

django



framework



Quem sou eu ?

Paola Katherine Pacheco

- Graduada em Análise de Sistemas pela Unesa
- Ex-graduanda de Estatística na Uerj
- Apaixonada por números , viagens e chocolate
- Desenvolvedora Back End Python
- Membro do Django Girls , Pyladies, Women Techmakers
- ex GDG Organizer Rio de Janeiro

about.me/pkcpweb

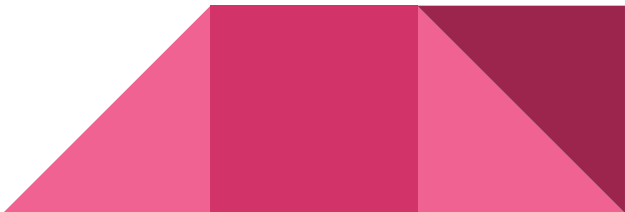


Antes de começar ...

Recomendo que você tenha noções de:

- Verbos HTTP

<https://www.caelum.com.br/apostila-vraptor-hibernate/rest/#11-3-o-triangulo-do-rest>

- REST <https://pt.wikipedia.org/wiki/REST>
 - Python e Django
 - API
- 

Django Rest Framework

Django Rest Framework é um ferramenta que facilita a criação de API's REST.

Documentação:

<http://www.django-rest-framework.org/>



Projeto que usaremos

Usaremos um projeto que já existe e já foi feito em outro tutorial, para que esse tutorial não seja muito longo.

É um projeto simples mas que com poucas alterações poderemos usar nesse treinamento.

<https://github.com/PKpacheco/meu-portfolio>



Virtual Env Wrapper

Para facilitar, usaremos o Virtual Env Wrapper.

O Virtual Env é um ambiente virtual que fará com que todas as dependências do projeto fiquem em um diretório só.

Para instalar digite o comando abaixo:

```
$ pip install virtualenvwrapper
```

(criar PastaDoProjeto)

E criaremos uma virtual env:

```
$ mkvirtualenv NomeDoProjeto
```



Virtual Env Wrapper

```
→ projeto-django-rest git:(master) mkvirtualenv projeto-drf
```

```
→ projeto-django-rest git:(master) mkvirtualenv projeto-drf
```

```
New python executable in /Users/paolakatherine/.virtualenvs/projeto-drf/bin/python2.7
```

```
Also creating executable in /Users/paolakatherine/.virtualenvs/projeto-drf/bin/python
```

```
Installing setuptools, pip, wheel...done.
```

```
virtualenvwrapper.user_scripts creating /Users/paolakatherine/.virtualenvs/projeto-drf/bin/predeactivate
```

```
virtualenvwrapper.user_scripts creating /Users/paolakatherine/.virtualenvs/projeto-drf/bin/postdeactivate
```

```
virtualenvwrapper.user_scripts creating /Users/paolakatherine/.virtualenvs/projeto-drf/bin/preactivate
```

```
virtualenvwrapper.user_scripts creating /Users/paolakatherine/.virtualenvs/projeto-drf/bin/postactivate
```

```
virtualenvwrapper.user_scripts creating /Users/paolakatherine/.virtualenvs/projeto-drf/bin/get_env_details
```

Instalando Django Rest Framework

\$ pip install djangorestframework

```
(projeto-drf) → projeto-django-rest git:(master) pip install djangorestframework
Collecting djangorestframework
  Downloading djangorestframework-3.5.3-py2.py3-none-any.whl (709kB)
    100% |#####| 716kB 1.1MB/s
Installing collected packages: djangorestframework
Successfully installed djangorestframework-3.5.3
```


Verificando dependências

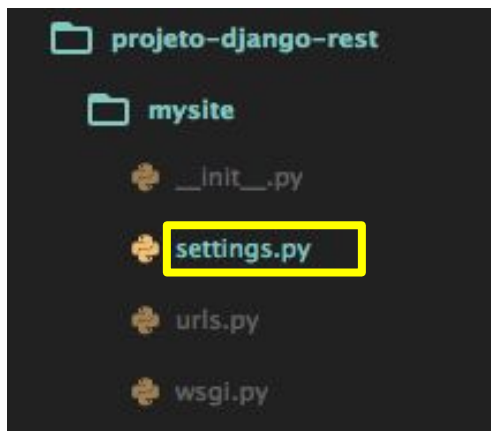
Podemos utilizar a mesma VirtualEnv do projeto antigo ou podemos criar uma nova. De qualquer maneira ambas precisam conter Django e o Django Rest Framework. Para isso , verifique as dependências do projeto com o comando abaixo

\$ pip freeze

```
(projeto-drf) → projeto-django-rest git:(master) x pip freeze  
Django==1.10.5  
djangorestframework==3.5.3
```

Adicionando Django Rest Framework

Precisamos adicionar em `mysite/settings.py` o `rest_framework` para que o projeto identifique a app.



```
INSTALLED_APPS = (  
    ... 'django.contrib.admin',  
    ... 'django.contrib.auth',  
    ... 'django.contrib.contenttypes',  
    ... 'django.contrib.sessions',  
    ... 'django.contrib.messages',  
    ... 'django.contrib.staticfiles',  
    ... 'portfolios',  
    ... 'rest_framework',  
)
```

Criando uma app API

Para administrar nosso rest framework criaremos uma app

\$ python manage.py startapp api

```
(projeto-drf) → projeto-django-rest git:(master) ✕ python manage.py startapp api
```

projeto-django-rest

api

mysite

portfolios

* .gitignore

* db.sqlite3

</> Makefile

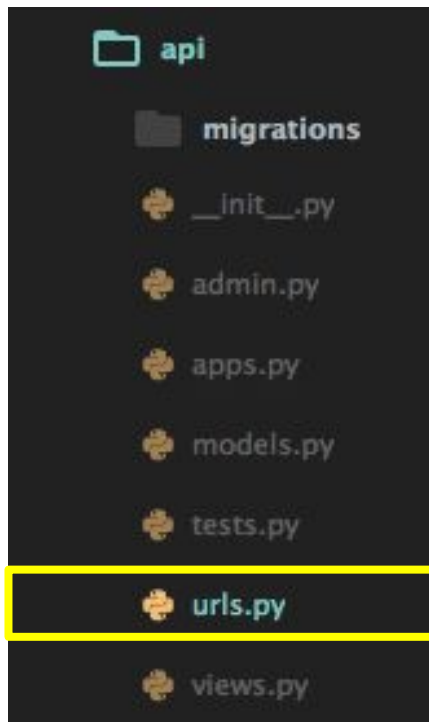
🔧 manage.py

📖 README.md

📄 requirements.txt

Criando arquivo urls.py

Criaremos o **arquivo urls.py** dentro da nossa nova app **api**



Adicionando rota da app api

Em `mysite/urls.py` adicionar a rota nova

```
urlpatterns = [  
    url(r'^admin/', include(admin.site.urls)),  
    url(r'^api/', include('api.urls')),  
    url(r'', include('portfolios.urls')),  
]
```

Serialização

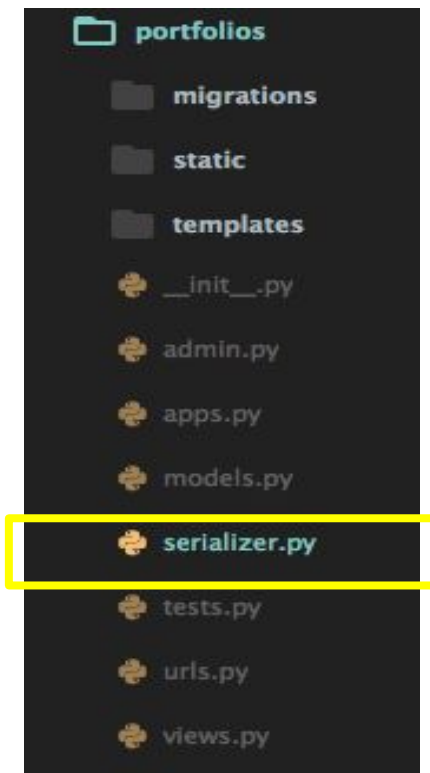
Serialização é o processo de tradução de **estruturas de dados** ou estado de **objeto** em um formato que possa ser armazenado (por exemplo, em um **arquivo** ou **buffer** de memória, ou transmitido por meio de um enlace de conexão de **rede**) e reconstruído posteriormente no mesmo ou em outro ambiente computacional.

<https://pt.wikipedia.org/wiki/Serializa%C3%A7%C3%A3o>



Criando arquivo serializer.py

Criaremos o arquivo `serializer.py` dentro de `portfolios`



Serializer.py

Serializaremos somente alguns dados do nosso modelo, para que nossos testes a frente se tornem mais curtos.

```
serializer.py x
# coding: utf-8

from rest_framework import serializers
from .models import DadosPessoais

class DadosPessoaisSerializer(serializers.ModelSerializer):

    class Meta:
        model = DadosPessoais
        depth = 1
        fields = ['id', 'name', 'adress', 'city', 'cep', 'phone', 'mobile']
```


Acrescentando código em views.py

Alguns ajustes em **portfolios/views.py** para que ele tenha interação com a API, afinal no projeto antigo ele só renderizava para o template.

```
from django.shortcuts import render

# Código do projeto antigo

def portfolio_exibir(request):
    ... pessoa = DadosPessoais.objects.all()
    ... context = {'pessoa': pessoa}

    ... return render(request, 'portfolios/portfolio_exibir.html', context)
```

Acrescentando código em `views.py`

Agora `portfolios/views.py` importa algumas coisas de `rest_framework` e tem um *class* específico para API.

Faremos um primeiro exemplo somente com GET, ou seja somente a exibição dos dados.

```
views.py x
# coding: utf-8

from .serializer import DadosPessoaisSerializer
from .models import DadosPessoais

from rest_framework.response import Response
from rest_framework.views import APIView
```

Acrescentando código em views.py

Faremos um *class* exibindo todos os nossos cadastros

```
class PortfolioListView(APIView):  
    ... serializer_class = DadosPessoaisSerializer  
  
    ... def get(self, request, format=None):  
        ... serializer = self.serializer_class(DadosPessoais.objects.all(), many=True)  
        ... return Response(serializer.data)
```

Urls.py

Em `api/urls.py` coloque o código abaixo. Com isso direcionamos uma rota para exibir o conteúdo de *PortfolioListView*

```
# coding: utf-8
from django.conf.urls import url
from portfolios.views import PortfolioListView

helper_patterns = [
    url(r'^portfolios/$', PortfolioListView.as_view(), name='portfolios'),
]

urlpatterns = helper_patterns
```

Admin

Coloque o projeto para rodar

\$make run

Lembrando que esse novo projeto tem um MakeFile.

Mas caso queira poderá rodar com

\$python manage.py runserver

Adicione uma nova pessoa e salve.

Administração do Django

[Início](#) > [Portfolios](#) > [Dados Pessoais](#) > Adicionar Dados Pessoais

Adicionar Dados Pessoais

Nome:

Endereço:

Cidade:

Cep:

Telefone:

Celular:

Administração do Django

Início › Portfolios › Dados Pessoais

✓ O Dados Pessoais "Maria da Silva" foi adicionado com sucesso.

Selecione Dados Pessoais para modificar

Ação:



Ir

0 de 1 selecionados

☐

DADOS PESSOAIS

☐

Maria da Silva

1 Dados Pessoais

Acessando o Rest Framework no browser

Definimos em `urls.py` que `api.urls` se chamaria *api* e que nosso `PortfolioListView` se chamaria *portfolios*, por isso para acessar devemos colocar o endereço: `http://localhost:8000/api/portfolios/`

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^api/', include('api.urls')),
    url(r'', include('portfolios.urls')),
]
```

```
# coding: utf-8
from django.conf.urls import url
from portfolios.views import Portf

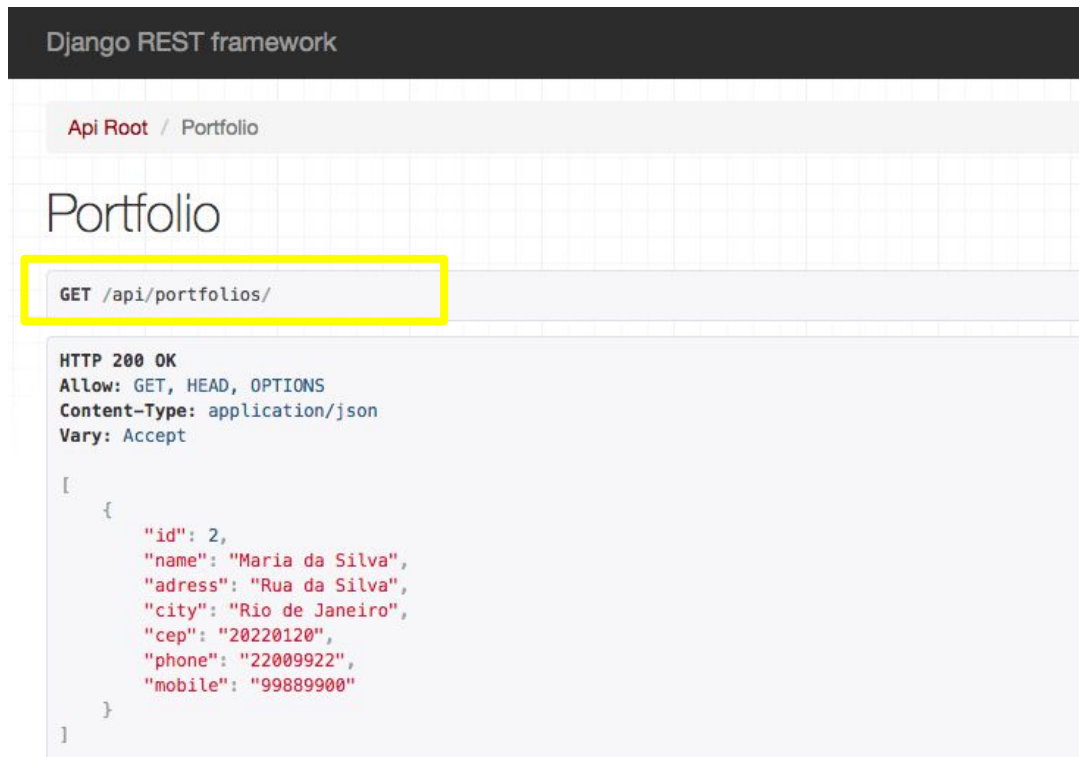
helper_patterns = [
    url(r'^portfolios/$', Portf

]

urlpatterns = helper_patterns
```

Acessando o Rest Framework no browser

<http://localhost:8000/api/portfolios/>



Django Rest Framework

Dessa maneira que fizemos o **portfolios/views.py** a medida que fomos acrescentando pessoas pelo Admin vamos gerar uma lista em **<http://localhost:8000/api/portfolios/>**

Django REST framework

Portfolio List

GET /api/portfolios/

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

```
[
  {
    "id": 2,
    "name": "Maria da Silva",
    "adress": "Rua da Silva",
    "city": "Rio de Janeiro",
    "cep": "20220120",
    "phone": "22009922",
    "mobile": "99889900"
  },
  {
    "id": 3,
    "name": "teste",
    "adress": "teste",
    "city": "teste",
    "cep": "010010101",
    "phone": "010101010",
    "mobile": "0101010"
  }
]
```

Localizar por ID

Se eu quiser buscar por um ID por exemplo o 2 que é o ID da Maria

<http://localhost:8000/api/portfolios/2>

Page not found (404)

Request Method: GET

Request URL: <http://localhost:8000/api/portfolios/2/>

Using the URLconf defined in `mysite.urls`, Django tried these URL patterns, in this order:

1. `^admin/`
2. `^api/ ^portfolios/$` [name='portfolios']
3. `^api/ ^$` [name='api-root']
4. `^api/ ^\.(?P<format>[a-z0-9]+)/?$` [name='api-root']

The current URL, `api/portfolios/2/`, didn't match any of these.

You're seeing this error because you have `DEBUG = True` in your Django settings file. Change that to `False`, and Django will display a standard 404 page.

Não é possível !

Alterando a views.py

Para que seja possível a exibição pelo ID e a lista
precisaremos alterar nossa view.

portfolios/views.py



Alterando views.py

Vamos acrescentar a exibição por ID em `portfolios/views.py`.

```
class PortfolioView(APIView):  
  
    def get(self, request, pk, format=None):  
        user = DadosPessoais.objects.get(pk=pk)  
        serializer = DadosPessoaisSerializer(user)  
        return Response(serializer.data)
```

Urls.py

Precisamos colocar nossas novas rotas no arquivo `api/urls.py`
Lembrando que para exibição por ID precisamos colocar **Regex** na rota.

```
# coding: utf-8
from django.conf.urls import url
from portfolios.views import PortfolioView, PortfolioListView

helper_patterns = [
    url(r'^portfolios/$', PortfolioListView.as_view(), name='portfolios'),
    url(r'^portfolios/(?P<pk>[0-9]+)/$', PortfolioView.as_view(), name='get_portfolio')
]

urlpatterns = helper_patterns
```

Regex

Um pouco de Regex:

`^` para o início do texto

`$` para o final do texto

`\d` para um dígito

`+` para indicar que o item anterior deve ser repetido pelo menos uma vez

`()` para capturar parte do padrão

Exibição por ID

Dessa maneira que fizemos a **portfolio/views.py** continua igual ao que estava sendo apresentado antes em

<http://localhost:8000/api/portfolios/>

Portfolio List

GET /api/portfolios/

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

```
[
  {
    "id": 2,
    "name": "Maria da Silva",
    "adress": "Rua da Silva",
    "city": "Rio de Janeiro",
    "cep": "20220120",
    "phone": "22009922",
    "mobile": "99889900"
  },
  {
    "id": 3,
    "name": "teste",
    "adress": "teste",
    "city": "teste",
    "cep": "010010101",
    "phone": "010101010",
    "mobile": "0101010"
  }
]
```

Localizar por ID

Porém se eu quiser buscar por um ID por exemplo o 2 que é o ID da Maria
<http://localhost:8000/api/portfolios/2>

Portfolio

```
GET /api/portfolios/2/
```

```
HTTP 200 OK
```

```
Allow: GET, HEAD, OPTIONS
```

```
Content-Type: application/json
```

```
Vary: Accept
```

```
{  
  "id": 2,  
  "name": "Maria da Silva",  
  "adress": "Rua da Silva",  
  "city": "Rio de Janeiro",  
  "cep": "20220120",  
  "phone": "22009922",  
  "mobile": "99889900"  
}
```


Postman

Fizemos até agora o método *GET*.

Para testar podemos usar o navegador ou uma ferramenta chamada Postman, onde nela podemos visualizar erros em vários formatos e testar outros métodos HTTPS como POST, PUT...

<https://www.getpostman.com/docs/introduction>



Postman

The screenshot shows the Postman application interface. At the top, there is a tab for the request URL: `http://localhost:8000/`. To the right, it says "No Environment" with a dropdown arrow, and there are icons for a preview and settings. Below this, the request method is set to "GET" and the URL is `http://localhost:8000/api/portfolios/`. There is a "Params" section, a blue "Send" button, and a "Save" button. The response is displayed in the "Body" tab, showing a JSON array of two objects. The first object represents a user with ID 2, and the second object represents a test user with ID 3. The status bar at the top right indicates "Status: 200 OK" and "Time: 40 ms".

http://localhost:8000/ x +

No Environment v

GET v http://localhost:8000/api/portfolios/ Params Send Save

Body Cookies Headers (6) Tests Status: 200 OK Time: 40 ms

Pretty Raw Preview JSON v

```
1 [
2   {
3     "id": 2,
4     "name": "Maria da Silva",
5     "adress": "Rua da Silva",
6     "city": "Rio de Janeiro",
7     "cep": "20220120",
8     "phone": "22009922",
9     "mobile": "99889900"
10  },
11  {
12    "id": 3,
13    "name": "teste",
14    "adress": "teste",
15    "city": "teste",
16    "cep": "010010101",
17    "phone": "010101010",
18    "mobile": "0101010"
19  },
20 ]
```

Post

Vamos adicionar o POST no arquivo `portfolios/views.py`

```
class PortfolioListView(APIView):
    ... serializer_class = DadosPessoaisSerializer

    ... def get(self, request, format=None):
    ...     serializer = self.serializer_class(DadosPessoais.objects.all(), many=True)
    ...     return Response(serializer.data)

    ... def post(self, request, format=None):
    ...     serializer = self.serializer_class(data=request.data)
    ...     if serializer.is_valid():
    ...         serializer.save()
    ...         return Response(serializer.data, status=status.HTTP_201_CREATED)
    ...     else:
    ...         return Response({"message": "403 Forbidden"}, status=status.HTTP_409_CONFLICT)
```

Post - Mensagem de erro tipo 1

Não mencionei no vídeo sobre como alterar as mensagens de erro. Essas seriam uma das maneiras possíveis de informar os erros. O serializer contém mensagens de erros prontas, podemos usar *serializer.errors*

```
· return Response(serializer.errors, status=status.HTTP_409_CONFLICT)
```

Post - Mensagem de erro tipo 1

A exibição do erro usando o *serializer.errors*, fica dessa maneira ao lado

Portfolio List

POST /api/portfolios/

HTTP 409 Conflict

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{
  "mobile": [
    "Este campo não pode ser em branco."
  ],
  "phone": [
    "Este campo não pode ser em branco."
  ],
  "adress": [
    "Este campo não pode ser em branco."
  ],
  "name": [
    "Este campo não pode ser em branco."
  ],
  "cep": [
    "Este campo não pode ser em branco."
  ]
}
```

Post - Mensagem de erro tipo 2

Essa é a outra maneira, onde colocamos uma mensagem qualquer.

```
return Response({"message": "Error"}, status=status.HTTP_409_CONFLICT)
```

Portfolio List

POST /api/portfolios/

HTTP 409 Conflict

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{  
  "message": "Error"  
}
```

Urls.py

Nosso arquivo `api/urls.py` continua o mesmo.

```
# coding: utf-8
from django.conf.urls import url
from portfolios.views import PortfolioListView, PortfolioView

helper_patterns = [
    ...url(r'^portfolios/$', PortfolioListView.as_view(), name='portfolios'),
    ...url(r'^portfolios/(?P<pk>[0-9]+)/$', PortfolioView.as_view(), name='get_portfolio')
]

urlpatterns = helper_patterns
```

Post

Indo em <http://localhost:8000/api/portfolios/> podemos ver no final da página o POST.

Raw data

HTML form

Nome

Endereço

Cidade

Cep

Telefone

Celular

POST

Post

Com isso podemos acrescentar uma pessoa diretamente por aqui e conferir no Admin se funcionou.

Django REST frameworkadmin

Raw dataHTML form

Nome

Adriana

Endereço

Rua da Adriana

Cidade

Rio de Janeiro

Cep

123123123

Telefone

123123123

Celular

123123123

POST

Post - Admin

Indo em <http://localhost:8000/admin> devemos encontrar as pessoas já cadastradas mais a última que cadastramos diretamente com POST

Administração do Django

Início > Portfolios > Dados Pessoais

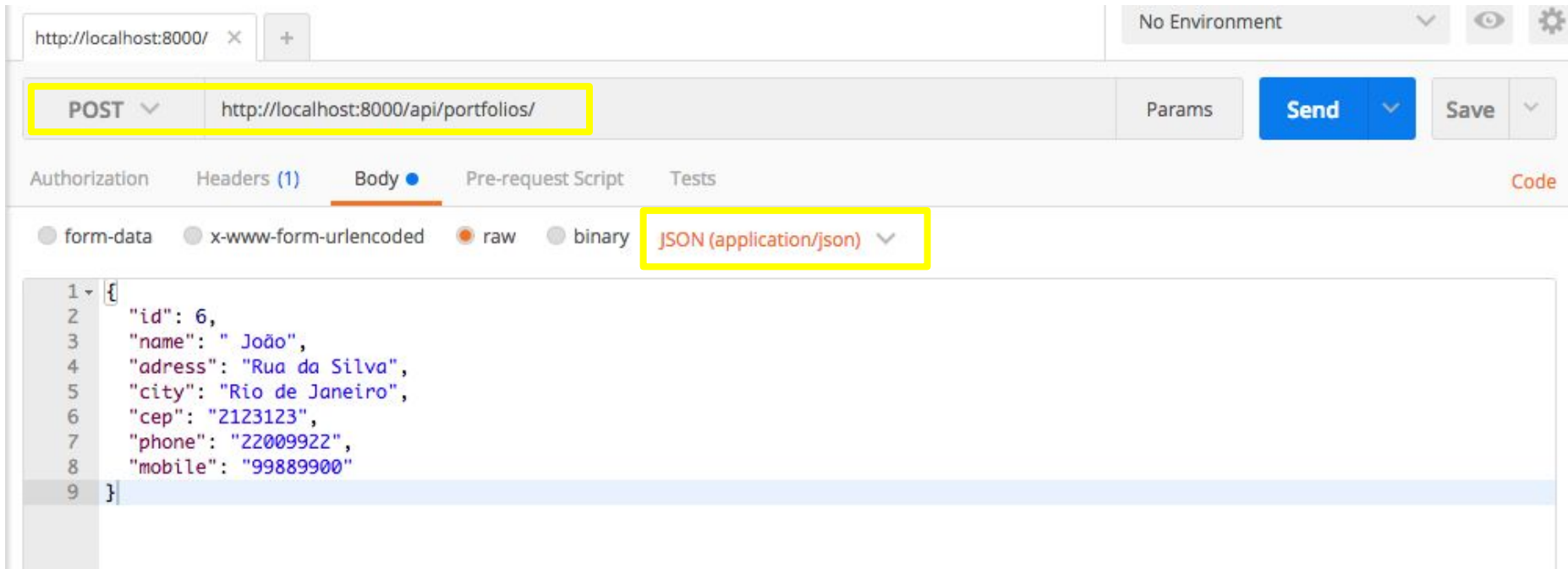
Selecione Dados Pessoais para modificar

Ação: ----- Ir 0 de 3 selecionados

<input type="checkbox"/>	DADOS PESSOAIS
<input type="checkbox"/>	Adriana
<input type="checkbox"/>	teste
<input type="checkbox"/>	Maria da Silva

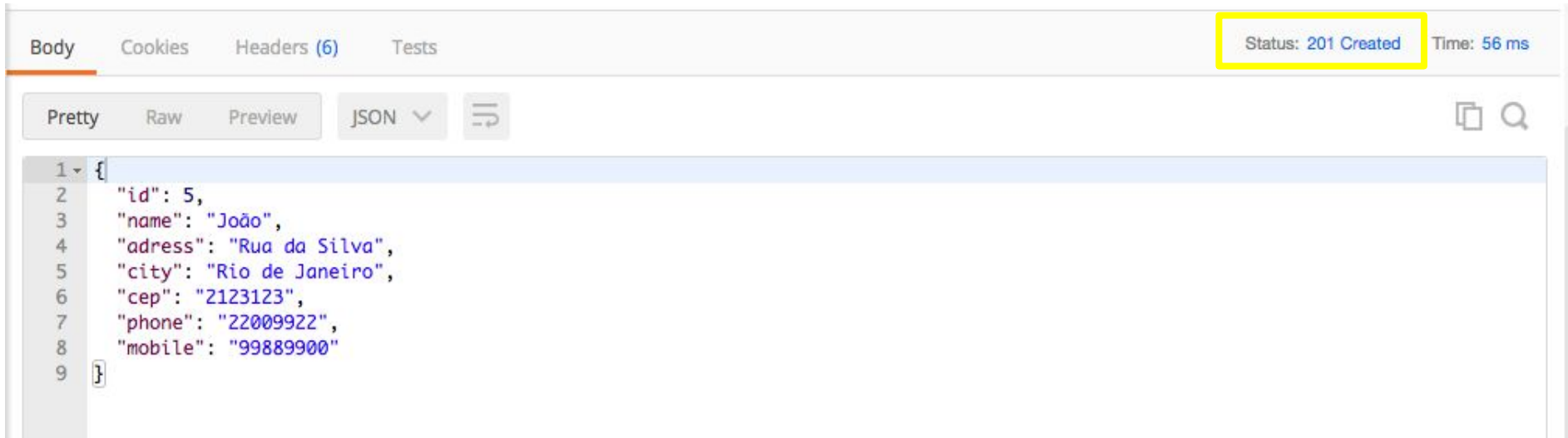
Postman

Pelo Postman podemos fazer o POST, selecionando a opção **POST**. Colocando o endereço para post - **http://localhost:8000/api/portfolios** Escrevendo no body com formato **JSON** (*todos os campos que foram serializados*) e clicando em **SEND**



Post

A resposta será indicada na parte abaixo da tela, junto com o status HTTP.



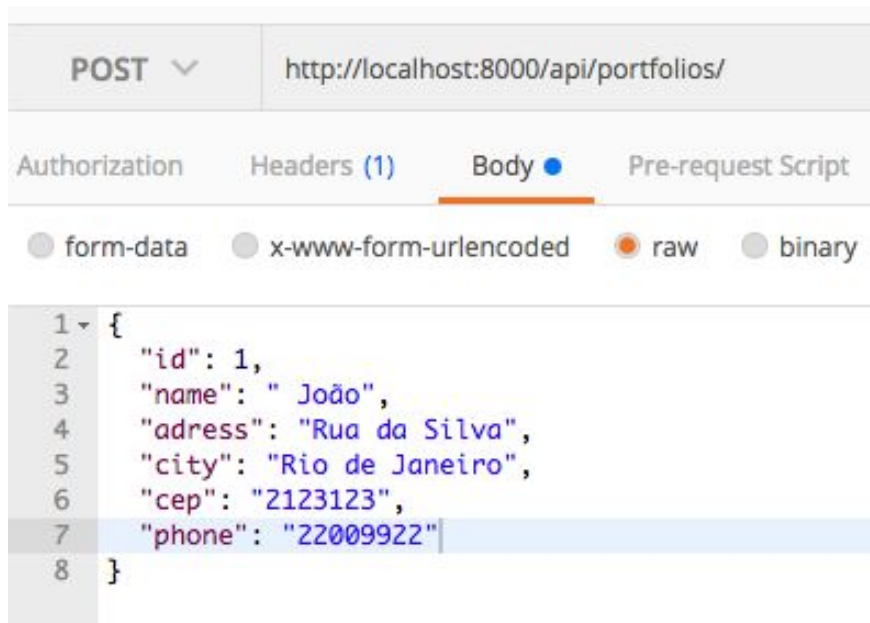
Postman

Todos os POSTS feitos no Postman, também podem ser verificados no Admin do projeto ou fazendo um GET diretamente no Postman ou no browser.



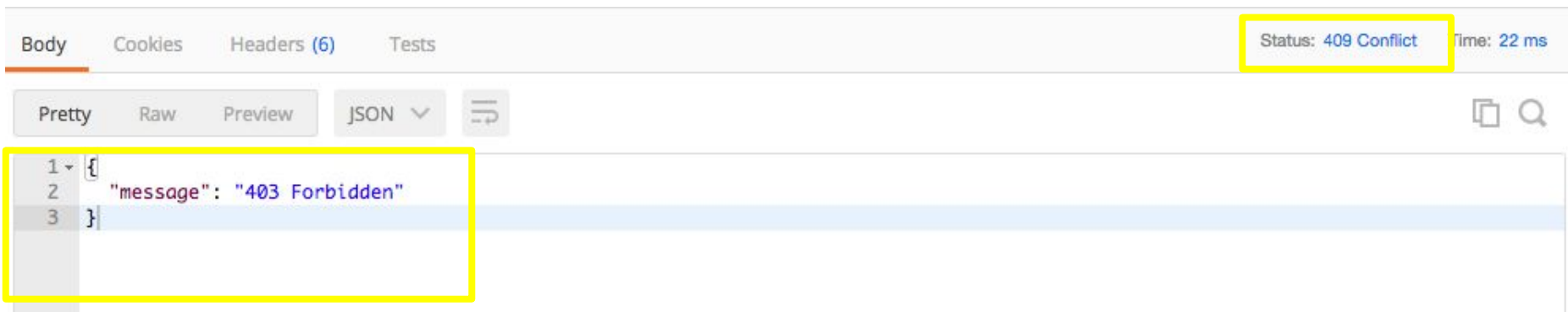
Post- Gerando erro no Postman

Em caso de erro, por exemplo, retirando um dos campos que é obrigatório .(*mobile*)



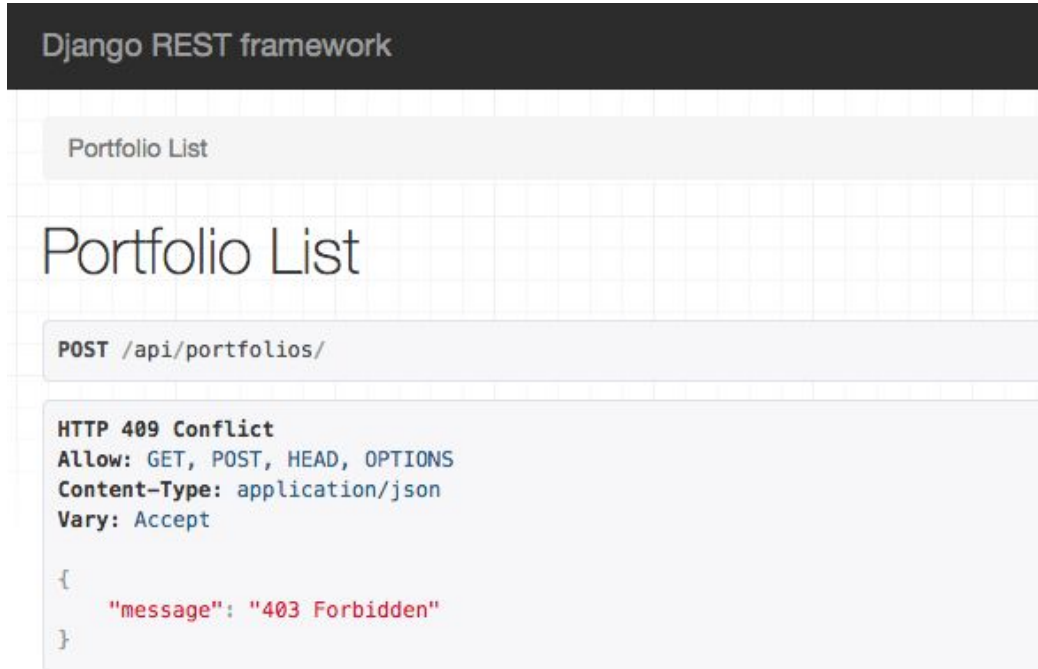
Post- Exibição de erro no Postman

O erro é apresentado com o status HTTP junto com uma mensagem configurável.



Erro no browser

Em caso de erro, por exemplo, retirando um dos campos que são obrigatórios.(mobile)



Onde posso aprender?

A documentação do DRF fornece um quickstart.

<http://www.django-rest-framework.org/>



Obrigada

about.me/pkcpweb

