Django RESTful API

좋은징조 - 김형훈

목차

1. Django RESTful API란?

- 서버와 클라이언트가 서로 통신하기 위해서는 REST API를 사용해야 한다. REST(Representational State Transfer)란, 자원을 이름으로 구분하여 해당 자원의 상태를 주고받는 모든 것을 의미한다. 즉, 자원 기반의 구조(ROA, Resource Oriented Architecture) 설계의 중심에 리소스가 있고 HTTP Method(POST, GET, PUT, DELETE)를 통해 리소스를 처리하도록 설계된 아키텍처이다.
- 네트워크 상에서 Server-Client 구조를 가지고 있으며 다양한 브라우저와 안드로이드, IOS와 같은 모바일에서도 통신이 가능하다. API(Application Programming Interface)는 데이터와 기능의 집합을 제공하여 컴퓨터 프로그램 간에 상호작용을 촉진하며 서로 정보를 교환 가능하도록 하는 것이다. REST API는 REST 기반으로 서비스 API를 구현한 것으로 HTTP 표준을 기반으로 구현한다. 우리는 REST API를 사용하기 위해 Django REST Framework를 통하여 코드를 작성한다. Django REST Framework는 웹 API를 구축하기 위한 Toolkit이다.

2. Django REST Framework 서비스 환경 구축

앞에 나온 'Django 서버 구축'을 기반으로 웹 서버를 구축할 디렉토리를 생성 후 이동한다.

```
> mkdir DjangoREST_tutorial
> cd DjangoREST_tutorial
```

Windows 기준으로 Python 기반의 가상환경을 만든다.

```
> python -m venv myenv
> myenv\Scripts\activate
```

웹 서버의 원활한 서비스를 위해 Django와 Django REST Framework 패키지를 설치한다.

```
(myenv) python -m pip install --upgrade pip
(myenv) pip install django~=2.0.0
(myenv) pip install djangorestframework
```

새로운 프로젝트와 앱을 생성한다.

```
(myenv) django-admin.py startproject tutorial
(myenv) cd tutorial
(myenv) django-admin.py startapp mysample
```

프로젝트 구조는 아래 그림과 같다.

```
DjangoREST_tutorial
    |- myenv
    |- tutorial
        |- mysameple
            |- migrations
            |- __init__.py
            |- admin.py
            |- apps.py
            |- models.py
            |- tests.py
            |- views.py
        |- __init__.py
        |- settings.py
        |- urls.py
        |- wsgi.py
    |- manage.py
```

---

3. Django REST Framework 서비스 구축

tutorial/settings.py

Django REST Framework와 등록한 앱을 사용하기 위해 INSTALLED_APPS에 'rest_framework'와 'tutorial.mysample'을 추가한다. 또한, Django REST Framework에 대한 Pagination과 JSONRenderer의 설정을 지정한다.

```
INSTALLED_APPS = [
    ...
    'rest_framework',
    'tutorial.mysample',
]
```

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': (
        'rest_framework.pagination.PageNumberPagination'
    ),
    'DEFAULT_RENDERER_CLASSES': (
        'rest_framework.renderer.JSONRenderer'
    ),
    'PAGE_SIZE': 10
}
```

mysample/models.py

MD5 필드를 가진 HashValues 모델을 정의한다.

```
from django.db import models

class HashValues(models.Model):
    MD5 = models.TextField()
```

models.py을 변경 후 데이터베이스와 동기화를 시켜야 한다.

```
(myenv) python manage.py makemigrations
(myenv) python manage.py migrate
```

mysample/serializers.py

serializer이란, models 객체와 querysets 같은 복잡한 데이터를 JSON, XML과 같은 데이터로 바꿔주는 역할을 한다.

```
from .models import HashValues
from rest_framework import serializers

class HashValuesSerializer(serializers.ModelSerializer):
    class Meta:
        model = HashValues
        fields = '__all__'
```

mysample/views.py

Django REST Framework APIView에 CBV(Class-Based-View)와 FBV(Function-Based-View) 2가지 방법이 있다. 여기서는 FBV 방식을 사용한다. @api_view()는 함수 기반 뷰를 만들겠다는 장식자이며 안에 요청 메서드 GET, POST를 명시한다. GET일 경우, 데이터베이스에 있는 데이터를 반환하고 POST일 경우, 요청 받은 데이터를 데이터베이스에 저장하고 반환한다.

```
from .models import HashValues
```

```
from .serializers import HashValuesSerializer
from rest_framework import status
from rest_framework.response import Response
from rest_framework.decorators import api_view, renderer_classes
from rest_framework.renderers import JSONRenderer

@api_view(['GET', 'POST'])
@renderer_classes((JSONRenderer,))
def HashValuesList(request):
    if request.method == 'GET': # 요청 방식이 GET이면
        queryset = HashValues.objects.all() # HashValues의 모든 객체를
        serializer = HashValuesSerializer(queryset, many=True) # 시리얼라이즈 하여
        return Response(serializer.data) # 응답으로 시리얼라이즈의 데이터를 반환한다

    elif request.method == 'POST': # 요청 방식이 POST이면
        serializer = HashValuesSerializer(data=request.data)
        # 요청 받은 데이터를 시리얼라이즈 하여
        if serializer.is_valid(): # 시리얼라이즈가 유효하면
            serializer.save() # 저장하고
            return Response(serializer.data) # 응답으로 시리얼라이즈의 데이터를 반환한다
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
        # 유효하지 않으면 400 에러를 반환한다
```

mysample/urls.py

mysample 안에 urls.py를 생성 후 클라이언트와 웹 서버가 서로 통신하기 위해 경로를 설정한다.

```
from django.urls import path, include
from tutorial.mysample import views

urlpatterns = [
    path('', views.HashValuesList),
    path('api-auth/', include('rest_framework.urls', namespace='rest_framework'))
]
```

tutorial/urls.py

tutorial/urls.py에서 mysample의 urls.py를 포함시켜야 mysample 기능이 수행 가능해진다. 추가하기 위해 경로를 설정한다.

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('mysample/', include('tutorial.mysample.urls')),
]
```

Test

서버 관리자를 등록한다.

```
(myenv) python manage.py createsuperuser
(myenv) Username : admin
(myenv) Email address : admin@admin.com
(myenv) Password : admin12345
(myenv) Password (again) : admin12345
```

아래 명령어로 서버를 실행시킨다.

```
(myenv) python manage.py runserver
```

http://127.0.0.1:8000/ 에 접속한다.