



# 8주차 DRF (2)



서강대 김동윤



# 목차

---

1. Model
2. View, Serializer
3. Mixins 활용 CRUD

# Model

---

## 1. 블로그의 Post 모델을 만들어줍시다.

```
from django.db import models

# 블로그는 다양한 포스트 단위로 구분
# 하나의 포스트에는 제목, 내용, 작성시간 정보가 들어있을 예정

# django.db.models.Model 을 상속 받는 클래스를 정의하게 되면
# django가 관리하는 하나의 데이터 모델이 생성되는 것
class Post(models.Model) :

    title = models.CharField(max_length=200)
    content = models.CharField(max_length=2048)
    #제목과 내용은 문자열로 된 필드,
    date = models.DateTimeField(auto_created=True, auto_now=True)
    # 작성 시간은 '자동으로 생성
    # 현재 시간이 자동으로 기록되는' DateTime 필드
```

현재 내용을 DB에 적용해줍니다.  
이를 마이그레이션(Migration) 작업  
이라고 합니다.

(DB와 Django 앱간의 데이터 모델 동기화)

```
DONGYUN@DESKTOP-HN4KK92 MINGW64 ~/Desktop/멋사/세션/drf(1)/drf (main|SPARSE)
$ python manage.py makemigrations
Migrations for 'blog':
  blog\migrations\0001_initial.py
    - Create model Post

DONGYUN@DESKTOP-HN4KK92 MINGW64 ~/Desktop/멋사/세션/drf(1)/drf (main|SPARSE)
$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, blog, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying blog.0001_initial... OK
  Applying sessions.0001_initial... OK
```

```
$ python manage.py makemigrations
```

해당 명령을 수행하면, Django가 데이터 모델을 추적하여 마이그레이션 데이터를 추출해 줍니다.  
마이그레이션 파일은 해당 앱 내에 migrations 폴더 내에 위치시켜 주지요.

```
$ python manage.py migrate
```

마이그레이션 데이터를 적용해야 합니다.  
INSTALLED\_APPS에 앱이 추가 되고,  
해당 마이그레이션 데이터가 추가되면 마이그레이션이 가능하게 됩니다. migrate 명령을 통해 마이그레이션을 진행해줍니다.

이제 view를 구성해 봅시다.

잠깐만!

Django에서는 View를 만드는 방법은 크게 2가지

- 1) 함수 기반 뷰(Function based view, view function)
- 2) 클래스 기반 뷰(Class-based view)

FBV는 함수로 정의하기에 직관적이고 구현하기 쉽다는 장점이 있지만, 확장성과 재사용성이 낮아 CBV(클래스 기반 뷰) 등장~!

하지만 CBV가 FBV를 완벽하게 대체하는 것은 아니라 정답은 없습니다.

- FBV
  - 직접 함수를 작성하여 request를 처리
  - 다수의 request 메소드를 사용할 경우, if 조건문으로 구분하여 처리
- CBV
  - django.views.View 클래스를 상속 생성
  - dispatch() 메소드가 HTTP 메소드 로직을 처리,
  - request가 GET이라면 해당 클래스 내의 get() 메소드를 실행

```
#views.py
from django.http import HttpResponse

def my_view(request):
    if request.method == 'GET':
        # Code block for GET request
        return HttpResponse('result')
```

```
#views.py
from django.http import HttpResponse
from django.views import View

class MyView(View):
    def get(self, request):
        # <view logic>
        return HttpResponse('result')
```

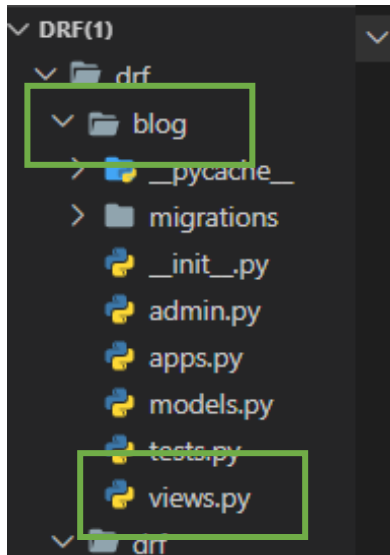
# View, Serializer

---



## 2. 이제 Serializer 를 구성해 봅시다.

Serializer : REST 로 데이터를 주고 받을 때, 모델을 어떻게 주고 받을 것인가를 정의하기 위한 클래스  
저희는 데이터 모델 그 자체를 주고 받을 것 이니, 기본적으로 모델 전체를 자동으로 변환해주는  
ModelSerializer 의 힘을 빌릴겁니다!



```
from django.shortcuts import render
from rest_framework.generics import GenericAPIView
from rest_framework import serializers
```

```
from drf.blog.models import Post
```

#blog\_api 위에 정의해주세요!  
#원래는 serializers.py로 따로 빼주는 것이 일반적이거나,  
오늘은 간단한 serializer만 제작할 것이니  
그냥 views.py에 작성해줍시다!

```
class PostSerializer(serializers.ModelSerializer) :
    class Meta :
        model = Post
        fields = '__all__'
```

- rest\_framework.serializers 에 포함되어 있는 ModelSerializer 는 Meta 클래스를 요구합니다.
- 클래스 안에 Meta 클래스를 정의하는 것으로, ModelSerializer가 자신이 필요한 정보들을 파악하여 자동으로 Serialize 를 수행하게 될 것입니다.
- ModelSerializer는 다양한 메타 데이터를 포함할 수 있지만, model 데이터만을 넘겨줘도, 저희가 쓰기에는 충분!
- Field는 모델에서 어떤 필드를 이용할 건지 명시해주는 것입니다!

## 3. 이제 view를 구성해 봅시다.

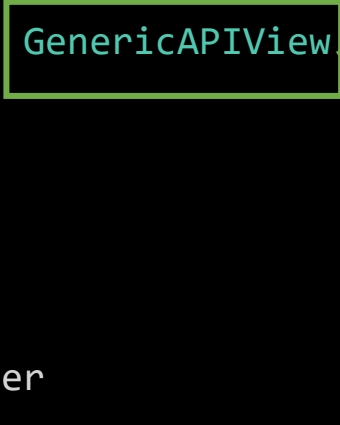
```
from django.shortcuts import render
from rest_framework.generics import GenericAPIView

from drf.blog.models import Post

class blog_api(GenericAPIView):

    queryset = Post.objects.all()
    serializer_class = PostSerializer

#serializer_class 는 PostSerializer 를 설정
```



- views 에도 REST API를 위한 클래스를 추가 합니다.
- 함수 기반으로 직접 추가해도 괜찮지만, 저희는 REST Framework를 통하여 쉽고 빠르게 확장을 할 것이므로 함수 기반이 아닌 클래스 기반으로!
- GenericAPIView 를 상속받아서, 일반적인 REST API View를 하나 만듭니다.
- GenericAPIView는 as\_view() 라는 함수를 가지고 있는데, 이 함수는 기본적인 REST Framework 웹 페이지를 하나 출력해 줍니다.

drf 를 사용하기 위해서는,  
기본적으로 queryset 이라는 멤버와  
serializer\_class 라는 멤버(또는 get\_serializer\_class() 라는 함수)를 제공해줘야 합니다.  
이를 통해서 GenericAPIView는  
기본적인 REST 기능을 수행할 수 있게 됩니다.

**Queryset** : 어떠한 모델을 보여줄것인가? 를 정의  
데이터베이스에서 전달 받은 객체의 목록 (Django ORM에서 발생한 자료형)  
# Post 데이터들을 보여줄 테니, 전체 Post 데이터를 반환

**Serializer** : REST 로 데이터를 주고 받을 때, 모델을 어떻게 주고 받을 것인가를 정의하기 위한 클래스  
우리는 데이터 모델 그 자체를 주고 받을 것 이니, 기본적으로 모델 전체를 자동으로 변환해주는 ModelSerializer 힘을 빌릴거예요~!

이제 GET 요청에 대한 처리를 구현할 차례 입니다.  
기본적으로 REST Framework는 GET / POST / PATCH / DELETE 등의  
요청에 대한 기본 처리를 제공 GenericAPIView 에서  
각각 get / post / delete 등의 이름으로 정의가 되어 있습니다.

직접 처리하는 것도 방법이지만,  
GenericAPIView는 다른 Mixin 클래스와의 조합으로  
쉽고 빠르게 구현하는 방법들을 제공해줍니다.  
rest\_framework.mixins 를 import 하고,  
ListModelMixins 라는 클래스 또한 다중 상속하여 해당 기능을 구현해 봅시다.

## 왜 믹스인 클래스를 사용할까?

만약 Django Rest Framework Mixin을 사용하지 않는다면,  
모든 기능을 View에 직접, 반복적으로 구현해야 할 것입니다.

하지만 API를 작업할 때  
목록을 보여주거나, 생성, 삭제, 수정 등은  
항상 사용되는 반복적인 일!

이러한 반복적인 기능을 하나의 Mixin 클래스로 제공한다면  
반복적인 일을 줄여주고  
가독성, 생산성을 높여줄 수 있어 사용합니다!

( 물론더 간소화할 수 있는 viewset 클래스도 존재하니,  
실습이 끝나고 여러분들이 직접 viewset으로 리팩토링 해보셔도 좋을 것 같습니다 ☺ )

# Mixins 활용한 메소드

---

Mixins 는 CBV 중 하나이기 때문에  
하나의 class 당 하나의 URL 에 대해서만  
처리를 할 수 있습니다.

/blog/ 에 대한 CBV

get : 블로그 목록  
post : 새 블로그 생성

/blog/<int:pk>/ 에 대한 CBV

get : pk 번 블로그 내용  
put : pk번 블로그 수정  
delete : pk번 블로그 삭제

## 1) ListModelMixin

- Queryset을 리스팅하는 믹스인
- `.list(request, *args, **kwargs)` 메소드로 호출하여 사용
- `GenericAPIView`의 `self.filter_queryset`, `self.get_queryset`, `self.get_serializer` 등의 메소드를 활용해 데이터베이스에 저장되어 있는 데이터들을 목록 형태로 response body로 리턴
- 성공 시, 200 OK response 리턴

## 2) CreateModelMixin

- 모델 인스턴스를 생성하고 저장하는 역할을 하는 믹스인
- `.create(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 201 Created 리턴
- 실패 시, 400 Bad Request 리턴

## 3) RetrieveModelMixin

- 존재하는 모델 인스턴스를 리턴해 주는 믹스인
- `.retrieve(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 200 OK response 리턴
- 실패 시, 404 Not Found 리턴

## 4) UpdateModelMixin

- 모델 인스턴스를 수정하여 저장해 주는 믹스인
- `.update(request, *args, **kwargs)` 메소드로 호출하여 사용
- 부분만 변경하고자 할 경우, `.partial_update(request, *args, **kwargs)` 메소드를 호출하여야 하며,
- 이 때 요청은 HTTP PATCH requests여야 함
- 성공 시, 200 OK response 리턴
- 실패 시, 404 Not Found 리턴

## 5) DestroyModelMixin

- 모델 인스턴스를 삭제하는 믹스인
- `.destroy(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 204 No Content 리턴
- 실패 시, 404 Not Found 리턴

## 1) ListModelMixin

- Queryset을 리스팅하는 믹스인
- `.list(request, *args, **kwargs)` 메소드로 호출하여 사용
- `GenericAPIView`의 `self.filter_queryset`, `self.get_queryset`, `self.get_serializer` 등의 메소드를 활용해 데이터베이스에 저장되어 있는 데이터들을 목록 형태로 response body로 리턴
- 성공 시, 200 OK response 리턴

## 2) CreateModelMixin

- 모델 인스턴스를 생성하고 저장하는 역할을 하는 믹스인
- `.create(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 201 Created 리턴
- 실패 시, 400 Bad Request 리턴

## 3) RetrieveModelMixin

- 존재하는 모델 인스턴스를 리턴해 주는 믹스인
- `.retrieve(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 200 OK response 리턴
- 실패 시, 404 Not Found 리턴

## 4) UpdateModelMixin

- 모델 인스턴스를 수정하여 저장해 주는 믹스인
- `.update(request, *args, **kwargs)` 메소드로 호출하여 사용
- 부분만 변경하고자 할 경우, `.partial_update(request, *args, **kwargs)` 메소드를 호출하여야 하며,
- 이 때 요청은 HTTP PATCH requests여야 함
- 성공 시, 200 OK response 리턴
- 실패 시, 404 Not Found 리턴

## 5) DestroyModelMixin

- 모델 인스턴스를 삭제하는 믹스인
- `.destroy(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 204 No Content 리턴
- 실패 시, 404 Not Found 리턴



```
from django.shortcuts import render
from rest_framework.generics import GenericAPIView
from rest_framework import serializers, mixins
# ListModelMixin 사용 위해 mixins 추가
from drf.blog.models import Post

class blog_api(GenericAPIView, mixins.ListModelMixin):
    queryset = Post.object.all()
    serializer_class = PostSerializer

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

ListModelMixin 클래스를 상속 받으면  
list 라는 함수가 상속받아집니다.  
get 메소드에서 이를 호출하는 것으로,  
기본적으로 목록 조회 기능을 요청할 수 있습니다.

- 다중 상속을 통해, GenericAPIView의 기능과 ListModelMixin의 기능을 같이 가져왔습니다.
- ListModelMixin은 GenericAPIView에 queryset과 serializer\_class를 기반으로 하여 데이터 List를 만들어주는 기능을 합니다.
- 기본적으로 REST Framework에서는 request, \*args, \*\*kwargs를 반드시 포함해서 처리하게 되어 있습니다.
- 기본적으로 온 요청에 대한 Parsing 작업을 하여 Request를 생성하고, 그 외에 여러 데이터는 \*args와 \*\*kwargs에 포함하여 오는 형태 입니다.
- (+) 파이썬에서 \*, \*\*는 주소값을 저장하는 의미가 아닙니다. 여러 개의 인수를 받을 때, 키워드 인수를 받을 때 사용하는 표시

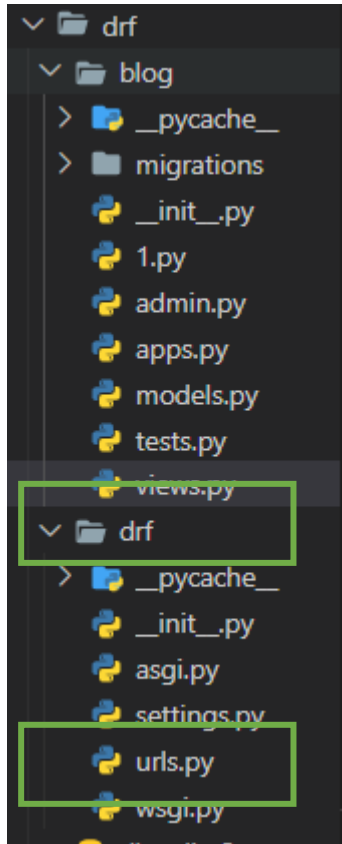
\*args는 "가변 인자"를 위한 변수  
함수의 인자를 몇 개 받을지 모르는 경우에 사용, 튜플 형태

\*\*kwargs는 keyword arguments의 약자  
딕셔너리 형태

```
def number_and_name(*args, **kwargs):  
    print(args, kwargs)  
  
number_and_name(1, 2, 3, name="홍길동")  
  
### 출력값 ###  
(1, 2, 3) #튜플 - args  
{'name': 'GilDong Hong'} # 딕셔너리 - kwargs
```

함수에 키-값 형태로 된 인자를 주면 파이썬은 자동으로 kwargs에 저장

5. view와 url을 연결해주기 위해 urls.py로 이동합시다!



```
from django.contrib import admin
from django.urls import path

from blog.views import blog_api

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/blog/', blog_api.as_view()),
]
```

\$ python manage.py runserver 을 이용해 서버를 실행 후,  
Url로 접속해봅시다!

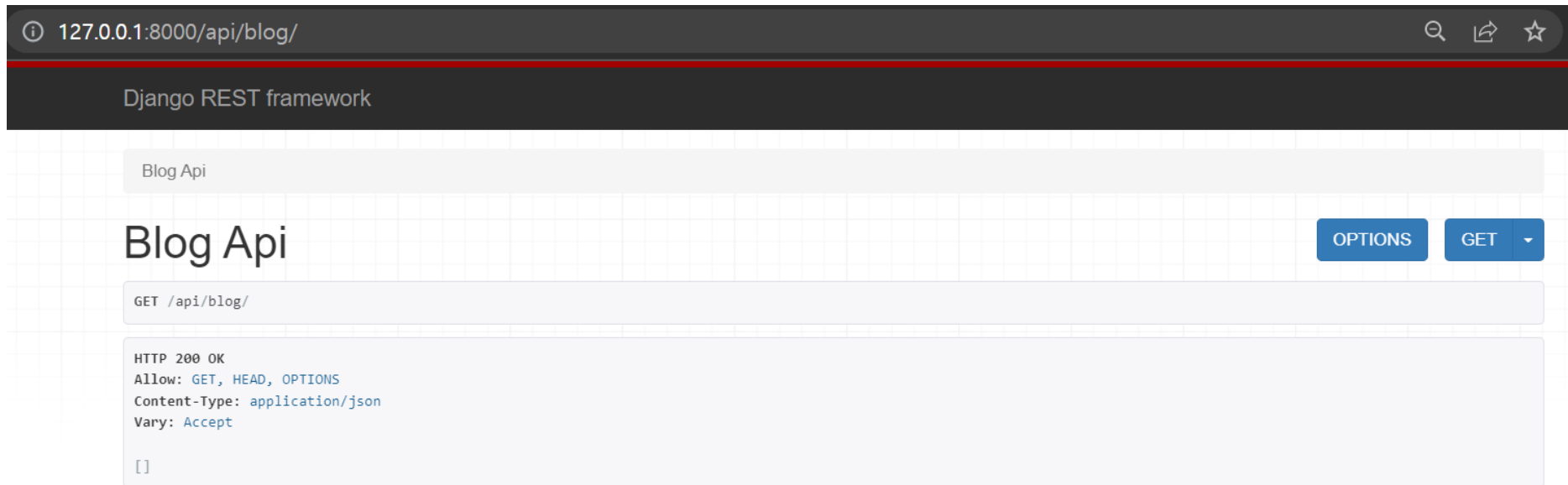
```
DONGYUN@DESKTOP-HN4KK92 MINGW64 ~/Desktop/멋사/세션/drf(1)/drf (main|
SPARSE)
$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified some issues:

WARNINGS:
?: (2_0.W001) Your URL pattern '^admin/' has a route that contains '(?P<', begins with a '^', or ends with a '$'. This was likely a n oversight when migrating to django.urls.path().

System check identified 1 issue (0 silenced).
May 10, 2022 - 16:48:24
Django version 4.0, using settings 'drf.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
Not Found: /
```

이런 식으로 나타나면 성공입니다!  
다만 지금 불러올 Post 리스트가 없어서  
아무것도 뜨지 않으니, 한번 Post 데이터들을 추가해봅시다.



6. Shell을 사용하여 Post 클래스 모델에 데이터를 추가, 저장해봅시다.

```
python manage.py shell #shell 실행
>>> from blog.models import Post #모델클래스 임포트
>>> from django.utils import timezone #타임존 모듈 임포트
>>> post1 = Post(title='제목1', content = 'post1의 내용', date = timezone.now())#데이터추가1
>>> post1.save() # 데이터 저장
>>> post2 = Post(title='제목2', content = 'post2의 내용', date = timezone.now())#데이터추가2
>>> post2.save()
>>> post3 = Post(title='제목3', content = 'post3의 내용', date = timezone.now())#데이터추가3
>>> post3.save()
>>> Post.objects.all() # 모델 데이터 전체 출력
>>> quit()
```

```
DONGYUN@DESKTOP-HN4KK92 MINGW64 ~/Desktop/멋사/세션 /drf(1)/drf (main|SPARSE)
$ python manage.py shell
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from blog.models import Post
>>> from django.utils import timezone
>>> post1 = Post(title='제목1', content = 'post의 내용', date = timezone.now())
>>> post1.save()
>>> Post.objects.all()
<QuerySet [<Post: Post object (1)>, <Post: Post object (2)>, <Post: Post object (3)>]>
>>> quit()
```

방금 내가 추가한 데이터들이  
이와 같이 잘 get 되어오면 성공입니다!

Django REST framework

Blog Api

## Blog Api

GET /api/blog/

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

```
[
  {
    "id": 4,
    "date": "2022-05-10T09:27:21.988887Z",
    "title": "제목1",
    "content": "post1의 내용"
  },
  {
    "id": 5,
    "date": "2022-05-10T09:27:22.017982Z",
    "title": "제목2",
    "content": "post2의 내용"
  },
  {
    "id": 6,
    "date": "2022-05-10T09:27:22.941522Z",
    "title": "제목3",
    "content": "post3의 내용"
  }
]
```

## 1) ListModelMixin

- Queryset을 리스팅하는 믹스인
- `.list(request, *args, **kwargs)` 메소드로 호출하여 사용
- `GenericAPIView`의 `self.filter_queryset`, `self.get_queryset`, `self.get_serializer` 등의 메소드를 활용해 데이터베이스에 저장되어 있는 데이터들을 목록 형태로 response body로 리턴
- 성공 시, 200 OK response 리턴

## 2) CreateModelMixin

- 모델 인스턴스를 생성하고 저장하는 역할을 하는 믹스인
- `.create(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 201 Created 리턴
- 실패 시, 400 Bad Request 리턴

## 3) RetrieveModelMixin

- 존재하는 모델 인스턴스를 리턴해 주는 믹스인
- `.retrieve(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 200 OK response 리턴
- 실패 시, 404 Not Found 리턴

## 4) UpdateModelMixin

- 모델 인스턴스를 수정하여 저장해 주는 믹스인
- `.update(request, *args, **kwargs)` 메소드로 호출하여 사용
- 부분만 변경하고자 할 경우, `.partial_update(request, *args, **kwargs)` 메소드를 호출하여야 하며,
- 이 때 요청은 HTTP PATCH requests여야 함
- 성공 시, 200 OK response 리턴
- 실패 시, 404 Not Found 리턴

## 5) DestroyModelMixin

- 모델 인스턴스를 삭제하는 믹스인
- `.destroy(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 204 No Content 리턴
- 실패 시, 404 Not Found 리턴



7. Post 클래스에 post api 메소드를 정의해봅시다.

```
from django.shortcuts import render
from rest_framework.generics import GenericAPIView
from rest_framework import serializers, mixins
# ListModelMixin 사용 위해 mixins 추가
from drf.blog.models import Post

class blog_api(GenericAPIView,
               mixins.ListModelMixin,
               mixins.CreateModelMixin):
    queryset = Post.objects.all()
    serializer_class = PostSerializer

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

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

8. Runserver 하시고 <http://127.0.0.1/api/blog> 접속해주세요!  
기존엔 없던 Post 요청을 보낼 수 있는 칸이 생성되었으면 성공입니다.  
한번 post 요청을 날려서 데이터를 생성해봅시다.

Raw dataHTML form

Title

제목4

Content

post4의 내용

POST

POST 요청으로 데이터가 201 생성되었다는 HTTP 상태코드가 뜨면 성공입니다!

## Blog Api

[OPTIONS](#)[GET](#) ▾

POST /api/blog/

HTTP 201 Created

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{
  "id": 7,
  "date": "2022-05-10T11:50:02.092420Z",
  "title": "제목4",
  "content": "P"
}
```

[Raw data](#)[HTML form](#)

Title

Content

[POST](#)

Mixins 는 CBV 중 하나이기 때문에  
하나의 class 당 하나의 URL 에 대해서만  
처리를 할 수 있습니다.

/blog/ 에 대한 CBV

get : 블로그 목록  
post : 새 블로그 생성

/blog/<int:pk>/ 에 대한 CBV

get : pk 번 블로그 내용  
put : pk번 블로그 수정  
delete : pk번 블로그 삭제

## 1) ListModelMixin

- Queryset을 리스팅하는 믹스인
- `.list(request, *args, **kwargs)` 메소드로 호출하여 사용
- `GenericAPIView`의 `self.filter_queryset`, `self.get_queryset`, `self.get_serializer` 등의 메소드를 활용해 데이터베이스에 저장되어 있는 데이터들을 목록 형태로 response body로 리턴
- 성공 시, 200 OK response 리턴

## 2) CreateModelMixin

- 모델 인스턴스를 생성하고 저장하는 역할을 하는 믹스인
- `.create(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 201 Created 리턴
- 실패 시, 400 Bad Request 리턴

## 3) RetrieveModelMixin

- 존재하는 모델 인스턴스를 리턴해 주는 믹스인
- `.retrieve(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 200 OK response 리턴
- 실패 시, 404 Not Found 리턴

## 4) UpdateModelMixin

- 모델 인스턴스를 수정하여 저장해 주는 믹스인
- `.update(request, *args, **kwargs)` 메소드로 호출하여 사용
- 부분만 변경하고자 할 경우, `.partial_update(request, *args, **kwargs)` 메소드를 호출하여야 하며,
- 이 때 요청은 HTTP PATCH requests여야 함
- 성공 시, 200 OK response 리턴
- 실패 시, 404 Not Found 리턴

## 5) DestroyModelMixin

- 모델 인스턴스를 삭제하는 믹스인
- `.destroy(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 204 No Content 리턴
- 실패 시, 404 Not Found 리턴

9. 이번엔 새로운 클래스를 만들어주고,  
여기다가 RetrieveModelMixin 추가 및 get 메소드를 구현할게요!

```
from django.shortcuts import render
from rest_framework.generics import GenericAPIView
from rest_framework import serializers, mixins
# ListModelMixin 사용 위해 mixins 추가
from drf.blog.models import Post

class blog_detail_api(GenericAPIView, mixins.RetrieveModelMixin):
    queryset = Post.objects.all()
    serializer_class = PostSerializer

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

why 새로운 클래스로 만드나요 😞 !?

지금부터 할 특정 인스턴스를 불러오기, 특정 인스턴스 수정/삭제하기는  
Url에 그 인스턴스의 id를 필요로 합니다.

따라서 하나의 class 당 하나의 URL 에 대해서만  
처리를 할 수 있는데 요청할 url이 달라져서 클래스를 분리하는 것이죠.

10. view를 연결할 새로운 url도 만들어줄게요!

```
from django.contrib import admin
from django.urls import path

from blog.views import blog_api, blog_detail_api

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/blog/', blog_api.as_view()),
    path('api/blog/<int:pk>/ ', blog_detail_api.as_view())
]
```

내가 get하길 원하는 인스턴스의 pk 값만 붙여서 보내주고,  
해당 인스턴스만 화면에 나온다면 성공입니다!

`http://127.0.0.1:8000/api/blog/4/`

## Blog Detail Api

`GET /api/blog/4/`

`HTTP 200 OK`

`Allow: GET, PUT, DELETE, HEAD, OPTIONS`

`Content-Type: application/json`

`Vary: Accept`

```
{
  "id": 4,
  "date": "2022-05-10T09:27:21.988887Z",
  "title": "제목1",
  "content": "post1의 내용"
}
```



## 1) ListModelMixin

- Queryset을 리스팅하는 믹스인
- `.list(request, *args, **kwargs)` 메소드로 호출하여 사용
- `GenericAPIView`의 `self.filter_queryset`, `self.get_queryset`, `self.get_serializer` 등의 메소드를 활용해 데이터베이스에 저장되어 있는 데이터들을 목록 형태로 response body로 리턴
- 성공 시, 200 OK response 리턴

## 2) CreateModelMixin

- 모델 인스턴스를 생성하고 저장하는 역할을 하는 믹스인
- `.create(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 201 Created 리턴
- 실패 시, 400 Bad Request 리턴

## 3) RetrieveModelMixin

- 존재하는 모델 인스턴스를 리턴해 주는 믹스인
- `.retrieve(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 200 OK response 리턴
- 실패 시, 404 Not Found 리턴

## 4) UpdateModelMixin

- 모델 인스턴스를 수정하여 저장해 주는 믹스인
- `.update(request, *args, **kwargs)` 메소드로 호출하여 사용
- 부분만 변경하고자 할 경우, `partial_update(request, *args, **kwargs)` 메소드를 호출하여야 하며,
- 이 때 요청은 HTTP PATCH requests여야 함
- 성공 시, 200 OK response 리턴
- 실패 시, 404 Not Found 리턴

## 5) DestroyModelMixin

- 모델 인스턴스를 삭제하는 믹스인
- `.destroy(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 204 No Content 리턴
- 실패 시, 404 Not Found 리턴

## 11. put 메소드를 지정해봅시다!

```
from django.shortcuts import render
from rest_framework.generics import GenericAPIView
from rest_framework import serializers, mixins
# ListModelMixin 사용 위해 mixins 추가
from drf.blog.models import Post

class blog_detail_api(GenericAPIView, mixins.RetrieveModelMixin,
                      mixins.UpdateModelMixin
                      ):
    queryset = Post.objects.all()
    serializer_class = PostSerializer

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

    def put(self, request, *args, **kwargs):
        return self.update(request, *args, **kwargs)
```

# DRF(2)

내가 수정하길 원하는 pk 값만 붙여서 보내주고,  
밑에 해당 pk값의 인스턴스를 put(수정)할 수 있는 화면이 나온다면 성공입니다!

## Blog Detail Api

GET /api/blog/4/

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

### BEFORE

```
{
  "id": 4,
  "date": "2022-05-10T09:27:21.988887Z",
  "title": "제목1",
  "content": "post1의 내용"
}
```

http://127.0.0.1:8000/api/blog/4/

PUT /api/blog/4/

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

### AFTER

```
{
  "id": 4,
  "date": "2022-05-10T12:55:46.329187Z",
  "title": "변경 제목1",
  "content": "post1의 변경 내용"
}
```

Media type:

application/json

Content:

```
{
  "id": 4,
  "date": "2022-05-10T09:27:21.988887Z",
  "title": "변경 제목1",
  "content": "post1의 변경 내용"
}
```

수정

PUT

## 1) ListModelMixin

- Queryset을 리스팅하는 믹스인
- `.list(request, *args, **kwargs)` 메소드로 호출하여 사용
- `GenericAPIView`의 `self.filter_queryset`, `self.get_queryset`, `self.get_serializer` 등의 메소드를 활용해 데이터베이스에 저장되어 있는 데이터들을 목록 형태로 response body로 리턴
- 성공 시, 200 OK response 리턴

## 2) CreateModelMixin

- 모델 인스턴스를 생성하고 저장하는 역할을 하는 믹스인
- `.create(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 201 Created 리턴
- 실패 시, 400 Bad Request 리턴

## 3) RetrieveModelMixin

- 존재하는 모델 인스턴스를 리턴해 주는 믹스인
- `.retrieve(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 200 OK response 리턴
- 실패 시, 404 Not Found 리턴

## 4) UpdateModelMixin

- 모델 인스턴스를 수정하여 저장해 주는 믹스인
- `.update(request, *args, **kwargs)` 메소드로 호출하여 사용
- 부분만 변경하고자 할 경우, `.partial_update(request, *args, **kwargs)` 메소드를 호출하여야 하며,
- 이 때 요청은 HTTP PATCH requests여야 함
- 성공 시, 200 OK response 리턴
- 실패 시, 404 Not Found 리턴

## 5) DestroyModelMixin

- 모델 인스턴스를 삭제하는 믹스인
- `.destroy(request, *args, **kwargs)` 메소드로 호출하여 사용
- 성공 시, 204 No Content 리턴
- 실패 시, 404 Not Found 리턴

## 12. 마지막으로 delete 메소드를 구현해봅시다!

```
from django.shortcuts import render
from rest_framework.generics import GenericAPIView
from rest_framework import serializers, mixins
# ListModelMixin 사용 위해 mixins 추가
from drf.blog.models import Post

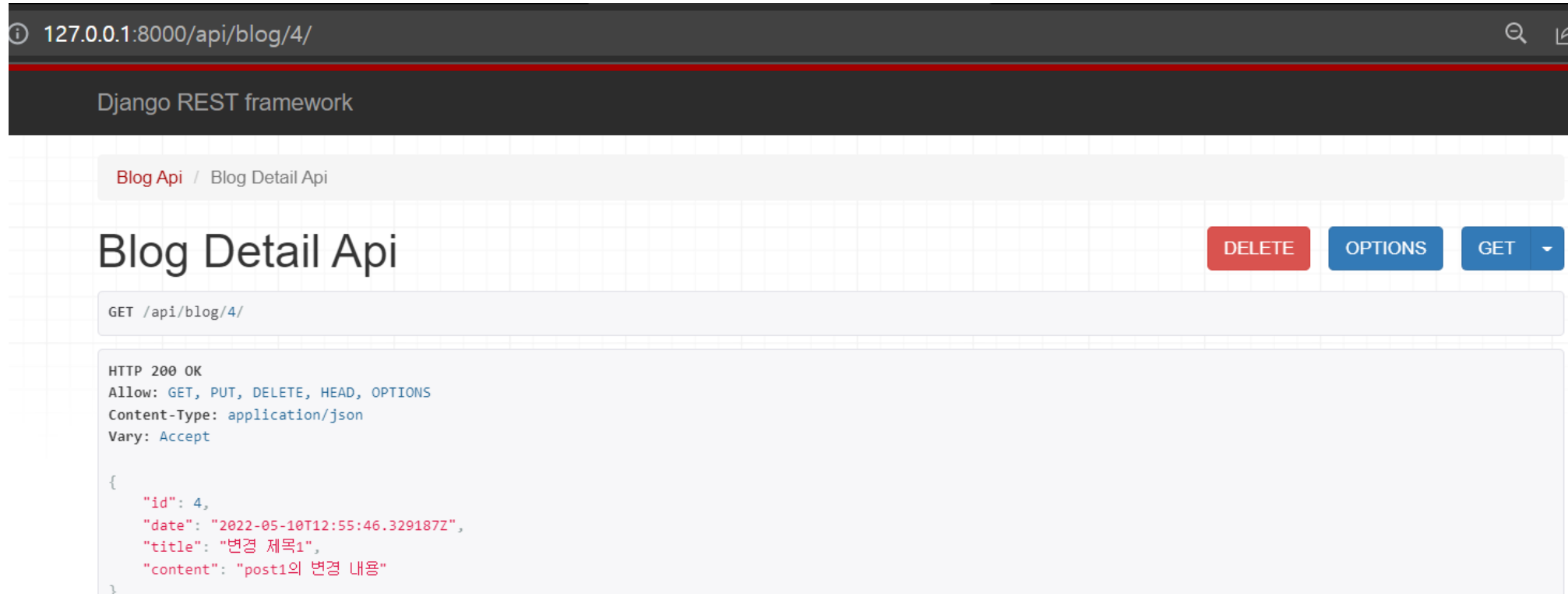
class blog_detail_api(GenericAPIView, mixins.RetrieveModelMixin):
    queryset = Post.objects.all()
    serializer_class = PostSerializer

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

    def put(self, request, *args, **kwargs):
        return self.update(request, *args, **kwargs)

    def delete(self, request, *args, **kwargs):
        return self.destroy(request, *args, **kwargs)
```

Pk 로 내가 원하는 post를 조회한다면  
DELETE 표시가 나타나고 삭제도 수행이 되면 성공입니다!



127.0.0.1:8000/api/blog/4/

Django REST framework

Blog Api / Blog Detail Api

## Blog Detail Api

DELETE OPTIONS GET ▾

GET /api/blog/4/

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

{
  "id": 4,
  "date": "2022-05-10T12:55:46.329187Z",
  "title": "변경 제목1",
  "content": "post1의 변경 내용"
}
```



수고하셨습니다



서강대 김동윤

