Django 초보 가이드

목차

-MVC, MFC

-Django 개념

-Project와 App – 웹프로젝트는 다수의 앱으로 구성되어 있음

-settings.py

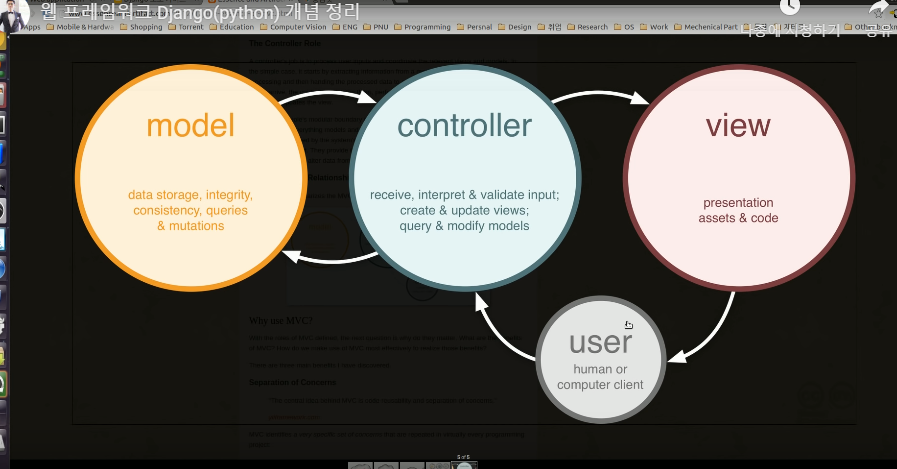
-manage.py

1.MVC & MTV

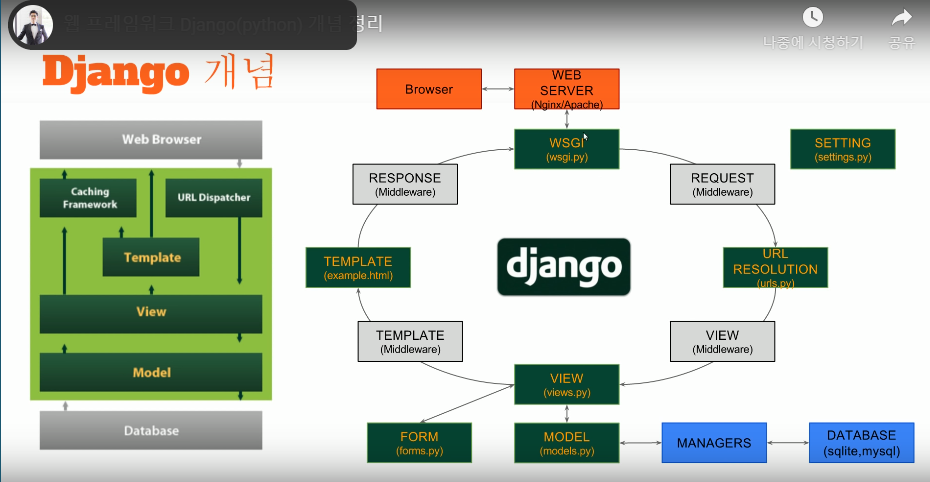
Model : 안전하게 데이터를 저장

View : 데이터를 적절하게 유저에게 보여줌

Control, Template(Django) : 사용자의 입력과 이벤트에 반응하여 Model과 View를 업데이트

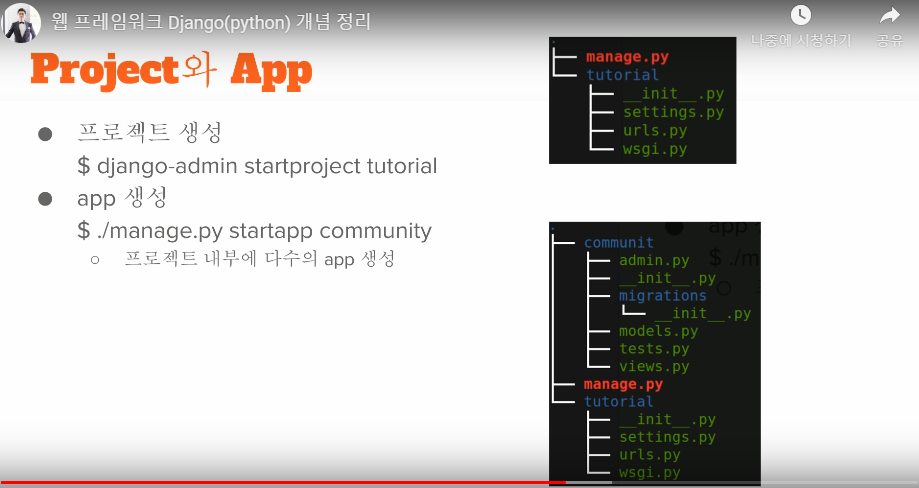


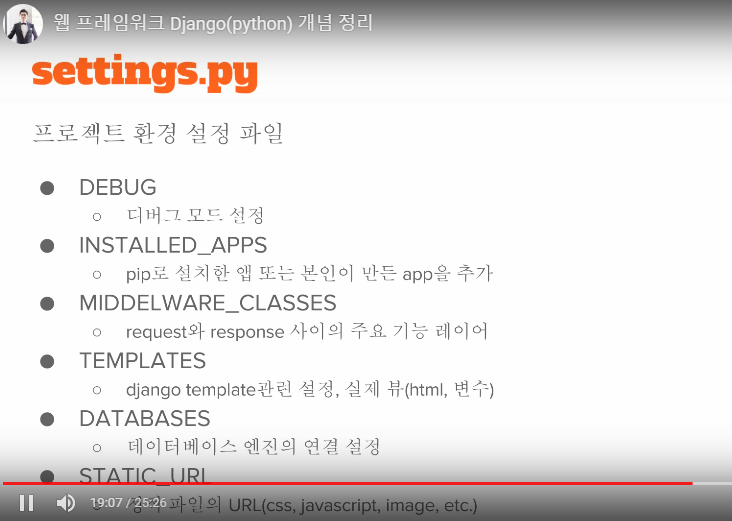
2. Django의 개념

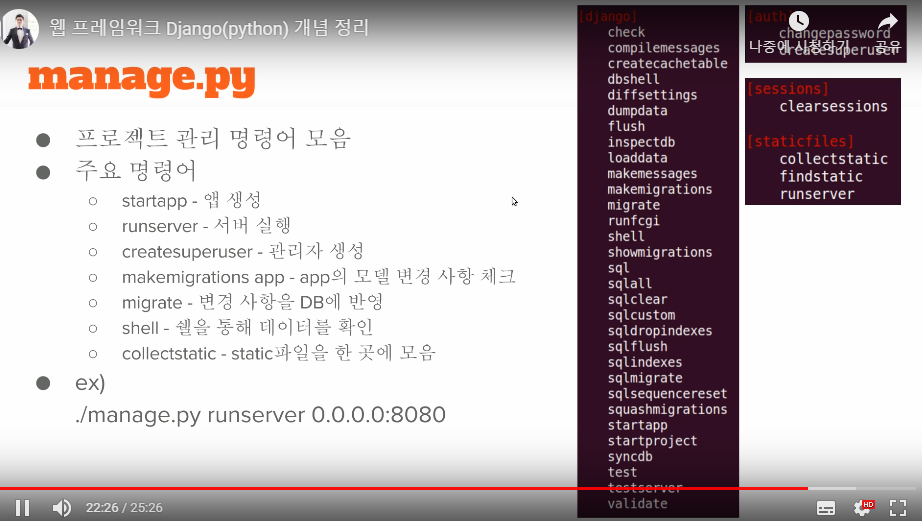


1. 브라우저에서 어떤 요청을 보내게 되면은 웹서버게이트웨이인터페이스(WSGI)에 먼저 신호가 들어온다. (이 파일은 우리가 크게 건드릴 일이 없다고 한다.)  
   이것의 역할은 웹서버와 장고를 적절히 결합시켜주는 역할이라고 한다.
2. 그 다음 신호를 urls.py에서 받게 된다. Urls.py는 정규표현식으로 구성되어 있다.  
   정규표현식에 맞게끔 특정한 VIEW로 보내주게 된다.
3. Views.py가 우리가 코드를 많이 작성하는 부분이라고 한다.  
   View에서 판단을 한다. 사용자의 요청에 의해서 데이터베이스에 데이터를 입력을 하던지, 데이터베이스로부터 데이터를 가지고와서 사용자에게 리스트를 보여줄 것인지
4. 결정을 하고 난 다음에 Model에 신호를 보내게 된다. 모델은 데이터베이스로부터 데이터를 가지고 올 때 사용
5. 모델에서 받은 데이터를 사용자에게 보여주기 위하여 UI 작업을 해야한다.
6. Template(html 파일)에서 UI를 만들어서 웹서버로 전송을 한다. Form과 같이 편리하게 화면을 만들 수 있다.

Project와 App







2강

웹 프레임워크 Django 실습

1. 프로젝트 & app 생성
2. 디렉토리 구조 확인
3. 관리자 페이지 확인
4. 글쓰기
5. 리스트
6. 글 보기

순서

Pip install Django – 으로 장고 설치

Django-admin startproject tutorial – 로 프로젝트 생성

Cd tutorial

<정보>Tutorial/urls.py에 외부에서 접근할 수 있는 url들이 표시된다.

./manage.py startapp community – 로 앱 생성 (윈도우면 앞에 python 붙여야 함)

Python ./manage.py migrate – settings에 설정되어있는 경로로 데이터베이스 생성

* 저걸 입력하면 장고가 필요한 기본적인 데이터베이스 테이블들이 생성되었음

Python ./manage.py createsuperuser - 슈퍼 유저 만들기

Id, ps : admin으로 생성하였음

Python ./manage.py runserver - 서버 실행 가능

만약에 포트번호 바꿔서 실행하고 싶다면 Python ./manage.py runserver 8080 -이렇게 적고 실행

서버를 돌린 다음 127.0.0.1:8080/admin 으로 어드민 페이지 접근 가능하다.

<정보> 어드민 페이지에서 기본적으로 user들을 관리할 수 있다. 특정 계정을 삭제할 수도 있다.

<정보> Group을 지정할 수 있다.

<정보> Model을 통해서 데이터베이스, 데이터를 만들면 어드민 계정을 통해서 그 데이터를 확인하고 수정하고 삭제할 수 있다. 따로 페이지를 만들지 않아도 되어서 편리하다.

Settings.py에서 INSTALLED\_APPS 안에 생성한 community 앱을 추가하여야 만든 앱을 사용할 수 있다.

그 다음 할 일은 community/models.py 코드를 생성하는 것이다.

<정보>Model은 필요한 데이터이고 기본적으로 class로 생성된다. 클래스는 기본적으로 models.Model을 상속하게 된다.

Models.py 부분에 아래와같이 클래스를 만들고

class Article(models.Model):

    name = models.CharField(max\_length=50)

    title = models.CharField(max\_length=50)

    contents = models.TextField()

    url = models.URLField()

    email = models.EmailField()

    cdate = models.DateTimeField(auto\_now\_add=True)

이제 데이터베이스에 적용하기 위하여

python manage.py makemigrations community - community라는 앱에 어떤 변화가 있는지 확인하는 단계

python manage.py migrate - 명령어를 주면 community라는 Table이 생성된다.

<정보> 여기까지 했으면 위 class에 대한 database 테이블을 db.sqlite3에 생성을 한 단계이다.

! 이제 글쓰기 form 을 만들어 보자

먼저 urls.py에서 User들이 접근할 수 있는 주소를 만들어야 한다.

from community.views import \*

path('write/', write, name='write'),

추가한다.

위 path라인 뜻은, write/ URL이 호출되면은 views.py 파일 안의 write 함수를 실행하게 한 것이다.

Views.py 에

# write함수는 request를 인자로 한다.

# 사용자의 요청이 request에 담긴다

def write(request):

    return render(request, 'write.html')

작성.

그리고 write.html 만들기 위하여 templates 폴더를 만든다.

그리고 write.html 파일 작성한다.

이후 runserver 한후 127.0.0.1:8000/write 하면 작성한 html 문서가 보인다.

Html 파일을 작성할 때, 장고에서는 작성한 model을 바탕으로 form을 편리하게 만들 수 있는 기능을 제공한다.

그러면 Community 폴더 하위에 forms.py 파일을 만든다.

from django.forms import ModelForm

from community.models import \*

class Form(ModelForm):

    class Meta:

        model = Article

        fields=['name', 'title', 'contents', 'url', 'email']

이러면 위처럼 Article이라는 Model을 사용하여 쉽게 Form을 생성할 수 있다.

ModelForm을 상속하는 Form 클래스를 하나 만들고 표시할 필드를 나열한다. 필드명은 Model에 있는 필드명이 되어야 한다.

Form은 views.py에서 생성이 가능하다.

from django.shortcuts import render

from community.forms import \*

# Create your views here.

# write함수는 request를 인자로 한다.

# 사용자의 요청이 request에 담긴다

def write(request):

    form = Form()

    return render(request, 'write.html', {'form':form})

community.forms 에서 import 부분을 추가하고,

Form() 객체를 생성한 다음 render 함수에서 ‘form’ 키값에 대한 value 부분에 생성한 변수를 전달한다.

그러면 write.html 탬플렛에 form 데이터가 전달이 된다.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>write</title>

</head>

<body>

 hello django

    <form action="" method="post">

        {% comment %}

            {{ form.as\_p}}

            {{ form.as\_ul}}

            {{ form.as\_table}}

        {% endcomment %}

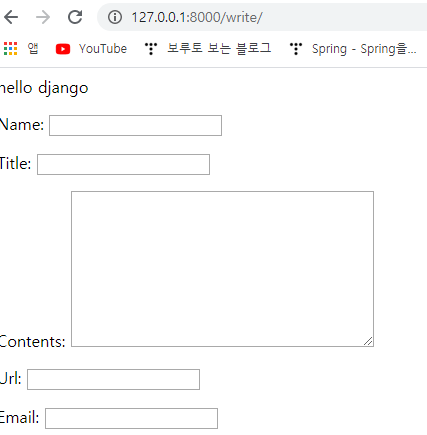
        {{ form.as\_p }}

    </form>

</body>

</html>

위처럼 <form> 태그 안에 {{form.as\_p}} 같은 명령어를 입력하여 form을 생성할 수 있다.



이후 브라우저에서 사용자가 위 Form을 작성하여 POST method로 request를 날렸을 때 전달받은 데이터를 데이터베이스에 저장하기 위한 부분 구현이 필요하다.

def write(request):

    if request.method == 'POST':

        form = Form(request.POST)

        if form.is\_valid():

            form.save()

    else:

        form = Form()

    return render(request, 'write.html', {'form':form})

위 부분 처럼 if 조건문에 method가 POST인지 조건을 걸고 form이 유효하다면 form.save() 함수를 통하여 데이터베이스에 저장한다. (이 단계에서 데이터베이스에 field가 저장이 된다.)



        {{ form.as\_p }}

        {% csrf\_token %}

Form 태그에 입력하고 submit 버튼 누르면 위쪽 forbidden 에러가 나온다.

그 이유는 위쪽 라인처럼 Form 태그안에

{% csrf\_token %}

를 안넣었기 때문이다.

Django 에서는 기본적으로 CSRF 토큰을 이용해 CSRF 공격을 방지한다.

# CSRF공격이란? Cross Stie Request Forgery는 웹 어플리케이션 취약점 중 하나로, 인터넷 사용자가 자신의 의지와는 무관하게 공격자가 의도한 행위(수정, 삭제, 등록 등)을 특정 웹사이트에 요청하게 만드는 공격

토큰의 동작 방식은

-클라이언트에서 해당 페이지를 접속하게 되면 Django 에서 자동으로 csrf\_token을 클라이언트로 보내어 cookie에 저장시키고, POST로 전송할 때 cookie의 csrf\_token이 함께 전송되어 인증하는 방식이다.

{% csrf\_token %}

토큰을 form에 넣고 다시 데이터제출 button을 누르면 정상적으로 form에 입력한 데이터가 데이터베이스에 저장이 된다.

이후 저장한 데이터를 보기 위한 페이지를 만들기 위해서 먼저

Urls.py 안에

path('list/', list, name='list'),

]

Path를 추가한다.

그리고 views.py 로 가서 list url을 처리하기 위한 함수를 만든다.

def list(request):

    return render(request, 'list.html')

이후 templates/ 하위에 list.html 파일을 만든다.

다시 views.py 파일로 돌아가서 list로 요청된 request를 처리할 함수를 정의한다.

def list(request):

    articleList = Article.objects.all()

    return render(request, 'list.html', {'articleList':articleList})

Article.objects.all()하면 Article이라고 하는 데이터베이스 Table에 있는 모든 COLUMN을 가져오게 되고, 그 Column을 articleList에 저장하게 된다.

그리고 위와 동일하게 객체를 하나 정의해서 articleList 값을 html로 전달한다.

List.html에서 전달받은 articleList 값을

<body>

{% for article in articleList %}

{{ article.title }} | {{ article.name }} | {{ article.cdate|date:"D d M Y" }}

{% endfor %}

</body>

처럼 가져와서 보여줄 수 있다.

<body>

    <ul>

        <li>제목 | 저자 | 날짜</li>

        {% for article in articleList %}

        <li>{{ article.title }} | {{ article.name }} | {{ article.cdate|date:"D d M Y" }}</li>

        {% endfor %}

    </ul>

</body>

Ul, li 태그 사용하여 정리

이후 urls.py로 돌아가서

path('view/<int:num>', view),

위의 뜻은 view/ 다음에 오는 숫자값을 num 에 저장하여 view 함수로 넘기겠다는 의미이다.

그리고 views.py에서 view 함수에서 위 url을 처리하도록 만든다.

Views.py

# 이때는 리스트가 아니라 게시글 하나임

def view(request, num="1"):

    # 모델을 생성하면 ID가 저절로 생성되는대, 해당 ID로 게시물에 접근할 수 있다.

    article = Article.objects.get(id=num)

    return render(request, 'view.html', {'article':article})

그리고 view.html 을 만든다.

<body>

    제 목 : {{ article.title }}

    <br>

    저 자 : {{ article.name }}

    <br>

    내 용 : {{ article.contents }}

    <br>

    Email : {{ article.email }}

    <br>

</body>

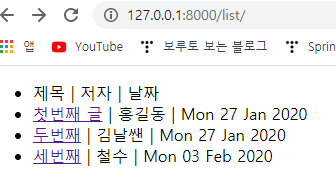
List.html에서

        {% for article in articleList %}

        <li><a href="/view/{{article.id}}">{{ article.title }}</a> | {{ article.name }} | {{ article.cdate|date:"D d M Y" }}</li>

        {% endfor %}

와 같이 링크를 걸어서



처럼 만들 수 있다.