A decorative horizontal bar is positioned above the title. It is composed of two segments: a teal segment on the left and an orange segment on the right.

# Técnico em Desenvolvimento de Sistemas - PW/BE

Criando API com Django Rest Framework



## Instalar o DRF no seu projeto django existente

Dentro do ambiente virtual do pipenv do seu projeto, execute o seguinte comando:

```
pipenv install djangorestframework
```



## Adicione o DRF no seu settings.py

Acesse o arquivo settings.py e após os aplicativos padrão do django, e antes do seu aplicativo (loja), adicione rest\_framework

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

## Criar a nossa view para receber requisições e retornar dados

```
views.py > ...  
from django.shortcuts import render  
from django.http import HttpResponse  
  
# Create your views here.  
def produtos_listar(request):  
    return HttpResponse('ok')
```


No arquivo `views.py`, importe a classe `HttpResponse`

Crie uma função chamada `produtos_listar` conforme a imagem



## Configure a URL para chamar sua função

No arquivo `urls.py` da sua aplicação (loja), crie um `urlpatterns` conforme a imagem ao lado e crie um path para a função criada na `views`

```
>  urls.py > ...  
from django.urls import path  
from . import views  
  
urlpatterns = [  
    path('produtos/', views.produtos_listar)  
]
```



## Certifique-se que o URL principal também está configurado corretamente

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

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

Adicione o path da sua aplicação (**loja**) no arquivo **urls.py** principal do projeto.



## Inicie o servidor e faça o teste

Se seguiu os passos  
corretamente, inicie o  
seu servidor, e acesse a  
URL para verificar a  
**resposta** da sua **API**

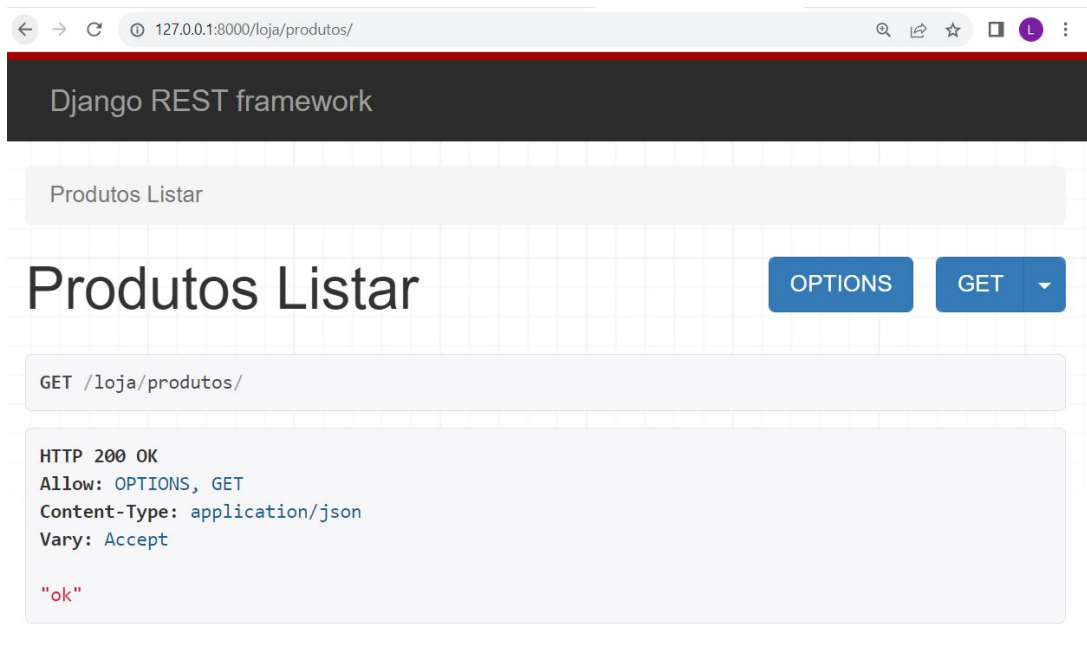


127.0.0.1:8000/loja/produtos/

ok

# DRF oferece uma interface amigável para a API

O que é chamado de ***browsable API***, nos permite ter acesso diretamente pelo browser a uma interface amigável e prática para testar nossa API





# Configurando a Browsable API

Importe o decorator `api_view` e a classe `Response` do `rest_framework`.

Faça os ajustes na sua função `produtos_listar`.

Acesse a sua API e verá uma interface mais detalhada

```
> python views.py > ...  
  
from django.shortcuts import render  
from django.http import HttpResponseRedirect  
from rest_framework.decorators import api_view  
from rest_framework.response import Response  
# Create your views here.  
  
@api_view()  
def produtos_listar(request):  
    return Response('ok')
```



## Passar parâmetros para uma função

```
@api_view()
def produto_detalhe(request, id):
    return Response(id)
```

Crie uma nova função no arquivo **views.py**, e entre os parênteses, após o request, adicione 'id'

Após isso altere o arquivo **urls.py** da sua aplicação para que o endpoint receba um parâmetro.

```
urlpatterns = [
    path('produtos/', views.produtos_listar),
    path('produtos/<id>/', views.produto_detalhe),
]
```



## Como faz para nossa API retornar dados de verdade?


Ao invés de apenas retornar uma mensagem de **OK**, a API deve ser construída para retornar uma lista de produtos ou um produto específico.

Mas como fazer isso?

## Passo 1 - Criar um Serializer (Serializador)

Na pasta da sua aplicação (**loja**), crie um arquivo chamado **serializer.py**

Dentro dele, importe a classe **serializers** do **rest\_framework** e depois crie uma classe que será responsável por serializar os dados

```
loja >  serializer.py > ...  
7  from rest_framework import serializers  
6  
5  class ProdutoSerializer(serializers.Serializer):  
4      id = serializers.IntegerField()  
3      titulo = serializers.CharField(max_length=255)  
2      preco_unitario = serializers.DecimalField(max_digits=6, decimal_places=2)  
1
```

## Passo 2 - Importe as classes no arquivo views.py



```
from .models import Produtos  
from .serializer import ProdutoSerializer
```



## Passo 3 - Altere a função produto\_detalhe

Ainda no arquivo `views.py`, altere a função que criamos anteriormente, para agora retornar dados ao invés da mensagem de OK

```
@api_view()
def produto_detalhe(request, id):
    produto = Produtos.objects.get(pk=id)
    serializer = ProdutoSerializer(produto)
    return Response(serializer.data)
```

## Passo 4 - Acesse a API no browser e veja o resultado

### Produto Detalhe

OPTIONS

GET



GET /loja/produtos/1/

HTTP 200 OK

Allow: GET, OPTIONS

Content-Type: application/json

Vary: Accept

```
{  
  "id": 1,  
  "titulo": "Coca cola lata 310ml",  
  "preco_unitario": "3.13"  
}
```

**Encontrou algum  
erro no resultado  
retornado pela API?**



## Para corrigirmos o problema de retornar um atributo decimal como string

Abra o arquivo `settings.py` e adicione ao final o código conforme a imagem ao lado

```
REST_FRAMEWORK = {  
    'COERCE_DECIMAL_TO_STRING': False  
}
```



# Confira o resultado



## Produto Detalhe

OPTIONS

GET



GET /loja/produtos/1/

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

```
{  
  "id": 1,  
  "titulo": "Coca cola lata 310ml",  
  "preco_unitario": 3.13  
}
```

# E se consultarmos a API com um id inexistente?



← → ↻ ⓘ 127.0.0.1:8000/loja/produtos/0/

DoesNotExist at /loja/produtos/0/

Produtos matching query does not exist.

**Request Method:** GET

**Request URL:** http://127.0.0.1:8000/loja/produtos/0/

**Django Version:** 4.1

**Exception Type:** DoesNotExist

**Exception Value:** Produtos matching query does not exist.

A nossa API **não deve** retornar erro, caso o ID informado não corresponda a um registro válido no banco de dados, devemos retornar o **STATUS 404**

## Acesse o arquivo views.py e adicione um try except

---

```
@api_view()
def produto_detalhe(request, id):
    try:
        produto = Produtos.objects.get(pk=id)
        serializer = ProdutoSerializer(produto)
        return Response(serializer.data)
    except Produtos.DoesNotExist:
        return Response(status=404)
```

# Confira o retorno da API

Produtos Listar / Produto Detalhe

## Produto Detalhe

OPTIONS

GET



GET /loja/produtos/0/

HTTP 404 Not Found

Allow: OPTIONS, GET

Content-Type: application/json

Vary: Accept

# Ao invés de definir 404, podemos melhorar o código



Importamos o módulo de `status` do `rest_framework`, com isso teremos um código mais legível no status

```
from rest_framework import status
```

```
@api_view()
def produto_detalhe(request, id):
    try:
        produto = Produtos.objects.get(pk=id)
        serializer = ProdutoSerializer(produto)
        return Response(serializer.data)
    except Produtos.DoesNotExist:
        return Response(status=status.HTTP_404_NOT_FOUND)
```

## Há como simplificar o try except?

```
views.py > ...  
from django.shortcuts import get_object_or_404  
  
@api_view()  
def produto_detalhe(request, id):  
    produto = get_object_or_404(Produtos, pk=id)  
    serializer = ProdutoSerializer(produto)  
    return Response(serializer.data)
```

# Retornar uma lista de produtos



```
@api_view()
def produtos_listar(request):
    queryset = Produtos.objects.all()
    serializer = ProdutoSerializer(queryset, many=True)
    return Response(serializer.data)
```

## Django REST framework

Produtos Listar

# Produtos Listar

OPTIONS

GET



GET /loja/produtos/

HTTP 200 OK

Allow: OPTIONS, GET

Content-Type: application/json

Vary: Accept

```
[
  {
    "id": 3,
    "titulo": "Sprite lata",
    "preco_unitario": 3.22
  },
  {
    "id": 4,
    "titulo": "Pepsi",
    "preco_unitario": 4.03
  },
  {
    "id": 1,
    "titulo": "Coca cola lata 310ml",
    "preco_unitario": 3.13
  },
]
```



# Podemos adicionar atributos customizados

```
class ProdutoSerializer(serializers.Serializer):
    id = serializers.IntegerField()
    titulo = serializers.CharField(max_length=255)
    preco_unitario = serializers.DecimalField(max_digits=6, decimal_places=2)

    preco_taxa = serializers.SerializerMethodField(method_name='calcular_taxa')

    def calcular_taxa(self, produto: Produtos):
        return produto.preco_unitario * Decimal(1.1 )
```

Criamos uma função que calcula 10% sobre o valor do produto e retornamos na API

Produtos Listar

# Produtos Listar

OPTIONS

GET



GET /loja/produtos/

HTTP 200 OK

Allow: OPTIONS, GET

Content-Type: application/json

Vary: Accept

```
[
  {
    "id": 3,
    "titulo": "Sprite lata",
    "preco_unitario": 3.22,
    "preco_taxa": 3.5420000000000003
  },
  {
    "id": 4,
    "titulo": "Pepsi",
    "preco_unitario": 4.03,
    "preco_taxa": 4.4330000000000001
  },
  {
    "id": 5,
    "titulo": "Coca Cola",
    "preco_unitario": 3.5,
    "preco_taxa": 3.85
  }
]
```

## Outra forma de serializar nossos dados



```
class ProdutoSerializer(serializers.ModelSerializer):  
    class Meta:  
        model = Produtos  
        fields = ['id', 'titulo', 'preco_unitario', 'preco_taxa']  
  
        preco_taxa = serializers.SerializerMethodField(method_name='calcular_taxa')  
  
    def calcular_taxa(self, produto: Produtos):  
        return produto.preco_unitario * Decimal(1.1 )
```

# Desserializar os dados

---

```
@api_view(['GET', 'POST'])
def produtos_listar(request):
    if request.method == 'GET':
        queryset = Produtos.objects.all()
        serializer = ProdutoSerializer(queryset, many=True)
        return Response(serializer.data)
    elif request.method == 'POST':
        serializer = ProdutoSerializer(data=request.data)
        return Response('ok')
```

# Validar e Salvar os dados

```
@api_view(['GET', 'POST'])
def produtos_listar(request):
    if request.method == 'GET':
        queryset = Produtos.objects.all()
        serializer = ProdutoSerializer(queryset, many=True)
        return Response(serializer.data)
    elif request.method == 'POST':
        serializer = ProdutoSerializer(data=request.data)
        if serializer.is_valid():
            print(serializer.validated_data)
            serializer.save()
            return Response('ok')
        else:
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

# Atualizar e deletar registros



```
@api_view(['GET', 'PUT', 'DELETE'])
✓ def produto_detalhe(request, id):
    produto = get_object_or_404(Produtos, pk=id)
    ✓ if request.method == 'GET':
        serializer = ProdutoSerializer(produto)
        return Response(serializer.data)
    ✓ elif request.method == 'PUT':
        serializer = ProdutoSerializer(produto, data=request.data)
        serializer.is_valid(raise_exception=True)
        serializer.save()
        return Response(serializer.data)
    ✓ elif request.method == 'DELETE':
        produto.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
```