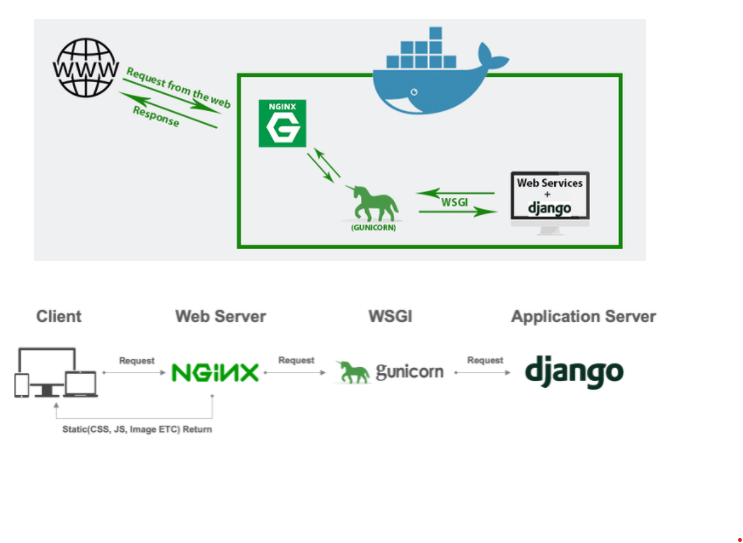
클라우드 환경에서 Django 배포하기

1. 프레임 워크 소개

우선적으로 웹사이트에 필요한 프레임워크가 무엇인지에 대해서 보고, 그에 따른 설정 방식에 대해서 서술!



일반적인 구조로 3-Tier로써 (WEB-WAS-DB) 에 대한 부분을 중점으로 생각하면 되고, 테스팅을 위해서 도커를 사용했지만, WAS 컨테이너만 올렸을 뿐 docker compose 에 대한 개념은 포함이 되어 있지 않으니 만약 클라우드 환경에서 EKS 를 기반으로 올리고 싶다면 이 점을 참고해 추가적으로 개발하는 것도 하나의 방법일 것! 간단한 프레임 워크들에 대해서 설명하자면

Nginx(WEB) : 웹 서버이고, 클라이언트에서 받아온 데이터를 리버스 프록시를 통해 WAS 서버에 넘겨주는 역할 이라 생각하면 됨

Gunicorn(WSGI) : 파이썬 기반의 WSGI 이며, Web server gateway interface 의 약자로 웹서버와 WAS 서버와 통신하기 위해 필요한 기반이라고 생각하면 됨

Django(WAS) : 파이썬 기반의 프레임 워크이며, 손쉽게 사이트를 구축 가능해, 짧은 프로젝트를 하기 위해 적합하다. 하지만 원래는 프론트와 백엔드를 구별해서 하는 것이 일반적(React, Vue.js 등)

1. 준비 환경

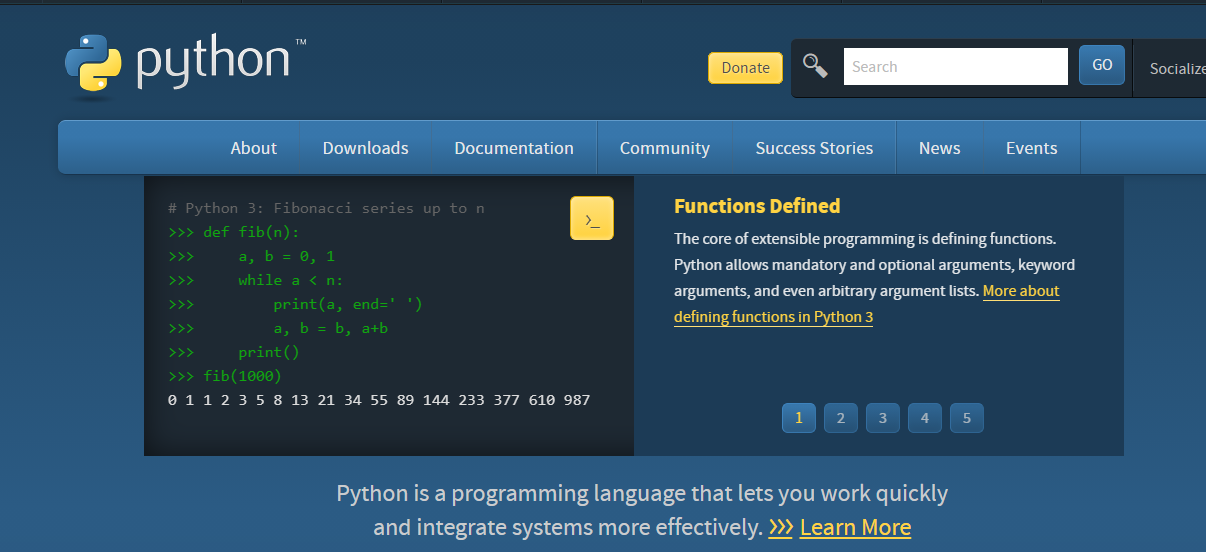
먼저 서버를 구축하기 전에 필요한 환경에 대해서 간략하게 기술

텍스트이(가) 표시된 사진

자동 생성된 설명

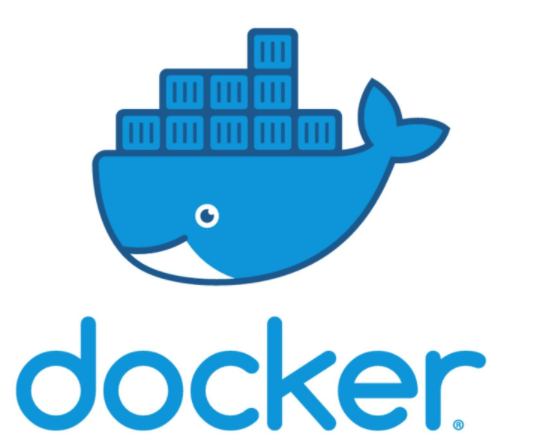
1. 개발 코드 작성에 필요한 프로그램(VSCODE)

개발할 때 문법 오류나 함수에 대해서 잡아주고 코드를 복사나 저장하기 편한 툴



1. 파이썬

개발용 언어 꼭 필요!



1. 도커

리눅스 환경에서 돌리기 위해서 사용하는 것, 근데 Django 를 테스팅 하는 정도면 충분히 로컬에서도 사용 가능하니 꼭 안 써도 상관 없음. 나는 리눅스 환경에서 하는 것을 기준으로 잡을 거라 도커를 사용할 것임

텍스트이(가) 표시된 사진

자동 생성된 설명

윈도우 환경에서 도커를 사용하기 위해서는 powershell 아니면 CMD 를 이용하면 되고

Docker login 을 이용해서 도커 허브 아이디와 비밀번호를 입력하면 사용 설정 끝



1. 우분투 OS

Red-hat 계열의 리눅스 운영체제인 우분투를 이용해서 해볼 예정.

모든 Django 와 nginx에 올라가는 운영체제는 이걸로 고정하고

클라우드(AWS)에 올릴 시에는 베스천호스트의 운영체제만 다름(amazon linux 2)

1. NGINX 시작하기(도커, 로컬은 안 해봄)

NGINX 를 구성하는 것은 이 차트에서는 매우 간편하게 이뤄진다. 하지만 여기서 주의해야 할 점은 NGINX 는 자체 서버를 돌리는 것이 목표가 아니라, WAS 서버와의 통신을 하기 위해서 연결하는 역할이라는 점을 참고해야 함.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 먼저 docker search nginx 를 이용해 nginx 가 설치된 이미지를 다운받는다. 여기서 주의할 점은 꼭 공식 사이트를 통해 다운 받도록 해야한다.(도커 허브를 통해서)

**텍스트이(가) 표시된 사진

자동 생성된 설명**

1. 다운 받은 이미지를 이용해서 컨테이너를 실행하는 명령어 사용 !

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 컨테이너 안으로 접속하기 위한 명령어를 이용해서 접속

텍스트이(가) 표시된 사진

자동 생성된 설명

1. Service 명령어를 통해서 nginx 데몬이 (상태/실행/중지) 를 설정 할 수 있다. 현재는 서버가 정상적으로 작동하는지에 대해서 확인하기 위함.
2. 원래는 apt-get install nginx를 통해서 까는 방법도 있음. 이게 더 편함(ubuntu 이미지)

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트, 명판이(가) 표시된 사진

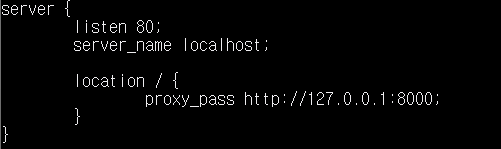
자동 생성된 설명

1. 설정 값은 Asia 의 Seoul 로 맞춰서 설치 진행

텍스트이(가) 표시된 사진

자동 생성된 설명

1. Nginx에서 proxy\_pass를 이용하기 위해서 설정 파일을 수정해야 함. Nginx.conf를 확인하면 conf.d 아래에 있는 설정파일을 받아들인다 써져 있음. 따라서 conf.d 에 내가 원하는 설정 값을 넣어서 서버가 실행되도록 해야함.



1. /etc/nginx/conf.d 로 들어가서 default.conf 라는 위와 같은 파일 하나 생성

**“Listen 80”** 의 의미는 80번의 포트번호로 들어올 경우 서버를 접속할 수 있다는 의미.

**“Server\_name localhost”** 은 DNS 와 같은 개념이라고 생각하면 된다. 인터넷 창에서 저 이름을 쳐야지 접속 할 수 있다.

**“Location / “** 의 의미는 url 의 끝 부분이 **“ / ”**로 끝나는 주소는 아래 proxy\_pass 로 보낸다 라는 의미임. (나중에 클라우드에 nginx를 올릴 경우 다시 나올 예정) 그리고 저장.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 컨테이너에서는 포트번호를 열어줘야 들어올 수 있는 구조이다. 그래서 임의의 포트 번호인 8000번을 이용해서 현재 컨테이너의 80번 포트로 포트 포워딩 하겠다는 의미의 -p 8000:80 이라고 입력 한 후 컨테이너를 재시작 하면 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 정상적으로 진행되는 것을 확인 할 수 있다.(“127.0.0.1:8000” 입력)
2. Service nginx (status/stop/start/restart) -> nginx 서버와 관련된 명령어(상태/정지/시작)

Nginx -t -> nginx 가 컴파일 될 수 있는지 상태 체크(정상이면 성공이라 뜸)

1. Django 설치 및 가동

**텍스트이(가) 표시된 사진

자동 생성된 설명**

1. 먼저 파이썬을 설치

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 파이썬 라이브러리를 설치하기 위한 pip를 설치

텍스트이(가) 표시된 사진

자동 생성된 설명

1. Pip를 이용해서 Django 와 gunicorn 설치

텍스트이(가) 표시된 사진

자동 생성된 설명

1. Django 가 설치되었다면 위에의 명령어를 입력해 프로젝트를 생성한다.
2. Project 는 설정 파일들이 담겨져 있는 전체적인 구조를 담당하는 역할이라고 생각하면 되고, 사이트를 구성하는 것들은 app 들로 한다고 보면 된다. 프로젝트 명과 app 명들은 원하는 이름으로 설정하면 됨.  
   Django-admin startproject “원하는 프로젝트명”  
   Django-admin startapp “원하는 app명”

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 정상적으로 설치가 되었는지 확인하기 위해선 일단 manage.py를 이용해서 서버를 가동해보면 된다.(포트번호는 원하는 대로 설정가능)

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 로컬 인터넷 창에서 “127.0.0.1:8000” 이나 “localhost:8000” 를 입력해서 접속
2. Gunicorn 연동(Gunicorn + Django)

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 시스템 데몬을 이용하기 위해서 systemctl를 설치하도록 한다. (데몬이란? 백그라운드에서 돌아가는 서버라고 생각하면 됨)

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

1. 먼저 gunicorn 을 통해서 위에 Django 서버를 돌릴 수 있는지 확인 과정을 거친다.(이 과정은 manage.py 파일이 있는 폴더 내에서 실행하면 정상적으로 작동할 것)



[Unit]

Description=gunicorn daemon

After=network.target

[Service]

User=root

Group=root

WorkingDirectory=/root/final\_was

ExecStart= /usr/local/bin/gunicorn --bind 127.0.0.1:8000 final\_was.wsgi:application

[Install]

WantedBy=multi-user.target

1. 그렇다면 위에 있는 gunicorn의 서비스 데몬을 자동화 하기 위해서 systemctl에서 설정파일을 수정해 서버가 열리는 동시에 실행 되도록 만듬.(프로젝트명은 “final\_was” 로 변경)

$ sudo systemctl daemon-reload # 작성한 서비스 적용

$ sudo systemctl start gunicorn # 작성한 서비스 실행하기

$ sudo systemctl enable gunicorn # 서버 재시작시 자동으로 실행

$ sudo systemctl status gunicorn.service # 실행한 서비스 상태 보기

$ sudo systemctl stop gunicorn # 서비스 중지

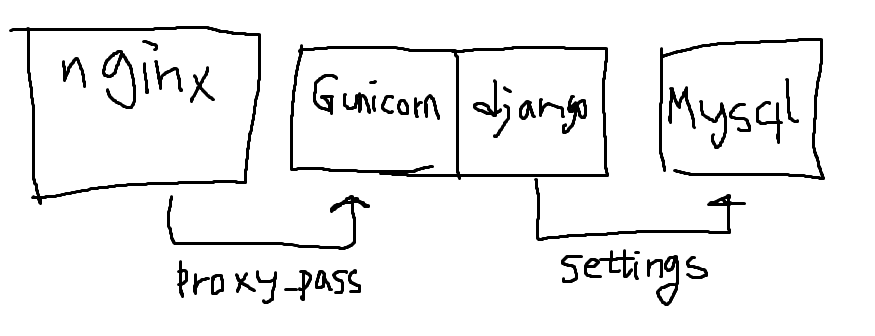
$ sudo systemctl restart gunicorn # 서비스 재시작

1. 위의 설정내용을 바탕으로 서비스를 등록이 되었으면 자동으로 실행하도록 설정

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 위와 같은 사진이 나오면 정상적으로 작동되며, 홈페이지도 정상적으로 접속 가능.
2. Nginx 연동(Nginx + Gunicorn)



이제 nginx를 통해 gunicorn 서버로 들어 갈 수 있도록 프록시 설정을 변경할 예정

텍스트이(가) 표시된 사진

자동 생성된 설명

Listen 포트를 80번으로 한 다음 nginx의 주소를 입력한 경우 gunicorn으로 매핑할 예정

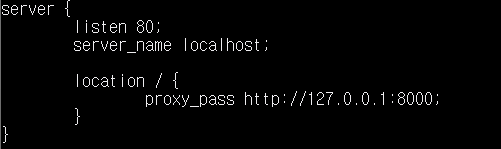
텍스트이(가) 표시된 사진

자동 생성된 설명

여기 그림에서 사용할 폴더는 **“sites-available”** 과 **“sites-enabled”**

일단 지난번에 진행했던 conf.d/default.conf 는 **제거**하고 진행





final\_was에 위에 적었던 값 다시 재입력하고 저장한다. 그런 다음에 아래와 명령어 입력

$ ln -s /etc/nginx/sites-available/final\_was /etc/nginx/sites-enabled/final\_was

$ rm -rf /etc/nginx/sites-enabled/default

위의 명령어를 입력 한 후에 접속을 시도해보면, nginx의 주소(끝이 / 로 끝나는)를 입력시에 Django 의 서버로 매핑이 되는 것을 확인 할 수 있다. 클라우드 상에서 3-tier 로 구성하는 경우 위의 같은 구분을 통해 서버로 넘겨주는 활동이 자주 일어나니 꼭 참고.

(만약 도커 상에서는 안된다고 하면 ec2를 이용해서 해보면 될 것, 도커는 포트의 문제 떄문에 nginx의 코드를 칠 경우 Bad Gateway 문제가 나올 것)

1. Django 작업

`-- final\_was

|-- final\_was

| |-- \_\_init\_\_.py

| |-- \_\_pycache\_\_

| | |-- \_\_init\_\_.cpython-38.pyc

| | |-- settings.cpython-38.pyc

| | |-- urls.cpython-38.pyc

| | `-- wsgi.cpython-38.pyc

| |-- asgi.py

| |-- settings.py ## Django 의 설정값 변경 가능(데이터베이스 연결, static, app)

| |-- urls.py ## 주소창에 대한 부분 매핑 가능

| `-- wsgi.py ## Gunicorn 과 관련된 부분, 나중에 static 과 관련

|-- blog ## 포토폴리오 app

|-- mypage ## API 연동 전용 app

|-- user ## 회원가입에 관련된 app

`-- manage.py ## 서버를 구동하거나, 데이터베이스 스키마 이전에 사용

`-- db.sqlite3 ## Django의 기본적인 데이터베이스

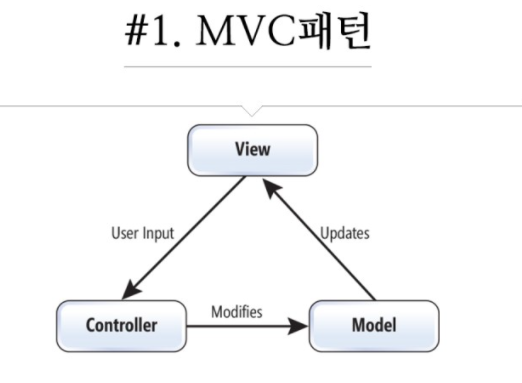
위와 같은 구조로 진행이 되며, 트리구조로 표현하기에는 파일과 폴더가 너무 많아서 중요한 부분(주석 단 부분)만 참고해서 코딩하는 방식에 대해서 서술하도록 하겠음.

파일에 대해서는 github에 올릴 것이므로, 참고하면 됨.

**MVC 모델**

Django 는 대표적인 MVC 모델이라고 하는데, 여기서 MVC 라고 하면

Model , View , Control 의 세가지의 약어라고 생각하면 된다.



여기서 Model 은 간단하게 말해서 “데이터베이스 스키마”에 관한 것 생각하고

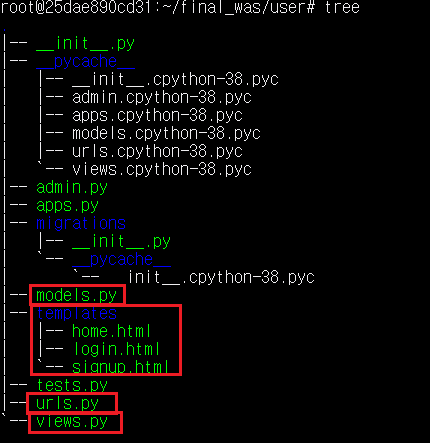
View 는 우리가 보여지는 화면에서의 데이터 처리를 생각하면 되고

Controller 는 이를 처리하기 위한 컨트롤 구조라고 생각하면 된다.

우리가 개발을 하게 될 Django 프로젝트에서는 위와 같은 구조가 거의 동일하게 작용한다고 보면 되고, 이를 담당하는 파일들이 있다.

이 파일들에 대한 처리 과정을 이해하는 것이 우선적으로 깃허브에 있는 파일들에 대한 개념을 잡는데에 우선되야 할 것.

트리구조를 예로 들어서 보여주면



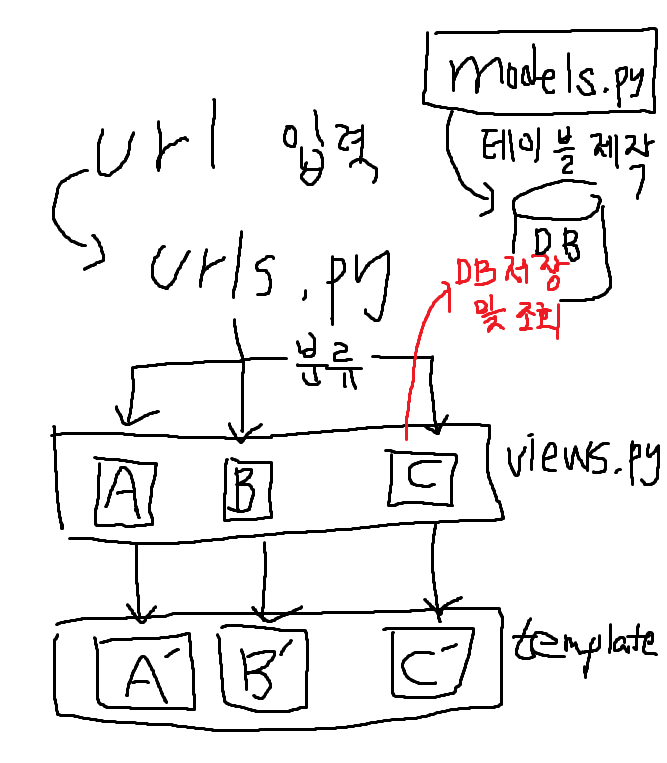
빨간색 네모박스안에 있는 파일들이 위에서 언급한 MVC 패턴에 부합하는 파일이라고 보면 된다.

**Urls.py : url 주소에 맞춰서 함수를 실행하는 것이 가능(control)**

**Views.py : 데이터 처리와 관련된 내용(view)**

**Models.py : 데이터베이스 스키마에 관련된 내용(model)**

다른 apps 들도 위와 같은 구조로 되어 있으며 간단하게 그림으로 묘사해보면



위와 같은 구조로 진행되는 것을 볼 수 있다. Models.py 를 통해 미리 규명한 데이터베이스 테이블에 대한 스키마를 저장하도록 하며 , views.py 에서는 그를 통해 데이터를 조회 및 처리하는 역할로써 해당 처리가 완료하면 templates 에 있는 html 에 데이터를 보내줌으로써, 우리가 보는 웹페이지를 구성 할 수 있도록 만든다.

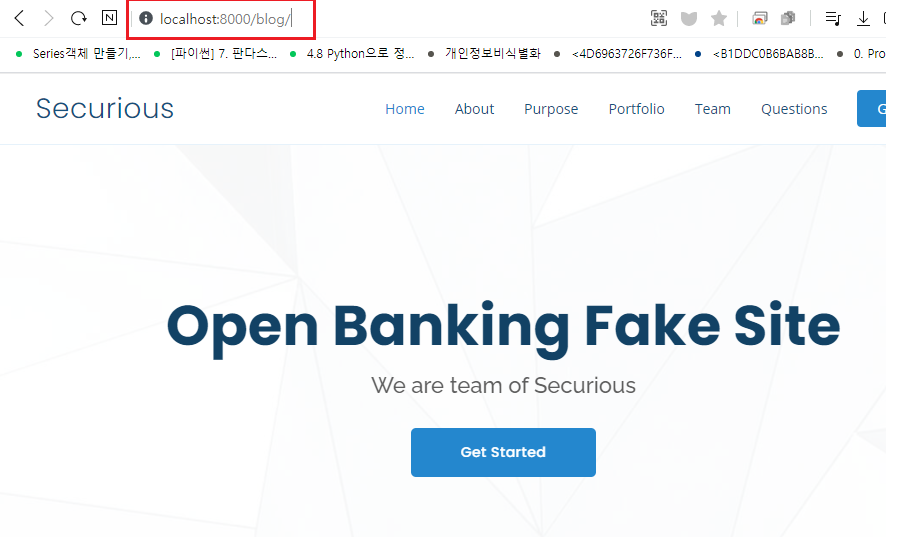
**urls.py**

텍스트이(가) 표시된 사진

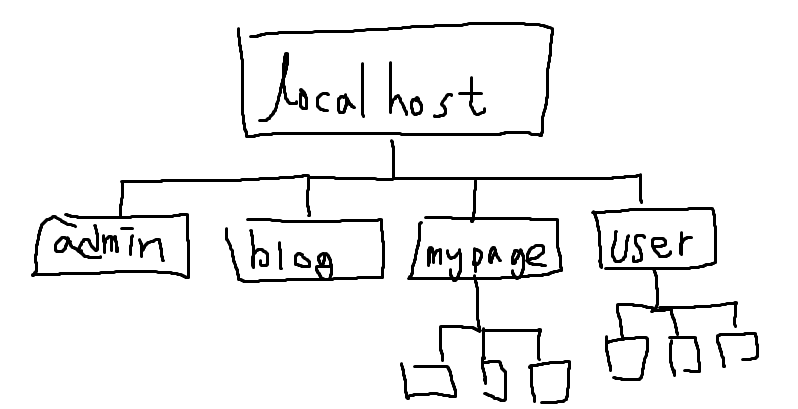
자동 생성된 설명

Urls.py 에서 하는 역할은 우리가 웹페이지에 입력하는 주소를 통해 원하는 위치로 이동할 수 있도록 만드는 역할이다. 먼저 프로젝트에서 앱으로 넘어가는 구조로 구성했으며,

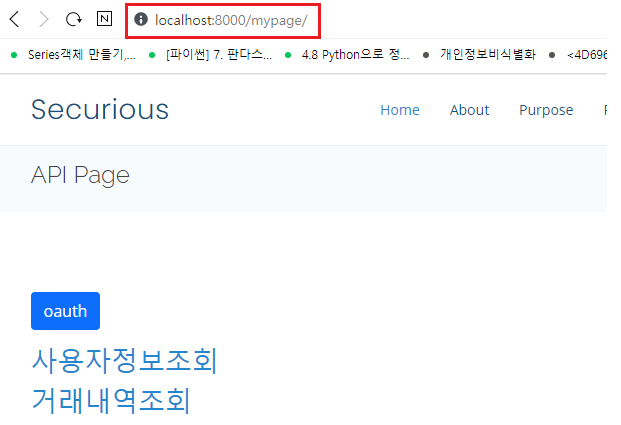
Path 안에 있는 값들을 통해 원하는 위치로 이동 할 수 있다. Include 를 이용한 것은 app 내에서 구체적인 주소를 매핑하기 위해서 앱 내에서도 urls.py 를 따로 만들어 처리 할 수 있도록 만든 것이다.



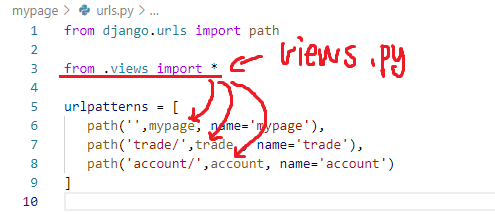
위에 쳐져 있는 빨간색 박스의 주소를 잘 보면, 맨 마지막 문구가 “blog/” 로 끝나는 문구로, 그에 따라 blog app에 해당되는 데이터를 표현할 수 있도록 위치가 이동된 것을 볼 수 있다.



이런 구조로 되어 있다 이해하면 된다.



위 사이트를 기준으로 설명하면 mypage로 접근하기 위해서 urls를 구성했으며, 구체적인 정보를 처리하기 위해서 app 내에서 새로운 urls.py를 만들어 처리하도록 했다.



Mypage의 urls.py 이고, path 함수에 들어가 있는 인자 값을 보면

첫번째 인자에는 앞선 경우와 동일하게 url 주소에 관련된 매핑이 되어 있다고 하면 두번째 인자에서는 views.py 에서의 데이터 처리에 관련된 값과 매핑이 되어있다 생각하면 된다. 위의 urls.py 에 의해서 해당 페이지로 접속시에 views.py 에 있는 함수들이 적용이 되게 되고, 그를 통해 홈페이지를 확인 할 수 있는 구조이다.

**Views.py**

텍스트이(가) 표시된 사진

자동 생성된 설명

Mypage의 Views.py를 기준으로 보면



앞선 urls.py 에서 사용되었던 함수의 이름이 views.py 와 동일하게 파란색 박스에 표시가 되어 있는 것을 확인 할 수 있다.

따라서 urls.py 에서 매핑 된 주소는 views 의 함수내에 있는 데이터 처리 과정을 거친 후에 반환되는 과정이라고 이해하면 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

데이터를 반환하는 과정은 데이터베이스에서 데이터를 가져온 다음 해당 html 과 함께 데이터를 넘겨주는 방식이 일반적이다.(objects.all() <- 데이터베이스의 정보를 가져온다)



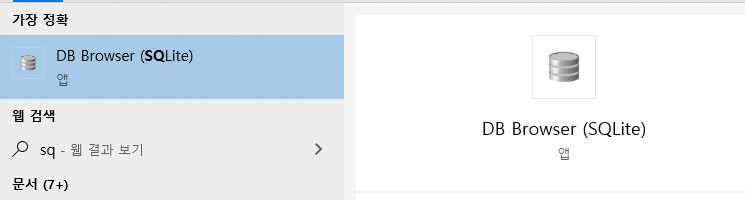
함수에 대해서 확인하던 중에 mypage.save() 라는 커맨드와 함께 처리가 되어 있는 것을 확인 할 수 있는데, 이것은 데이터베이스에 저장하는 과정중의 하나로써 views에서 데이터 처리와 동시에 데이터베이스에 저장도 가능하다.

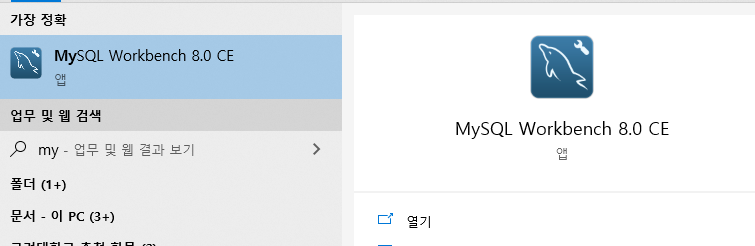
**Models.py**

텍스트이(가) 표시된 사진

자동 생성된 설명

Mypage에 있는 models.py 를 확인해보면 데이터베이스에 대한 스키마가 저장되어 있는 것을 확인할 수 있다. 스키마를 등록하기 위해서는 manage.py 에서의 migrate 과정이 필요하며, 그 과정이 마무리가 되었을 경우 데이터베이스를 자유자재로 사용 가능하다.





지금은 Django의 기본 내장되어 있는 데이터베이스에 대한 고려이므로, Sqlite에 대한 브라우져로 데이터를 확인 가능하지만, 일반적으로 아래 그림과 같은 워크벤치를 이용해 데이터베이스를 조회 및 처리가 가능하다.

테이블이(가) 표시된 사진

자동 생성된 설명

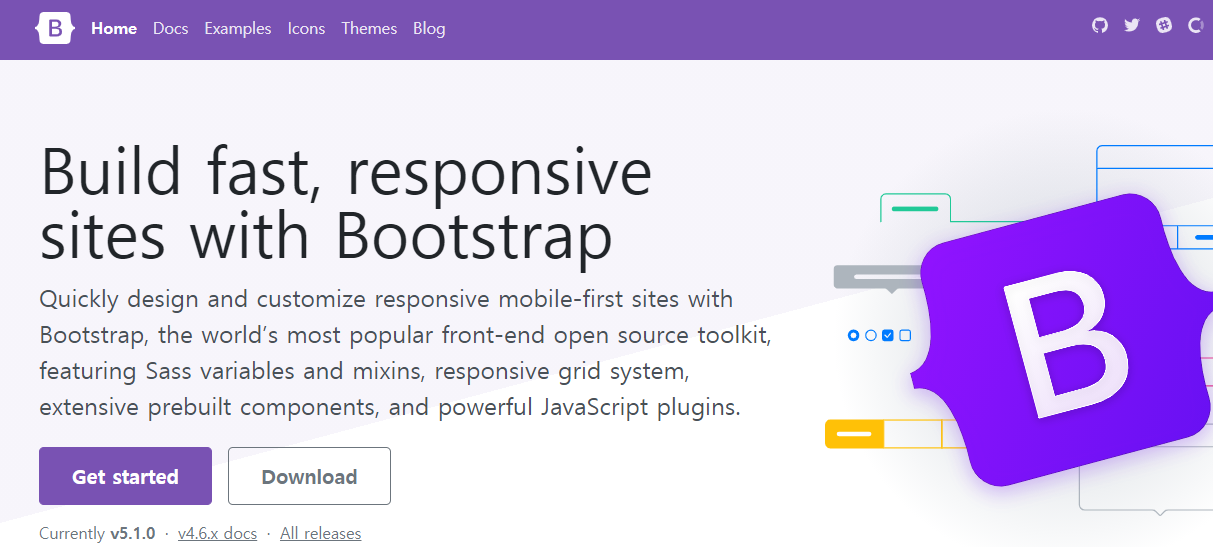
해당 데이터베이스를 확인해보면 models.py에서 등록한 스키마들에 대한 기록이

남아있는 것을 확인할 수 있다.



실제로 클라우드 상에서 서버를 운용할 경우 위와 같은 MySQL에 저장된 내용을 확인가능.

**Static 적용**



홈페이지를 꾸밀 때 가장 중요한 요소는 프론트 엔드와 백엔드에 관련된 부분인데, 프로젝트를 하면서 가장 시간을 잡아 먹는 요소도 프론트 엔드다.

예쁜 홈페이지를 꾸미기 위해서 디자인을 고려해야 하고, 주제에 맞는 특성을 고려해서 선택해야 하므로 생각보다 손쉽게 진행이 안될 경우가 생겼었는데, 우선 홈페이지를 꾸미기 위해서 어떤 과정을 거쳐야 하는지 django에서 처리 과정을 간단하게 서술

텍스트이(가) 표시된 사진

자동 생성된 설명

앞선 views.py 에서 return 되는 값을 보면 html문에 대해 반환을 하는 것을 확인할 수 있다.

이때 html에 대한 부분이 어디에 저장되어 있는지 확인해야 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Views.py 와 같은 폴더 내에 templates 폴더가 있고, 그곳에서 html을 반환하는 형식이다.(app을 새로 생성시에 html을 적용하기 위해서 templates를 새로 만들어야 함)

텍스트이(가) 표시된 사진

자동 생성된 설명

Static 파일을 html 파일에 저장하기 위해서는 서버가 static 파일에 대한 부분을 찾을 수 있도록 매핑하게 만드는 것이 제일 중요하며, 그를 하기 위해선 세가지의 주의점이 있다.

1. Html 파일의 상단에 “{% load static %}” 입력
2. Static 파일을 찾을 수 있도록 “{% static “static 파일의 위치” %}” 로 변경
3. Settings.py 의 설정 값을 변경 하는 것

텍스트이(가) 표시된 사진

자동 생성된 설명

해당되는 Static파일의 위치를 STATICFILES\_DIRS 에다가 app 이름과 함께 위의 사진과 저장

위 세가지 사항을 준수하면 외부에서 가져온 프론트 엔드 파일을 활용해 내가 원하는 웹사이트를 구축할 수 있다.

**\*추가**

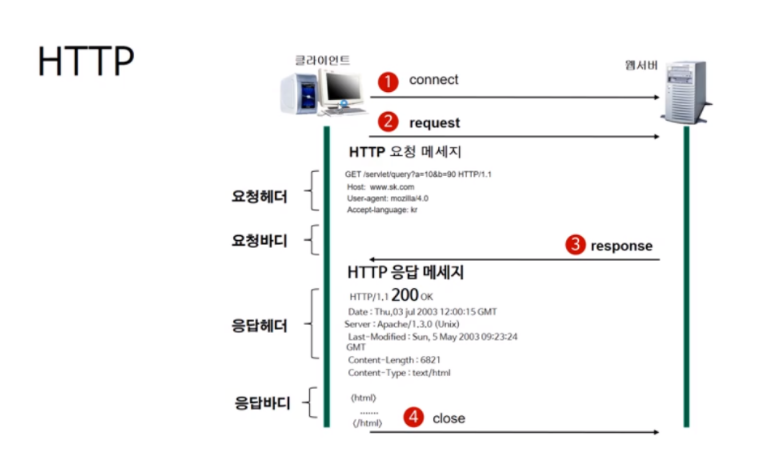
텍스트이(가) 표시된 사진

자동 생성된 설명

Gunicorn에서 django로 바인딩 하면서 static 파일에 대한 처리가 안되는 경향이 있는데, 그것은 프로젝트에 있는 wsgi.py 를 위와 같이 수정하면 됨(apt-get install Django-static)

**HTML 프로토콜**

웹 서비스 개발에서 사용되는 메서드인데, http 프로토콜에서는 데이터 전송을 위해 지원하는 7가지 메서드 중 2가지의 메서드에 대해서 다뤄볼 예정(GET,POST)



정확하게 말하면 요청헤더 부분의 메소드 부분에 해당됨.



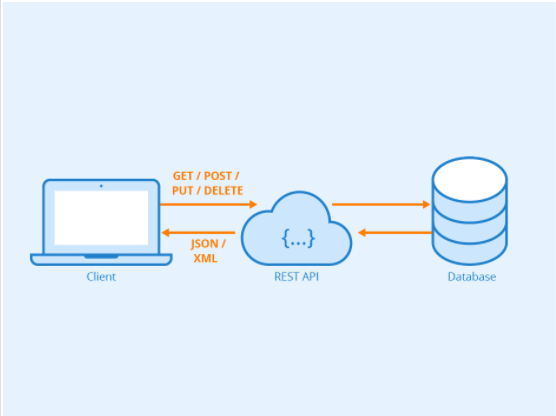
회원가입(user/signup.html)에 사용된 form 태그의 method 를 확인해보면 POST 형식으로 보내는 것을 확인 할 수 있으며, 이를 통해 버튼을 누를 경우 POST 형식으로 데이터가 전송 되는 형식으로 보면 된다.



User 의 views 파일을 통해 구체적인 프로세스를 확인해보면, 데이터를 처리하도록 만드는 버튼을 누를 시에 POST 형식의 request 로 데이터가 전송이 되는 것이고 , 그것을 views.py 에서 처리할 수 있도록 한 코드

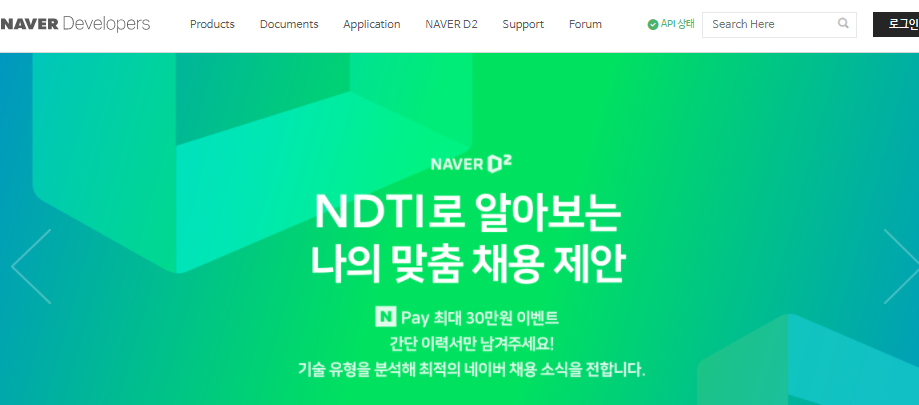
그리고 그에 대한 처리 값을 반환하도록 만든 개념이라 보면 됨.

**API 사용하기**



위에서 클라이언트가 보낸 요청에 대해서 처리할 수 있도록 예시를 통해 확인해봤는데, 그를 통해 API 활용에 대해서 간단하게 서술

인터넷에서 데이터를 주고 받는데 사용되는 프로토콜은 HTTP API 형식과 RESTful API 형식 이라고 생각하면 됨. 데이터를 주고 받는데 특성화한 형식이라고 보면 되고, 해당 프로토콜을 사용할 시에 올바른 형식으로 사용하는 것이 중요함!



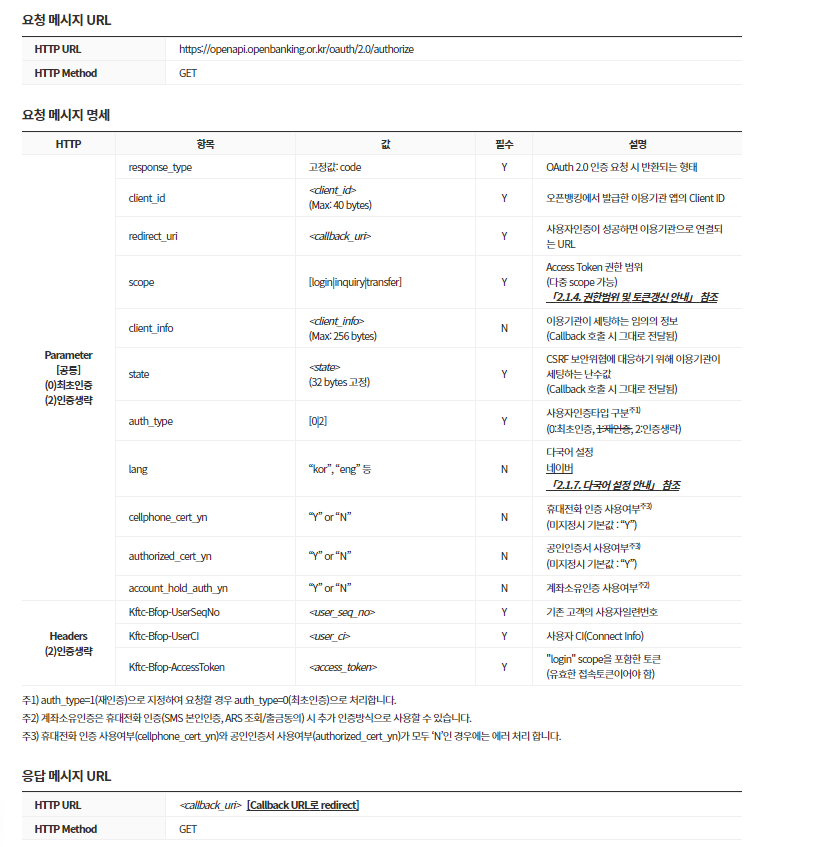
오픈API 를 사용해서 내가 원하는 정보를 가져올 수 있도록 구성할 수 있으며, 또한 그를 활용해서 서비스를 제공하는 것도 가능!

위 프로젝트에서는 금융결제원의 API를 이용했으므로, 그것을 중점으로 파악해보겠슴



금융결제원에서 API 를 사용하기 위해선 위와 같은 프로세스를 거쳐야 하는데, 일단 구체적인 내용에 대해서는 금융결재원에서 제시하는 문서자료를 통해서 확인 가능

[금융결제원 오픈API 개발자사이트 (kftc.or.kr)](https://developers.kftc.or.kr/dev/doc/open-banking)



사용자 인증을 받기 위해서 필요한 값들이 이렇게 있는데, 이것이 나타내는 것은 사실

위에서 **인터넷에서 데이터를 주고 받는 형식(HTTP,REST) 를 이용해서 데이터를 전송하라는 의미로 이해하고 진행하면 됨.**

텍스트이(가) 표시된 사진

자동 생성된 설명

인증이 다 완료되었다고 가정하고 토근 발급 API에 대한 활용을 예로 들어 설명하면

텍스트이(가) 표시된 사진

자동 생성된 설명

Request 에 필요한 데이터는 해당 문서에 구체적으로 적혀져 있음.

(code, client\_id,client\_secret,redirect\_url,grant\_type)

그렇다면 위의 조건을 충족할 수 있는 **데이터를 직접 만들어서 전송하면 됨**



먼저 파이썬 코드에서 데이터 전송을 위한 requests 라이브러리를 이용해서 금융결제원에서 요구하는 데이터 형식을 맞춰 데이터를 만들고 전송 메소드에 맞춰서 전송하고 그에 대한 반환 값을 이용해 처리하는 구조라고 보면 됨.

텍스트이(가) 표시된 사진

자동 생성된 설명

그 결과로 이런 식으로 처리 가능

구체적인 코딩 내용에 대한 부분은 너무 상세하고 많아서 다 적지 못했지만, 프로젝트 하면서 중요한 부분에 대한 내용에 대해서 서술하도록 노력했고, 궁금한점이 있으면 연락주거나 댓글 ㄱㄱ!