

# ESCOLA E FACULDADE DE TECNOLOGIA SENAI ROBERTO MANGE

## DESENVOLVIMENTO DE SISTEMAS

JULHO DE 2022 — SENAI-SP



ESCOLA SENAI "ROBERTO MANGE"  
UMI. REACTIVADO DA INDÚSTRIA

**SENAI**  
SÃO PAULO

## CRIANDO UMA API USANDO A BIBLIOTECA FACILITADORA “ModelViewSet”

Como viu em sala de aula, nas views do django estávamos usando a biblioteca APIView para criarmos nossa API e definirmos o que iria ser executado em cada método http da API (Get, Post, Put, Delete, etc....).

É importante usarmos a APIView para entendermos o que de fato acontece dentro do Django e também termos o poder de controlar o que realmente queremos que seja feito, deixando mais flexível e em nossas mãos o código.

```
from rest_framework.views import APIView
from rest_framework.response import Response
from django.http import HttpResponse
from rest_framework import status

class PeopleAPIView(APIView):
    def delete(self, request, peopleId = ''):
        # try:
        # busca o usuario com o id
        peopleFound = People.objects.get(id=peopleId)
        peopleFound.delete() #deleta o usuario com o id encontrado
        return Response(status=status.HTTP_200_OK, data="People successfully deleted!")
        # except People.DoesNotExist:
        # return Response(status=status.HTTP_404_NOT_FOUND,data="People not Found!")
    def put(self, request, peopleId = ''):
        #o people já existente no banco:
        peopleFound = People.objects.get(id=peopleId)
        #o people com os dados novos
```

Código usando a APIView

## CRIANDO UMA API USANDO A BIBLIOTECA FACILITADORA “ModelViewSet”

No entanto.... Há uma outra biblioteca chamada **ModelViewSet** que é capaz de abstrair vários desses comportamentos básicos de uma API nos entregando praticamente pronto todos os métodos básicos das APIs, sendo extremamente útil nos casos em que queremos rapidez no desenvolvimento e não temos a necessidade de alterar o comportamento básico dos métodos!!!!

Para usar a ModelViewSet é necessário que você já tenha a biblioteca **django-rest-framework** que é a mesma quando estávamos usando a APIView, portanto caso já possua pode pular os passos 1, 2, e 3.

1. **pip install django-rest-framework** → com o terminal na venv ativada, instale a biblioteca do django p/ trabalhar via API;
2. Informe o django que você deseja usar a biblioteca instalada em **INSTALLED\_APPS** dentro de **settings.py**:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'rest_framework',  
    'corsheaders',  
    'main',  
]
```



## CRIANDO UMA API USANDO A BIBLIOTECA FACILITADORA “ModelViewSet”

Para usar a ModelViewSet é necessário que você já tenha a biblioteca **django-rest-framework** que é a mesma quando estávamos usando a APIView, portanto caso já possua pode pular os passos 1, 2, e 3.

3. Adicione as configurações da REST\_FRAMEWORK também dentro do **settings.py** (neste caso estamos definindo que nossa API não terá autenticação/autorização):

```
REST_FRAMEWORK = {  
    'DEFAULT_PERMISSION_CLASSES': [  
        'rest_framework.permissions.AllowAny'  
    ]  
}
```

4. Em **views.py** faça a importação da classe que usaremos para facilitar nossa API:

```
views.py 3, U x  
main > views.py > ...  
1  from .models import *  
2  from .serializers import *  
3  |  
4  from rest_framework.viewsets import ModelViewSet  
5  |  
6  from rest_framework.response import Response  
7  |  
8  |
```

## CRIANDO UMA API USANDO A BIBLIOTECA FACILITADORA “ModelViewSet”

5. Na views.py faça a utilização da biblioteca importada criando as classes que definirão o comportamento da API, exemplo:

```
views.py U X
main > views.py > ...
1 from .models import *
2 from .serializers import *
3 from rest_framework.viewsets import ModelViewSet
4
5 class PeopleAPIView(ModelViewSet):
6     queryset = People.objects.all()
7     serializer_class = PeopleSerializer
8
9 class PlanetAPIView(ModelViewSet):
10    queryset = Planet.objects.all()
11    serializer_class = PlanetSerializer
12
13 class StarshipsAPIView(ModelViewSet):
14    queryset = Starships.objects.all()
15    serializer_class = StarshipsSerializer
```

- Usando a ModelViewSet eliminamos a necessidade de definir o que é necessário ser feito no Get,Post,Put,Delete e Patch, pois a biblioteca faz essas operações padrões apenas sendo necessário passar duas informações:
  - **queryset** → informa qual a tabela do banco será usada
  - **serializer\_class** → informa qual o serializer será usado



## CRIANDO UMA API USANDO A BIBLIOTECA FACILITADORA “ModelViewSet”

6. Em `urls.py` do *Aplicativo*, defina o roteamento de cada ponto de acesso da API (endpoint), informando qual a rota e para qual view será redirecionada:

```
#importando as views que criamos da nossa api:
from .views import *
#importando a DefaultRouter que irá ajudar na definição das rotas da api
from rest_framework.routers import DefaultRouter

router = DefaultRouter()
router.register(r'people', PeopleAPIView)
router.register(r'planet', PlanetAPIView)
router.register(r'starship', StarshipsAPIView)

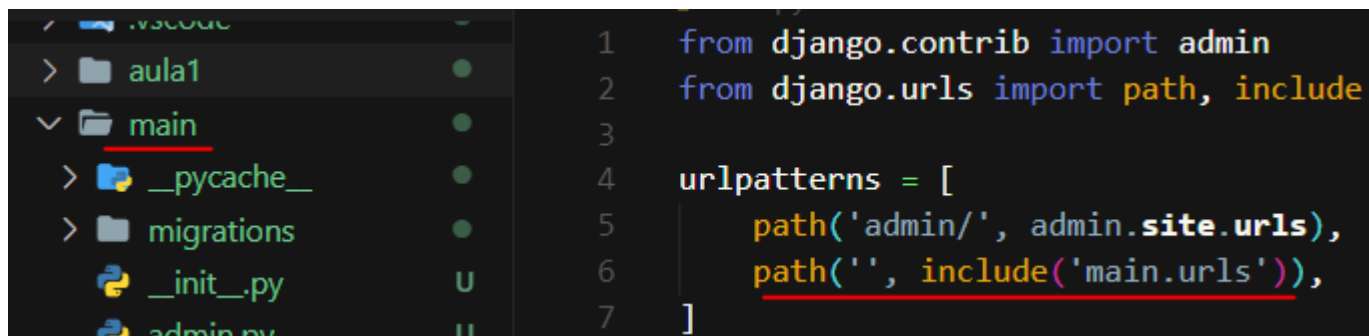
urlpatterns = router.urls
```



## CRIANDO UMA API USANDO A BIBLIOTECA FACILITADORA “ModelViewSet”

7. Em `urls.py` do *Projeto*, inclua as `urls` que acabou de criar no aplicativo registrando-as nas `urls` do projeto:

**Observação: você deve usar o nome do aplicativo que criou, no meu caso o aplicativo foi criado como main, portanto eu utilizo o nome `main.urls`**

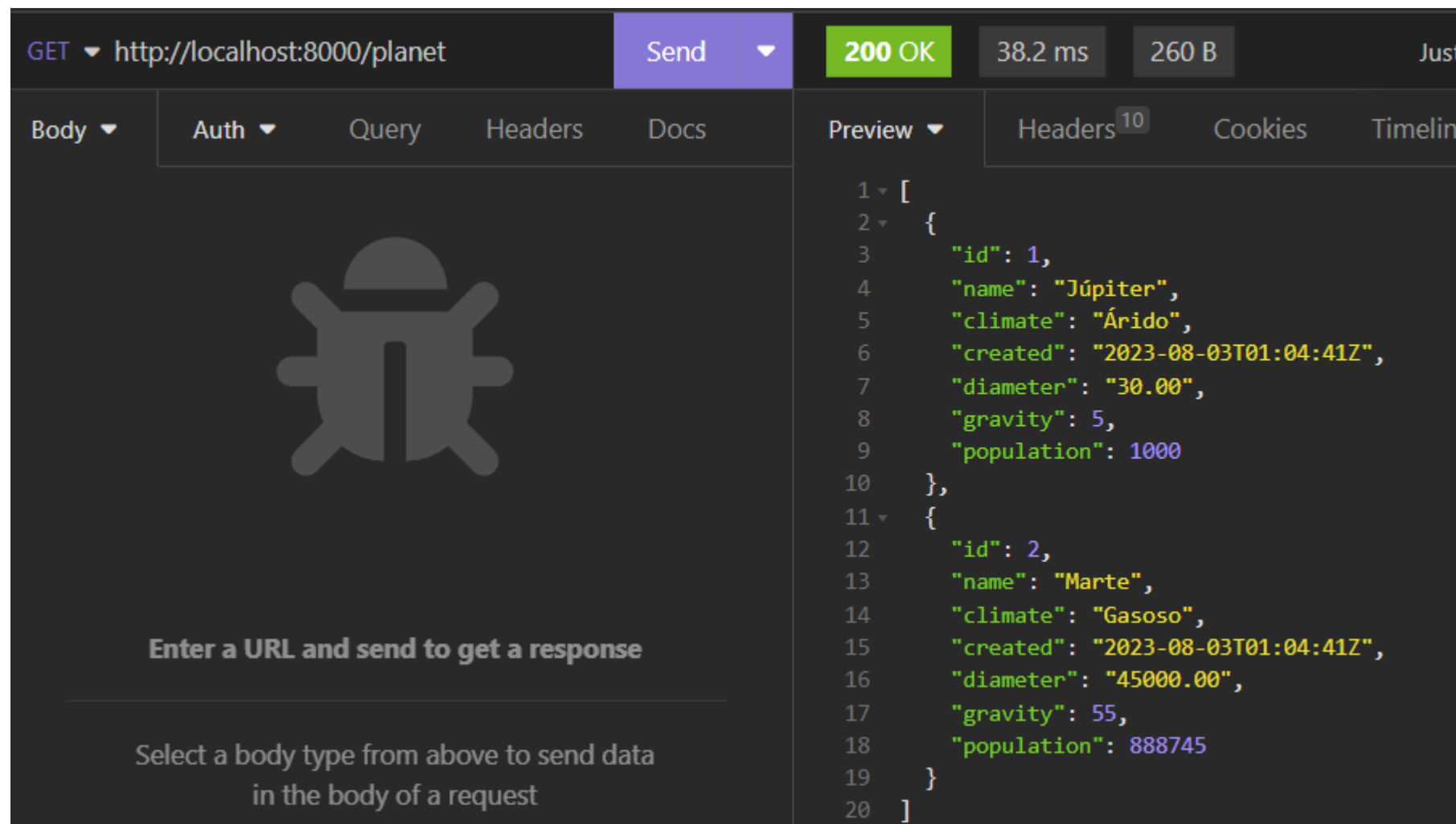


```
1 from django.contrib import admin
2 from django.urls import path, include
3
4 urlpatterns = [
5     path('admin/', admin.site.urls),
6     path('', include('main.urls')),
7 ]
```



## CRIANDO UMA API USANDO A BIBLIOTECA FACILITADORA “ModelViewSet”

TESTANDO: (para efeito de teste evidenciarei apenas o Planet)



GET ▼ http://localhost:8000/planet Send ▼ 200 OK 38.2 ms 260 B Just

Body ▼ Auth ▼ Query Headers Docs Preview ▼ Headers <sup>10</sup> Cookies Timeline

Enter a URL and send to get a response

Select a body type from above to send data in the body of a request

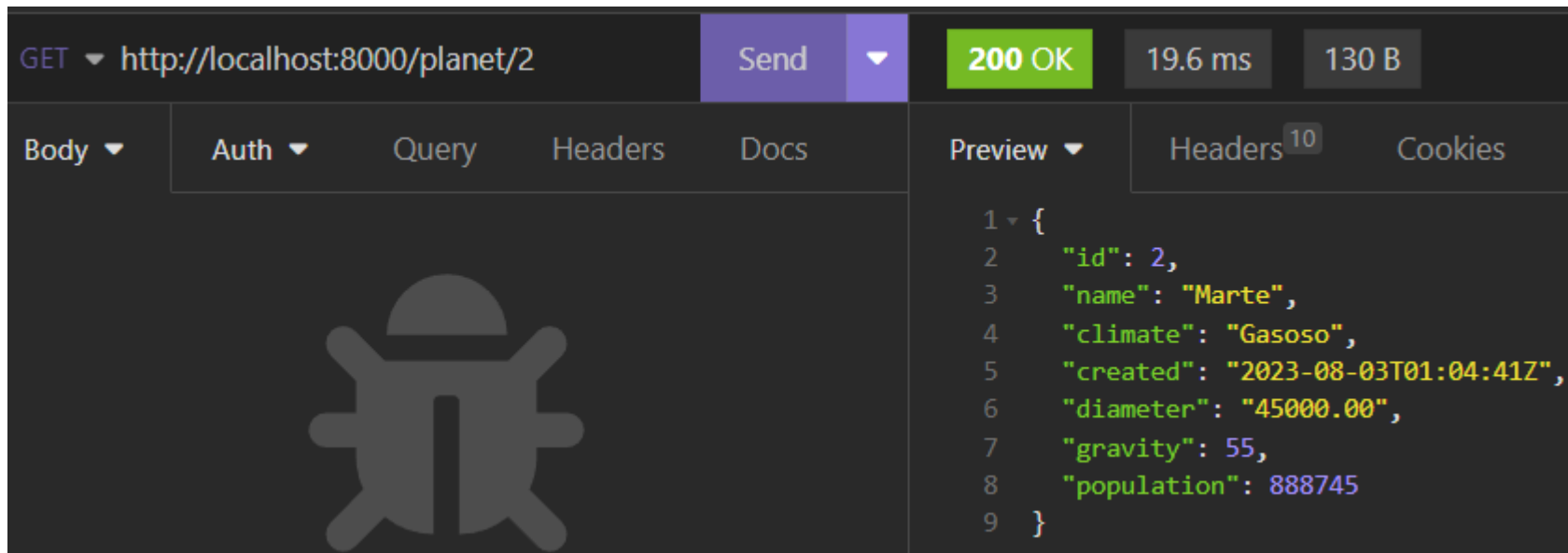
```
1 [
2   {
3     "id": 1,
4     "name": "Júpiter",
5     "climate": "Árido",
6     "created": "2023-08-03T01:04:41Z",
7     "diameter": "30.00",
8     "gravity": 5,
9     "population": 1000
10  },
11  {
12    "id": 2,
13    "name": "Marte",
14    "climate": "Gasoso",
15    "created": "2023-08-03T01:04:41Z",
16    "diameter": "45000.00",
17    "gravity": 55,
18    "population": 888745
19  }
20 ]
```

Solicitando todos os planetas



## CRIANDO UMA API USANDO A BIBLIOTECA FACILITADORA “ModelViewSet”

TESTANDO: (para efeito de teste evidenciarei apenas o Planet)



GET http://localhost:8000/planet/2 Send 200 OK 19.6 ms 130 B

Body Auth Query Headers Docs Preview Headers 10 Cookies

```
1 {  
2   "id": 2,  
3   "name": "Marte",  
4   "climate": "Gasoso",  
5   "created": "2023-08-03T01:04:41Z",  
6   "diameter": "45000.00",  
7   "gravity": 55,  
8   "population": 888745  
9 }
```

Solicitando um planeta  
por Id

## CRIANDO UMA API USANDO A BIBLIOTECA FACILITADORA “ModelViewSet”

TESTANDO: (para efeito de teste evidenciarei apenas o Planet)

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8000/planet/
- Status:** 201 Created
- Time:** 14.1 ms
- Size:** 133 B
- JSON Body:**

```
1 {
2   "name": "Saturno",
3   "climate": "Gasoso",
4   "diameter": "20.00",
5   "gravity": 4,
6   "population": 3000
7 }
```
- Preview:**

```
1 {
2   "id": 4,
3   "name": "Saturno",
4   "climate": "Gasoso",
5   "created": "2023-09-17T17:34:14.605879Z",
6   "diameter": "20.00",
7   "gravity": 4,
8   "population": 3000
9 }
```

Cadastrando um novo planeta

## CRIANDO UMA API USANDO A BIBLIOTECA FACILITADORA “ModelViewSet”

TESTANDO: (para efeito de teste evidenciarei apenas o Planet)

The screenshot shows a REST client interface with the following details:

- Method:** PUT
- URL:** http://localhost:8000/planet/4/
- Status:** 200 OK
- Time:** 16.9 ms
- Size:** 133 B
- JSON Body:**

```
1 {
2   "name": "Saturno",
3   "climate": "Gasoso",
4   "diameter": "40.00",
5   "gravity": 4,
6   "population": 3000
7 }
```
- Preview Body:**

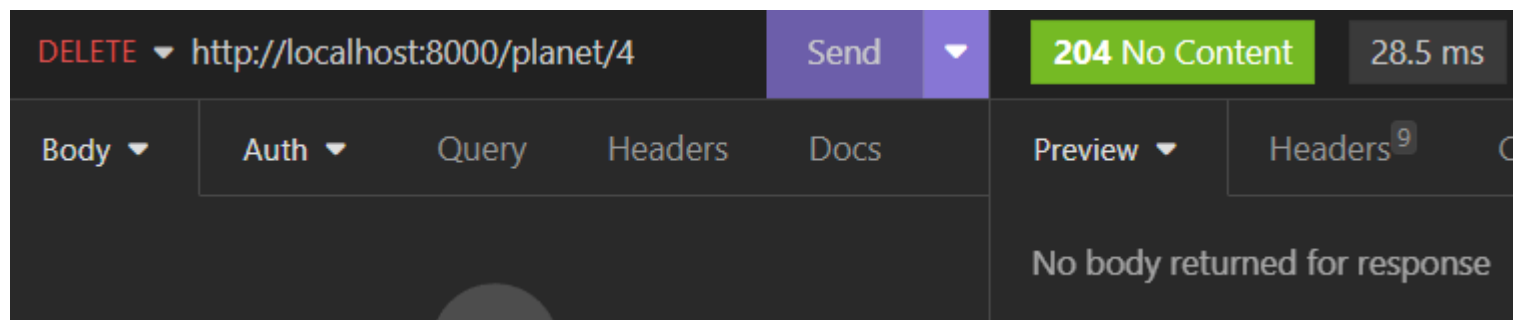
```
1 {
2   "id": 4,
3   "name": "Saturno",
4   "climate": "Gasoso",
5   "created": "2023-09-17T17:34:14.605879Z",
6   "diameter": "40.00",
7   "gravity": 4,
8   "population": 3000
9 }
```

Alterando o cadastro  
de um planeta já  
existente



## CRIANDO UMA API USANDO A BIBLIOTECA FACILITADORA “ModelViewSet”

TESTANDO: (para efeito de teste evidenciarei apenas o Planet)



Deletando um planeta

## ADICIONADO FILTROS DE BUSCA COM “DjangoFilter”

Já aprendemos a fazer uma API com os métodos básicos, no entanto é muito comum necessitarmos de um backend que seja capaz de dar informações de uma forma mais específica, usando os chamados filtros.

Neste caso, podemos por exemplo, solicitar todos os Planetas que tenham uma gravidade em específico, ou algum nome em comum, etc...

Para isso usaremos a biblioteca chamada **django-filter**, que pode ser implementada tanto usando a APIView ou a ModelViewSet, no entanto usarei esta última por consistir em uma implementação mais enxuta.

Para evitar repetição dos mesmos passos já informados nos slides anteriores, demonstrarei apenas os passos necessários para implementar o django-filter já considerando que tenha realizado todos os procedimentos anteriores para criar o django como API usando ModelViewSet!

1. **pip install django-filter** → com o terminal na venv ativada, instale a biblioteca do django p/ trabalhar com filtros;



## ADICIONADO FILTROS DE BUSCA COM “DjangoFilter”

2. Importe a classe `DjangoFilterBackend` → necessário para que o django-filter seja integrado com a restframework

```
views.py U X
main > views.py > ...
1  from .models import *
2  from .serializers import *
3  from rest_framework.viewsets import ModelViewSet
4  from django_filters.rest_framework import DjangoFilterBackend
5
```

## ADICIONADO FILTROS DE BUSCA COM “DjangoFilter”

3. Implemente a classe `DjangoFilterBackend` importada e defina os campos usados como filtro → estes procedimentos são necessários para que a classe `ModelViewSet` (responsável por facilitar a criação de nossa API conforme informado nos slides anteriores) possa se integrar com a biblioteca `django-filter`. Definimos também em `filterset_fields` os campos que desejamos que sejam usados como filtro (observe que estes campos precisam existir dentro de sua model)

```
class PeopleAPIView(ModelViewSet):
    queryset = People.objects.all()
    serializer_class = PeopleSerializer
    filter_backends = [DjangoFilterBackend]
    filterset_fields = ['name', 'eyeColor', 'height', 'gender']

class PlanetAPIView(ModelViewSet):
    queryset = Planet.objects.all()
    serializer_class = PlanetSerializer
    filter_backends = [DjangoFilterBackend]
    filterset_fields = ['name', 'climate', 'diameter']

class StarshipsAPIView(ModelViewSet):
    queryset = Starships.objects.all()
    serializer_class = StarshipsSerializer
    filter_backends = [DjangoFilterBackend]
    filterset_fields = ['name', 'model', 'passengers']
```

## ADICIONADO FILTROS DE BUSCA COM “DjangoFilter”

TESTANDO: (para efeito de teste evidenciarei apenas o Planet)

The screenshot shows the Insomnia API client interface. At the top, a GET request to `http://localhost:8000/planet` is shown with a status of 200 OK, 22 ms, and 132 B. Below the request bar, the 'Query' tab is selected, showing a single query parameter: `climate=Gasoso`. The 'URL PREVIEW' section shows the full URL: `http://localhost:8000/planet?climate=Gasoso`. The 'Preview' tab on the right shows the JSON response:

```
[
  {
    "id": 2,
    "name": "Marte",
    "climate": "Gasoso",
    "created": "2023-08-03T01:04:41Z",
    "diameter": "45000.00",
    "gravity": 55,
    "population": 888745
  }
]
```

Perceba que neste caso estamos usando o filtro dos planetas que possuem o 'climate' (clima) que especificarmos, pois definimos anteriormente nas `views.py` que este campo será usado em `filterset_fields` da Planet view.

Obs.: conforme observou, após usarmos o filtro no Insomnia a url ficou: <http://localhost:8000/planet?climate=Gasoso>  
É importante notar que nas APIs os filtros são por padrão enviados para o backend usando os chamados **parâmetros**.  
Estes parâmetros são enviados na url da API dessa forma: **? + nome do filtro + = + valor do filtro**  
Podemos também mandar mais de um parâmetro sendo necessário apenas usar o **&** para separá-los,  
por exemplo, se quisermos filtrar por clima e também diâmetro do planeta então a url ficará:  
**`http://localhost:8000/planet?climate=Gasoso&diameter=55`**