2강 장고 초기 설정시 사용한 명령어 징리

django-admin startproject mysite . – 현재 path에 startproject 생성

django-admin startapp todo – 현재 path에 startapp 생성

python manage.py migrate –DB 생성

python manage.py createsuperuser – ID, PS admin으로 관리자 계정 생성

파일 설정 순서

Settings.py

* INSTALLED\_APPS = [] 안에 생성한 Startapp을 넣어야 한다.
* 'todo.apps.TodoConfig',

Templates 경로를 지정하여 준다

* 'DIRS': [os.path.join(BASE\_DIR, 'templates')]

-

타임존 설정 변경

TIME\_ZONE = 'Asia/Seoul'

USE\_TZ = False

-STATICFILES\_DIRS 추가

STATICFILES\_DIRS = [os.path.join(BASE\_DIR, 'static')]

Models.py

Urls.py

Start Project Urls.py에 아래부분 추가

from django.urls import path, include

path('todo/', include('todo.urls')),

추가

그리고 이전 강의와 다르게 startApp에 위 urls.py 파일을 복사하여 아래와 같이 내용을 입력하였다. (todo/vonly/ 같은 url을 처리하려 하는것인가..? 아직은 정확히 모르겠다.)

from django.urls import path

from . import views

app\_name = 'todo'

urlpatterns = [

    path('vonly/', views.TodoVueOnlyTV.as\_view(), name='vonly'),

]

Views.py

아래 코드 입력

from django.views.generic import TemplateView

# Create your views here.

class TodoVueOnlyTV(TemplateView):

    template\_name = 'todo/todo\_vue\_only.html'

templates

templates/todo/todo\_vue\_only.html 위치시키고 (1강에서 작성하였던 Vue로 작성한 html 파일이다.)

서버를 실행한 후 127.0.0.1:8000/todo/vonly/ 입력하여 출력되는 todo\_vue\_only.html을 보면



처럼 글자가 출력되지 않은걸 볼 수 있다.

그 이유는

Vue.js **머스태쉬** 문법이 Django의 Template 문법과 달라서 발생한 문제이다.

그래서 해결책은 Vue 인스턴스에서

        var vm = new Vue({

            el: '#app',

            delimiters: ['{', '}'],

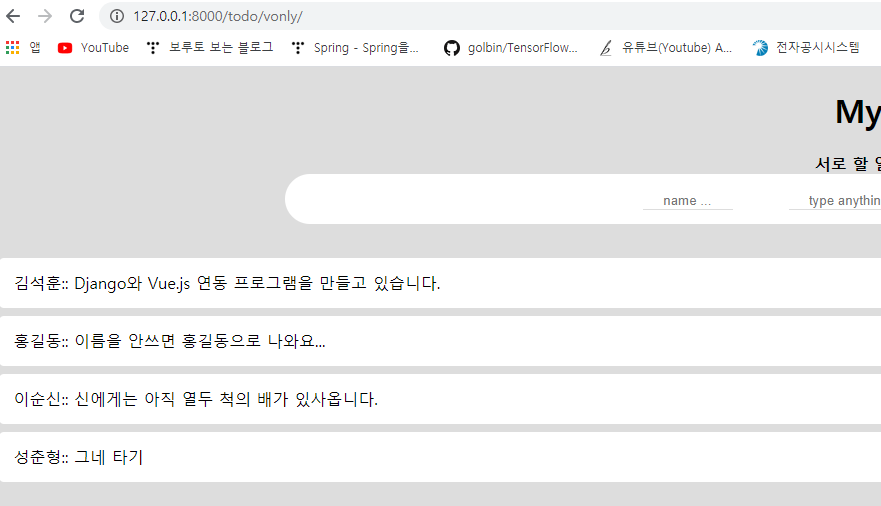
delimiter를 위처럼 바꾸고 (위 라인의 의미는 vue **머스태쉬** 문법에서 중괄호를 하나만 사용하겠다는 의미이다.),

그리고 vue 표현식 적는 부분에서 아래처럼

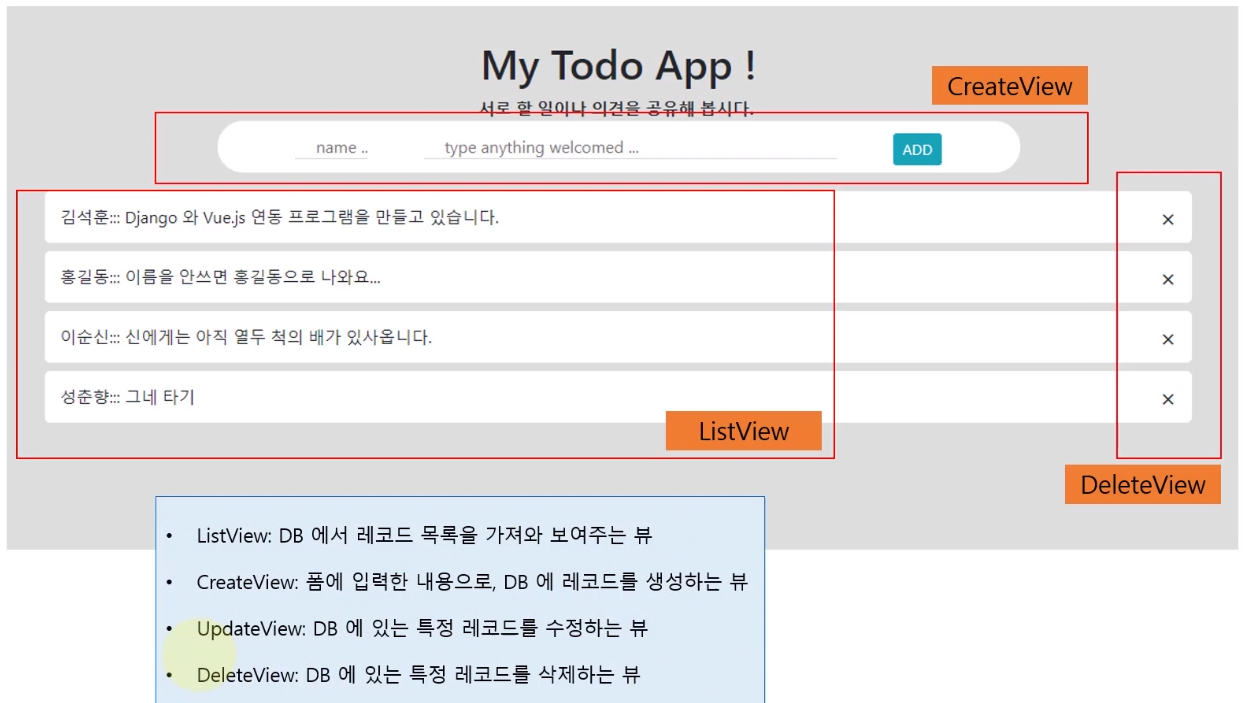
                <span>{{ todo.name }}:: {{todo.item}}</span>

을 아래로 변경하면 의도한대로 html 화면이 보여지는 것을 볼 수 있다.

                <span>{ todo.name }:: {todo.item}</span>



3강



장고에서는 위와 같이 기능을 가지는 클래스형뷰들이 많이 구현되어 있다.

기 구현된 클래스형뷰들을 잘 활용하면 개발 공수를 많이 줄일 수 있다.

Todo/models.py에 아래 테이블을 만든다.

class Todo(models.Model):

    name = models.CharField('NAME', max\_length=5, blank=True)

    todo = models.CharField('TODO', max\_length=50)

    def \_\_str\_\_(self):

        return self.todo

####

테이블을 신규로 정의할 때는 admin 사이트에서도 보일 수 있도록, admin.py 파일에도 등록을 해주어야 한다.

from todo.models import Todo

# Register your models here.

@admin.register(Todo)

class TodoAdmin(admin.ModelAdmin):

    list\_display = ('id', 'name', 'todo')

$%^$#@# 모델에 변경사항이 발생하면 이를 데이터베이스 테이블에 반영을 해주어야 한다.

python manage.py showmigrations – 로 현재 상태를 보고,

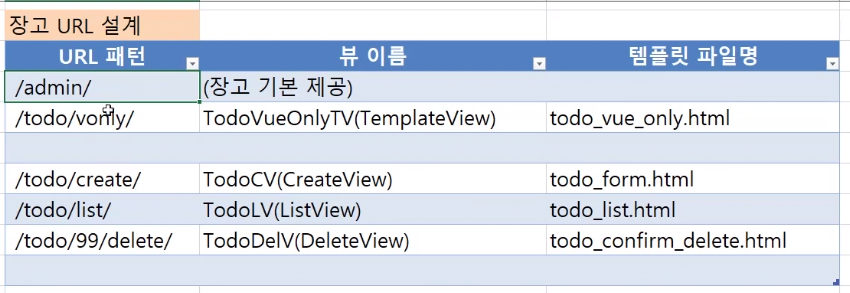
python manage.py makemigrations – 로 migrations 파일을 만든 다음에

python manage.py migrate – 로 데이터베이스에 반영을 해서 테이블을 만든다.

위 같이 Model 작업이 끝난 후에는 admin 페이지에서 model을 확인해 볼 수 있다.



Model 작업이 잘 완료된걸 확인할 수 있음



URL 패턴, 뷰 이름, 템플릿 파일명은 항상 1-1 Mapping 되므로 위처럼 정리를 해놓는게 좋다

위 표를 참조하여 Urls.py에 url 추가.

path('create/', views.TodoCV.as\_view(), name='create'),

    path('list/', views.TodoLV.as\_view(), name='list'),

    path('<int:pk>/delete/', views.TodoDelV.as\_view(), name='delete'),

<int:pk> 이런 표현은 숫자가 들어오면 INT로 변환해서 View로 넘겨주는 역할을 한다.

모든 View의 공통적인 속성(RedirectView 제외)으로 template\_name이 있다.

class TodoCV(CreateView):

    model = Todo

    fields = '\_\_all\_\_'

    template\_name = 'todo/todo\_form.html'

    success\_url = reverse\_lazy('todo:list')

class TodoLV(ListView):

    model = Todo

    template\_name = 'todo/todo\_list.html'

class TodoDelV(DeleteView):

    model = Todo

    template\_name = 'todo/todo\_confirm\_delete.html'

    success\_url = reverse\_lazy('todo:list')

CreateView는 Form 을 만들기 위해서 Model 이 필요하다.

그리고 Form 에 필요한 Field 생성 \_\_all\_\_ 모든 필드 쓴다는 의미

Form 처리가 끝난 후에는 redirect를 할 success\_url이 필요하다. (todo/list)

ListView는 Table에서 가져와야 하기 때문에 마찬가지로 Model을 정의해야 한다.

DeleteView는 특정 table에서 특정 record 하나를 삭제해야 하기 때문에 table을 지정해 주어야 한다.

Deleteview도 삭제 처리가 된 후에는 redirect가 필요하기에 redirect url을 지정한다.

!@#$ success\_url을 지정할 때는 reverse(), reverser\_lazy() 함수 중 하나를 사용해야 하는대, 여기서는 reverse\_lazy()함수를 지정해야 한다. (정확히 이해는 못했지만, 저 라인이 실행되는 시점에는 아직 urls.py 모듈이 실행되지 않은 상태여서 그렇다고 한다.)

# Django 로 todo 앱 코딩하기

## HTML 코딩하기

Todo\_form.html 을 생성한 후 vue.js 코드들을 덜어내고 순수 HTML 코드로 form을 작성한다.

<form class="inputBox" action='.' method="post"> {% csrf\_token %}

            <input class="name" name="name" type="text" placeholder="name ...">

            <input class="item" name="todo" type="text"

                placeholder="type anything welcomed ...">

            <button type="submit">ADD</button>

        </form>

저렇게 form을 만들고 todo/create url으로 접근 후 form 에 정보를 입력하고 저장하면 데이터베이스에 저장된 것을 확인할 수 있다. (admin 페이지 들어가서 확인 가능)

!@#$

개인적으로 views.py에서

from django.views.generic import TemplateView, CreateView, ListView, DeleteVie

class TodoCV(CreateView):

    model = Todo

    fields = '\_\_all\_\_'

    template\_name = 'todo/todo\_form.html'

    success\_url = reverse\_lazy('todo:list')

Class에 todo\_form.html에서 입력된 form의 data를 데이터베이스에 저장하는 어떤 로직도 보이지 않아서 어떻게 동작하는건지 의문이 든다.

잘은 모르겠지만 혼자 생각해본 결과 TodoCV 클래스에서 CreateView를 받는대(라이브러리), CreateView 안에 model, fields 등 속성과 DB 저장하는 로직이 이미 구현되어 있고,

우리는 views.py에서 위 속성들을 오버라이드 받아서 편하게 사용할 수 있는(?) 구조인 것으로 추정된다.)

Todo\_list.html을 vue.js 코드 제거하고 Django 탬플릿 사용하여 작성하는 과정임

listView에서는 object\_list라는 Context 변수를 넘겨준다.

        <ul class="todoList">

            {% for todo in object\_list %}

            <li>

                <span>{{ todo.name }}:: {{ todo.todo}}</span>

                <span class="removeBtn"><a href="{% url 'todo:delete' todo.id %}">&#x00D7</a></span>

            </li>

            {% endfor %}

        </ul>

위에서 삭제 버튼을 누르면 todo.id를 넘기는 delete url이 실행되도록 하여놨음.

Deleteview.html 을 vue.js 코드 제거하고 Django 탬플릿 사용하여 작성하는 과정임

Deleteview에서는 object 라는 Context 변수를 넘겨준다.

<p>Are you sure to delete {{ object }} ? </p>

        <br>

        <form action="." method="post"> {% csrf\_token %}

            <button type="submit">Confirms</button>

        </form>

위 코드 입력 후 todo/list 에서 record 삭제 버튼 눌렀을 때, 웹페이지와 admin페이지에서 해당 record가 정상적으로 삭제처리 된 것을 볼 수 있다.

이것도 어떤 로직으로 동작한것인지 잘 모르겠는대,

class TodoDelV(DeleteView):

    model = Todo

    template\_name = 'todo/todo\_confirm\_delete.html'

    success\_url = reverse\_lazy('todo:list')

아마 DeleteView 라는 클래스형뷰를 사용하여 가능했던 일인 것 같다.

=>추후 클래스형뷰에 대한 학습이 필요하다.

마지막으로 이름을 안 쓰고 todoItem만 쓰고 입력을 했을 때, default로 홍길동 이름이 나오도록 보완이 필요하다.

이런 기능은 form 처리하는 부분에 넣을 수 있고, database 에서 save 하는 부분에 넣을 수도 있는대 데이터베이스 save하는 부분에 넣는게 일반적인 방법이다.

Models.py에 기존 만들어놓은 Todo Class에서

상위 클래스의 save 메서드를 오버라이드하고 name이 입력 안된경우 체크하는 if문을 넣고 상위 클래스의 save()메서드를 호출한다.

class Todo(models.Model):

    name = models.CharField('NAME', max\_length=5, blank=True)

    todo = models.CharField('TODO', max\_length=50)

    def \_\_str\_\_(self):

        return self.todo

    def save(self, force\_insert=False, using=None, update\_fields=None):

        if not self.name:

            self.name = "홍길동"

        super().save()

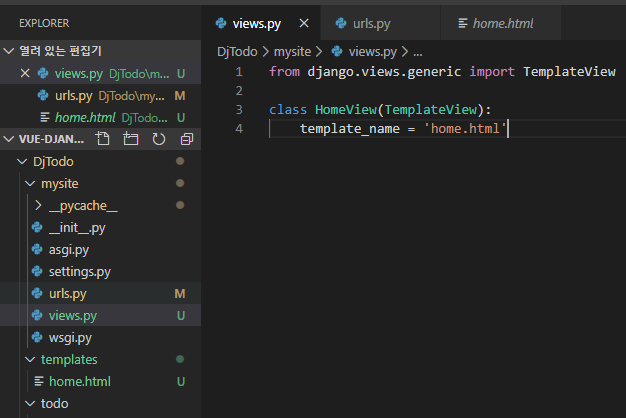
# Django 로 첫페이지 만들기

여기서 사이트의 첫페이지를 만드는대, 첫페이지는 특정 앱에 속한 것이 아니므로 프로젝트 디렉터리에 만들것이라고 한다.

from .views import HomeView

    path('', HomeView.as\_view(), name='home'),

StartProject 아래 urls.py에 위처럼 root 페이지 path를 추가한다.



그리고 views.py 파일 만들고 위 class만들어 저장한다.

!@#$

위 부분에서 아직 이해가 안되는건 StartProject의 templates는 StartApp의 templates와 달리 왜 프로젝트 바깥에 존재하게 되는걸까??? 정확한 이유를 아직 모르겠음.

이어서 base.html 파일 아래 외부 라이브러리(CDN)들을 사용한다.

Bootstrap.css

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">

Fontawesome.css, not glyphicon

Favicon.ico

<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css">

    <link rel="shortcut icon" href="{% static 'img/favicon.ico' %}">

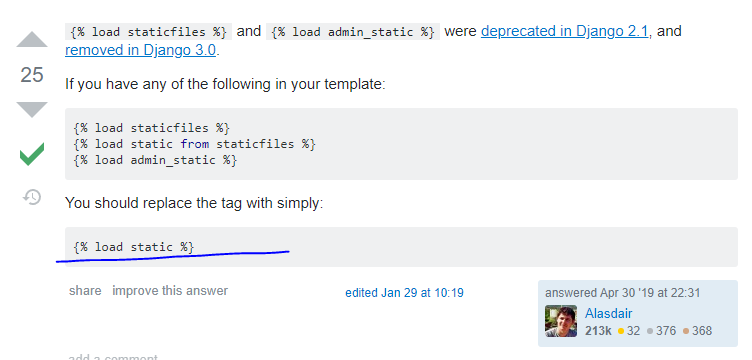
위에서 장고의 static TEMPLATE 코드를 사용하기 위해서 파일 재일 위에

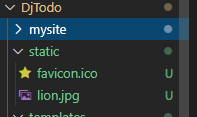
{% load staticfiles %}

를 적어주어야 한다.

!@#$ 주의!!! - <https://stackoverflow.com/questions/55929472/django-templatesyntaxerror-staticfiles-is-not-a-registered-tag-library>

Django 3.0 에서부터는 아래 이미지처럼 {% load static %} 으로 태그를 적어야 한다.





그리고 사용할 favicon.ico를 static폴더를 만들고 아래에 저장시킨다.

그리고 아래 자바스크립트 파일도 추가한다.

Bootstrap.js

Jquery.js – 부트스트랩.js를 위하여 필요

Popper.js – 부트스트랩.js를 위하여 필요

<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js" integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js" integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6" crossorigin="anonymous"></script>

그리고 base.html을 만든 후, bootstrap 공식 document를 열고 navbar 코드를 긁어와서 저장한다.

<예제로 배우는 파이썬 프로그래밍 url, Static 파일에 대한 설명 나와있음>

<http://pythonstudy.xyz/python/article/314-Static-%ED%8C%8C%EC%9D%BC>

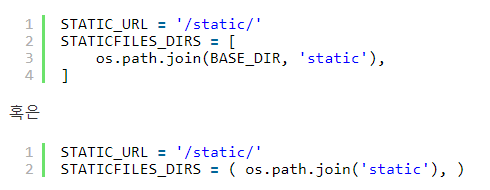
Static 파일 설명

-------------------------------------------------

웹사이트는 일반적으로 자바스크립트, CSS, 이미지 등의 파일들을 사용하는데, 이러한 파일들을 Django 에서는 Static 파일이라 부른다.

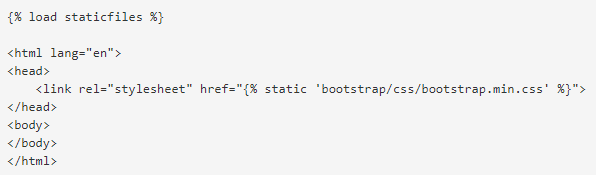
이러한 Static 파일들을 체계적으로 관리하기 위하여 일반적으로 Django 프로젝트 홈 디렉토리 밑에 “static” 이라는 서브 폴더를 만들어 그곳에 static 파일들을 넣는다.

Static 폴더에 파일들을 넣고 사용하기 위해서는 settings.py 에 하나의 셋팅을 추가해 주어야 한다. 즉, settings.py 파일에서 아래와 같이 static 파일들을 찾는 경로를 나타내는 STATICFILES\_DIRS 라는 변수를 설정해야 한다. 경로가 여러 개일 수 있지만, 여기서는 BASE\_DIR/static 폴더 하나를 지정하였다.

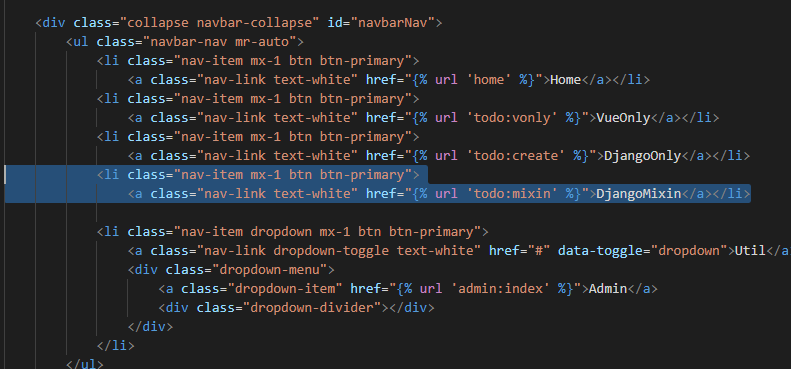


Static 파일들은 주로 템플릿에서 사용되는데, settings.py 설정을 마친 후 static 파일들을 사용하기 위해서는, 템플릿 상단에 {% load static %} 태그를 먼저 명시해 주어야 한다. 그리고 실제 static 파일을 가리키기 위해서는 아래 link 태그에서 보이듯이 “{% static ‘리소스명’ %}” 와 같이 static 템플릿 태그를 사용하여 해당 리소스를 지정한다. 이때 리소스명에는 “static/” 폴더명 다음의 경로만 지정한다.

(아래는 Django 2.x 버전이라서 <% load staticfiles %> 로 적혀있다. 3.x버전에서는 <% load static %>이다



강사분이 첨부한 base.html을 복붙하여 사용하였을 때 아래 드래그된 부분을 안지우면 에러가 나서 지우고 진행하였음. 기억할 것



그리고 home.html에서

{% extends 'base.html' %}

위 템플릿 문법으로 템플릿 확장 기능을 사용하는걸 볼 수 있음.

마찬가지로 아래 링크에 템플릿 확장에 관한 설명들이 잘 나와있다.

<http://pythonstudy.xyz/python/article/312-Django-%ED%85%9C%ED%94%8C%EB%A6%BF-%ED%99%95%EC%9E%A5>

<title>{% block title %}home.html{% endblock %}</title>

{% block extra-style %}{% endblock %}

{% block content %}{% endblock %}

{% block footer %}{% endblock %}

{% block extra-script %}{% endblock %}

상속 받는 html에서 원하는 코드를 집어넣을 수 있도록(접근 가능하도록) ‘block 태그명’ 으로 block 구문을 만들어 준다.

이후 todo App 의 탬플릿 밑에 있는 todo\_form.html 과 todo\_list.html, todo\_confirm\_delete.html을 base\_html을 상속하여 강의 따라 변경시킨다.

todo\_vue\_only.html 파일의 경우 기존에 작성해놓은 vue.js 코드를 base.html에서 작성해 놓았던

{% block extra-script %}

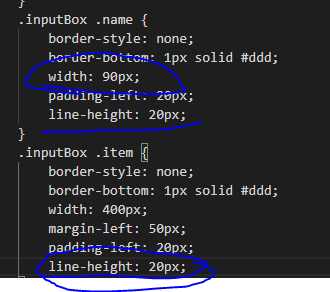
{% endblock %}

태그로 감싼다.

그다음 todo\_form.html, todo\_confirm\_delete.html 에서 bootstrap class 적용하여 버튼 바꿈.

<button type="submit" class="btn btn-danger btn-sm">Confirms</button>

와 같이.

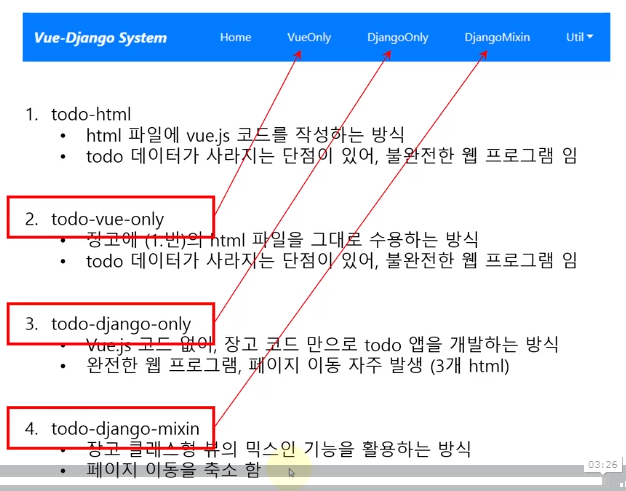


이후 todo\_vue\_only.html 에서 css 속성에 위 동그라미 친 부분 수정. (모양 더 예쁘게 만들기 위하여 시행하였음)

그리고 마지막으로 강사님이 미리 작성해놓은 코드에서 home.html을 복붙하고 정상 동작 확인하는 것으로 Django로 첫 페이지 만들기(bootstrap 으로 화면 만들기) 과정 끝남.

# Django로 todo앱 코딩하기

## 개발 방향 요약하기



이전까지 작성한

todo-vue-only 의 장점 : vue 코드를 사용하여 하나의 html 파일에서 이전에 작성하였던 기능 구현가능 하였음(single page)

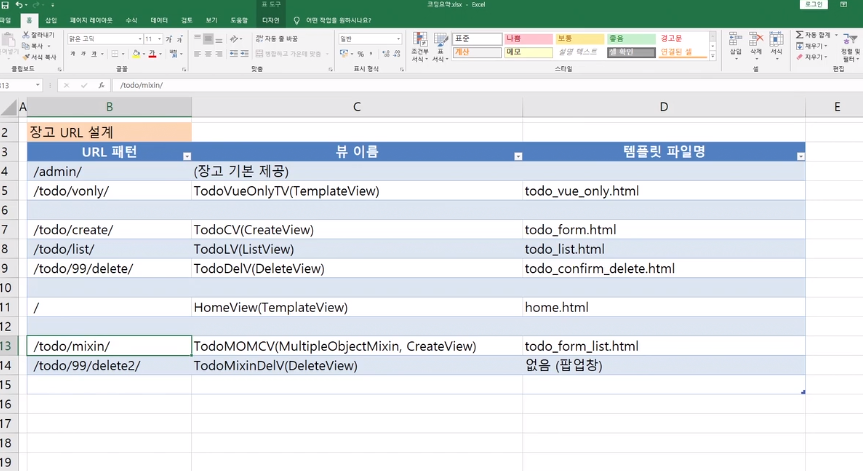
그에 반하여 todo-django-only 에서는 여러 개의 html 파일이 필요하였음. (vue의 single page 기능을 사용하지 못하였기 때문에)

여기서는 todo\_list.html, todo\_form.html, todo\_confirm\_delete.html

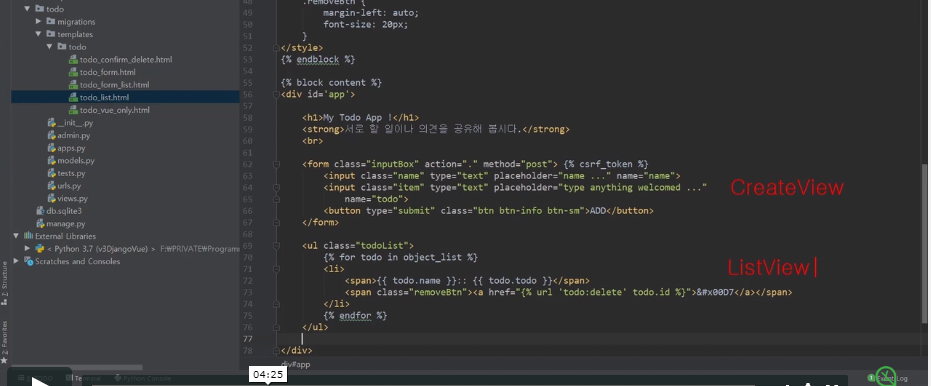
이후 이번 강의 스탭에서는 4번 항목의 django의 mixin 기능을 사용하여 페이지 이동을 축소하는 형식으로 개발을 진행한다고 한다.

## 클래스형뷰 Mixin 사용 하기(document보는법, mixin view 만드는법, 다중상속 등 내용 나옴)

코딩순서는 보통 settings -> model -> url -> view -> template 순으로 한다.



이후부터는 강의 들으며 코드 변경사항 일일이 적지는 않겠음. 단 중요한건 적을예정.



여기서 form 부분은 createView가 담당하고, list 부분은 ListView가 담당한다고 한다.

(아마 잘은 모르지만 기존 라이브러리에서 구현된 @@@@View 클래스들이 그렇게 구현되어 있나보다.)

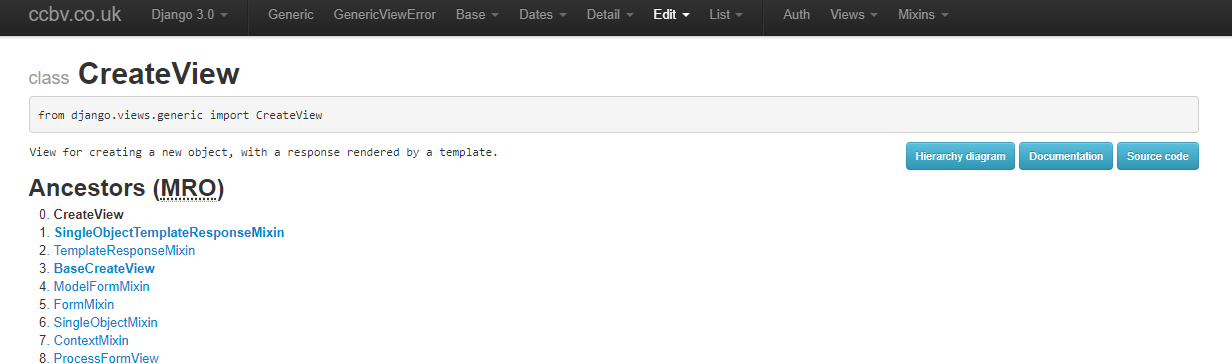
그래서 저 기능을 둘 다 사용하려면 이 강의의 title처럼 클래스형뷰를 mixin기능을 사용 해서 사용해야 한다고 한다.

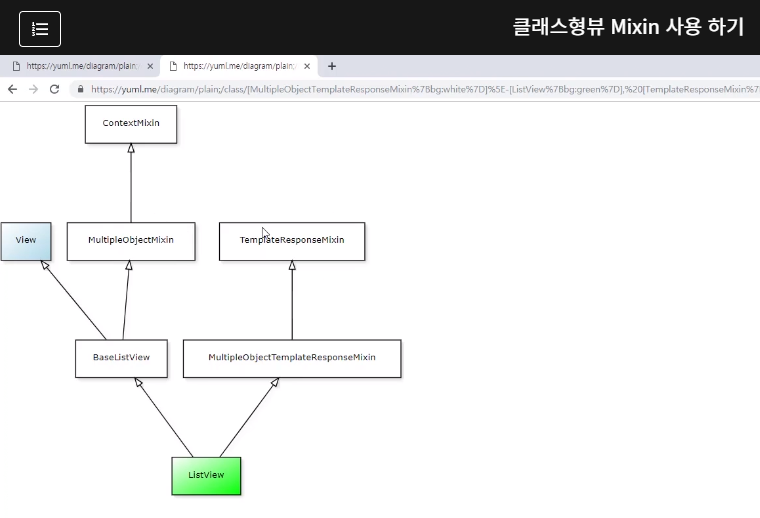
**그리고 템플릿을 코딩하면서 확인해야 할 사항은 view에서 template로 넘겨주어야 할 context변수가 어떤것들이 있는지 꼭 체크하여야 한다.**

위 예제에서는 form부분에서는 context 변수를 사용하지 않고 있고, list에서는 object\_list라는 context변수가 필요하다.

그래서 view에서 object\_list라는 Context 변수를 view에서 넘겨주어야 한다.

# <http://ccbv.co.uk/> 페이지에 장고 Classy Class-Based View에 관한 내용이 잘 정리되어 있다.





Django의 Class형 View는 공통적으로 사용되는 단위기능들을 위처럼 mixin class로 만들어두고 나서 mixin 클래스들을 조합해서 GenericView를 만드는 방식으로 구성되어 있다.

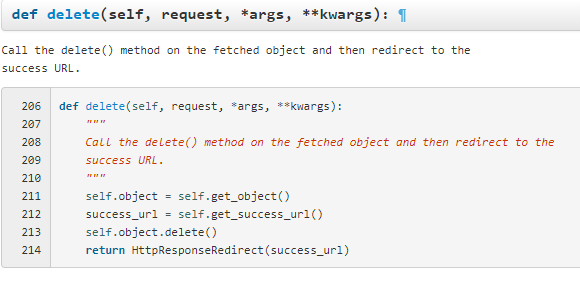
이 강의에서 다중상속, mixin view만드는 법 등 중요한 내용들 많이 나오는데, 당장은 이해하기 버거우니 우선 넘어가고 필요하면 후에 다시 볼 것

## Bootstrap 팝업창 만들기

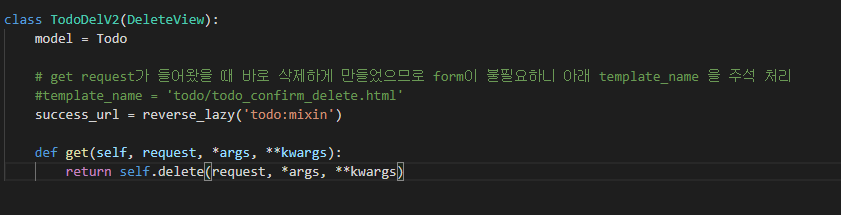
그냥 따라 쳤음.. 아래 내용 말고는 강의 다시 재생하며 공부할 것..

Deleteview에서는 post request를 받아야지 아래 문서의 delete()가 호출되어 DB의 데이터가 삭제되도록 되어있었다.





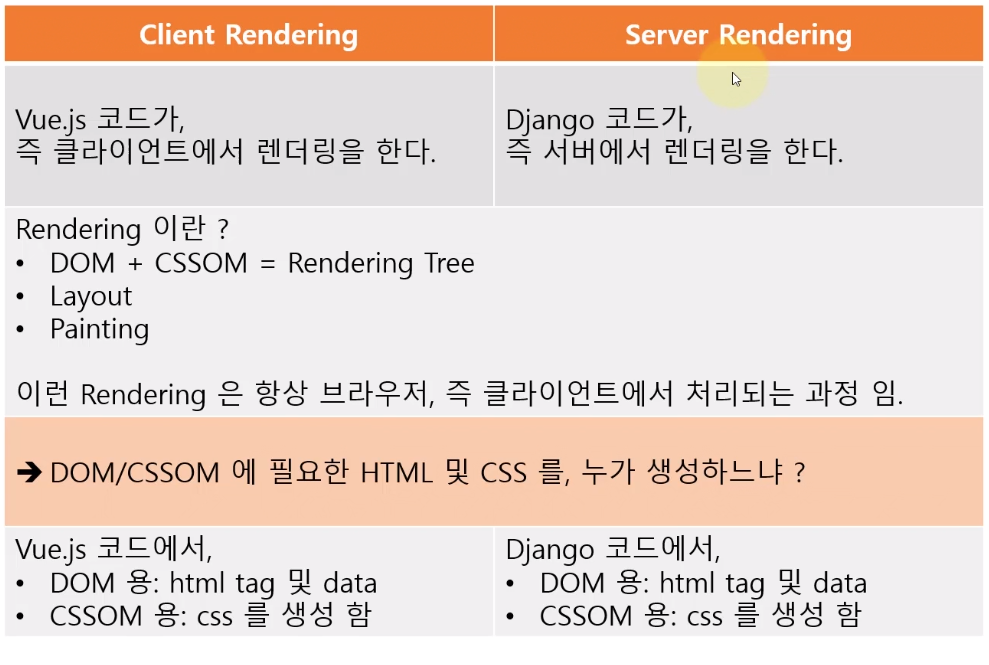
하지만 강의에서 GET Request를 받았을 때 delete()메서드를 바로 호출하게 하여 html 파일 전환없이 바로 데이터를 삭제할 수 있게 하였다.



# Client vs Server Rendering

## Client vs Server Rendering

일반적으로 VueOnly Menu에서 동작하는 방식을 클라이언트 렌더링, DjangoOnly나 DjangoMixin Menu에서 동작하는 방식을 서버 렌더링 이라혹 한다.



DjangoOnly 에서는 Django의 템플릿 엔진에서 html에서 css를 만들고 있어서 서버 렌더링을 하하 있다는 것을 알 수 있음

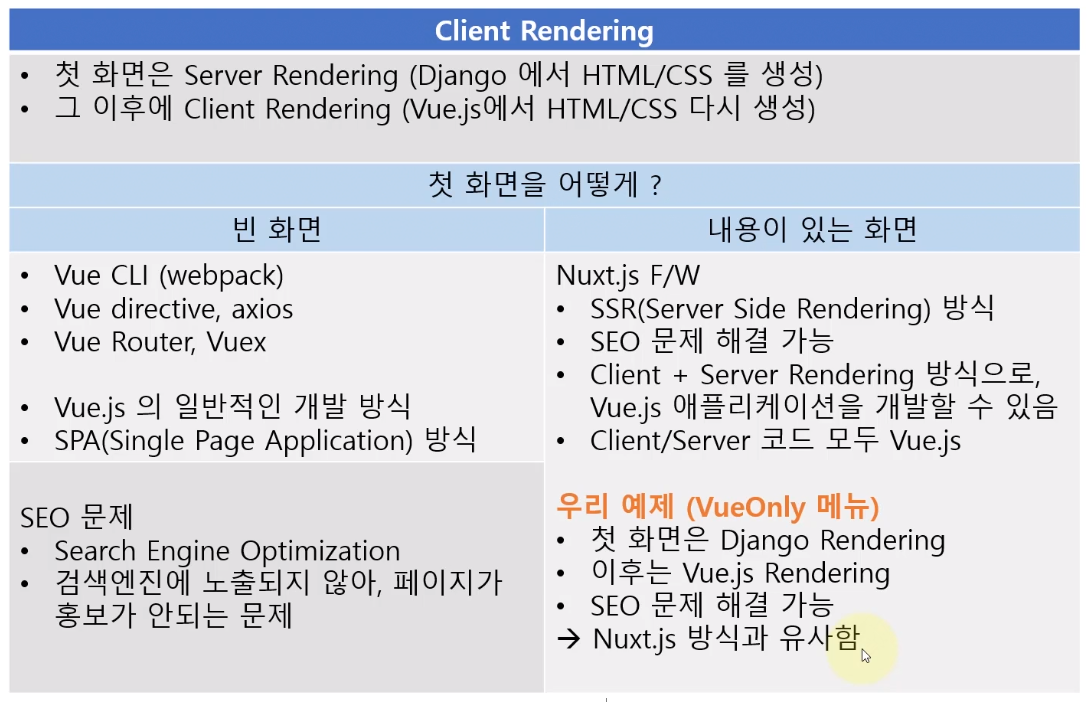
클라이언트 렌더링 방식으로 동작한다고 하는 VueOnly 메뉴에서는 첫번째 화면만 서버에서 html과 css를 만들고 있고, add 화면과 delete 화면은 Vue.js에서 html과 css를 만들고 있음을 알 수 있다.

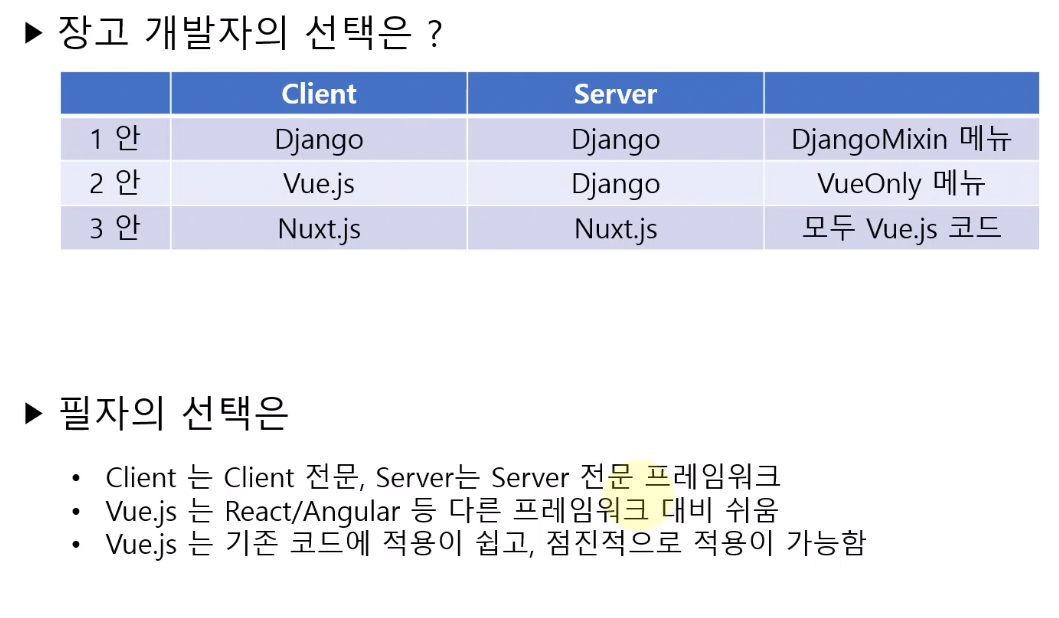
## SPA vs SSR

그래서 Client Rendering을 한다고 하더라도 첫 화면은 Server Rendering (Django 에서 HTML/CSS를 생성)

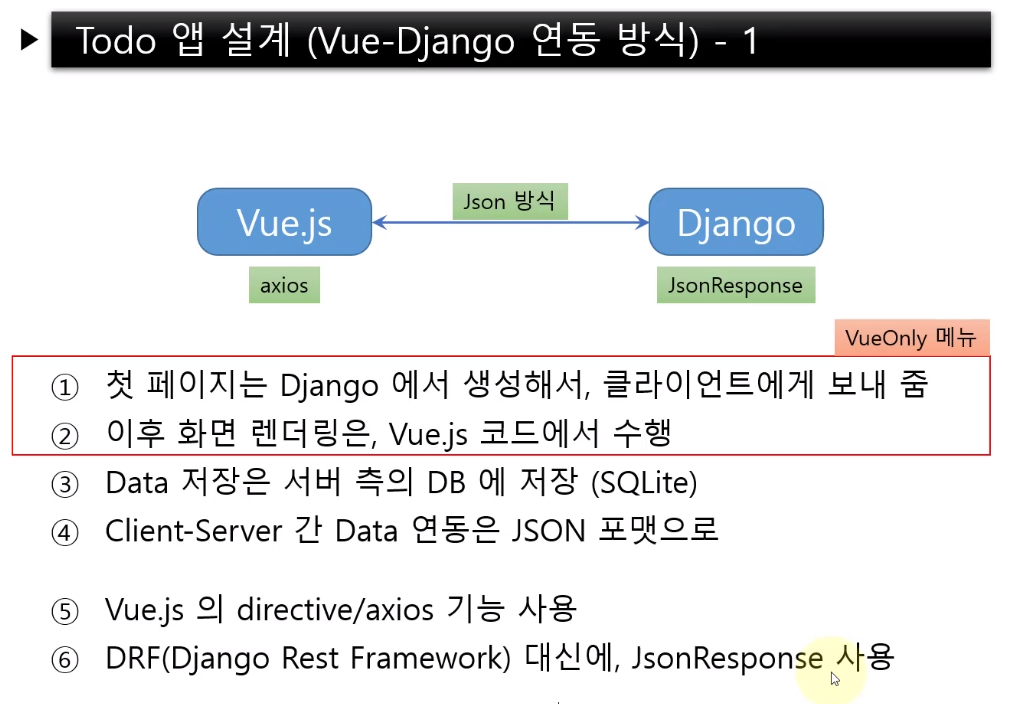
그 이후에 Client Rendering (Vue.js에서 HTML/CSS 다시 생성)

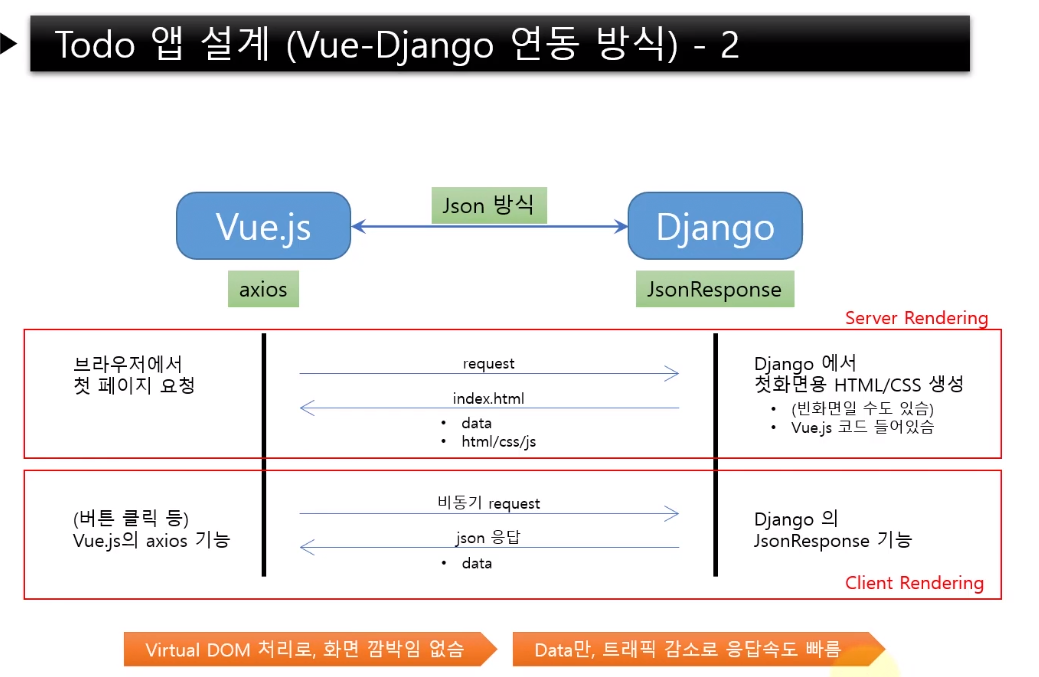
이럴 수 밖에 없는 이유는, 웹 프로그래밍의 동작 원리 상 첫 화면은 서버로부터 응답을 받아서 화면을 그려주기 때문이다.





## Vue-Django 연동 설계

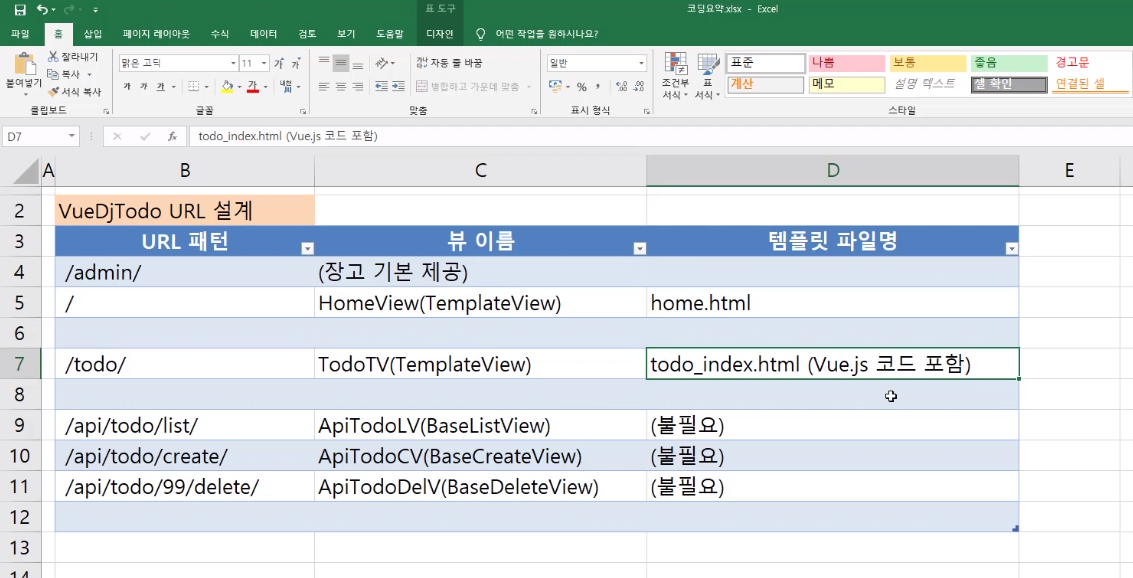




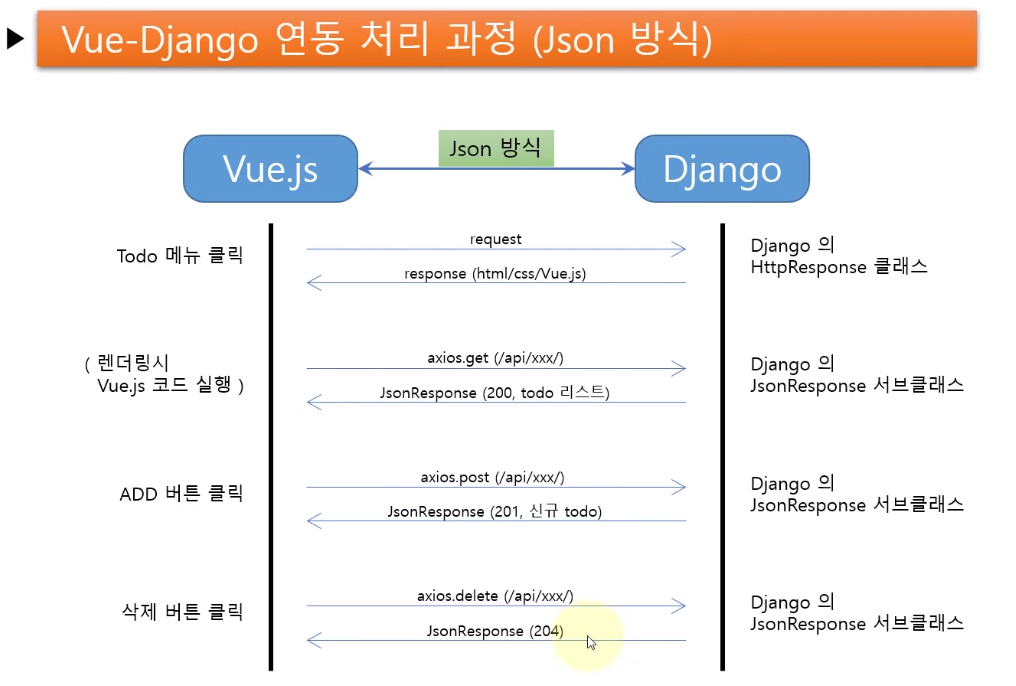
# Vue-Django 연동하기 (기본 코딩)

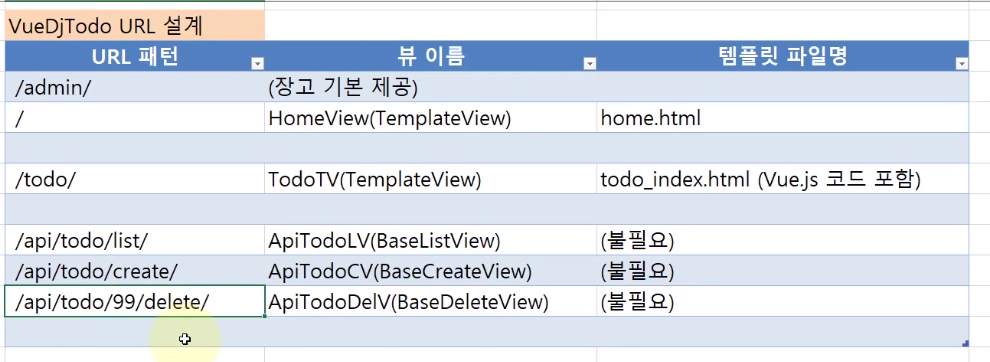
## Todo 앱 골격 만들기

## Todo 앱 기본 코딩하기



## JSON 연동 Sequence Chart





# Vue-Django 연동하기 (JSON 연동)

## Vue.js – axios.get() 코딩 하기

Vue.js 에서는 created, mouted, updated와 같은 라이프 사이클 훅을 제공하고 있다.

View instance가 생성되는 시점에 생성되는 메서드는 create 이다.

아래처럼 라이프 사이클 부분에서 axios 기능을 바로 사용할 수 있지만,

Method 부분에 axios 기능 사용하는 메서드를 정의해서 호출하는 방식으로 코딩하는게 일반적이라고 한다.

            created: function(){

                console.log("created()...");

                this.fetch\_all\_todo();

            },

            methods:{

                fetch\_all\_todo: function(){

                    console.log("fetch\_all\_todo()...");

                    var vm = this;

                    axios.get('/api/todo/list/')

                    .then(function (res){

                        console.log("GET RES", res);

                        vm.todoList = res.data;

                    })

                    .catch(function (err){

                        console.log("GET ERR", err);

                    })

                },

script src="https://unpkg.com/axios/dist/axios.min.js"></script>

axios 라이브러리를 사용하기 위해서는 CDN 링크를 걸어줘야 한다.

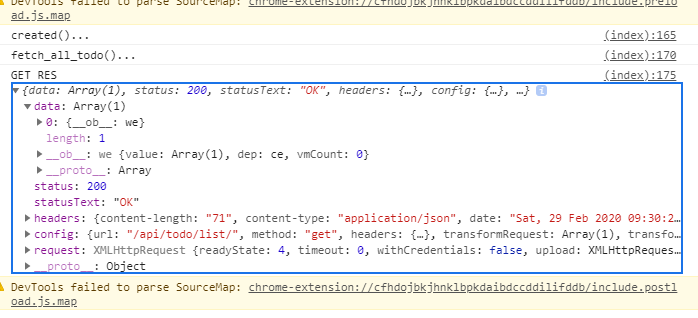
.then(function (res){

                        console.log("GET RES", res);

                        vm.todoList = res.data;

                    })

이 부분이 호출되었을 때 크롬 개발자도구 콘솔창에 서버가 보낸 response 데이터가 찍히는데,



서버에서 전송했던 ‘data’ 부분 외에 다른 데이터들이 같이 온 것을 볼 수 있음.

그래서 위에서 response를 저장할 때 res.data으로 저장해서 사용한 것이다.

### This 위치에 따른 참조하는 것 정리

자바스크립트에서 this는 위치에 따라 가르키는 것이 달라지기에 주의해야 한다.

Vue instance 안에서의 this는 vue instance 자신을 가르킨다.

Call back 함수 안에 정의된 this는 window 객체를 의미하기에 var vm=this처럼 로컬 임시 변수를 선언하여 사용할 수 있다. (로컬 변수 사용 안하면 화살표 함수 사용하여 해결할 수 있다.)

### Axios.get 문법 정리

Axios.get문법은 url을 먼저 넣고, url 호출이 성공했을 때 실행할 콜백 함수를 then()에 넣고, 실패했을 때 실행할 콜백 함수를 catch() 구문에 넣는다.

콜백함수에 서버로부터 받은 결과값을 저장할 인자의 이름은 임의로 정해서 넣으면 된다.

## Django – get. JsonResponse 코딩 하기

추후 다른 기능(앱)을 추가할 것을 생각해서 api기능들만 따로 모아놓기 위해서 api라는 별도 startapp 프로젝트를 만든다.

MTV 순서에 따라 코딩 진행.

1. Settings.py 에서 api 앱 등록

'api.apps.ApiConfig',

1. Urls.py 에서 api url 경로 등록

path('api/', include('api.urls')),

1. Api 앱에서 urls.py 파일 만들고 url 추가

#app\_name 스페이는 api로 하였음.

app\_name = 'api'

urlpatterns = [

    path('/todo/list/', views.ApiTodoLV.as\_view(), name = 'list'),

]

1. Views.py 코드 작성

<https://docs.djangoproject.com/ko/3.0/ref/request-response/#jsonresponse-objects>

from django.http import JsonResponse

from django.views.generic import ListView

# Create your views here.

from todo.models import Todo

class ApiTodoLV(ListView):

    model = Todo

    # template\_name 속성은 보여줄 html 파일이 있을때 쓰는 속성인대, 지금 jon response를 할 것이기에 이 경우는 보여줄 html이 없기 때문에 안씀

    # template\_name

    #client에서 get 방식으로 요청을 보낼 때

    def get(self, request, \*args, \*\*kwargs):

        tmpList = [

                    {'id':1, 'name':'d김석훈', 'todo':'Django와 Vue.js연동 프로그램을 만들고 있습니다.'},

                    {'id':2, 'name':'d홍길동', 'todo':'이름을 안쓰면 홍길동으로 나와요...'},

                    {'id':3, 'name':'d이순신', 'todo':'신에게는 아직 열두 척의 배가 있사옵니다.'},

                    {'id':4, 'name':'d성춘형', 'todo':'그네 타기'},

                    ]

        return JsonResponse(data=tmpList, safe=False)

    def render\_to\_response(self, context, \*\*response\_kwargs):

        todoList = list(context['object\_list'].values())

        return JsonResponse(data=todoList, safe=False)

todoList = list(context['object\_list'].values())

바로 위 코드에서 .values() 메서드는 Table에서 꺼내온 레코드들을 Dictionary 로 풀어주는 역할을 한다.

아래 링크에서 queryset에 적용할 수 있는 api들 확인할 수 있다.

<https://docs.djangoproject.com/en/3.0/ref/models/querysets/#values>

### 쿼리셋에서 values()메서드에 값 넣어서 데이터 필터링해서 가져올 수도 있다.

**>>>** Blog.objects.values()

<QuerySet [{'id': 1, 'name': 'Beatles Blog', 'tagline': 'All the latest Beatles news.'}]>

**>>>** Blog.objects.values('id', 'name')

<QuerySet [{'id': 1, 'name': 'Beatles Blog'}]>

중요

장고의 클래스형뷰를 활용하기 위해서는 <https://ccbv.co.uk/> 사이트를 많이 활용하라고 한다.

table에서 data를 가지고 오는 일은 listview가 하는데, listview는 기본적으로 httpResponse를 보낸다. 그런데 우리는 jsonResponse를 보내는 것이기 때문에 listView를 수정을 해주어야 한다.

JsonResponse objects 문서 내용

<https://docs.djangoproject.com/ko/3.0/ref/request-response/#jsonresponse-objects>

**class JsonResponse(data, encoder=DjangoJSONEncoder, safe=True, json\_dumps\_params=None, \*\*kwargs**

반드시

data에 list형 넣으면 safe = True 해야 한다.

Data에 다른 형의 data 넣으면 safe = False 해야 한다.

안그러면 에러난다.

### Ccbv 문서 보는법

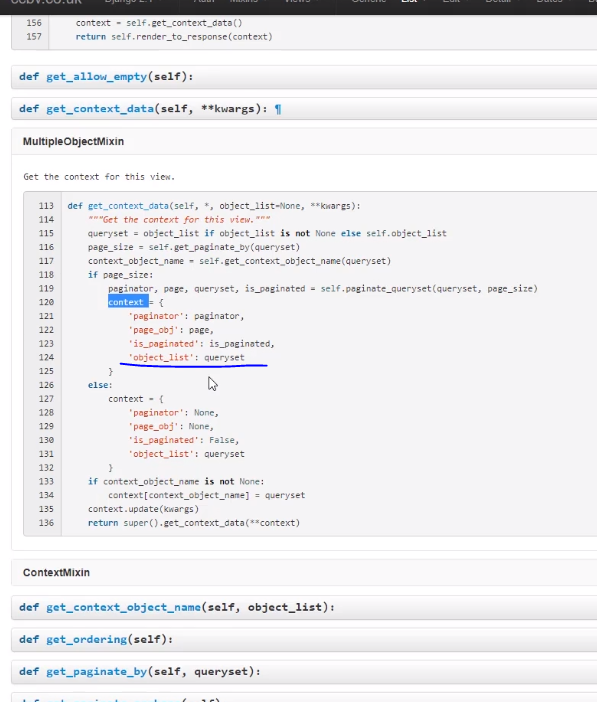
<https://ccbv.co.uk/projects/Django/3.0/django.views.generic.list/ListView/>

보면 Logic의 시작은 get()에서 시작하기에 get()을 먼저 본다.

보면 몇가지 일을 하고 나서 return self.render\_to\_response(context) 를 호출하는 것을 볼 수 있다.

문서에서 render\_to\_response() 메서드를 보면, response\_class0를 리턴하고 있는 것을 볼 수 있는데 이것이 http response class임을 알 수 있다.

* 그래서 우리는 이 부분을 JSON Response Class로 바꾸어주면 된다.
* 그래서 render\_to\_response()만 오버라이딩해서 수정하면 된다.



위에 밑줄 친 부분에서 db에서 model 클래스 정보 읽어온 데이터(queryset)을 ‘object\_list’ key 값에 저장하고 있는 것을 확인할 수 있다.