Django

MTV

- MVC와 항상 비교되는 개념이다.

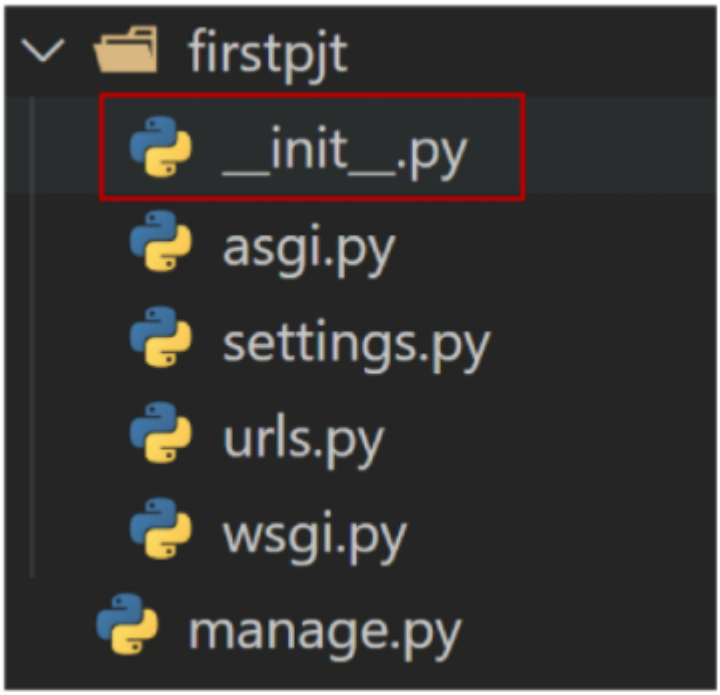
MVC	MTV
Model	Model
View	Template
Contoller	View

- Model : 데이터를 읽고, 저장하는 형태
- Templates : 표현하는 양식, HTML문서
- View : URL, Model, Template와 통신하며 요청에 응답

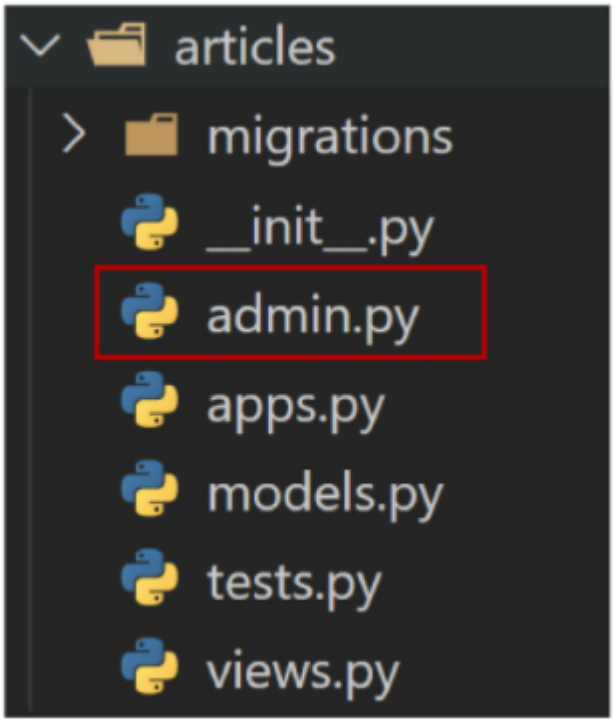
주요 명령어 들

- `django-admin startproject firstpjt .` : 프로젝트를 'firstpjt'라는 이름으로 해당 폴더 '.'에 생성한다.
- `python manage.py runserver` : 서버를 가동시킨다.
- `python manage.py startapp articles` : 앱을 'articles'라는 이름으로 생성한다.

생성한 프로젝트 구조



- `__init__.py`, `asgi.py`, `settings.py`, `urls.py`, `wsgi.py` 가 생성된다.
- `manage.py` 는 프로젝트보다 한 단계 상위에 생성된다.



- `__init__.py`, `admin.py`, `apps.py`, `models.py`, `tests.py`, `views.py`가 생성된다.

INSTALLED_APPS

- 반드시 app을 생성하고 등록해야한다!!

settings.py

- `INSTALLED_APPS` : 반드시 app을 생성하고 등록해야한다!!!!!!!!
- `LANGUAGE_CODE` : 사용자에게 제공되는 번역을 결정한다.
  - `ko-kr`이 한글
- `TIME_ZONE` : 시간대를 설정한다.
  - `Asia/Seoul`이 한국 시간대
  - USE_TZ가 True여야 작동한다.

DTL (Django Template Language)

- Django template에서 사용하는 built-in 함수들이다.
- python 코드가 절대 아니다.

1. Variable
   - `{{ variable }}`처럼 views.py에서 정의한 변수를 사용할 수 있다.
   - 밑줄이 들어갈 순 있지만, 밑줄로는 시작할 수 없다.
     - `_context` 는 불가능, `first_value`는 가능
2. Filters
   - 위의 variable을 수정할 때 사용한다.
   - `{{ variable|filter }}`와 같이 사용한다.
   - `{{ variable|date: 'mm dd' }}`와 같이 인자가 필요한 필터도 있다.

3. Tags
    - 반복, 논리등 제어 흐름을 만들어준다.
    - `{% if %} ~ {% endif %}`와 같이 사용하며, 종료태그가 필요할 수도, 아닐 수도 있다.
4. Comments
    - 주석을 표현한다.
    - `{# comment #}`와 같이 comment에 주석을 표현하면 된다. 줄바꿈은 허용되지 않는다.
    - `{% comment %} ~ {% endcomment %}` 안에 주석을 쓸 수도 있다. 줄바꿈 가능

Template 만들기!

- 데이터 흐름에 따라 작성하는데, 보통 아래와 같이 작성한다.
    1. urls.py
    2. views.py
    3. templates
- urls.py

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

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

```python
app_name = 'movies'
urlpatterns = [
    path('', views.index, name='index'),
    path('new/', views.new, name='new'),
    path('create/', views.create, name='create'),
```

`path` : 요청되는 URL에 views의 어떤 함수를 호출할 지 정해주는 함수

`include` : app의 urls.py를 불러올 때 사용한다. `appname.urls`로 표현해야한다!

`app_name` : app을 구분해 만약 같은 name을 가진 URL호출도 구분하여 호출할 수 있도록 함

app_name을 사용함으로 유지보수가 편해진다.

- views.py

```python
def create(request):
    if request.method == 'POST':
        movie = Movie()
        movie.title = request.POST.get('title')
        movie.audience = request.POST.get('audience')
        movie.release_date = request.POST.get('release_date')
        movie.genre = request.POST.get('genre')
        movie.score = float(request.POST.get('score'))
        movie.poster_url = request.POST.get('poster_url')
        movie.description = request.POST.get('description')
        movie.save()
    return redirect('movies:index')

def detail(request, pk):
    movie = Movie.objects.get(pk=pk)
    context = {
        'movie': movie,
    }
    return render(request, 'movies/detail.html', context)
```

`render` : 첫 번째 인자로 request, 두 번째 인자로 불러올 html의 디렉토리, 세 번째 인자로 context를 넣어주면 된다. context는 항상 **딕셔너리**!!

`redirect` : `app_name:name`으로 호출되는 함수이고, 새로 render하지 않고 이미 만들어진 다른 URL의 불러올 때 사용한다.

- templates (_.html)

```
<div class="container">
  <h1>INDEX</h1>
  <a href="{% url 'movies:new'%}"><bu
  {% for movie in movies %}
  <span class="rounded bg-success tex
  <span> </span><a class="text-decora
  <hr>
  {% endfor %}
```

```
<!-- greeting.html -->

<p>안녕하세요 저는 {{ info.name }} 입니다.</p>
<p>제가 좋아하는 음식은 {{ foods }} 입니다.</p>
<p>{{ foods.0 }}을 가장 좋아합니다.</p>

<a href="/index/">뒤로</a>
```

`{% url 'app_name:name' %}` : url을 호출할 때 사용한다.

`{{ context.key }}` : 넘겨받은 context를 사용할 때 이렇게 사용한다.

- 상속
  - `{% extends '' %}`로 미리 만들어진 skeleton을 상속받을 수 있다. 반드시 템플릿 최상단에 작성되어야한다!

- 상속받은 뒤, `{% block 블럭이름 %} ~ {% endblock %}`으로 블럭을 채워줄 수 있다.
- 이 때, settings.py에 TEMPLATES의
- `{% include '' %}` : 엄연히 상속은 아니지만, 비슷하게 템플릿을 로드할 순 있다.

HTML Form

- 웹에서 정보를 입력하는 여러 방식을 제공하고, 입력받은 데이터를 서버로 전송하는 역할을 담당한다.
- `action=""` : 데이터가 전송될 URL을 지정한다.
- `method=""` : 데이터를 전달하는 방식을 지정한다.
  - GET : 기본상태, URL에 노출되어 전달된다.
  - POST : 내부 데이터로 전송된다. django에선 CSRF를 적용시켜야한다. 아래 그림처럼 csrf_token 태그로 csrf적용 안쓰면 403 Error가 발생한다!

```
<form action="{% url 'movies:create' %}" method="POST" class="">
  {% csrf_token %}
  <label for="title">TITLE</label>
  <input type="text" name="title" id="title"><br>
  <label for="audience">AUDIENCE</label>
  <input type="text" name="audience" id="audience"><br>
```

- `<label>`의 for와 `<input>`의 id가 연결된다.
- `<input>`의 **name**이 POST나 GET에서 전달될 이름이다! - **name**이 가장 중요한 요소

Variable Routing

- URL주소의 일부를 변수로 사용하는 것.
- `path('/<int:pk>')`와 같이 `<>`로 묶여있는 것이 변수이다. 이를 통해 하나의 path에 여러 페이지를 연결시킬 수 있다.
- 이를 HTML파일에서 연결시킬때는, 아래와 같이 하면 된다.

```
href="{% url 'movies:detail' movie.pk %}"
```

- movies 앱의 detail 경로를 호출하는데, movie.pk의 variable routing을 사용해서!
- 이러면 아래의 path가 호출된다.

```
app_name = 'movies'
urlpatterns = [
    path('', views.index, name='index'),
    path('new/', views.new, name='new'),
    path('create/', views.create, name='create'),
    path('<int:pk>/', views.detail, name='detail'),
```

- app_name이 movies고, name이 detail인 4번째 줄이 호출된다.

Model

- 데이터에 대한 정보, Scheme를 가지고 있다.
- 저장된 데이터베이스의 구조를 python 구문으로 가지고 있다.
- 아래와 같은 데이터가 있다고 가정했을 때,

pk	name	created_at	updated_at
1	김김김	2022-03-11	2022-03-12
2	이이이	2022-03-11	2022-03-14

- 필드, 속성, 열 : pk, name, created_at, updated_at
- 튜플, 레코드 행 : 각 행의 정보들
- 테이블 : 전체를 뜻하는 말

ORM

- Object-Relational-Mapping
- python을 통해 DB를 다룰 수 있도록 해주는 프로그래밍 기술
- Django는 내장 Django ORM을 사용하고, DB로 SQLite를 사용한다.
- **ORM을 사용함으로써 웹 개발 속도가 올라간다 -> 생산성이 높아진다!**

Model

- models.py를 작성함으로 ORM을 사용할 수 있다!

```
class Article(models.Model):
    title = models.CharField(max_length=10)
    content = models.TextField()
```

- models.Model을 상속받아 model의 기본틀을 잡아준다.
- title과 content는 Article DB의 필드이고, models.~는 속성이다.
- CharField와 TextField의 차이
    - CharField(max_length=?)와 같이 최대 길이를 정해준다.
    - TextField()는 길이를 특별히 정하지 않는 문자열 필드일 때 주로 사용한다.
    - DateField(auto_now_add=True) : 최초로 생성될 때 현재의 날짜으로 갱신된다는 뜻
    - DateField(auto_now=True) : 최초 생성 + 수정될 때 현재의 날짜으로 갱신된다는 뜻

Migrations

1. `python manage.py makemigrations` : models.py에서 만든 model처럼 migration파일을 생성해준다. 아직 DB를 만든건 아니다
2. `python manage.py migrate` : 위에서 만든 migration파일을 바탕으로 실제 DB를 만들어준다.
3. `python manage.py sqlmigrate app_name 0001` : app_name의 1번 migration파일을 SQL문으로 하면 어떨까 궁금할 때 쓰는 명령이다.
4. `python manage.py showmigrations` : migration파일들이 실제 DB로 만들어졌는지 아닌지 확인해 볼 수 있다.

- 이를 통해 생성된 DB는 views.py에서 ORM으로 다룰 수 있어진다.

```
from .models import Article
```

- 위와 같이 models.py의 Scheme를 불러와준 뒤, 사용한다.
- `Article.objects.all()` : Article의 모든 튜플을 불러와준다.
- `Article.objects.get(필드=값, 필드=값...)` : 뒤의 조건에 맞는 튜플들만 불러온다.
- 나머지 명령어들은 CRUD에서 자세히 다룸

CRUD

- Create, Read, Update, Delete를 뜻한다.
- 대부분의 컴퓨터 소프트웨어가 가지는 기본적인 데이터 처리 기능
- CREATE

    1. 변수에 틀을 불러오고, 각 필드를 지정해준 뒤 저장하기

    ```
    article = Article()
    article.title = first_title
    article.content = first_content
    article.save()
    ```

    2. 초기 값과 함께 변수에 불러오고, 저장하기

    ```
    article = Article(title='second_title', content='second_content')
    article.save()
    ```

    3. create 메서드 사용하기

    ```
    Article.objects.create(title='third_title', content='third_content')
    ```

- READ
    - Article.objects.all()
    - Article.objects.get(필드=값) : 하나만 가져오고, 해당 조건에 맞는게 두 개 이상이면 오류
    - Article.objects.filter(필드=값) : 두 개 이상도 가져와서 Query set으로 출력해줌
- UPDATE
    - Article.objects.get(필드=값)으로 수정하기 원하는 값을 불러온 뒤, CREATE의 1번 방식처럼 해주면 update완성
- DELETE
    - Article.objects.get(필드=값)으로 지우고 싶은 값을 불러온 뒤, `article.delete()`를 해주면 지워진다.

Admin Site

- 서버의 관리자가 활용하기 위한 페이지
- admin.py를 작성해주자
- `python manage.py createsuperuser`를 입력해 관리자아이디를 만들어주는 것으로 시작한다.

```python
from django.contrib import admin
from .models import Article


class ArticleAdmin(admin.ModelAdmin):
    list_display = ('pk', 'title', 'content', 'created_at', 'updated_at',)


admin.site.register(Article, ArticleAdmin)
```

- class는 admin페이지에서 어떤 형식으로 model을 보여줄지 설정한 것이다.
- admin.site.register()로 admin페이지에 표시하도록 한다.