

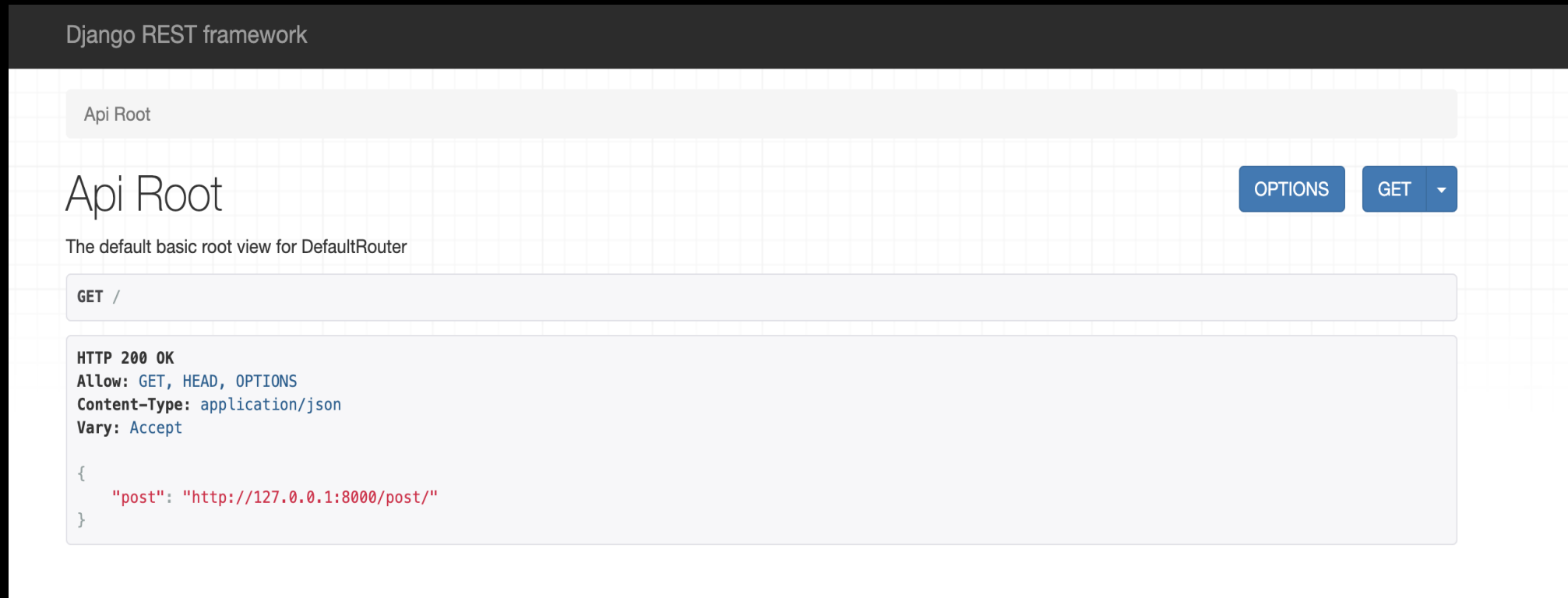
Django 심화강의

DjangoRestFramework를

원하는 대로 커스텀하기 위한

viewset&Router 이해하기

지난시간에 했던 실습 다시보기!!
`python manage.py runserver`



지난시간,,,맛봤던
DRF로 만든 REST API!

코드의 재활용성을 높이고 백엔드와 프론트엔드의 완벽한 분리를 하기 위해
지난주에 배웠던 것은?!

RESTful API

Django 안에서 RESTful API 서버를 쉽게 구축하여
다른 웹, 앱, 컴퓨터 등등에서 Django 데이터를 다룰 수 있도록
데이터구조화를 도와주는 프레임워크!!

Django "REST" Framework

Django REST Framework 되돌아보는 맛

RESTful이란?

: HTTP의 URL과 HTTP의 Method를 사용하여 API 사용 가독성을 높은 시스템!

CRUD !!!

하나의 API로

- 1) GET : 정보 읽기
- 2) POST : 정보를 추가하기
- 3) PUT : 정보를 업데이트하기
- 4) DELETE : 정보를 삭제하기

데이터를 처리할 수 있다!!

Django REST framework 되돌아보는 맛

MTV패턴!

Django!

Models.py → Views.py → urls.py

Django **Rest** Framework!

Models.py → **Serializers.py** → Views.py → urls.py

Django REST framework 되돌아보는 맛

MTV 패턴!

Django	Django Rest Framework
Form/ModelForm	Serializer/ModelSerializer
Model로부터 Field 읽어옴	
유효성 검사	
HTML Form	JSON 문자열

지난 실습으로 되돌아보기!
저번주에 진행했던 폴더를 열어주세요!

Django REST framework 되돌아보는 맛

Models.py

DB를 작성하기
원하는 모델 만들기

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=100)
    body = models.TextField()
```

Serializers.py

원하는 모델의

Serializer 만들기

**** Serializer 란?!**

데이터베이스의 데이터를
queryset, 모델 인스턴스를
XML, JSON 형식으로
직렬화 하기

ModelForm의 역할!

```
serializer.py
1 from .models import Post
2 from rest_framework import serializers
3 #json파일을 직렬화하는 프레임워크
4
5 class PostSerializer(serializers.ModelSerializer):
6     class Meta:
7         model = Post
8         fields = '__all__'
9         read_only_fields = ('title',)
```

Api Root / Post List

Post List

GET /post/?format=api

HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json

OPTION

GET

json

api

```
[{"id":1,"title":"안녕하세요","body":"안녕하세요"}, {"id":3,"title":"모르겠다","body":"모르겠다"}, {"id":5,"title":"","body":"d"}, {"id":6,"title":"","body":"d"}, {"id":7,"title":"","body":"d"}, {"id":8,"title":"","body":"d"}, {"id":9,"title":"","body":"d"}, {"id":10,"title":"","body":"d"}, {"id":12,"title":"","body":"d"}, {"id":13,"title":"","body":"d"}, {"id":14,"title":"","body":"d"}, {"id":15,"title":"fdfd","body":"fdfdfd"}]
```

Django REST framework 되돌아보는 맛

Views.py

원하는 모델의
viewset 만들기

**** viewset 란?!**

CRUD와 CRUD URL을
한번에 만들기 위한!

Django
뷰의 집합

views.py

```
1 from django.shortcuts import render
2 from rest_framework import viewsets
3 from .models import Post
4 from .serializer import PostSerializer
5
6 class PostViewSet(viewsets.ModelViewSet):
7     queryset = Post.objects.all()
8     serializer_class = PostSerializer
```



- 1) 반복되는 로직을 단일클래스 안에 결합함!으로
코드의 복잡함을 줄인다.
- 2) 라우터를 사용함으로써,
사용자가 url 설정파일을 다루지 않아도 된다!!

urls.py

ViewSet으로
만들어진 url

Router로 맵핑하기

**** Router 란?!**

ViewSet에 연결되어 있어
자동적으로 URL 주소를
연결해줌

urls.py

```
from django.urls import path
from . import views

app_name = 'post'
urlpatterns = [
    #post
    path('detail/<int:pk>', views.PostDetailView.as_view(), name = 'detail'),
    path('new', views.PostCreateView.as_view(), name='new'),
    path('edit/<int:pk>', views.PostUpdateView.as_view(), name='edit'),
    path('delete/<int:pk>', views.PostDeleteView.as_view(), name = 'delete'),
]
```



```
from django.urls import path, include
from rest_framework.routers import DefaultRouter
# 참고 레스트 프레임워크는 라우터를 통해 url을 전달함
from . import views
appname = 'post'

router = DefaultRouter()
router.register('post', views.PostViewSet)

urlpatterns = [
    path('', include(router.urls)), # 실제로 url을 작성하게 하는 것
]
```



- 1) 자동적으로 url 라우터를 사용함으로써,
사용자가 url 설정파일을 다루지 않아도 된다!!

REST API는 정말 적은 코드로 web API를 구현 할 수 있는 방법.



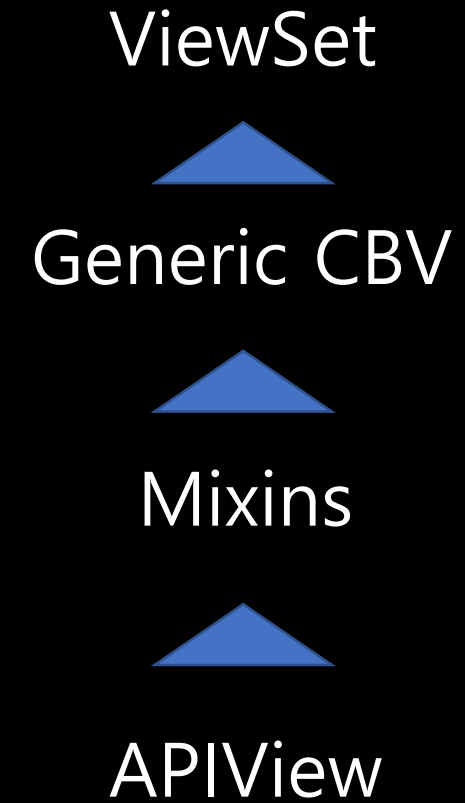
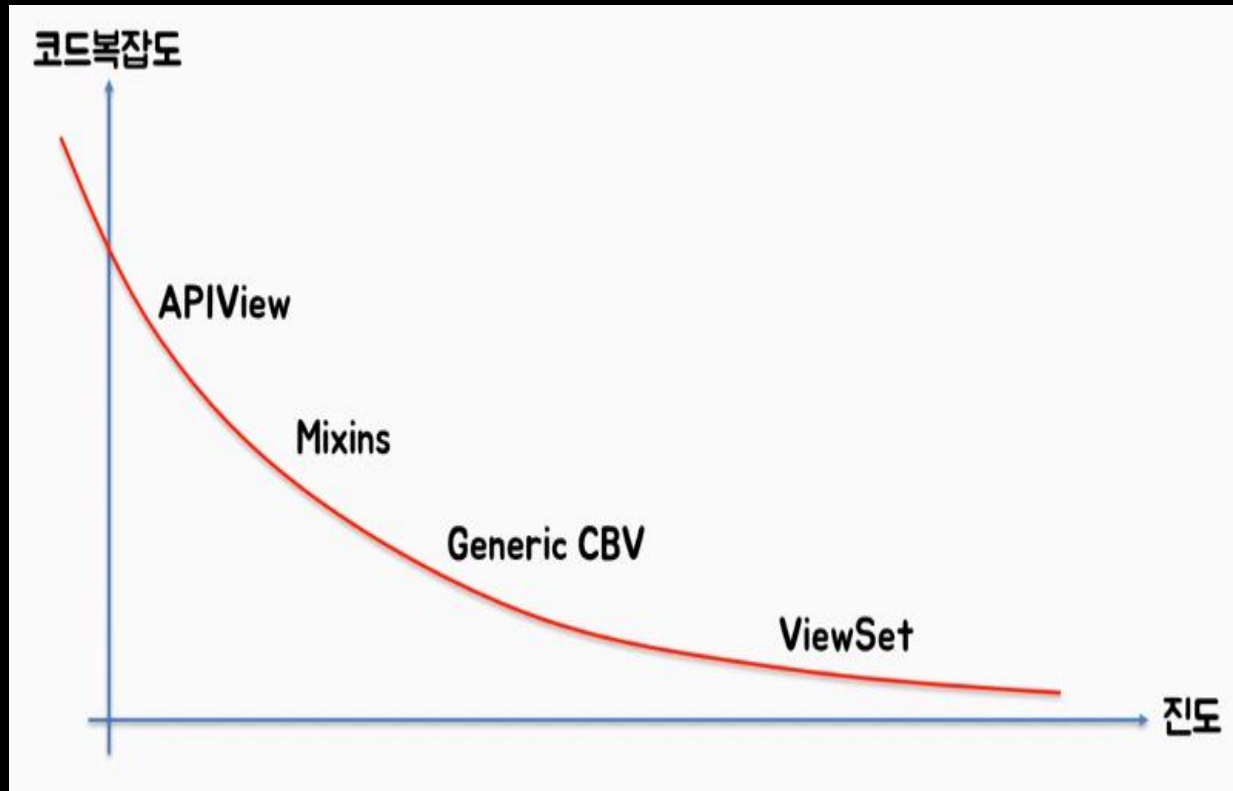
Django CBV때 배운
상속 클래스의 개념으로 **viewset**이 이루어지기 때문!!

코드의 낭비를 줄이고 원하는 대로 기능을 **control**하고 싶다면?!



오늘은 간단하게
ViewSet에 CBV로 상속되어지는 과정을 배워보면서
원하는 대로 ViewSet을 다뤄보고,
예상치 못한 버그에 대처하는 방법을
알아가보자!

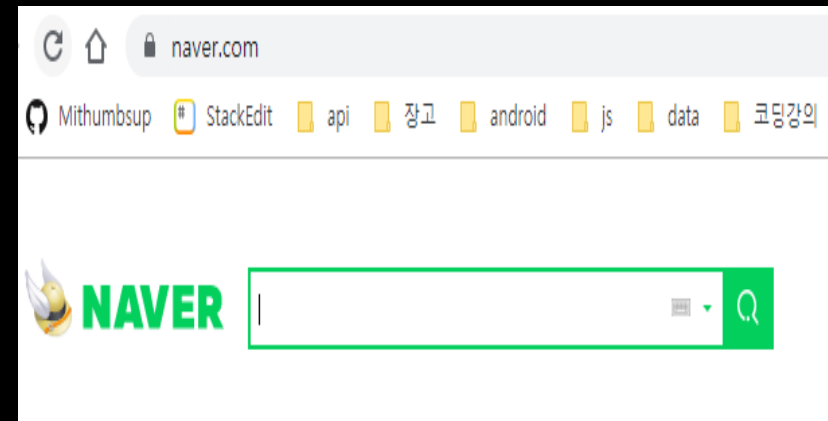
ViewSet에 이르는 과정



View

= 함수로도 작성할 수 있고 클래스로도 작성할 수 있는

요청(request)를 받아서 응답(response)를 반환해주는 호출 가능한 객체~



Django "REST" Framework

공식 홈페이지를 보고 Class Based Views 알아갈 예정~~

<https://www.django-rest-framework.org/tutorial/3-class-based-views/>

The screenshot shows the Django REST Framework website. The navigation bar includes links for Home, Tutorial, API Guide, Topics, and Community. A search bar and navigation buttons (Previous, Next, GitHub) are also present. The main content area is titled "Class-based Views" and explains that this pattern helps keep code DRY. A sidebar on the left lists the tutorial steps: Quickstart, 1 - Serialization, 2 - Requests and responses, 3 - Class based views (highlighted), 4 - Authentication and permissions, 5 - Relationships and hyperlinked APIs, and 6 - Viewsets and routers. Below the sidebar is a Sauce Labs advertisement. The main text area shows a code snippet for a class-based view, `SnippetList`, which inherits from `APIView`. The code includes imports for `Snippet`, `SnippetSerializer`, `Http404`, `APIView`, `Response`, and `status`. The `get` method is implemented to list all snippets or create a new one.

```
from snippets.models import Snippet
from snippets.serializers import SnippetSerializer
from django.http import Http404
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status

class SnippetList(APIView):
    """
    List all snippets, or create a new snippet.
    """
    def get(self, request, format=None):
        snippets = Snippet.objects.all()
        serializer = SnippetSerializer(snippets, many=True)
        return Response(serializer.data)

    def post(self, request, format=None):
```


RESTful API

하나의 API로 HTTP의 Method를 사용하여 데이터를 처리할 수 있다!!
CRUD!!

- 1) GET : 정보 읽기
- 2) POST : 정보를 추가하기
- 3) PUT : 정보를 업데이트하기
- 4) DELETE : 정보를 삭제하기

Class [](APIView) :

def <내가 필요한 HTTP Method>:

그 Http Method로 어떻게 처리할지
직접 정의하기

viewset을 다루기 전!
제일 먼저 viewset의 가장 근본 상속 클래스인

APIView

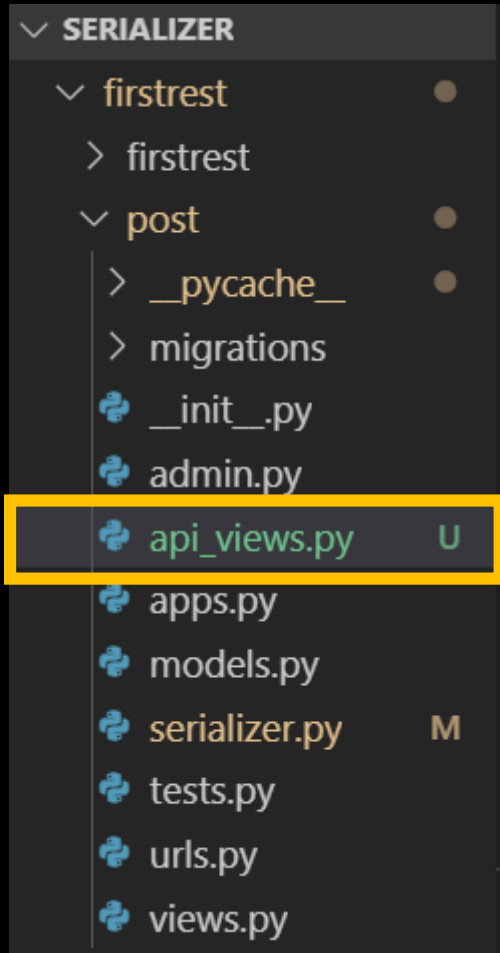
2가지 사용방법!

1. @api_view : 함수 기반 뷰에서 사용하기

2. APIView : 클래스 기반 뷰에서 사용하기

APIView를 클래스 기반 뷰에서 사용하기 위해

[app 이름] / **api_view.py** 만들기



```
#데이터를 처리하는 과정
from .models import Post
from .serializer import PostSerializer
# status에 따라 직접 Response를 내입맞대로 처리할 것 > APIview를 사용하는 의의!!
from django.http import Http404 # Get Object or 404 직접 구현
from rest_framework.response import Response
from rest_framework import status
# APIview를 상속받은 CBV
from rest_framework.views import APIView
# PostDetail 클래스의 get_object 메소드 대신 이거 써도 된다
# from django.shortcuts import get_object_or_404
```

-> APIview를 상속하여 view를 만들 때는
status와 response 를 임포트 해와서
직접 응답 과정을 만든다!!

```
class PostList(APIView):
    def get(self, request): -> 정보 읽기
        posts = Post.objects.all()
        # 쿼리셋 넘기기 (many=True인자)
        serializer = PostSerializer(posts, many=True)
        # 직접 Response 리턴해주기 : serializer.data > data를 넘겨주기
        return Response(serializer.data)

    def post(self, request): -> 정보를 추가하기
        serializer = PostSerializer(data=request.data)
        if serializer.is_valid(): # 직접 유효성 검사
            serializer.save() # 문제가 없으면 저장
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST) #오류페이지 보내주기
```

APIView를 클래스 기반 뷰에서 사용하기

api_views.py

Http 메서드로
데이터 전송하는
방식

```
# PostList 클래스와는 달리 pk값을 받음 (메소드에 pk인자)
class PostDetail(APIView):
    # get_object_or_404를 구현해주는 helper function
    def get_object(self, pk):
        try:
            return Post.objects.get(pk=pk)
        except Post.DoesNotExist:
            raise Http404

    def get(self, request, pk, format=None): -> 정보 읽기
        post = self.get_object(pk)
        # post = get_object_or_404(Post, pk)
        serializer = PostSerializer(post)
        return Response(serializer.data)

    # 위 post 메소드와 비슷비슷한 논리
    def put(self, request, pk, format=None): -> 정보 수정하기
        post = self.get_object(pk)
        serializer = PostSerializer(post, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    def delete(self, request, pk, format=None): -> 정보 삭제하기
        post = self.get_object(pk)
        post.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
```

APIView를 클래스 기반 뷰에서 사용하기



```
from rest_framework.urlpatterns import format_suffix_patterns
from django.urls import path
from post import api_views
```

Default Router 사용X > API ROOT 없음

```
urlpatterns = [
    path('post/', api_views.PostList.as_view()),
    path('post/<int:pk>', api_views.PostDetail.as_view()),
]
```

```
urlpatterns = format_suffix_patterns(urlpatterns)
```

APIView를 클래스 기반 뷰에서 사용하기

python manage.py runserver

Api Root / Post List

Post List

OPTIONS GET ▾

GET /post/

json
api

HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[]

Raw data HTML form

Title

Body

POST

Post List / Post Detail

Post Detail

DELETE OPTIONS GET ▾

GET /post/2

HTTP 404 Not Found
Allow: GET, PUT, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
 "detail": "Not found."
}

Http Method

Media type: application/json ▾

Content:

PUT

가장 근본 상속 클래스인 APIView

```
from post.models import Post, Prudct , , , ,
```

```
class PostList(APIView):
```

```
def get(self, request):  
    posts = Post.objects.all()  
    serializer = PostSerializer  
    return Response(serializer.
```

```
def post(self, request):  
    serializer = PostSerializer  
    if serializer.is_valid():  
        serializer.save()  
        return Response(seriali  
    return Response(serializer.
```

```
class PrudctList(APIView):
```

```
def get(self, request):  
    posts = Post.objects.all()  
    serializer = PostSerializer(posts, many=True)  
    return Response(serializer.data)
```

```
def post(self, request):  
    serializer = PostSerializer(data=request.data)  
    if serializer.is_valid():  
        serializer.save()  
        return Response(serializer.data, status=status.HTTP_201_CREATED)  
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

각 모델에 적용되는
get/ put / delete / post 작업을 위한
대부분의 코드는
비슷한 논리로 작성됨

사용하는 모델이 늘어날수록
불필요한 코드도 늘어난다!

코드의 낭비를 줄이기 위한 해답!!!



Mixin 사용하기

Mixin 사용하기

[app 이름] / **mixin_view.py** 만들기

▼ SERIALIZER

▼ firstrest

> firstrest

▼ post

> __pycache__

> migrations

__init__.py

admin.py

api_views.py U

apps.py

mixin_views.py U

models.py

serializer.py M

tests.py

urls.py M

views.py

db.sqlite3 M

manage.py

```
# 데이터 처리 대상 : 모델, Serializer import 시키기
from post.models import Post
from post.serializer import PostSerializer
```

```
from rest_framework import generics
from rest_framework import mixins
```

-> Mixin을 상속하여 view를 만들 때는
generics와 mixins를 импорт 한다!!

```
class PostList(mixins.ListModelMixin, mixins.CreateModelMixin,
               1) 2) generics.GenericAPIView):
    3)
```

```
    queryset = Post.objects.all() # 쿼리셋 등록!
    serializer_class = PostSerializer # Serializer 클래스 등록!
```

```
# get은 list메소드를 내보내는 메소드
```

```
1) { def get(self, request, *args, **kwargs):
    return self.list(request, *args, **kwargs)
```

```
# post는 create을 내보내는 메소드
```

```
2) { def post(self, request, *args, **kwargs):
    return self.create(request, *args, **kwargs)
```

-> Mixin의 메서드를
사용할 때는
믹스인 클래스를
정의해주고
믹스인 메서드를
사용한다.

참고 사이트 :
[https://github.com/encode/django-rest-](https://github.com/encode/django-rest-framework)

Mixin을 다루기 위해 먼저 알아야할 2가지

1. `from rest_framework import generics`

https://github.com/encode/django-rest-framework/blob/master/rest_framework/generics.py

```
class GenericAPIView(views.APIView):
```

GenericAPIView 를 상속!

```
Base class for all other generic views.
```

```
"""
```

```
# You'll need to either set these attributes,
```

```
# or override get_queryset()/get_serializer_class().
```

```
# If you are overriding a view method, it is important that you call
```

```
# get_queryset() instead of accessing the queryset property directly,
```

```
# as queryset will get evaluated only once, and those results are cached
```

```
# for all subsequent requests.
```

```
queryset = None
```

```
serializer_class = None
```

`queryset` 사용할 모델 정의 하기!!

`serializer_class` 사용할 모델 직렬화 하기!!

Mixin을 다루기 위해 먼저 알아야할 2가지

2. `from rest_framework import mixins`

https://github.com/encode/django-rest-framework/blob/master/rest_framework/mixins.py

```
def get(self, request):  
    posts = Post.objects.all()  
    serializer = PostSerializer(posts, many=True)  
    return Response(serializer.data)
```



Mixin 클래스의 메서드를 상속!
Http 메서드와 한 줄로 작성된 리턴 문을 사용함!

```
# get은 list메소드를 내보내는 메소드  
def get(self, request, *args, **kwargs):  
    return self.list(request, *args, **kwargs)
```

Http 메서드	Mixin 클래스	Mixin 메서드 (리턴문)
----------	-----------	-----------------

get

```
class ListModelMixin
```

```
class RetrieveModelMixin
```

list 객체 조회하기 (리스트)
retrieve 객체 조회하기 (디테일)

put

```
class UpdateModelMixin:
```

update 객체 수정하기

post

```
class CreateModelMixin:
```

create 객체 생성하기

delete

```
class DestroyModelMixin:
```

destroy 객체 삭제하기

Mixin 사용하기



mixin_views.py

```
class PostDetail(mixins.RetrieveModelMixin, mixins.UpdateModelMixin,
                 mixins.DestroyModelMixin, generics.GenericAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer

    # DetailView의 get은 retrieve을 내보내는 메소드
    def get(self, request, *args, **kwargs):
        return self.retrieve(request, *args, **kwargs)

    # put은 update을 내보내는 메소드
    def put(self, request, *args, **kwargs):
        return self.update(request, *args, **kwargs)

    # delete은 destroy를 내보내는 메소드
    def delete(self, request, *args, **kwargs):
        return self.destroy(request, *args, **kwargs)
```

Mixin 사용하기



```
from rest_framework.urlpatterns import format_suffix_patterns
from django.urls import path
from post import mixin_views
```

Default Router 사용X > API ROOT 없음

```
urlpatterns = [
    path('post/', mixin_views.PostList.as_view()),
    path('post/<int:pk>', mixin_views.PostDetail.as_view()),
]
```

```
urlpatterns = format_suffix_patterns(urlpatterns)
```

Mixin 사용하기

python manage.py runserver

Api Root / Post List

Post List

OPTIONS GET ▾

GET /post/

json
api

HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[]

Raw data HTML form

Title

Body

POST

Post List / Post Detail

Post Detail

DELETE OPTIONS GET ▾

GET /post/2

HTTP 404 Not Found
Allow: GET, PUT, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
 "detail": "Not found."
}

Http Method

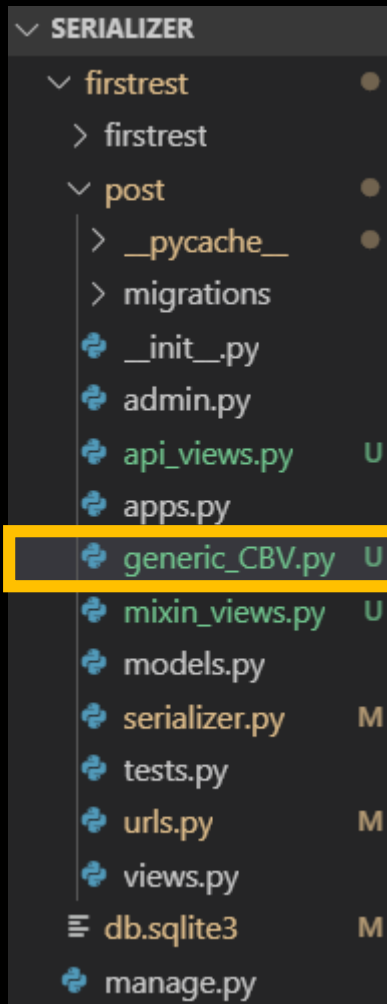
Media type: application/json ▾

Content:

PUT

더더욱 간단한 !!!
Generic class 사용하기

더더욱 간단한 !!! Generic class 사용하기



```
from post.models import Post
from post.serializer import PostSerializer
from rest_framework import generics
```

https://github.com/encode/django-rest-framework/blob/master/rest_framework/generics.py

더더욱 간단한 !!! Generic class 사용하기

List만들기

Mixin

https://github.com/encode/django-o-rest-framework/blob/0e1c5d313232a131bb4a1a414abf921744ab40e0/rest_framework/generics.py#L232

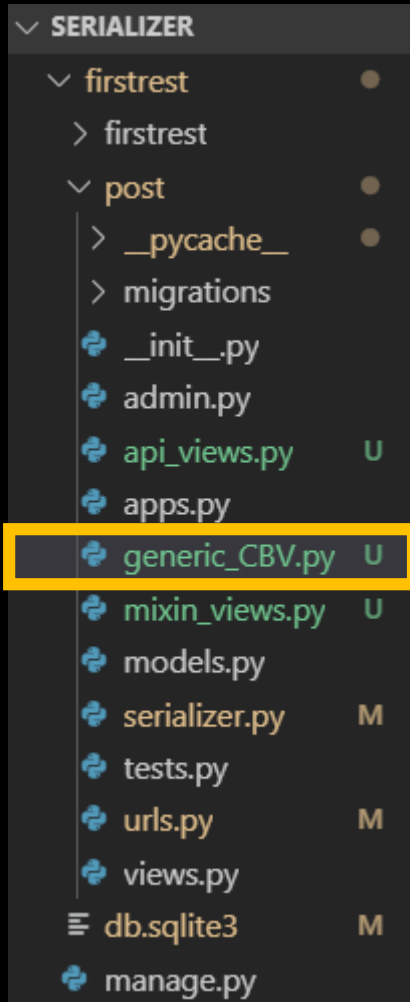
```
class SnippetList(mixins.ListModelMixin,
                  mixins.CreateModelMixin,
                  generics.GenericAPIView):
    queryset = Snippet.objects.all()
    serializer_class = SnippetSerializer

    def get(self, request, *args, **kwargs):
        return self.list(request, *args, **kwargs)

    def post(self, request, *args, **kwargs):
        return self.create(request, *args, **kwargs)
```

Generic_class

```
class PostList(generics.ListCreateAPIView):
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```



더더욱 간단한 !!! Generic class 사용하기

Detail만들기

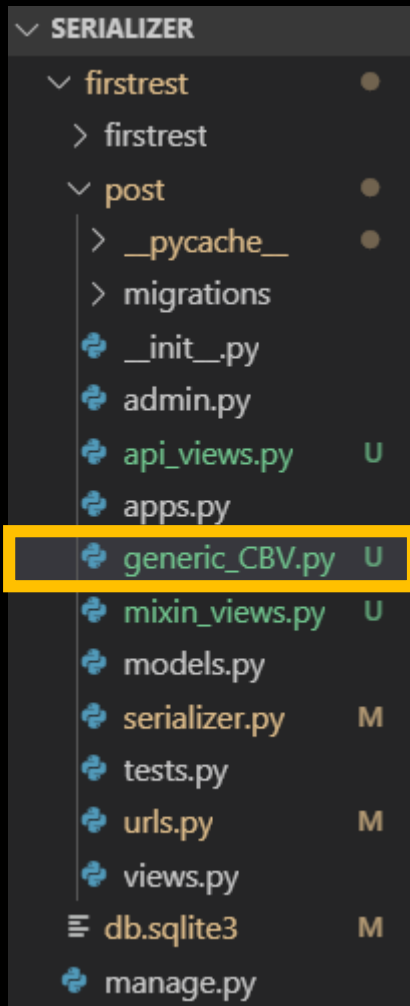
Mixin

https://github.com/encode/django-rest-framework/blob/0e1c5d313232a131bb4a1a414abf921744ab40e0/rest_framework/generics.py#L232

```
class RetrieveDestroyAPIView(mixins.RetrieveModelMixin,  
                              mixins.DestroyModelMixin,  
                              GenericAPIView):  
  
    """  
    Concrete view for retrieving or deleting a model instance.  
    """  
  
    def get(self, request, *args, **kwargs):  
        return self.retrieve(request, *args, **kwargs)  
  
    def delete(self, request, *args, **kwargs):  
        return self.destroy(request, *args, **kwargs)
```

Generic_class

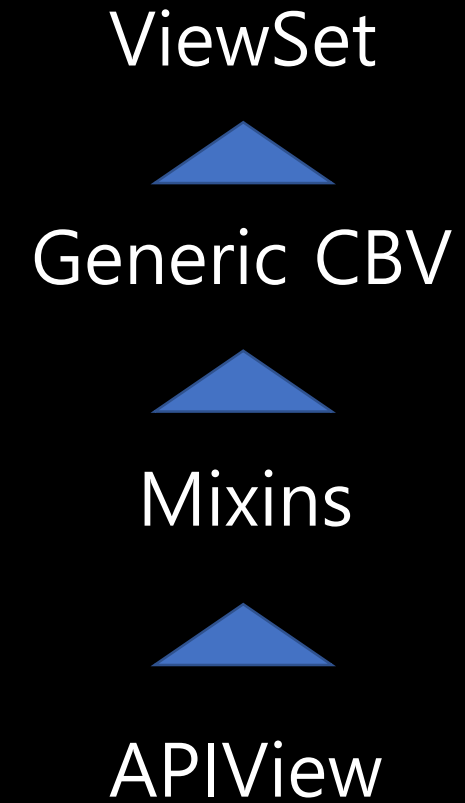
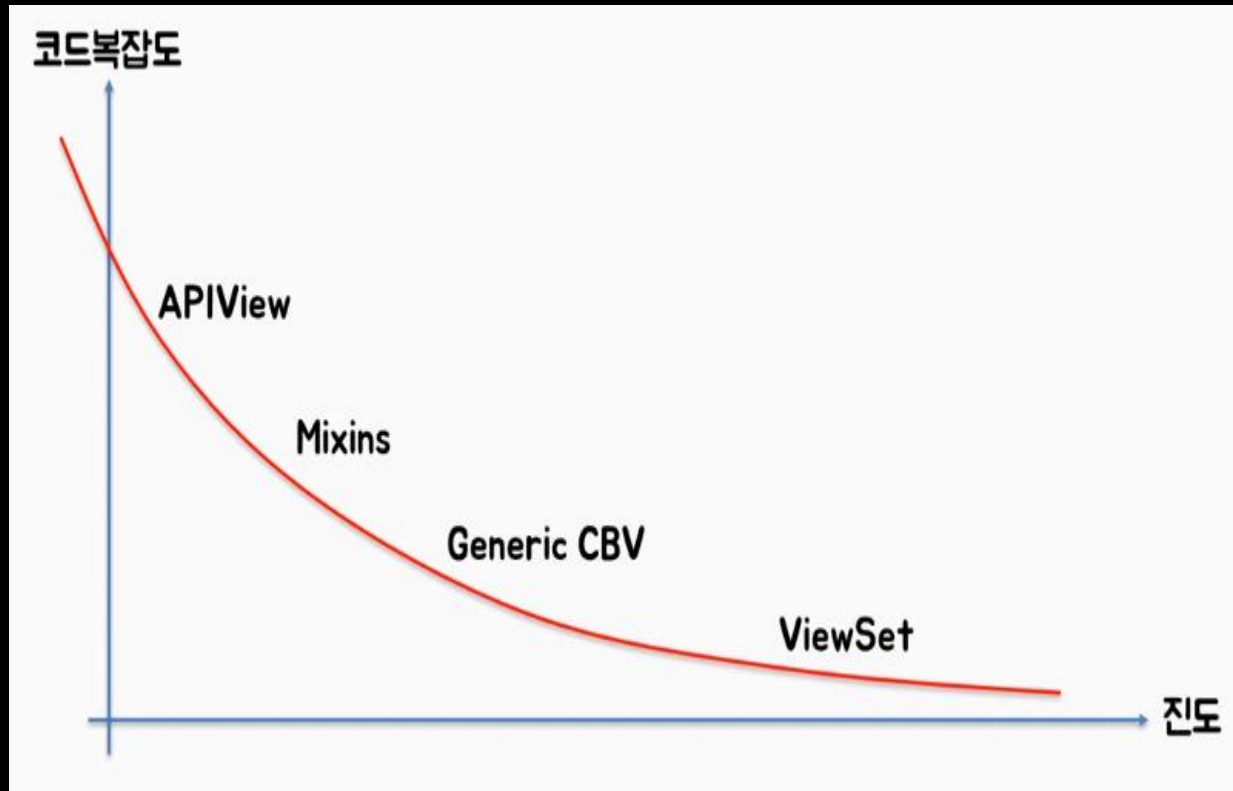
```
class PostDetail(generics.RetrieveUpdateDestroyAPIView):  
    queryset = Post.objects.all()  
    serializer_class = PostSerializer
```



viewset과 router를
이해하기 위한 간단한 상속 개념 알기

끝

ViewSet에 이르는 과정



Django "REST" Framework

공식 홈페이지를 보고 viewsets-and-routers 알아가보자~~

<https://www.django-rest-framework.org/tutorial/6-viewsets-and-routers/>

The screenshot shows the Django REST Framework website. The navigation bar includes links for Home, Tutorial, API Guide, Topics, and Community, along with a search bar and GitHub link. The 'Tutorial' dropdown menu is open, showing a list of tutorials from Quickstart to Viewsets and routers. The 'Class-based Views' tutorial is selected. The main content area displays the 'Class-based Views' section, which explains that class-based views are a powerful pattern for keeping code DRY. It also shows a code snippet for a class-based view, `SnippetList`, which inherits from `APIView` and implements the `get` method to return a list of snippets.

Django REST framework Home Tutorial API Guide Topics Community Search Previous Next GitHub

Tutorial 3: Class-based Views
Rewriting our API using class-based views
Using mixins
Using generic class-based views

Class-based Views
Class-based views, rather than function based views. As we'll see this is a powerful pattern helps us keep our code DRY.

Class-based views
We start by rewriting the root view as a class-based view. All this involves is a little bit of refactoring of `views.py`.

```
from snippets.models import Snippet
from snippets.serializers import SnippetSerializer
from django.http import Http404
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status

class SnippetList(APIView):
    """
    List all snippets, or create a new snippet.
    """
    def get(self, request, format=None):
        snippets = Snippet.objects.all()
        serializer = SnippetSerializer(snippets, many=True)
        return Response(serializer.data)

    def post(self, request, format=None):
```

viewset과 router 알아보기

1) 읽기 전용 viewsets -> list + detail

```
class ReadOnlyModelViewSet(mixins.RetrieveModelMixin,  
                             mixins.ListModelMixin,  
                             GenericViewSet):
```

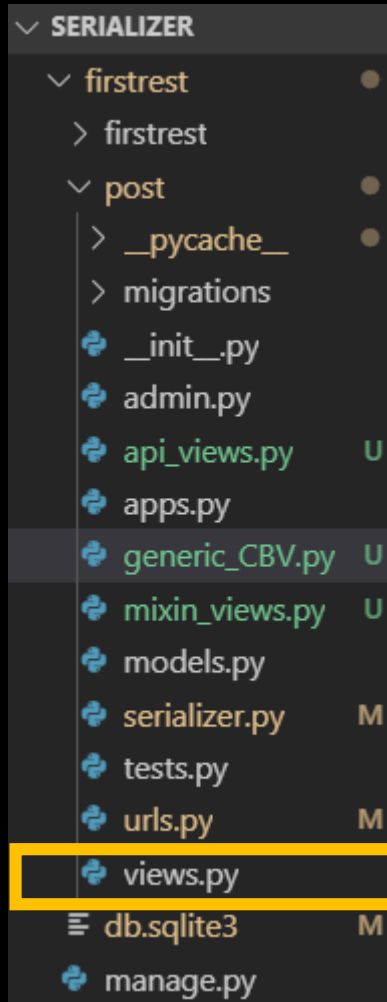
List()
Retrieve()

2) CRUD 전용 viewsets

```
class ModelViewSet(mixins.CreateModelMixin,  
                   mixins.RetrieveModelMixin,  
                   mixins.UpdateModelMixin,  
                   mixins.DestroyModelMixin,  
                   mixins.ListModelMixin,  
                   GenericViewSet):
```

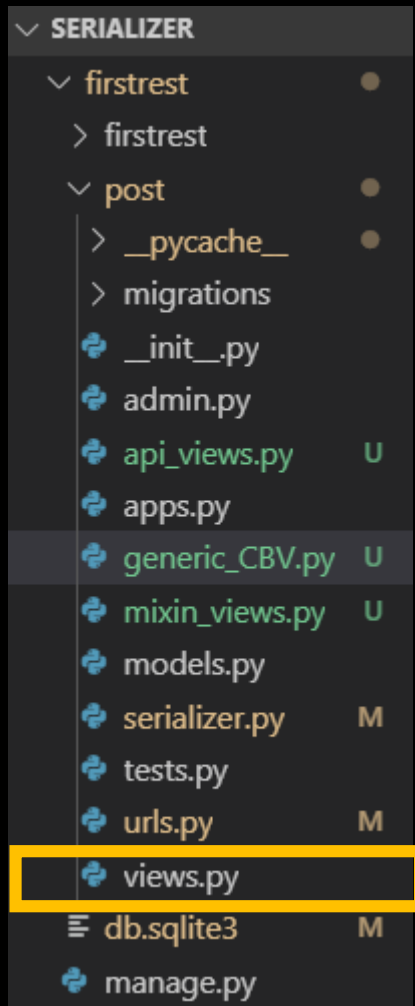
List()
Retrieve()
Create()
Update()
Partial_update()
Destroy()

viewset과 router 알아보기



```
from post.models import Post
from post.serializer import PostSerializer
from rest_framework import viewsets
```

viewset과 router 알아보기



```
# ReadOnlyModelViewSet은 말 그대로  
# ListView, DetailView의 조회만 가능
```

```
class PostViewSet(viewsets.ReadOnlyModelViewSet):  
    queryset = Post.objects.all()  
    serializer_class = PostSerializer
```

ReadOnlyModelViewSet은
기본적으로 Listview와 detailview에서 사용되고 있는 기능들인
List() - 작성된 글의 목록을 조회하는 기능,
Retrive() - 작성된 글을 검색하는 기능
즉 ! 글들을 읽기만 하는 기능들을 모아둔 viewset 이다!

viewset과 router 알아보기

https://github.com/encode/django-rest-framework/blob/master/rest_framework/viewsets.py

링크를 따라 들어가면
ViewSet에 관련한 상속파일들을
볼 수 있다는 것

```
class ReadOnlyModelViewSet(mixins.RetrieveModelMixin,  
                             mixins.ListModelMixin,  
                             GenericViewSet):
```

```
    """
```

```
    A viewset that provides default
```

```
    """
```

```
    pass
```

Definition References

Defined on line 199

199 `class GenericViewSet(ViewSetMixin, generics.GenericAPIView):`

ReadOnlyModelViewSet과 ModelViewSet의
기능들이 어떤 걸 상속해서
작동하는 지 보다 보면
커스텀이 어떤 식으로 되었는 지
어떤 기능을 추가로 사용할 수 있는지
알 수 있쥬

```
class GenericViewSet(ViewSetMixin, generics.GenericAPIView):
```

```
    Base class for all other generic views.
```

```
    """
```

```
    # You'll need to either set these attributes,
```

```
    # or override `get_queryset()`/`get_serializer_class()`.
```

```
    # If you are overriding a view method, it is important that you call
```

```
    # `get_queryset()` instead of accessing the `queryset` property directly,
```

```
    # as `queryset` will get evaluated only once, and those results are cached
```

```
    # for all subsequent requests.
```

```
    queryset = None
```

```
    serializer_class = None
```

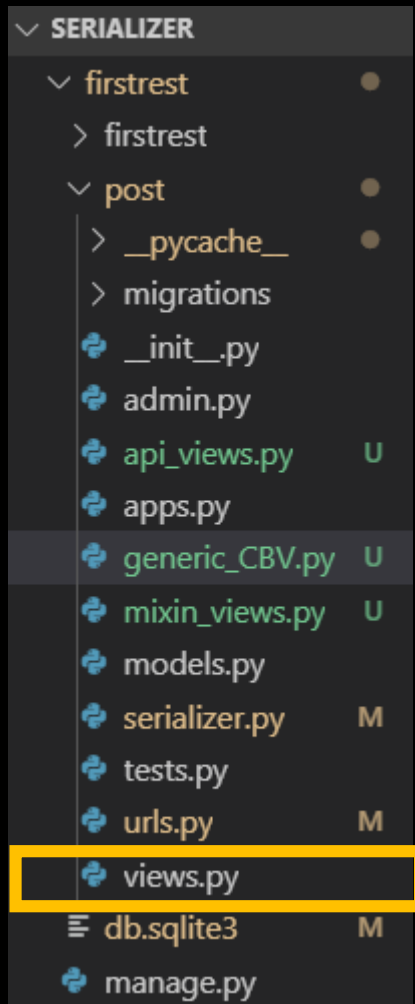

글을 읽기만 하는 것뿐만 아니라
CRUD까지 할 수 있는 끝판왕 viewset은 없으까,,,

그게 바로!!
CRUD 전용 viewsets 인 **ModelViewSet!!**
요걸로 읽기부터 crud까지 한방에 모든 걸 끝내버리자

```
class ModelViewSet(mixins.CreateModelMixin,  
                    mixins.RetrieveModelMixin,  
                    mixins.UpdateModelMixin,  
                    mixins.DestroyModelMixin,  
                    mixins.ListModelMixin,  
                    GenericViewSet):
```

viewset과 router 알아보기

2) CRUD 전용 ModelViewsets 만들기



```
from post.models import Post
from post.serializer import PostSerializer
from rest_framework import viewsets
```

```
# @action처리
from rest_framework import renderers
from rest_framework.decorators import action
from rest_framework.response import Response
from django.http import HttpResponse
```

viewset과 router 알아보기

2) CRUD 전용 ModelViewsets 만들기

Views.py

ModelViewSet은
기본적으로

Listview와 detailview에서
사용되고 있는 기능들인

List()

- 작성된 글의 목록을 조회기능,

Retrive()

- 작성된 글을 검색하는 기능

추가로

글들을 읽고

crud 하는 기능들을 모아둔

viewset 이다!

1)

2)

```
from post.models import Post
from post.serializer import PostSerializer
from rest_framework import viewsets

# @action처리
from rest_framework import renderers
from rest_framework.decorators import action
from rest_framework.response import Response
from django.http import HttpResponseRedirect

# Create your views here.

# ModelViewSet은 ListView와 DetailView에 대한 CRUD가 모두 가능
class PostViewSet(viewsets.ModelViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer

    # @action(method=['post'])
    @action(detail=True, renderer_classes=[renderers.StaticHTMLRenderer])
    # 그냥 압을 띄우는 custom api
    def highlight(self, request, *args, **kwargs):
        return HttpResponseRedirect("압")
        #return Response(snippet.highlighted)
```

viewset과 router 알아보기

2) CRUD 전용 ModelViewsets 만들기

Views.py

1)

```
class PostViewSet(viewsets.ModelViewSet):  
    queryset = Post.objects.all()  
    serializer_class = PostSerializer
```

작성한 글 목록을 불러오고
그 글에 대한 정보를 볼 수 있는 기능을 했던
ReadOnlyViewSet과 동일한 코드지만!

상속받은 클래스는 (viewsets.ModelViewSet)
동일한 코드를 반복해서 입력하더라도
ModelViewSet을 상속받아 작성한 거라면!
클래스에 미리 정의되어진 기능들을 적용하여 사용할 수 있다!

viewset과 router 알아보기

2) CRUD 전용 ModelViewsets 만들기

Views.py

2)

```
from rest_framework.decorators import action
from rest_framework.response import Response
```

```
# @action(method=['post'])
@action(detail=True, renderer_classes=[renderers.StaticHTMLRenderer])
# 그냥 압을 띄우는 custom api
def highlight(self, request, *args, **kwargs):
    return HttpResponse("압")
    #return Response(snippet.highlighted)
```

`@action(detail=True, renderer_classes=[renderers.StaticHTMLRenderer])`

이것은 읽고, 검색하고, crud 기능 말고
다른 기능을 추가로 적용하고 싶을 때
@action을 활용해서!
꾸밀 수 있다는 것!

Response를 어떤 형식으로 Rendering 시킬 것인가
(rest_framework/renderer.py)

- e.g.) `renderer.JSONRenderer`
- e.g.) `renderer.BrowsableAPIRenderer`
- etc) `StaticHTMLRenderer, TemplateHTMLRenderer`

viewset과 router 알아보기

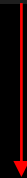
2) CRUD 전용 ModelViewsets 만들기

Views.py

2)

```
from rest_framework.decorators import action
from rest_framework.response import Response

# @action(method=['post'])
# @action(detail=True, renderer_classes=[renderers.StaticHTMLRenderer])
# 그냥 압을 띄우는 custom api
def highlight(self, request, *args, **kwargs):
    return HttpResponse("압")
    #return Response(snippet.highlighted)
```



highlight 기능은
요청 받았을 때 "압"이라고 쓰여진 페이지로 보낼 수 있는 기능을
사용자가 직접 만들어서 정의한 함수!

@action 으로 직접 사용자가 함수를 정의해서 만들 수 있다~

viewset과 router 알아보기

2) CRUD 전용 ModelViewsets 만들기

Views.py

2)

```
from rest_framework.decorators import action
from rest_framework.response import Response
```

```
# @action(method=['post'])
@action(detail=True, renderer_classes=[renderers.StaticHTMLRenderer])
# 그냥 압을 띄우는 custom api
def highlight(self, request, *args, **kwargs):
    return HttpResponse("압")
    #return Response(snippet.highlighted)
```



Custom API 로 작성된 메서드들은
default 는 GET방식으로 호출이 되면 처리됨!!
action format 인자로 설정이 가능하다
이렇게 ~ `# @action(method=['post'])`
즉 원한다면 HTTP method인 post/put/delete 로
받아서 사용할 수 있다는 점!

viewset과 router 알아보기

 urls.py

```
from django.urls import path, include
from rest_framework.routers import DefaultRouter
# 장고 레스트 프레임워크는 라우터를 통해 url을 전달함
from . import views
appname = 'post'

router = DefaultRouter()
router.register('post', views.PostViewSet)

urlpatterns = [
    path('', include(router.urls)), # 실제로 url을 작성하게 하는 것
]
```


드디어 적용한 뷰셋과 라우터

```
python manage.py runserver
```

ROUTER

router 알아보기

```
urlpatterns = [  
    # 127.0.0.1:8000/post == ListView  
    path('post/', views.PostList.as_view()),  
    # 127.0.0.1:8000/post/<pk> == DetailView  
    path('post/<int:pk>', views.PostDetail.as_view()),  
]
```

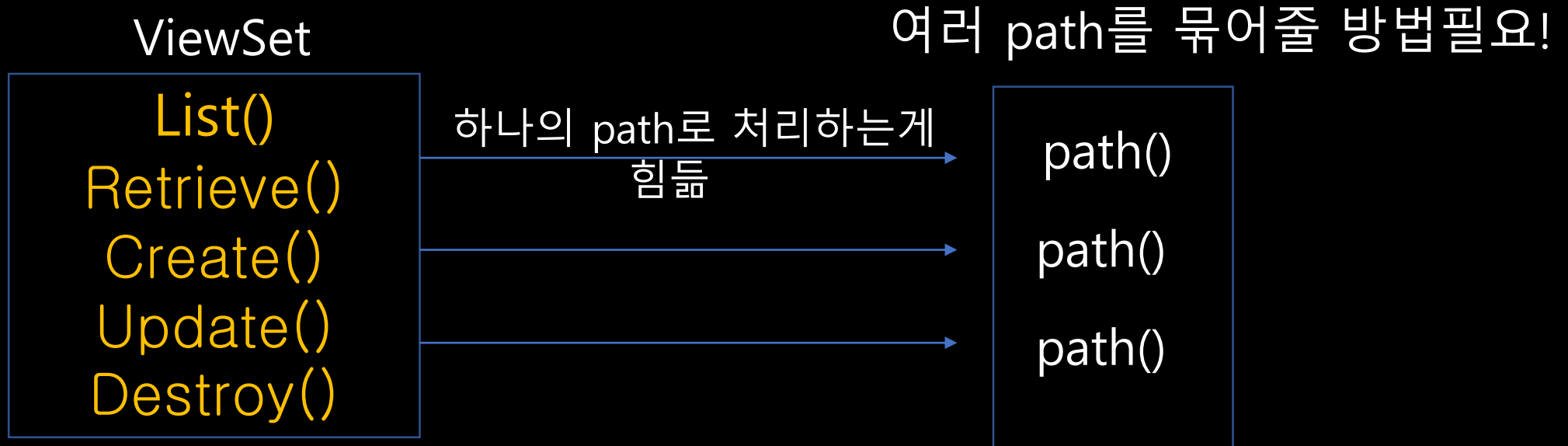


```
router = DefaultRouter()  
router.register('post', views.PostViewSet)  
  
urlpatterns = [  
    path('', include(router.urls)), # 실제로 url을 작성하게 하는 것  
]
```

router 알아보기

ViewSet 는 하나의 path 함수로는 처리할 수 없다

서로 다른 2개 이상의 path 함수들을 묶어주는 과정이 필요!



router 알아보기
여러 path를 묶어줄 방법필요!

path()
path()
path()

Path함수의 두번째 인자로
오는 함수를 묶는다!

path('post/', api_views.PostList.as_view()),
path(요청을 처리할 url, 요청을 인자로 받아 처리할 함수, namespace),

router 알아보기
여러 path를 묶어줄 방법필요!

Path함수의 두번째 인자로
오는 함수를 묶는다!

path('post/', api_views.PostList.as_view()),
path(요청을 처리할 url, 요청을 인자로 받아 처리할 함수, namespace),

as_view({'http_method' : '처리할_함수'})

```
mypath = PostViewSet.as_view({  
    'get': 'retrieve',  
    'put': 'update',  
    'patch': 'partial_update',  
    'delete': 'destroy'  
})  
  
urlpatterns = [  
    path('', mypath)  
]
```

router 알아보기

코드의 낭비를 줄이고 원하는 대로 기능을 **control**하고 싶다면?!

router

router.**register**(요청을 처리할 url, views.모델ViewSet, namespace),


```
urlpatterns = [  
    path('', include(router.urls)), # 실제로 url을 작성하게 하는 것  
]
```

CRUD 와 같이 기본적으로 router에 정의된 url들을
사용함

router 알아보기

코드의 낭비를 줄이고 원하는 대로 기능을 **control**하고 싶다면?!

router



<https://github.com/encode/django-rest-framework/blob/master/docs/api-guide/routers.md>

원한다면 라우터가 구성하는 urlpatterns를 살펴보시길!

DjangoRestFramework를

원하는 대로 커스텀하기 위한

viewset&Router 이해하기

예제 코드가 있는

<https://www.django-rest-framework.org/tutorial3/class-based-views/>

이 페이지를 잘 읽고 따라가 보길 바랄게요!

고생하셨습니다