Python Web Programming

2. REST and Django

최강타

cminor_delf@naver.com

Server?

- WebSocket, TCP, UDP server
    - Thread (while(True)) + something...
        - 지속적인 연결 유지 + 상태 공유
    - 서버의 입력, 출력 등은 서버 프로그램 자체에서 정의
        - 일반적인 방법론은 있지만 표준은 존재하지 않음
        - 단순하게 정의한 것이 가볍고 빠를 수도 있다.
            - 주로 게임 등에서 사용

Responsibility of Server

- 무엇을 목표로 하는 서버인가?

  - FPS 게임 서버
    - 단일 목적 클라이언트(게임 프로그램)
    - 사용자(다수)간 지속적인 상태 공유
    - 입력(이동, 공격 등)에 대한 빠른 반응 및 전파

  - 웹 사이트 서버
    - 범용 클라이언트 (웹 브라우저)가 해석 가능한 표준화된 입/출력
    - 무상태성 (요청과 요청 간의 불연속 및 독립)
    - 분산 네트워크 및 보안을 고려한 정확하고 안전한 트랜잭션(Transaction)

  ➔ 네트워크 프로그래밍 ≠ 웹 프로그래밍

REST Architecture

- REST : Representational State Transfer
    - CRUD(Create, Read, Update, Delete) using HTTP Requests (GET POST PUT DELETE)
    - Stateless(무상태성)
    - Resource : URI를 통해 특정 가능한 자산을 중심으로 시스템 구축
    - URI(Uniform Resource Identifier)
      : URL (Uniform Resource Locator)의 상위 개념
        - 단순하게, URI = URL + a (식별자 등)
- RESTful API : REST 구조를 따르는 API

REST Example

- Resource : https://www.mygame.com/users/
  - User resource를 저장하기 위한 실제 위치 (URL)

- URI : https://www.mygame.com/users/2718
  - 2718 식별자를 가지는 User 정보에 대한 URI

- REST :
  - GET https://www.mygame.com/users/2718 2718번 User 정보 조회
  - POST https://www.mygame.com/users/2719 2719번 User 생성
  - DELETE https://www.mygame.com/users/2718 2718번 User 삭제
  - PUT https://www.mygame.com/users/2719 2719번 User 정보 전체 수정
  - PATCH https://www.mygame.com/users/2719 2719번 User 정보 일부 수정

Representation of REST

- REST를 통한 요청과 응답에는 명령어(GET, POST, etc.) 외에도 많은 데이터가 들어간다.
  - Header
    - Media Type : Body에 해당하는 데이터가 무엇인지(text/html, application/json 등)
    - Body 길이가 몇 바이트인지
    - 인증서 구성
    - etc.
  - Body
    - 실제로 전달하고자 하는 데이터
    - 대다수의 경우 JSON(Key-Value Dict)이 기본

Django

- django 설치, Project 생성하기

- 서버 열기 : python manage.py runserver
  - 데이터베이스 반영 : python manage.py migrate

- 프로젝트 단위 설정
  - settings.py : app 추가
  - urls.py : 전체 사이트의 접속 url 관리 ➔

Django Create App

- python manage.py startapp main

- settings.py

```
INSTALLED_APPS = [
        'django.contrib.admin',
        'django.contrib.auth',
        'django.contrib.contenttype
        s', 'django.contrib.sessions',
        'django.contrib.messages',
        'django.contrib.staticfiles',
        'main',     #Add this!
]
```

views.py (in main folder)

```
from django.shortcuts import render # Create your views here.

def index(request):
        return render(request,'main/index.html')
```

Django Create App

- /main/templates/main/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Title</title>
</head>
<body>
        Hello, World!
</body>
</html>
```

/main/urls.py

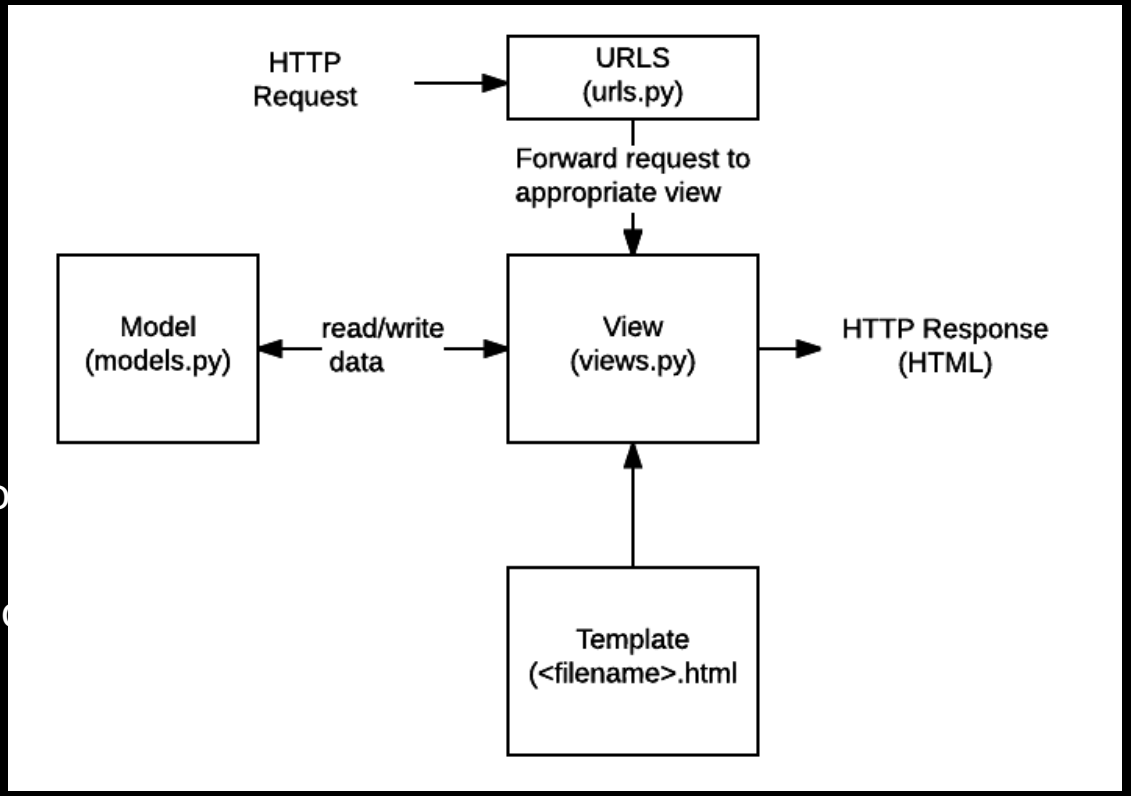
```
from django.urls import path
from . import views                    # . == this path
urlpatterns = [ path('', views.index, name='index'), ]
```

```
#urls.py (in Project Folder)
from django.contrib import admin
from django.urls import path, include


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

Django Pattern

- Django의 기본 구조
  - MVT Pattern
    - Model
    - View
    - Template
  - Model : Database mapping
  - View : Defines HTTP Response and API Endpo
  - Template : Rendering Documents with Functi
  - URLs : Entry of View



URL

- urls.py (in app directory) : 해당 앱에서 접근 가능한 주소의 목록
  - 127.0.0.1:8000/ 아래에

```
from django.urls import path
from . import views
urlpatterns = [
        path('',views.index,name='index'),
        path('hello',views.hello1,name='hello'),
        path('hello/',views.hello2), # /의 유무?
        path('a/b/c',views.index), # 의미없는 슬래시 구분?
]
```

View

- views.py : URL과 연결되어 호출 가능한 메서드의 목록
  - 호출 결과는 Response, Render (HTML Template), Redirect 등 다양함.

  - 함수 기반 뷰(Below) vs 클래스 기반 뷰?

```python
from django.shortcuts import render, redirect
from django.http import HttpResponse, JsonResponse, Http404
from django.views import View

def index(request):
	return render(request,'main/index.html')

def hello(request):
	return HttpResponse("<h1>Hello, World!</h1><h2>Well...</h2>")

def toGoogle(request):
	return redirect("https://google.com")
```

```python
// continued...
def not_found(request):
	raise Http404("Sorry... no page here.")

def api_example(request):
	data = {
		"message": "Hello, this is a JSON response",
		"status": "success"
	}
	return JsonResponse(data)
```

Function-based View

- Function-based :

```
def content(request):
    if request.method == 'POST':
        #process(request)
        return HttpResponse("Form submitted!")
    elif request.method == 'GET':
        data = {
            "message": "Hello, this is a JSON response",
            "status": "success"
        }
        return JsonResponse(data)
    else:
        return HttpResponse("Invalid request.")
```

Class-based View

- views.py

```
class contentView(View):
    def get(self, request):
        return HttpResponse("This is a GET request")

    def post(self, request):
        #process(request)
        return HttpResponse("This is a POST request")
```

- urls.py

```
urlpatterns = [
    path('cview',views.contentView.as_view())
]
```

Templates

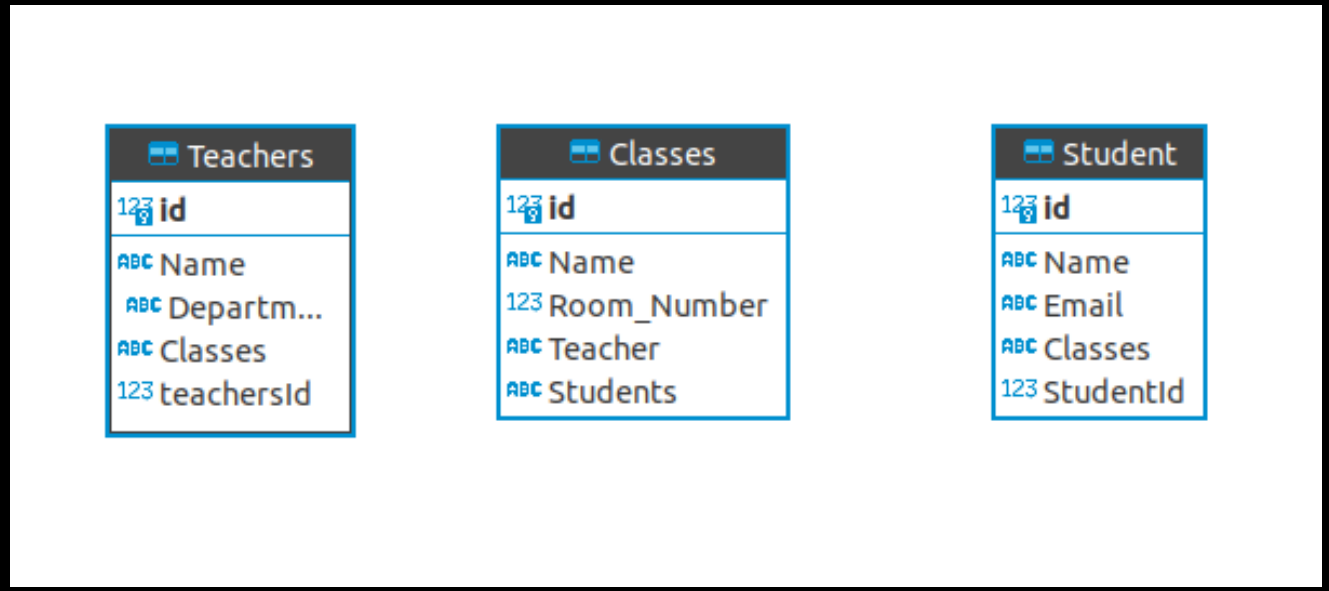
- 단순히 HTML 파일을 출력하는 것이 아닌, 특정한 데이터를 포함하는 양식을 렌더링(?)
    - index.html with Context:

```
def index(request):
    context = {
        'title': 'My Home Page’,
        'content': 'Welcome to my homepage!'
    }
    return render(request, 'main/index.html', context)
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ title }}</title>
</head>
<body>
    <h1>{{ content }}</h1>
</body>
</html>
```

Models

- 서버의 데이터 모델을 정의하기 위한 부분
  - 즉, 데이터베이스를 구성하고 상호작용하기 위한 요소들을 정의 ➔ Database Schema



DB : Database

- 데이터베이스는 데이터를 저장하는 곳이다?

  ➔ 단순히 저장이 목적인 경우 Storage를 사용

- 데이터베이스는 데이터를 특정한 구조에 따라 체계적으로 저장, 관리하는 것이다.

  - 데이터베이스의 구조는 여러 개의 표를 사용하는 관계형(Relational) DB가 일반적이다.

    - 관계형 DB를 관리하고 사용하기 위한 언어를 SQL (Structured Query Language)이라고 한다.

    - 표가 아닌 다른 구조 (노드 등)를 가지는 경우를 일반적으로 NoSQL 이라고 부른다.

models.py (schema) 예시

```
from django.db import models

class Author(models.Model): # 저자 테이블
        name = models.CharField(max_length=100)
        birth_date = models.DateField(null=True, blank=True)

        def __str__(self):
                return self.name # 관리 화면에서 저자의 이름을 표시


class Book(models.Model): # 책 테이블
        title = models.CharField(max_length=200)
        author = models.ForeignKey(Author, on_delete=models.CASCADE)
        published_date = models.DateField()
        isbn = models.CharField(max_length=13, unique=True)

        def __str__(self):
                return self.title # 관리 화면에서 책의 제목을 표시하기 위한 문자열
```

Forms

- Form : 웹에서 데이터를 입력하기 위한 양식
    - Template : 출력(렌더링)하기 위한 양식

- HTML의 <form> 태그를 사용해서 Form 페이지를 만들 수 있다.
    - 하지만, Django의 Model과 직접적인 연결을 위해서 별도의 클래스를 정의할 필요가 있다.

Models.py + Forms.py

models.py

```
from django.db import models

class Student(models.Model):
        name = models.CharField(max_length=100)
        student_id = models.CharField(max_length=10,
         unique=True)

        def __str__(self):
                return f"{self.name} ({self.student_id})"
```

forms.py

```
from django import forms
from .models import Student

class StudentForm(forms.ModelForm):
        class Meta:
                model = Student
                fields = ['name', 'student_id']
```

makemigrations, migrate, runserver

Form 클래스 내부의 Meta 클래스는 모델의 필드를 가져오는 역할

EOF