 복습

MTV 패턴

Model – 데이터베이스 설계

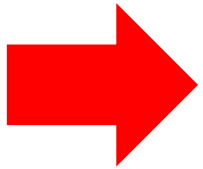
- 데이터베이스에 저장되는 데이터의 영역

Template – 화면 UI 설계

- 사용자에게 보여지는 영역, 화면
- HTML

View – 프로그램 로직 설계

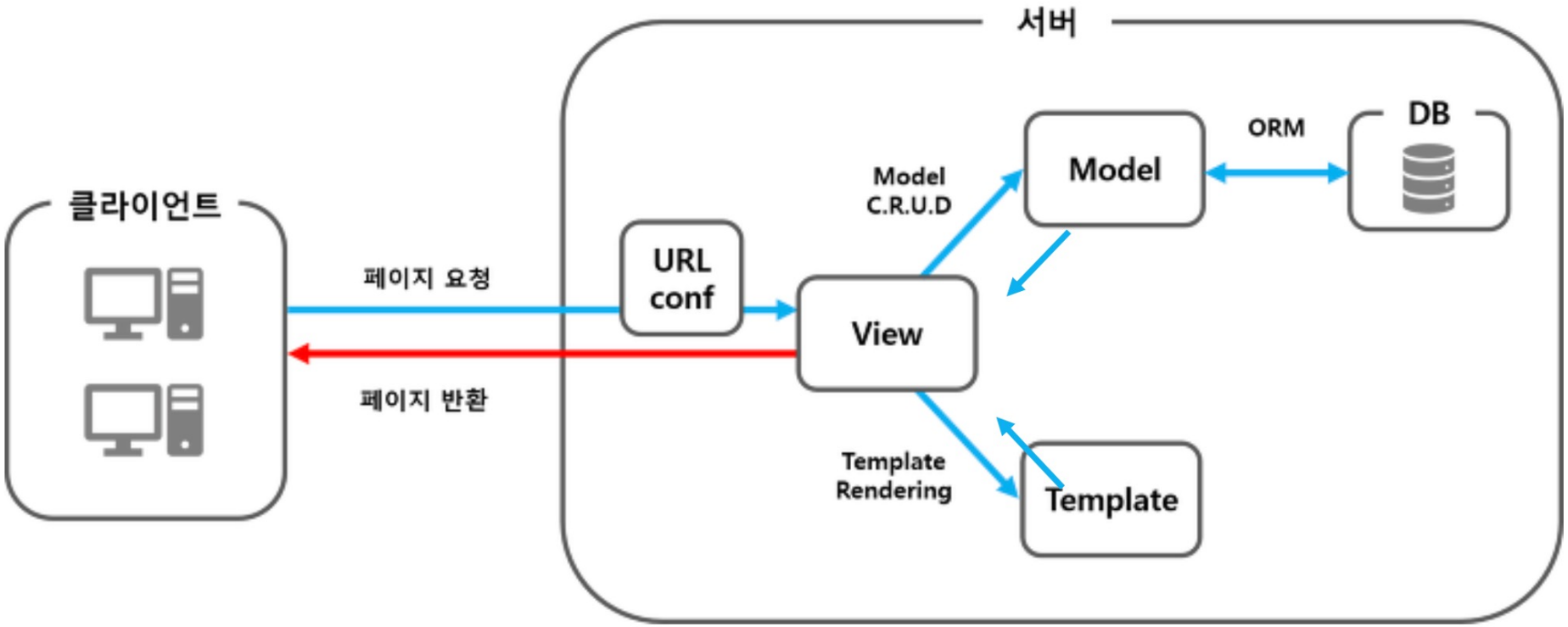
- 요청에 따라 Model에서 필요한 데이터를 가져와 처리
- 처리 결과를 Template에 전달



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

Django 복습

Django의 동작 과정

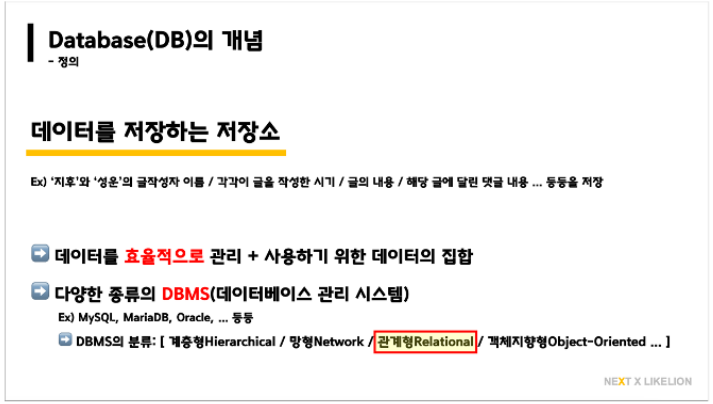


Django 복습

ORM

ORM

Django의 DB 처리 방식



심화!

데이터베이스 : PC에 전자적으로 저장되는 구조화된 정보나 데이터의 조직적인 집합

DBMS : 데이터를 한 곳에 모아 저장소를 만들고, 해당 저장소에 여러 사용자가 데이터를 저장 및 관리를 할 수 있도록 도와주는 소프트웨어

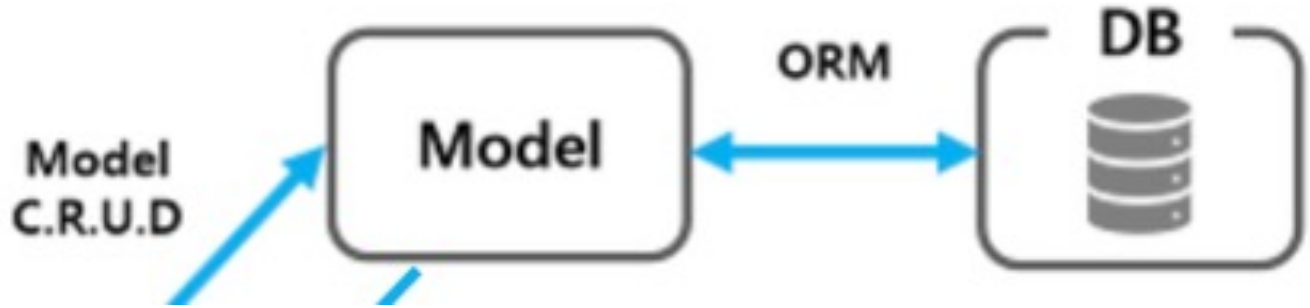
RDBMS : 관계형 데이터베이스를 관리하는 시스템

NoSQL : Not Only SQL의 약자로, 기존 전통적인 방식의 DB 방식(ACID 등)을 채택하지 않으면서도 뛰어난 확장성과 성능을 보장해주는 데이터베이스

SQL(Strucured Query Language) : RDBMS의 데이터를 관리하기 위해 사용되는 특수 목적의 프로그래밍 언어

Django 복습

ORM



Django의 DB처리는 **ORM(Object Relational Mapping)기법**을 사용한다.
객체(Object)와 **관계형 데이터베이스(Relational)**를 **연결(Mapping)**해준다.
=데이터베이스와 객체 지향 프로그래밍 언어(Python, Java)간의 호환되지 않는 데이터를 변환 및 연결하는 기법

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)

models.py에서 DB 변경사항이 생기면 반영해야된다.
=> 테이블 및 필드를 생성, 삭제, 변경 등 DB에 대한 변경 사항을 알려주는 정보인 **마이그레이션**을 도입

Django 복습

Model & Admin

```
# project/blog/models.py
```

```python
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
```

중요! models.py에 변경사항이 생길 때 마다,
python manage.py makemigrations와
python manage.py migrate를 해주세요!!

```
# project/blog/admin.py
```

```python
from django.contrib import admin
from .models import Post
# Register your models here.
admin.site.register(Post)
```

Django 복습

url

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

```
from django.contrib import admin
from django.urls import path
from blog 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('update/<int:post_pk>/', views.update, name="update"),
    path('delete/<int:post_pk>/', views.delete, name="delete"),
]
```

Django 복습

Template

https://docs.djangoproject.com/ko/4.1/ref/templates/language/
https://docs.djangoproject.com/ko/4.1/howto/custom-template-tags/

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

```
{% if (조건문) %}
    ~~~~
{% endif %}
```

```
{% for~ in ~ %}
    ~~~~
{% endfor %}
```

```
{% url 'index' %}
```

```
{{객체.어트리뷰트}}
```

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

Django 복습

Template

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

```
<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>
```

```
# project/blog/templates/new.html
```

```
<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>
```

Django 복습

Template

```
# project/blog/templates/detail.html
```

```
<body>
    <div>
        <h2>제목</h2>
        <p>{{ post.title }}</p>
    </div>
    <div>
        <h2>내용</h2>
        <p>{{ post.content }}</p>
    </div>
    <a href="{% url 'home' %}">홈으로</a>
    <a href="{% url 'update' post.pk %}">수정하기</a>
    <a href="{% url 'delete' post.pk %}">삭제하기</a>
</body>
```

```
# project/blog/templates/update.html
```

```
<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>
```

Django 복습

View

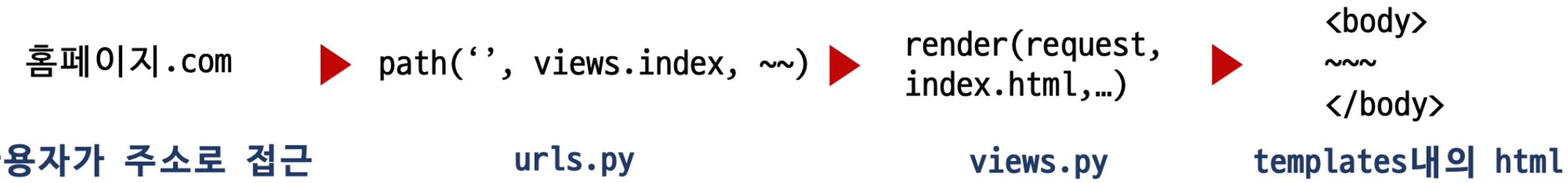
```
# project/blog/views.py
```

```python
from django.shortcuts import render, redirect
from .models import Post

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

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

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')
```

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

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

def update(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, 'update.html', {'post':post})

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

    return redirect('home')
```

Django 복습

Django 동작 과정



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)

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 })
```

Post 모델을 전부 가져와서,

posts라는 변수에 저장한다.

posts는 파이썬 변수.
하지만, html은 파이썬 변수를 인식할 수 없다.

render 함수를 통해,

위에서 저장한 posts 변수를,

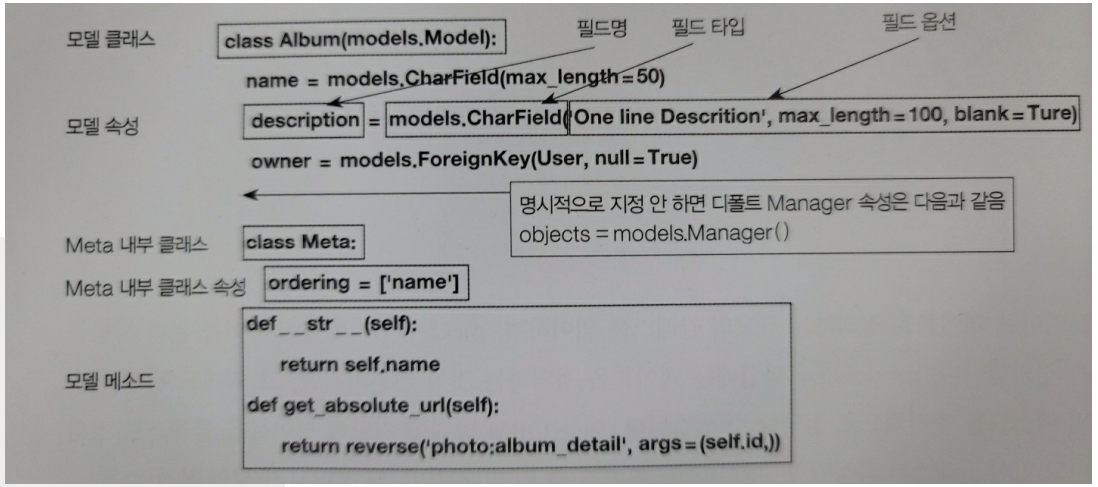
posts라는 이름으로,

home.html에 전달하고 해당 html을 불러온다.

쉬는 시간 후, Django 심화 : App 분리 시작하겠습니다.

Django 심화

Model 심화




```python
from django.db import models

class MyModelName(models.Model):
    """A typical class defining a model, derived from the Model class."""

    # Fields
    my_field_name = models.CharField(max_length=20, help_text='Enter field documentation')
    ...

    # Metadata
    class Meta:
        ordering = ['-my_field_name']

    # Methods
    def get_absolute_url(self):
        """Returns the url to access a particular instance of MyModelName."""
        return reverse('model-detail-view', args=[str(self.id)])

    def __str__(self):
        """String for representing the MyModelName object (in Admin site etc.)."""
        return self.field_name
```

Django 심화

Model 심화

project/blog/Models.py

```python
from django.db import models
from django.contrib.auth.models import User
from django.urls import reverse

# Create your models here.

class Post(models.Model):
# 모델 속성 : 데이터베이스 테이블의 컬럼 = 모델 클래스의 속성 = 모델 클래스의 필드 (ORM)
    title = models.CharField(verbose_name='TITLE', max_length=50)
    # CharField는 max_length가 필수. form에서 input tag
    content = models.TextField()
    # TextField는 form에서 textarea tag
    create_dt = models.DateTimeField(auto_now_add=True)
    # auto_now_add는 생성 시각 기준으로 자동 생성
    update_dt = models.DateTimeField(auto_now=True)
    # auto_now는 저장 시각 기준으로 자동 생성
    author = models.ForeignKey(User, on_delete=models.CASCADE, related_name="my_posts",
null=True)
    # ForignKey는 1:N에서 N에서 설정. CASCADE가 있는 속성에 해당하는 데이터 삭제 시, 그를
포함하는 모든 객체 삭제. related_name은 역참조할 때 사용

    class Meta:   # 메타 데이터 : 데이터에 대한 데이터. Meta 내부 클래스를 정의해 사용
        verbose_name = 'post'
        # 모델 객체의 별칭.
        verbose_name_plural = 'posts'
        # 별칭의 복수형 명칭
        ordering = ('-create_dt', 'author')
        # 모델 객체 리스트 출력시 정렬 기준. -면 내림차순. 위의 경우에는 create_dt 기준으로
내림차순 후 author 기준으로 오름차순

    # 모델 메소드 : 클래스 메소드가 아니라 객체 메소드. 항상 self인자를 가지고 있으며, 테이블 단위가
아니라 레코드 단위에 영향을 미친다. (테이블 단위는 views에서 Post.objects.all() 등으로 사용)
    def __str__(self):
        # 자신(객체)의 문자열 표현을 반환한다.
        # return self.title
        return f'{self.title}입니다.'


    def get_absolute_url(self):
        # 자신(객체)의 url을 반환한다. detail을 보여주는 경우 위 메소드를 사용하면 편하다.
        return reverse(f'blog:post_detail', args=[self.id])

    # create_dt기준으로 앞 뒤 객체 반환.
    def get_previous(self):
        return self.get_previous_by_create_dt()

    def get_next(self):
        return self.get_next_by_create_dt()

class Comment(models.Model):
    post = models.ForeignKey(Post, on_delete=models.CASCADE, related_name='comments')
    author = models.ForeignKey(User, on_delete=models.CASCADE,
related_name="my_comments")
    content = models.TextField()
    create_dt = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ('-create_dt', 'post', 'author')

    def __str__(self):
        return f'{self.author}-{self.content}'
```

모델이 변경되면?

Django 심화

Admin 심화

```
# project/blog/admin.py
from django.contrib import admin
from blog.models import Post, Comment
# Register your models here.

class CommentInline(admin.TabularInline):
    # TabularInline은 같은 admin page에서 다른 model을 수정할 수 있는 권한을 부여한다.
    model = Comment

@admin.register(Post)
# 데코레이터. admin.site.register(Post,PostAdmin)와 동일하다.
class PostAdmin(admin.ModelAdmin):
    # admin에서 아이디, 제목, 작성자, 작성 시각이 보이게 설정함.
    # 카테고리로 필터 가능
    # 게시글에 딸려있는 이미지와 댓글을 접근 가능하도록 설정하였음.
    list_display = ['id', 'title', 'author', 'create_dt']
    # 각 객체들이 위 순서로 보이게 설정
    list_display_links = ['id', 'title']
    # id말고 제목을 눌러도 해당 글의 정보를 볼 수 있는 페이지로 이동
    list_filter = ['create_dt', 'author']
    # create_dt와 author를 기준으로 필터 생성
    search_fields = ['title', 'author']
    # title과 author를 검색 가능
    inlines = [
        CommentInline,
    ]
    # 이제 Post 모델에서 연관된 Comment를 확인 및 수정할 수 있다.

@admin.register(Comment)
class CommentAdmin(admin.ModelAdmin):
    list_display = ('id', 'post', 'content', 'author')
```

Admin 페이지에 접속!

Django 심화

Url 심화

```
# project/project/urls.py
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('blog/', include('blog.urls')),
    # blog/로 시작되는 url이면, blog하위에 있는 url을 불러온다.
]
```

```
# project/blog/urls.py
from django.urls import path
from blog import views

app_name = 'blog'
# 중복을 피하기 위한 url 네임스페이스
urlpatterns = [
    # /blog/
    path('', views.home, name = 'home'),
    path('new/', views.new, name="new"),
    path('detail/<int:post_pk>/', views.detail, name="detail"),
    path('update/<int:post_pk>/', views.update, name="update"),
    path('delete/<int:post_pk>/', views.delete, name="delete"),
]
```

변경된 홈(http://127.0.0.1:8000/blog/)으로 접속!

Django 심화

Url 심화

127.0.0.1:8000/blog/

NoReverseMatch at /blog/

Reverse for 'detail' not found. 'detail' is not a valid view function or pattern name.

Request Method:	GET
Request URL:	http://127.0.0.1:8000/blog/
Django Version:	3.2.5
Exception Type:	NoReverseMatch
Exception Value:	Reverse for 'detail' not found. 'detail' is not a valid view function or pattern name.
Exception Location:	/Users/beomjin/.local/share/virtualenvs/session16-j4rP95ae/lib/python3.8/site-packages/django/urls/resolvers.py, line 694, in _reverse_with_prefix
Python Executable:	/Users/beomjin/.local/share/virtualenvs/session16-j4rP95ae/bin/python
Python Version:	3.8.2
Python Path:	['/Users/beomjin/NEXTLIKELION/session16/project', '/Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Versions/3.8/lib/python38.zip', '/Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Versions/3.8/lib/python3.8', '/Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Versions/3.8/lib/python3.8/lib-dynload', '/Users/beomjin/.local/share/virtualenvs/session16-j4rP95ae/lib/python3.8/site-packages']
Server time:	Thu, 01 Jul 2021 17:38:28 +0900

Error during template rendering

In template /Users/beomjin/NEXTLIKELION/session16/project/blog/templates/home.html, error at line 7

Reverse for 'detail' not found. 'detail' is not a valid view function or pattern name.

```
1  <body>
2      <h1>독후감 블로그</h1>
3      <div>
4          <ul>
5              {% for post in posts %}
6              <li>
7                  <a href="{% url 'detail' post.pk %}">{{ post.title }}</a>
8              </li>
9              {% endfor %}
10         </ul>
11     </div>
12     <a href="{% url 'new' %}">글 쓰러가기</a>
13 </body>
```

Url의 논리적인 이름을 사용하는 곳에, app_name:을 붙여줍시다!

```
<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>
```

```
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')
```

변경! detail.html도 확인 후 변경해주세요!

```
<body>
    <h1>독후감 블로그</h1>
    <div>
        <ul>
            {% for post in posts %}
            <li>
                <a href="{% url 'blog:detail' post.pk %}">{{ post.title }}</a>
            </li>
            {% endfor %}
        </ul>
    </div>
    <a href="{% url 'blog:new' %}">글 쓰러가기</a>
</body>
```

변경! Update와 delete도 확인 후 변경해주세요!

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

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

Django 심화

accounts 앱 생성

App 생성 => settings에 추가 => models 작성 => 마이그레이션 적용
=> admin, urls, templates, views 작성

```
# project/accounts/models.py
from django.db import models
from django.contrib.auth.models import User

# Create your models here.
class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    age = models.PositiveIntegerField(help_text="User Age",
blank=True, null=True)
```

```
# project/accounts/admin.py
from django.contrib import admin
from .models import Profile
# Register your models here.
admin.site.register(Profile)
```

```
# project/accounts/urls.py
from django.urls import path
from accounts import views

app_name = 'accounts'
urlpatterns = [
    # /accounts/
    path('', views.home, name = 'home'),
]
```

```
# project/accounts/templates/home.html
<div>
    <h1>독후감 블로그-회원관리</h1>
    <a href="{% url 'blog:home' %}">독후감 블로그</a>
</div>
```

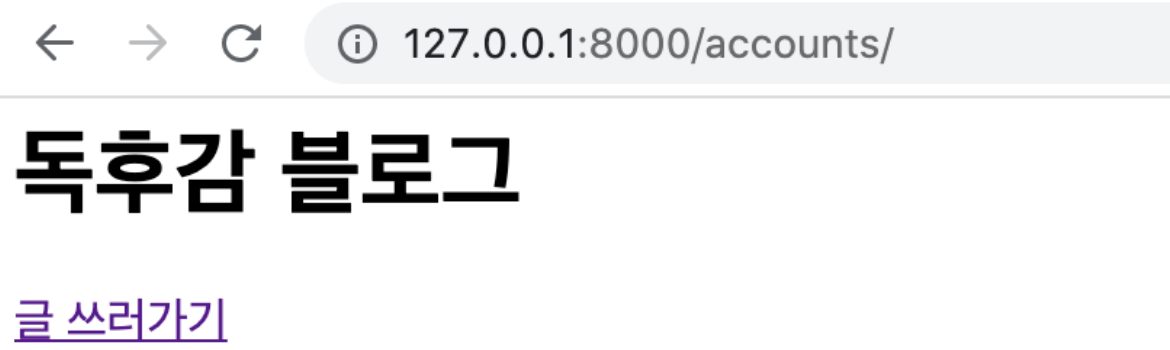
project/project/urls.py 에서 accounts include!

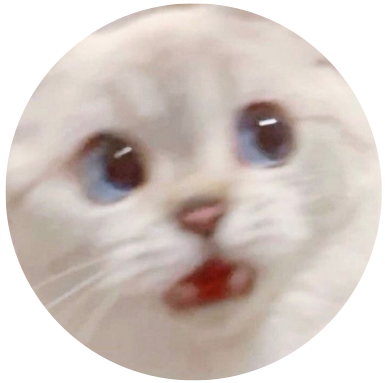
```
# project/accounts/views.py
from django.shortcuts import render

# Create your views here.
def home(request):
    return render(request, 'home.html')
```

http://127.0.0.1:8000/accounts/에 접속!

Django 심화

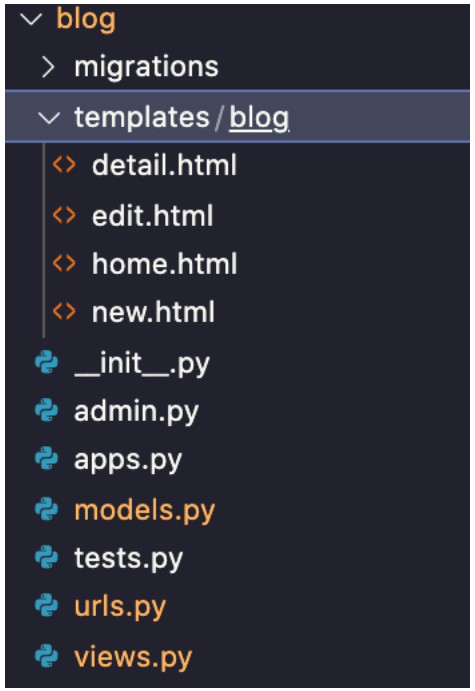
Url 심화





Django 심화

Url 심화



project/blog/views.py

```python
def home(request):
    posts = Post.objects.all()

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

주의!!!

App_name/templates/App_name/
에다가 html파일을 모아둘 것!

기존 방식으로 작성 시, 오류 없이 실행은 됩니다.
그러나 보여질 템플릿에 대한 결정은, **templates 폴더를 검색하여 처음에 발견한 파일을 사용**하는 방식으로 작동되고 있습니다.
즉, 해당 App의 template을 자동으로 가져오는 것이 아닙니다.

view에서 템플릿을 참고하는 경우에는, **App_name/file_name(파일구조)**
으로 작성해주시면 됩니다!

기타 static 파일들(image, css, javascript)도 이유는 다르지만
(collectstatic 시 중복 문제)
App_name/static/App_name/
에다가 파일을 모아두는 것이 좋습니다.

project/blog/templates/blog/home.html

```
<h1>독후감 블로그</h1>
<div>
    <ul>
        {% for post in posts %}
        <li>
            <a href="{% url 'blog:detail' post.pk %}">{{ post.title }}</a>
        </li>
        {% endfor %}
    </ul>
</div>
<a href="{% url 'blog:new' %}">글 쓰러가기</a>
```

쉬는 시간 후, Django 심화 : 상속 시작하겠습니다.

Django 심화

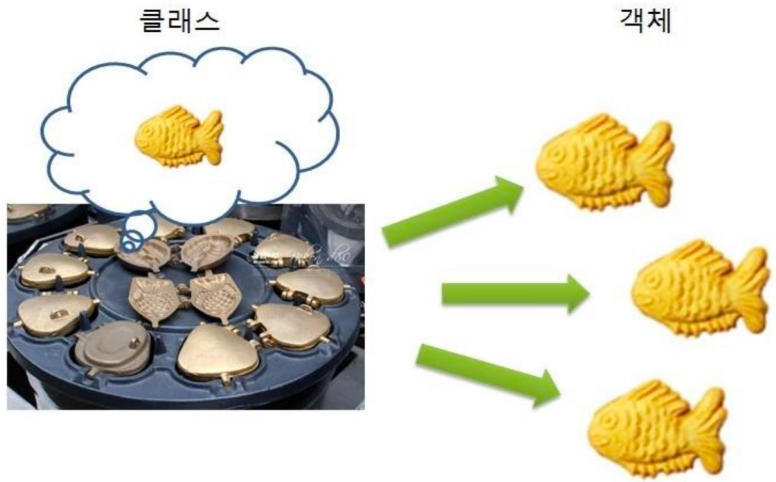
파이썬 객체

객체(object, instance)는 서로 연관된 데이터(attiribute)들과 그것들을 조작하기 위한 함수(method)들을 하나의 집합에 모아놓은 것을 말합니다.

객체 지향 프로그래밍에서 객체를 만들기 위해서는 클래스(class)가 필요합니다.

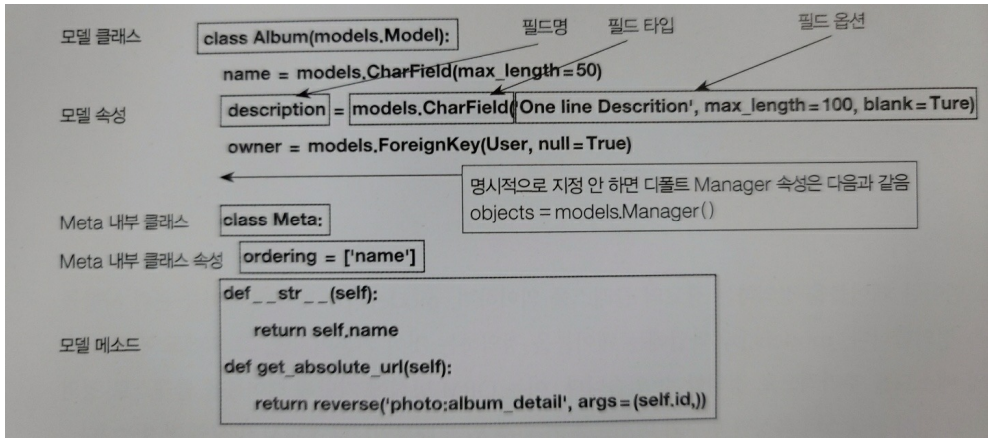
Class

Class를 가장 잘 설명 해주는 그림




```
class FishBreadPan:
    pass
```

```
fishBread1 = FishBreadPan()
fishBread2 = FishBreadPan()
fishBread3 = FishBreadPan()
```



Django 심화

객체 상속

보통 상속은 기존 클래스를 변경하지 않고
기능을 추가하거나 기존 기능을 변경하려고 할 때 사용합니다!
(Django에서 기본적으로 제공되는 것들은 수정 불가. auth, Form 등)

```python
class Person:
    def __init__(self, name, age):
        print(f'Person 생성 중... 인자는 {name}, {age}')
        self.name = name
        self.age = age
    def greeting(self):
        print(f'반갑습니다. 저는 {self.name}이고, {self.age}살입니다.')
    def get_name(self):
        print(f'제 이름은 {self.name}입니다.')
    def get_age(self):
        print(f'제 나이는 {self.age}살입니다.')

a = Person('김고대', 23)
a.greeting()
a.get_name()
a.get_age()
```

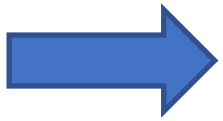
```python
class Student(Person):
    def __init__(self, name, age, gpa):
        super().__init__(name, age)
        self.age = age
        self.gpa = gpa
    def greeting(self):
        print(f'반갑습니다. 저는 {self.name}이고, {self.age}살입니다. 학점은 {self.gpa}입니다.')
    def get_gpa(self):
        print(f'제 학점은 {self.gpa}입니다.')

b = Student('김개발', 24, 4.0)
b.greeting()
b.get_name()
b.get_age()
b.get_gpa()
```

Django 심화

객체 상속

```
class FourCal:
    def __init__(self, first, second):
        self.first = first
        self.second = second
    def setdata(self, first, second):
        self.first = first
        self.second = second
    def add(self):
        result = self.first + self.second
        return result
    def mul(self):
        result = self.first * self.second
        return result
    def sub(self):
        result = self.first - self.second
        return result
    def div(self):
        result = self.first / self.second
        return result
```

Second가 0일 때 에러 발생



```
>>> a = FourCal(4, 0)
>>> a.div()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    result = self.first / self.second
ZeroDivisionError: division by zero
```

메소드 오버라이딩

```
class SafeFourCal(FourCal):
    def div(self):
        if self.second == 0:  # 나누는 값이 0인 경우 0을 리턴하도록 수정
            return 0
        else:
            return self.first / self.second
```

문제 해결!

```
>>> a = SafeFourCal(4, 0)
>>> a.div()
0
```

여기부터는 모르셔도 됩니다!

Django 의 위치는?

가상환경 실행 중 which python 명령어 입력으로 가상환경 경로 찾기
~~~/virtualenv/{가상환경이름}/lib/{python버전}/site-packages/django/

Django 내부 기능 궁금하시거나, 나중에 기능 상속을 하신다면 보세요!
Django 깃헙을 참고하는 것도 좋습니다. https://github.com/django/django
~~~

User의 확장 기법

proxy 모델 사용하기
- 부가적인 사용자 정보를 저장할 필요 없이, 기본 매니저(objects)를 변경하거나 메서드를 추가 하는 정도의 변화 등 파이썬 수준의 동작만을 변경할 때 사용한다.
- User모델을 상속하고, Meta Class에서 Proxy=True를 입력해서 proxy 모델 클래스인 것을 선언할 수 있다.

User 모델과 일대일관계의 프로필 테이블 추가하기
- Django의 인증 시스템을 그대로 활용하고, 로그인이나 권한 부여 등과 상관 없는 사용자 정보 필드를 저장하고자 할 때 사용한다.
- models.Model을 상속하여 작성한 모델에, 기존 User 모델과 OneToOneField로 일대일관계를 맺어서 사용자에 관한 정보를 저장한다.

AbstractUser 모델 상속한 사용자 정의 User 모델 사용하기
- User Model(로그인 인증 처리 부분 등)을 그대로 가져다 쓰는 대신 몇몇 필드만 재정의할 때 사용한다.
- settings.py에 AUTH_USER_MODEL = '{App_name}.{model_name}' 를 추가해주어야 한다.

AbstractBaseUser 모델 상속한 사용자 정의 User 모델 사용하기
- 필드를 재정의할 뿐만 아니라, User의 인증 및 권환 관리(로그인 인증 처리 절차) 등을 수정할 때 사용한다.
- settings.py에 AUTH_USER_MODEL = '{App_name}.{model_name}' 를 추가해주어야 한다.
- Manager(기존의 objects)를 만들어서 설정해주어야한다.

Django 심심화

Django User 확장

```
# project/accounts/models.py
```

```python
from django.db import models
from django.contrib.auth.models import AbstractUser

# Create your models here.
class Profile(AbstractUser):
    age = models.PositiveIntegerField(help_text="User Age", blank=True, null=True)
```

```
# project/accounts/admin.py
```

```python
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin as BaseUserAdmin
from .models import Profile
from .views import CreateUserForm
# Register your models here.
class UserAdmin(BaseUserAdmin):
    add_form = CreateUserForm
admin.site.register(Profile)
```

Django 심심화

Django User 확장

```
# project/accounts/urls.py
from django.urls import path
from accounts import views

app_name = 'accounts'
urlpatterns = [
    # /accounts/
    path('', views.home, name = 'home'),
    path('signup', views.signup, name = 'signup'),
]
```

```
# project/accounts/templates/accounts/home.html
<div>
    <h1>독후감 블로그-회원관리</h1>
    <a href="{% url 'blog:home' %}">독후감 블로그</a>
    <a href="{% url 'accounts:signup' %}">회원가입</a>
</div>
```

```
# project/accounts/templates/accounts/signup.html
<h2>회원가입</h2>
<form method="post" action="">
{% csrf_token %}
{{ form.as_p }}
<input type="submit" value="회원가입" />
</form>
```

Django 심심화

Django User 확장

```
# project/project/settings.py
AUTH_USER_MODEL = 'accounts.Profile'
```

```
# project/accounts/views.py
from django.shortcuts import render, redirect
from .models import Profile
from django.contrib.auth.forms import UserCreationForm

# Create your views here.
def home(request):
    return render(request, 'accounts/home.html')

class CreateUserForm(UserCreationForm):
    class Meta:
        model=Profile
        fields=['username', 'email', 'age', 'password1', 'password2']

def signup(request):
    form = CreateUserForm
    if request.method == "POST":
        form = CreateUserForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('accounts:home')
    return render(request, 'accounts/signup.html', {'form': form})
```

```
# project/blog/models.py
from accounts.models import Profile
```

User를 싹 다 Profile로 변경

Django 심심화

Form

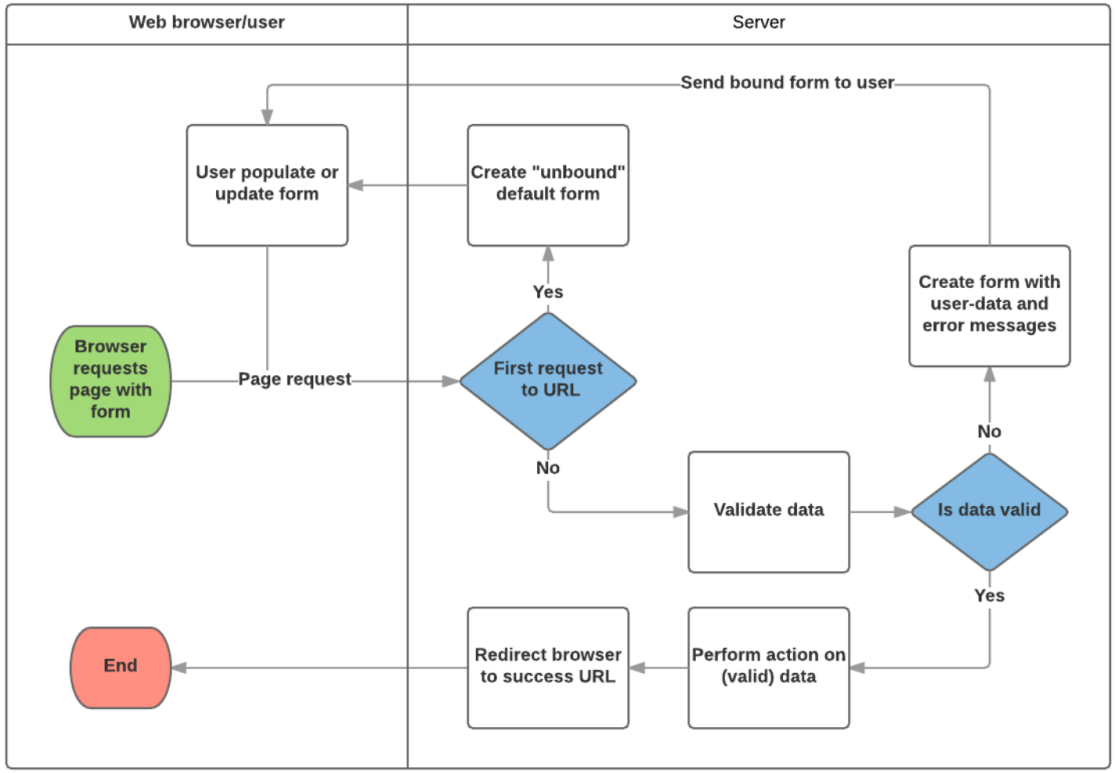
장고의 주기능 중 하나
Model처럼 Form클래스를 정의하면,
Model이 DB랑 상호작용 하는 것처럼 Form클래스와 프론트엔드가 상호작용한다.

Form클래스를 상속받아서 따로 forms.py를 만들어 사용하며,

- 입력폼 HTML 생성(form.as_table(), form.as_p(), form.as_ul() 등)
- **입력폼 값 검증(validation) 및 값 변환**
- 검증을 통과한 값들을 **사전타입**(cleaned_data)으로 제공

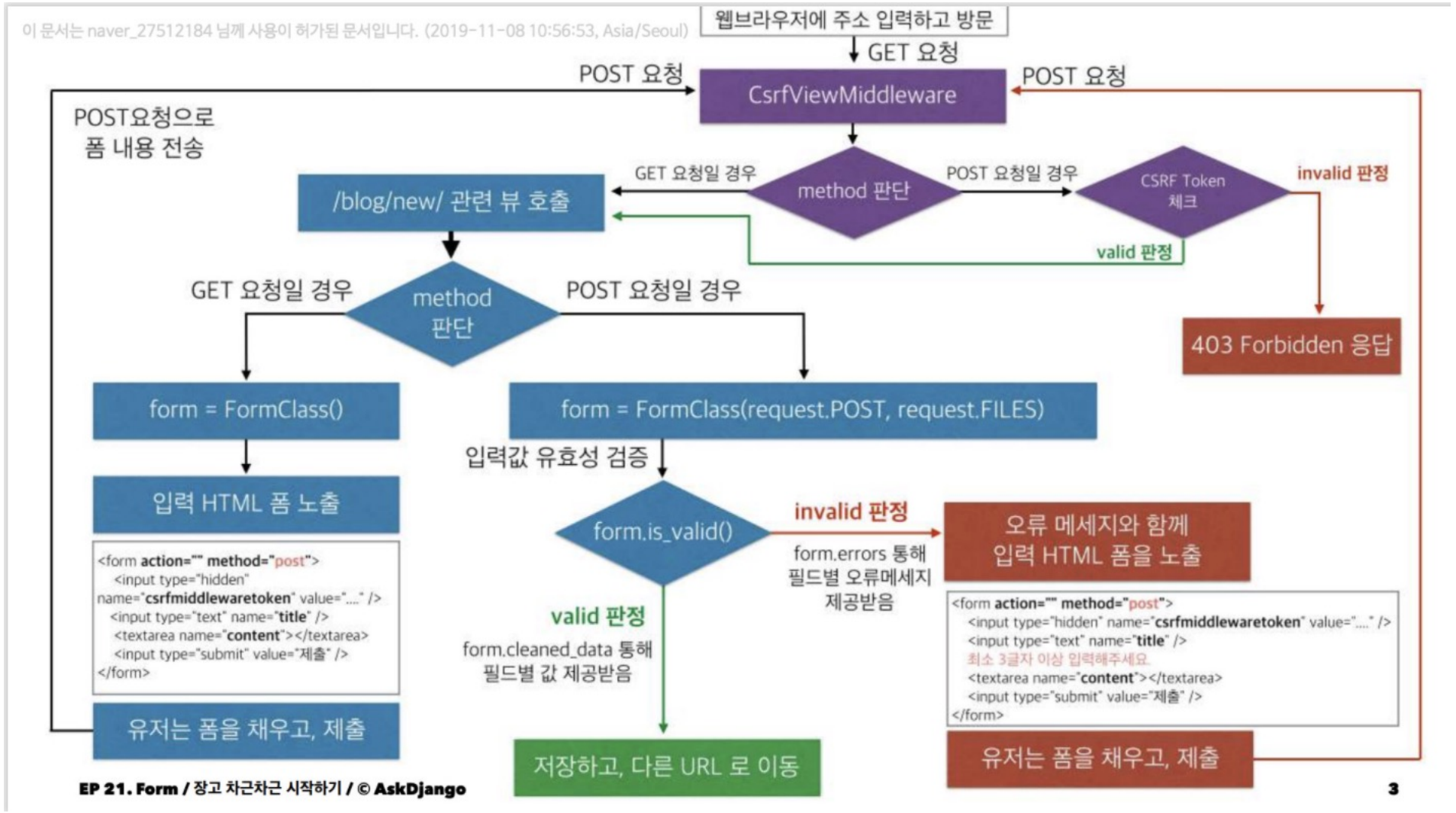
등의 역할을 수행한다.

일반 폼, 모델 폼, 폼셋 등의 종류가 있다.



Django 심심화

Form



Django 심심화

Form 용어

Widget
<input>, <textarea>같은 HTML 폼 위젯에 대응되는 클래스로, 해당 위젯을 HTML로 렌더링한다.

Field
각 필드에 대한 유효성을 담당하는 클래스이다.
예) EmailField는 데이터가 유효한 이메일 주소인지 확인한다.

Form
폼 자체에 대한 유효성 검증 규칙 및 HTML로서의 표시 방법을 알고 있는 필드의 모음이다.

Form Media
필요한 CSS와 JS를 제공해준다.

Bound Form
데이터 집합에 바인딩 되어 있으면, 해당 데이터의 유효성을 검사하고 HTML로 렌더링

Unbound Form
유효성을 검사할 데이터가 없으므로 유효성 검사를 하지 못하며 빈 양식을 렌더링