Django

웹 서버 & 웹 클라이언트

웹 서버와 웹 클라이언트
- HTTP(S) 프로토콜로 통신하는 클라이언트와 서버를 개발하는 것
- 보통은 웹 서버를 개발하는 경우가 많아서 파이썬 웹 프로그래밍이라고 하면 장고와 같은 웹 프레임워크를 사용한 웹 서버 개발을 의미한다.
- HTTP는 통신 방식으로 TCP/IP 프로토콜 위에서 동작. TCP/IP 동작에 필수적인 IP 주소를 가져야 함

웹 클라이언트
- 웹 브라우저를 사용하여 요청
- 리눅스 curl 명령을 사용하여 요청 : curl http://www.example.com
- Telnet을 사용하여 요청 : telnet www.example.com 80
- 직접 만든 클라이언트로 요청 :
  code example.py > python example.py
  python -c "import urllib.request;print(urllib.request.urlopen('http://www.example.com').read().decode('utf-8')) ≫

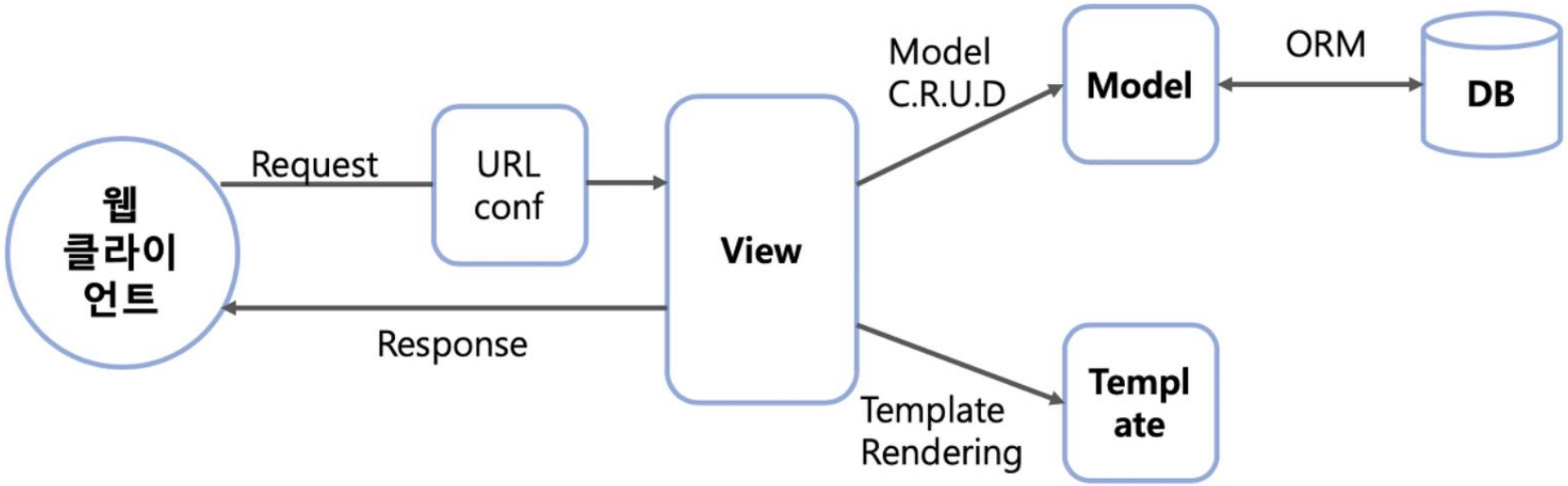
MVT

MVT 패턴이란 ?

Django framework에서 기본적으로 사용되는 design pattern이며 이 design pattern에서 유명한 것이 M(model).V(view).C(controller) 패턴이다. 하지만 django에선 V와 C 부분이 각각 Template, View(django view)로 치환

장고 철학 : https://docs.djangoproject.com/ko/3.0/misc/design-philosophies

- 클라이언트로부터 요청을 받으면 URLconf를 이용하여 URL을 분석합니다.
- URL 분석 결과를 통해 해당 URL에 대한 처리를 담당할 뷰를 결정합니다.
- 뷰는 자신의 로직을 실행하면서, 만일 데이터베이스 처리가 필요하면 모델을 통해 처리하고 그 결과를 반환받습니다.
- 뷰는 자신의 로직 처리가 끝나면 템플릿을 사용하여 클라이언트에 전송할 HTML파일을 생성합니다.
- 뷰는 최종 결과로 HTML 파일을 클라이언트에 보내 응답합니다.



MVT

https://butter-shower.tistory.com/49

MVT 코딩 순서
1.프로젝트 뼈대 만들기 : 프로젝트 및 앱 개발에 필요한 디렉토리와 파일 생성
2.모델 코딩하기 : 테이블 관련 사항을 개발(models.py, admin.py 파일)
3.URLconf 코딩하기 : URL 및 뷰 매핑 관계를 정의 (urls.py 파일)
4.템플릿 코딩하기 : 화면 UI 개발 (templates/디렉토리 하위의 *.html 파일들)
5.뷰 코딩하기 : 어플리케이션 로직 개발 (views.py 파일)

Model :
데이터베이스 테이블을 정의하는 장고의 클래스를 의미하며 models.py 파일에 테이블 관련 사항들을 정의
그 외 관련 변수 및 메소드를 추가적으로 정의할 수 있으며 ORM 방식에 기반해 클래스의 특징인 변수와 메소드를 포함
URLconf :
클라이언트로부터 받은 요청에 들어있는 URL이 url.py 파일에 처리함수(뷰)와 매핑하는 파이썬 코드를 작성
“ path('articles/2003/', views.special_case_2003)”에서 article/2003 부분이 URL이고 views.special_case_2003 부분이 처리함수(뷰)이다.
View :
웹 요청을 받아서 데이터 베이스 접속 등 해당 어플리케이션의 로직에 맞는 처리를 하고, 그 결과 데이터를 HTML로 변환하기 위하여 템플릿 처리를 한 후에 최종 HTML로 된 응답 데이터를 웹 클라이언트로 반환하는 역할을 한다.
Template
화면 UI 정의이며 템플릿 파일은 *.html 확장자를 가지며, 장고의 템플릿 시스템 문법에 맞게 작성한다. 유의할 점은 템플릿 파일을 적절한 디렉토리에 위치시켜야 한다는 점이다. 장고에서 템플릿 파일을 찾을 때는 TEMPLATE_DIRS 및 INSTALLED_APPS에서 지정된 앱의 디렉토리를 검색한다. 이 항목들은 프로젝트 설정 파일인 settings.py에 정의되어 있다.

Templates

backend 항목은 사용할 탬플릿 엔진을 지정
장고의 자체 템플릿 : django.template.backends.django.DjangoTemplates',
프로젝트 템플릿 파일 위치 : 'DIRS': [os.path.join(BASE_DIR, 'templates')]
애플리케이션 내의 템플릿 디렉토리에서도 찾을 지 여부를 지정 : 'APP_DIRS': True

개발자는 템플릿 문법에 따라 템플릿 코드만 작성하며 최종 HTML 텍스트 파일은 장고가 알아서 만들어 준다.

제네릭 뷰의 디폴트 템플릿 : 제네릭 뷰에서 template_name 속성을 지정하지 않는 경우에 사용하는 템플릿 파일 이름

템플릿 내부 처리 과정
- 템플릿 설정에 따라 엔진 객체를 생성
- 템플릿 파일 로딩 및 템플릿 객체를 생성
  템플릿 파일을 찾은 결과는 하나의 파일이지만 {% extends %} 또는 {% include %}태그가 있는 경우는 여러 개의 파일을 찾게 됨
- 렌더링을 실시해 최종 HTML 텍스트 파일을 생성

Templates

backend 항목은 사용할 탬플릿 엔진을 지정
장고의 자체 템플릿 : django.template.backends.django.DjangoTemplates',
프로젝트 템플릿 파일 위치 : 'DIRS': [os.path.join(BASE_DIR, 'templates')]
애플리케이션 내의 템플릿 디렉토리에서도 찾을 지 여부를 지정 : 'APP_DIRS': True

개발자는 템플릿 문법에 따라 템플릿 코드만 작성하며 최종 HTML 텍스트 파일은 장고가 알아서 만들어 준다.

제네릭 뷰의 디폴트 템플릿 : 제네릭 뷰에서 template_name 속성을 지정하지 않는 경우에 사용하는 템플릿 파일 이름

템플릿 태그는 파이썬을 HTML로 바꿔주어, 빠르고 쉽게 동적인 웹 사이트를 만들 수 있게한다.
extends : 템플릿 상속
- load : 빌트인 템플릿태그/필터 외에 추가 로딩
- 각 장고앱의 templatetags/ 디렉토리 내, 파일명을 지정
  (django/contrib/humanize/tempaltetags/humanize.py)

include : 템플릿 가져오기
- 현재의 context가 그대로 전달.
- with옵션을 통해 추가 키워드 인자 전달
- only 추가옵션을 통해 지정

block … endblock : 블락 영역 지정
- 템플릿 상속을 위한 영역 지정

comment … endcomment : 주석 영역 지정

Templates

Django의 템플릿시스템은 템플릿코드를 해석해서 템플릿 파일을 만드는데 이과정을 렌더링이라고 하며 결과물인 템플릿 파일은 HTML, XML, JSON 등의 텍스트 파일이다.

템플릿 언어란 파이썬 변수 및 문법을 html 안에서 쓸 수 있도록 장고에서 제공해주는 언어
변수를 템플릿 언어로 쓰기 위해서는 {{ 변수 }} , 다음과 같은 형태로 표현. 또한 ' . ' 을 이용해서 변수의 속성으로 접근. {{ blog.title }}
필터로 인해서 변수에 여러가지 효과를 줄 수 있으며 필터는 | (파이프) 를 이용해서 적용할 수 있다. {{ 변수 | length }}
태그는 {% tag %} 다음과 같은 모양을 하고 있으며 제일 많이 쓰는 것은 반복문과 제어문이다.
Built-in Template tags
템플릿 태그는 {% %}로 구성되어 있습니다.
이전 토픽에서 사용하였던 {% load static %}와 같이 사용할 수도 있고, 위의 예제에서 볼 수 있듯이 if문 또는 for문와 같이 흐름을 제어할 수 있도록 작성할 수도 있습니다.
HTML 문서는 프로그래밍 언어가 아니고 마크업 언어이기 때문에 단지 문서를 웹에서 띄워주는 역할을 하는 것이라고 배웠습니다. 그렇기 때문에 HTML 자체는 프로그래밍적 로직을 구현할 수 없습니다만 if문, for문와 같은 템플릿 태그를 사용한다면 가능할 수 있습니다.
따라서 앞에서 배웠던 템플릿 변수와 템플릿 태그를 활용한다면 간단한 처리도 가능하게 됩니다.

{% extends %}와 같이 단독으로 사용할 수 있는 템플릿 태그들도 있지만, {% if %} 처럼 뒤에 {% endif %} 템플릿 태그를 반드시 닫아주어야 하는 것들도 있습니다.
템플릿 코멘트 : 한줄 코멘트 {# #} , 여러 줄을 코멘트 {% comment %} {% endcomment %}

애플리케이션 설계

장고 프로젝트 생성 : django-admin startproject do_it_django_prj .
데이터베이스 생성 : python manage.py makemigrations > python manage.py migrate
관리자 계정 생성 : python manage.py createsuperuser

앱 생성 : python manage.py startapp blog single_pages
settings.py에 blog 앱과 single_pages 앱 등록
테이블 모습 확인 : python manage.py runserver

관리자 페이지에서 첫 포스트 작성
blog/admin.py : 관리자 페이지에 post 모델 등록
blog/models.py : Post 모델 정의

표지판 역할 : urls.py
do_it_django_prj/urls.py : 사용자가 웹 방문 시 어떤 페이지로 들어가야 하는지 알려줌
blog/urls.py : urlpatterns 리스트에 url과 url이 들어올 때 어떻게 처리할지 명시

애플리케이션 설계

FBV(Function based view)로 페이지 만들기
path('', views.index) : url이 'blog/'로 끝나면 views.py에 정의되어 있는 index() 함수를 실행
blog/views.py : 함수 정의
blog/templates/bolg/index.html: 템플릿 파일 작성

CBV(Class based view)로 페이지 만들기
path('', views.PostList.as_view()),: url이 'blog/'로 끝나면
blog/views.py : ListView를 사용, model = Post라고 선언
blog/templates/bolg/post_list.html: 템플릿 파일 작성
index() 함수를 대체하는 PostList를 ListView 클래스를 상속해 작성

애플리케이션 설계

정적 파일 관리 : blog/templates/blog/post_list.html
css 파일 경로 지정 : {% load static %}를 추가하여 static 파일을 사용하겠다고 선언
base.html
<head> 태그 안에 원래 bootstrap.min.css 파일 링크 수정
blog/templates/blog/post_list.html : 실제 포스트 내용이 표시되도록 수정

포스트에 이미지 올리기
이미지에 폴더 지정
Pillow 라이브러리 설치하고 마이그레이션
이미지 업로드 테스트
미디어 파일을 위한 URL
python -m pip istall Pillow
blog/models.py : imageField로 head_image 필드 추가
모델 변경 후 migrations

포스트에 파일 올리기 : blog/models.py

부트스트랩

https://getbootstrap.com/
https://bootswatch.com/