Práctica 1

Servicio simple sin parámetros

1. Crear el archivo data.py

```python
import json
from .models import Project

class Data:
    def get_test_data(self):
        aux = {'nombre':'Juan Pérez'}
        return json.dumps(aux)
```

2. Crear el archivo api_g.py

```python
from projects.data import Data
import json
```

```python
class DataTest(APIView):
    def get(self, request):
        cat = Data()
        return Response({'resultado':cat.get_test_data()}, status=status.HTTP_200_OK)
```

3. En el archivo urls.py

```python
from rest_framework import routers
from .api import ProjectViewSet
from .api_g import DataTest, ProjectServices
from django.urls import path, include

router = routers.DefaultRouter()

router.register('api/projects', ProjectViewSet, 'projects')
# urlpatterns = router.urls
urlpatterns = [
    path('api/', include(router.urls)),
    path('dataTest/', DataTest.as_view()),
```

4. Verificar su funcionamiento desde el navegador y mediante la herramienta thunder client

Práctica 2

Servicio simple con parámetros

1. En el archivo data.py

```python
    def get_suma(self, num1, num2):
        suma = num1 + num2
        res = {'suma':suma}
        return json.dumps(res)
```

2. En el archivo serializers.py

```python
from rest_framework import serializers
from .models import Project
```

```python
class SumaSerializer(serializers.Serializer):
    num1 = serializers.IntegerField()
    num2 = serializers.IntegerField()
```

3. En el archivo archivo api_g.py

```python
class DataTest(APIView):
    def get(self, request):
        cat = Data()
        return Response({'resultado':cat.get_test_data()}, status=status.HTTP_200_OK)

    def post(self, request):
        serializer = SumaSerializer(data=request.data)
        if serializer.is_valid():
            num1 = serializer.validated_data['num1']
            num2 = serializer.validated_data['num2']
            cat = Data()
            return Response({'resultado':cat.get_suma(num1, num2)}, status=status.HTTP_200_OK)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

4. En el archivo urls.py, completa el path

```python
from rest_framework import routers
from .api import ProjectViewSet
from .api_g import DataTest, ProjectServices
from django.urls import path, include

router = routers.DefaultRouter()

router.register('api/projects', ProjectViewSet, 'projects')
# urlpatterns = router.urls
urlpatterns = [
    path('api/', include(router.urls)),
    path('dataTest/', DataTest.as_view()),
    path('projectServices/', ProjectServices.as_view()),
]
```

5. Verificar su funcionamiento desde el navegador y mediante la herramienta thunder client
    a. Verifica cuando los parámetros no se envían
    b. Verifica cuando el tipo de dato no es entero
    c. Observa que se tiene un servicio en el método POST y otro en el GET

Práctica 3

Servicio con parámetros en json y regreso de modelo de datos

1. En el archivo data.py

```python
    def get_projects(self, parameters=None):
        projects = Project.objects.all()
        if parameters:
            if 'title' in parameters:
                # projects = projects.filter(title__startswith = parameters['title'])
                projects = projects.filter(title__contains = parameters['title'])
            if 'description' in parameters:
                projects = projects.filter(description__contains = parameters['description'])
            if 'quantity' in parameters:
                projects = projects.filter(quantity__gt= parameters['quantity'])
        return projects
```

   Observe las diferentes formas que se tiene para filtrar la información y el encadenamiento utilizado

2. En el archivo serializers.py, completar el serializador para los parámetros (observe que es un objeto tipo Json)

```python
class ProjectParameterSerializer(serializers.Serializer):
    parameters = serializers.JSONField()
```

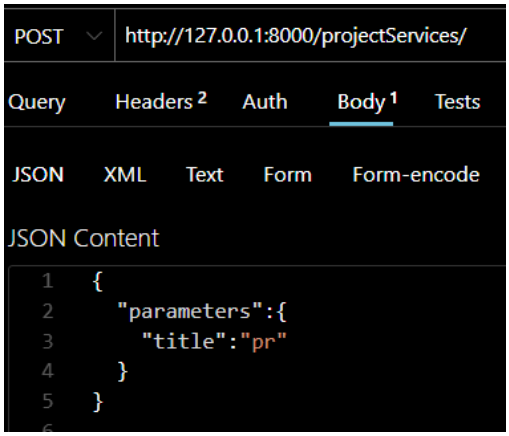
3. En el archivo api_g, crear la clase ProjectServices

```python
from .serializers import SumaSerializer, ProjectParameterSerializer
from django.core.serializers.json import DjangoJSONEncoder
```

```python
class ProjectServices(APIView):
    def post(self, request):
        serializer = ProjectParameterSerializer(data = request.data)
        if serializer.is_valid():
            parameters  = serializer.validated_data['parameters']
            cat = Data()
            res = cat.get_projects(parameters)
            res2 = json.dumps(list(res.values()), cls=DjangoJSONEncoder)
            return Response({'resultado':res2}, status=status.HTTP_200_OK)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Observe que en la transformación a JSON se tiene DjangoJSONEncoder, esto es para que los objetos también los pueda convertir, por ejemplo la FECHA.

4. Verifique su funcionamiento realizando diferentes corridas con los parámetros diferentes.



Práctica 4

APITestCase

1. Crear las pruebas

```python
from rest_framework.test import APITestCase
import json
from projects.models import Project

class ProjectServiceTest(APITestCase):
    def setUp(self):
        self.url = '/projectServices/'
        Project.objects.create(title = "Proyecto 1", description = "Descipción uno")
        Project.objects.create(title = "Proyecto 2", description = "Descipción dos")
        Project.objects.create(title = "Proyecto 3", description = "Descipción tres")
        Project.objects.create(title = "Proyecto 4", description = "Descipción cuatro")
```

```python
def test_without_parameters(self):
    data = {
        'parameters':{}
    }
    response = self.client.post(self.url, data, format='json')
    self.assertEqual(response.status_code, 200)
    data = json.loads(response.data['resultado'])
    self.assertEquals(len(data), 4)
```

```python
def test_without_parameters(self):
    data = {
        'parameters':{'title':"Proyecto 2"}
    }
    response = self.client.post(self.url, data, format='json')
    self.assertEqual(response.status_code, 200)
    data = json.loads(response.data['resultado'])
    self.assertEqual(data[0]["description"], "Descipción dos")
```