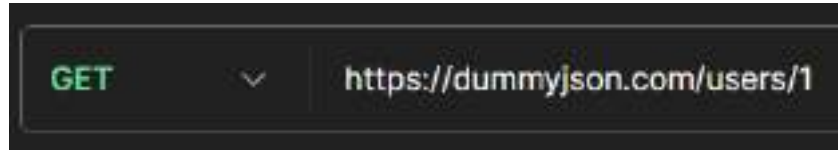
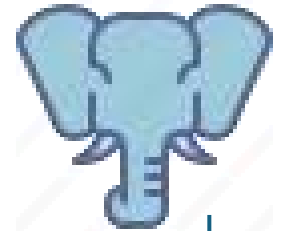


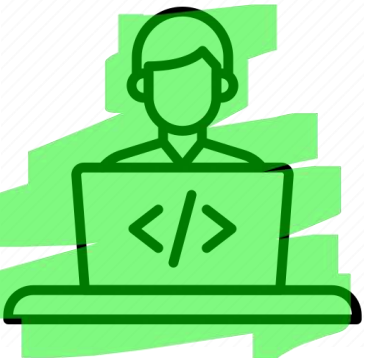


Ciclo de Requests



```
from rest_framework import serializers
from .models import MockUser

class MockUserSerializer(serializers.ModelSerializer):
    class Meta:
        model = MockUser
        fields = ['id', 'username', 'email']
```



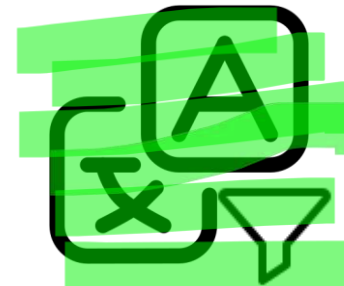
request



endpoint
urls



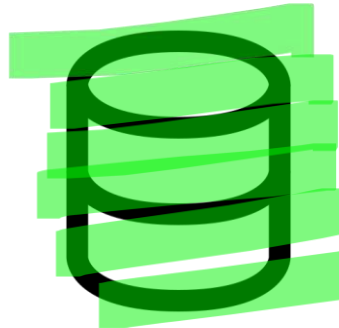
viewsets



Serializers
JSON=Django



models



database

```
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import MockUserViewSet

router = DefaultRouter()
router.register(r'mock-users', MockUserViewSet)

urlpatterns = [
    path('', include(router.urls)),
]
```

```
class MockUserViewSet(viewsets.ModelViewSet):
    queryset = MockUser.objects.all()
    serializer_class = MockUserSerializer
```

```
from django.db import models

class MockUser(models.Model):
    username = models.CharField(max_length=100)
    email = models.EmailField(unique=True)

    def __str__(self):
        return self.username
```

Para melhorar nosso projeto, agora vamos adicionar:

 **Login (Permissões)**

 **Paginação**

 **Filtros**



Mas primeiro vamos arrumar isso aqui...

Rode seu projeto e acesse:

<http://localhost:8000/sales/>

Product id	Product object (2)
Client id	Client object (5)
Employee id	Employee object (1)
Nrf	Employee object (1)
	Employee object (2)
	Employee object (3)
POST	

Vá para o seu MODELS.py e adicione:




```
class Client(ModelBase):
```

```
    name = models.CharField(
        db_column='tx_nome',
        max_length=70,
        null=False
    )
    age = models.IntegerField(
        db_column='nb_idade',
        null=False
    )
    rg = models.CharField(
        db_column='tx_rg',
        max_length=12,
        null=False
    )
    cpf = models.CharField(
        db_column='tx_cpf',
        max_length=12,
        null=False
    )
```

```
    def __str__(self):
        return f"{self.id} - {self.name}"
```

```
class Product(ModelBase):
```

```
    description = models.TextField(
        db_column='description',
        null=False
    )
    quantity = models.IntegerField(
        db_column='quantity',
        null=False,
        default=0
    )
```

```
    def __str__(self):
        return f"{self.description[:30]}... - Qtd: {self.quantity}"
```

```
class Employee(ModelBase):
```

```
    name = models.CharField(
        db_column='tx_nome',
        max_length=70,
        null=False
    )
    registration = models.CharField(
        db_column='tx_registro',
        max_length=15,
        null=False
    )
```

```
    def __str__(self):
        return f"{self.id} - {self.name}"
```

```
class Sale(ModelBase):
    product = models.ForeignKey(
        Product,
        db_column='id_product',
        null=False,
        on_delete=models.DO_NOTHING
    )
    client = models.ForeignKey(
        Client,
        db_column='id_client',
        null=False,
        on_delete=models.DO_NOTHING
    )
    employee = models.ForeignKey(
        Employee,
        db_column='id_employee',
        null=False,
        on_delete=models.DO_NOTHING
    )
    nrf = models.CharField(
        db_column='tx_nrf',
        max_length=255,
        null=False
    )
```

```
def __str__(self):
    return (f"Nf:{self.nrf} - Cliente:{self.client.name}, "
            f"Produto:{self.product.description[:30]}...", "
            f"Funcionario:{self.employee.name}")
```

Depois de adicionar eles, pode rodar seu projeto:

Acesse <http://localhost/sales/>

Product id	Aula de Ingles com material di... - Qtd: 1
Client id	2 - Fulano da Silva
Employee id	1 - Beltrano dos Santos
Nrf	1 - Beltrano dos Santos
	2 - Fulana Rosa
	3 - Teste
	4 - Fabricio

Para melhorar nosso projeto, agora vamos adicionar:

→ **Login (Permissões)**

Paginação

Filtros



Login (Permissões)

Primeiramente vamos criar um usuário, mas não um usuário qualquer, um SUPERUSUÁRIO.

```
oProject1> python manage.py createsuperuser
```

O sistema vai pedir que você preencha:

- nome
- email
- senha
- repetir senha para confirmar



Login (Permissões)

No exemplo abaixo estou criando um SuperUsuário de nome “jon”, email “jon@gmail.com”, senha: “jon”.

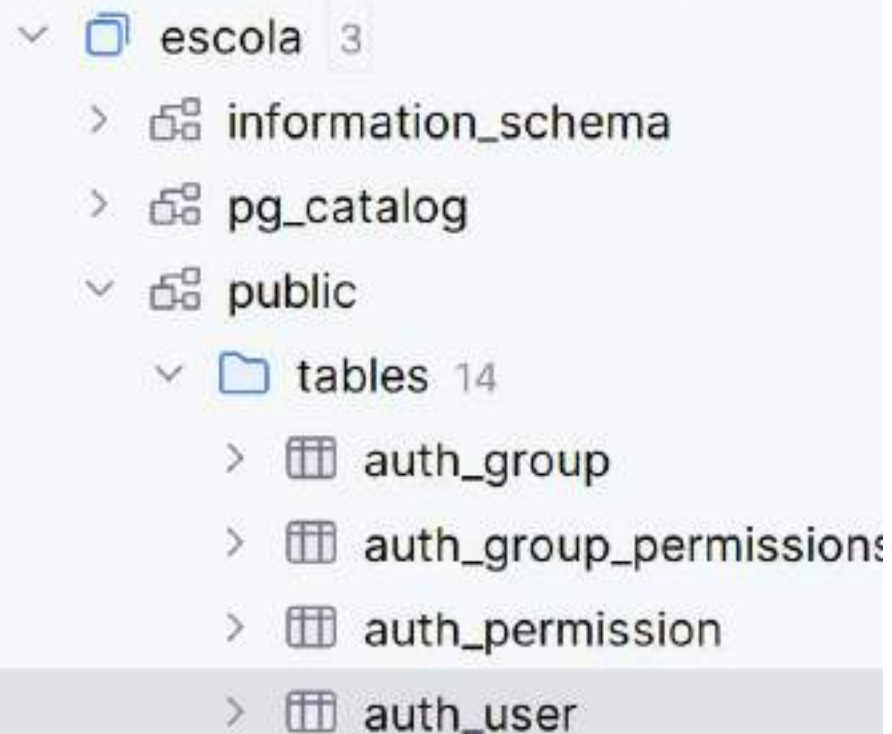
```
(.venv) PS C:\Users\jonatas.lopez\PycharmProjects\ujangoProject1> python3
Username (leave blank to use 'jonatas.lopez'): jon
Email address: jon@gmail.com
Password:      <- aqui eu digitei: jon (por questões de segurança, a senha não vai aparecer
Password (again):  <- aqui eu digitei novamente: jon
The password is too similar to the username. <- validação: nome e senha iguais.
This password is too short. It must contain at least 8 characters.
This password is too common.                  <- validação: senha maior que 8, e senha comum.
Bypass password validation and create user anyway? [y/N]: y
```

<- conclusão: descartei as validações e cliquei em y (sim)



```
Bypass password validation and
Superuser created successfully.
```

Ver o usuário criado no postgres



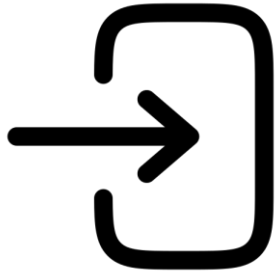
WHERE		ORDER BY								
	id	password	last_login	is_superuser	username	first_name	last_name	email	is_staff	is_active
1	1	pbkdf2_sha256\$10000000\$X3g...	<null>	• true	jon			jon@gmail.com	• true	• true

<- aqui a senha “jon” foi encriptada



@fpftech.educacional

viewsets



Depois de adicionar os amarelos...

Rode o Django novamente

```
from rest_framework import viewsets, permissions
from escola.models import Client, Product, Employee, Sale
from escola.serializers import ClientSerializer, ProductSerializer, EmployeeSerializer, SaleSerializer

2 usages
class ClientViewSet(viewsets.ModelViewSet):
    queryset = Client.objects.all()
    serializer_class = ClientSerializer
    permission_classes = [permissions.IsAuthenticated]

2 usages
class ProductViewSet(viewsets.ModelViewSet):
    queryset = Product.objects.all()
    serializer_class = ProductSerializer
    permission_classes = [permissions.IsAuthenticated]

2 usages
class EmployeeViewSet(viewsets.ModelViewSet):
    queryset = Employee.objects.all()
    serializer_class = EmployeeSerializer
    permission_classes = [permissions.IsAuthenticated]

2 usages
class SaleViewSet(viewsets.ModelViewSet):
    queryset = Sale.objects.all()
    serializer_class = SaleSerializer
    permission_classes = [permissions.IsAuthenticated]
```





Faça o Login com seu superuser

Django REST framework

Log in

Api Root / Client List

Client List

GET

GET /clients/

HTTP 403 Forbidden

Allow: GET, POST, HEAD, OPTIONS

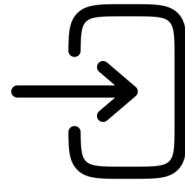
Content-Type: application/json

Vary: Accept

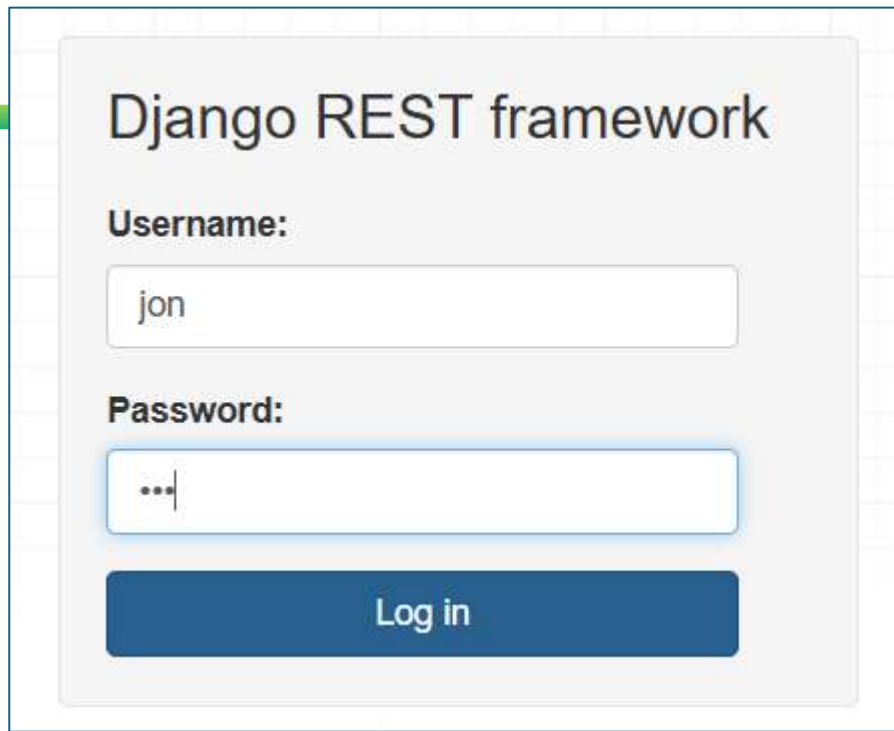
```
{  
  "detail": "Authentication credentials were not provided."  
}
```



@fpftech.educacional



Agora sim...



Django REST framework

Username:

Password:

Log in

Obs: existem outros tipos de permissões tbm:

AllowAny: Url liberado sem necessidade de autenticação;

IsAuthenticated: Url só libera se houver um usuário autenticado;

IsAdminUser: Verifica se usuário tem privilégios de admin (is_superuser)

Client List

GET /clients/

HTTP 200 OK

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
[
  {
    "id": 2,
    "name": "Fulano da Silva",
    "age": 23,
    "rg": "123456-7",
    "cpf": "123456789-01"
  },
  {
    "id": 3,
    "name": "Siclano da Silva",
    "age": 41
```



Para melhorar nosso projeto, agora vamos adicionar:

Login (Permissões)

- Create superuser
- Adicionar permissões nos viewsets



Paginação

Filtros



Quando se tem muitos dados no banco, não precisa trazer todos eles de uma única vez.

3 tipos de paginação mais comuns:

- **PageNumberPagination**= É o mais simples, vem com um número certo de items por página.

Exemplo: <http://localhost:8000/client/?page=1>

- **CursorPagination**= Igual o anterior, só que você pode esconder a informação da página na url.

Exemplo: [http://localhost:8000/client/? cursor=cj0xJnA9MjAyNC0xMC0wMSswMCUzQTA1JT](http://localhost:8000/client/?cursor=cj0xJnA9MjAyNC0xMC0wMSswMCUzQTA1JT)

- **LimitOffset**= É o mais recomendado, flexibilidade de definir quantidade que pode vim da lista.
(vamos usar esse)

Exemplo: <http://localhost:8000/client/?limit=2&offset=2>



Implementando Paginação

No settings.py, criar uma nova variável chamada “REST_FRAMEWORK”

- PageNumberPagination=

```
122
123     DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
124
125     REST_FRAMEWORK = {
126         'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
127         'PAGE_SIZE': 10
128     }
```

- CursorPagination=

```
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.CursorPagination',
    'PAGE_SIZE': 10
}
```



- LimitOffset= (vamos usar esse)

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.LimitOffsetPagination'
}
```



Como funciona o LimitOffset



Limit= 2 e Offset 0

Value 1	Value 2	Value 3
Value 4	Value 5	Value 6
Value 7	Value 8	Value 9

Limit 4, Offset 5

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

Limit= 2 e Offset 1

Value 1	Value 2	Value 3
Value 4	Value 5	Value 6
Value 7	Value 8	Value 9

Limit 6, Offset 2

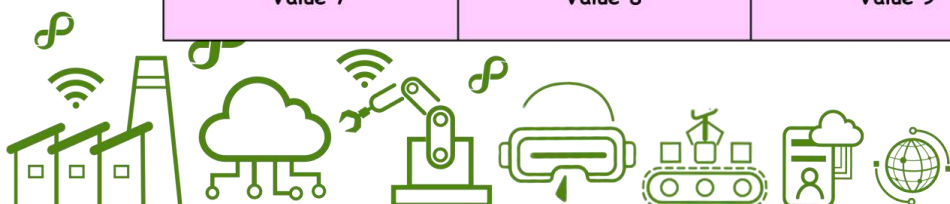
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

Limit= 6 e Offset 3

Value 1	Value 2	Value 3
Value 4	Value 5	Value 6
Value 7	Value 8	Value 9

Limit 3, Offset 9

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13



Itens por página

10

1 - 10 de 16



Rodando o proj.



```
GET /clients/?limit=3&offset=1
```

HTTP 200 OK

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{
  "count": 7,
  "next": "http://localhost:8000/clients/?limit=3&offset=4",
  "previous": "http://localhost:8000/clients/?limit=3",
  "results": [
    {
      "id": 3,
      "name": "Siclano da Silva",
      "age": 41,
      "rg": "234567-8",
      "cpf": "234567890-12"
    },
    {
      "id": 4,
      "name": "Siclano Bertrano",
      "age": 19,
      "rg": "12345498-9",
      "cpf": "234456859-48"
    },
    {
      "id": 6,
      "name": "Siclano Bertrano",
      "age": 19,
      "rg": "12345498-9",
      "cpf": "234456859-48"
    }
  ]
}
```



Para melhorar nosso projeto, agora vamos adicionar:

Login (Permissões)

- Create superuser
- Adicionar permissões nos viewsets

Paginação

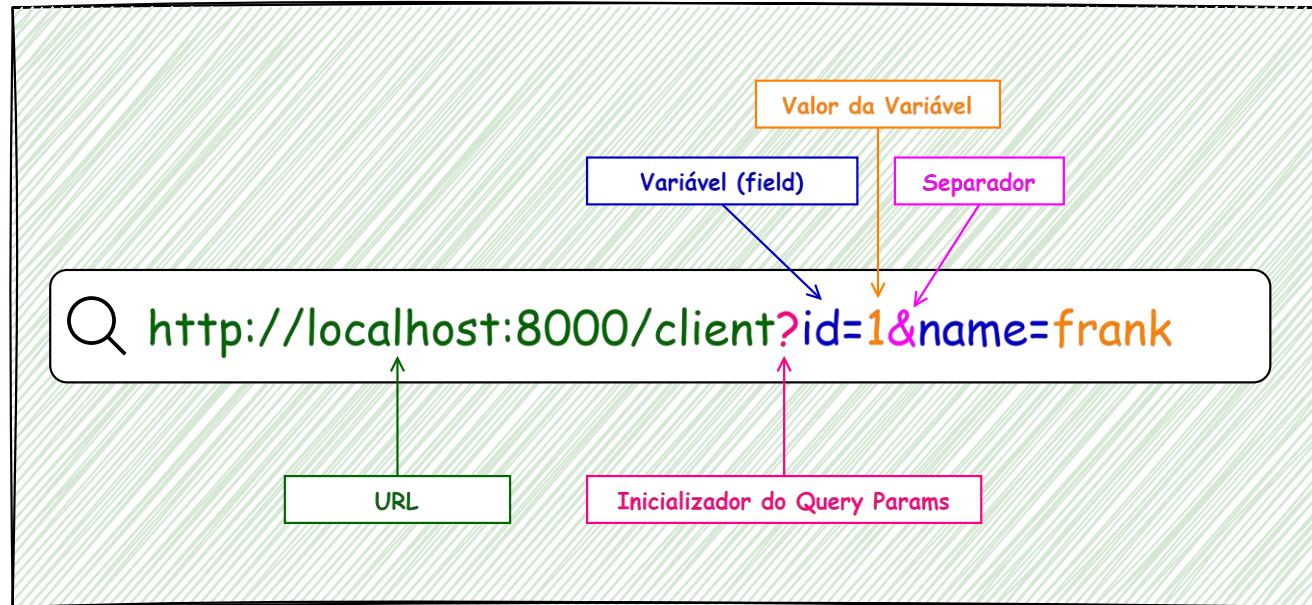
- Adicionar variável “REST_FRAMEWORK” no settings.py
- LimitOffset



Filtros



Instalar biblioteca que faz filtros dinâmicos baseados em parâmetros de consulta (query parameters) na URL.



```
project1> pip install django-filter
```

```
project1> pip freeze > requirements.txt
```

```
asgiref==3.8.1  
Django==5.2.1  
django-filter==25.1  
djangorestframework==3.16.0  
psycopg2-binary==2.9.10  
sqlparse==0.5.3  
tzdata==2025.2
```



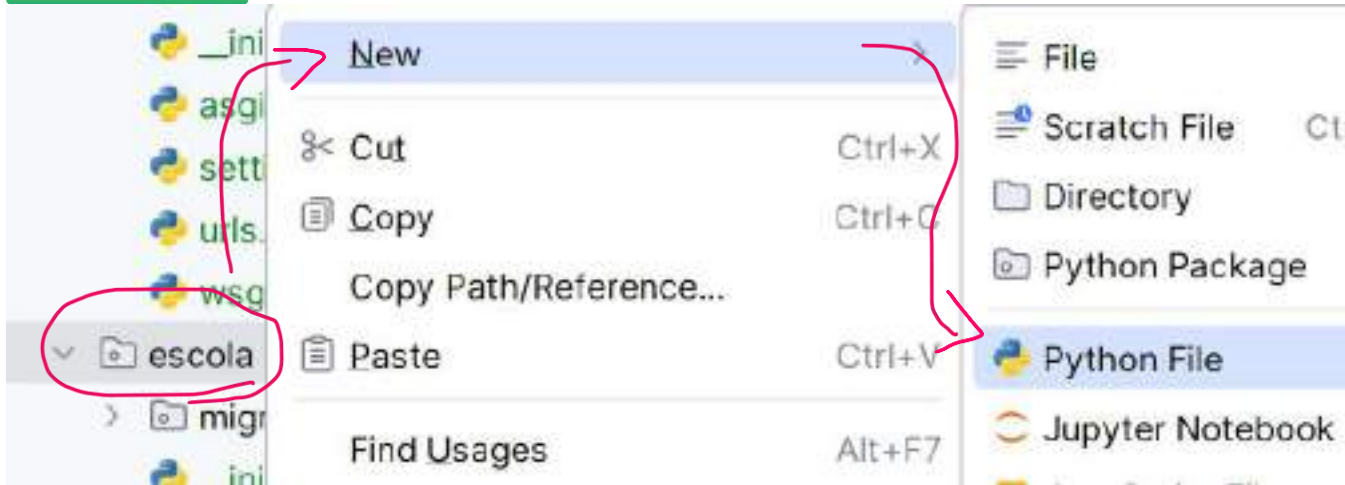
Filtro, Atualizando o settings.py

```
1  INSTALLED_APPS = [  
2      'django.contrib.admin',  
3      'django.contrib.auth',  
4      'django.contrib.contenttypes',  
5      'django.contrib.sessions',  
6      'django.contrib.messages',  
7      'django.contrib.staticfiles',  
8      'escola.apps.EscolaConfig',  
9      'rest_framework',  
10     'django_filters'  
11 ]
```

```
REST_FRAMEWORK = {  
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.LimitOffsetPagination',  
    'DEFAULT_FILTER_BACKENDS': 'django_filters.rest_framework.DjangoFilterBackend'  
}
```



Crie um novo arquivo "filters" no seu aplicativo

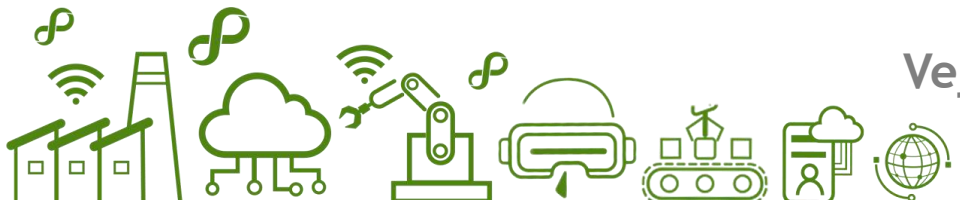


```
class ClientFilter(django_filters.FilterSet):  
    name = django_filters.CharFilter(lookup_expr='icontains')  
    cpf = django_filters.CharFilter(lookup_expr='exact')  
    rg = django_filters.CharFilter(lookup_expr='exact')  
  
    class Meta:  
        model = Client  
        fields = ['id', 'name', 'age', 'cpf', 'rg']
```

<- lookup expression faz a filtragem.

<- retornam os campos.

Veja completo em www.dontpad.com/NDS03/filters



Agora

atualize
seu
view.py
pra ter
os filtros

```
from rest_framework import viewsets, permissions
from escola.models import Client, Product, Employee, Sale
from escola.serializers import ClientSerializer, ProductSerializer, EmployeeSerializer, SaleSerializer
from escola.filters import ClientFilter, ProductFilter, EmployeeFilter, SaleFilter
from django_filters.rest_framework import DjangoFilterBackend
```

2 usages

```
class ClientViewSet(viewsets.ModelViewSet):
    queryset = Client.objects.all()
    serializer_class = ClientSerializer
    permission_classes = [permissions.IsAuthenticated]
    filter_backends = [DjangoFilterBackend]
    filterset_class = ClientFilter
```

2 usages

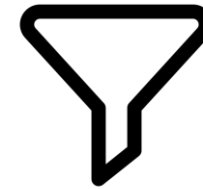
```
class ProductViewSet(viewsets.ModelViewSet):
    queryset = Product.objects.all()
    serializer_class = ProductSerializer
    permission_classes = [permissions.IsAuthenticated]
    filter_backends = [DjangoFilterBackend]
    filterset_class = ProductFilter
```

2 usages

```
class EmployeeViewSet(viewsets.ModelViewSet):
    queryset = Employee.objects.all()
    serializer_class = EmployeeSerializer
    permission_classes = [permissions.IsAuthenticated]
    filter_backends = [DjangoFilterBackend]
    filterset_class = EmployeeFilter
```

2 usages

```
class SaleViewSet(viewsets.ModelViewSet):
    queryset = Sale.objects.all()
    serializer_class = SaleSerializer
    permission_classes = [permissions.IsAuthenticated]
    filter_backends = [DjangoFilterBackend]
    filterset_class = SaleFilter
```



Pronto!

Agora é só
rodar



Product List

 Filters

OPTIONS

GET

GET /products/

HTTP 200 OK

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
[
  {
    "id": 1,
    "description": "Aula de Ing
    "quantity": 1
  },
  {
    "id": 2
```

Filters

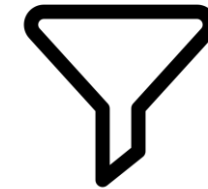
Field filters

Description contains:

Quantity:

Submit

url.../?name=Fernando&age=27



```
GET /clients/?name=Bernardo
```

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

```
[
  {
    "id": 8,
    "name": "Bernardo",
    "age": 19,
    "rg": "1231029-3",
    "cpf": "12334123-34"
  }
]
```

```
GET /employees/?name=Fulana
```

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

```
[
  {
    "id": 2,
    "created_at": null,
    "modified_at": null,
    "active": true,
    "name": "Fulana Rosa",
    "registration": "345678"
  }
]
```

```
GET /products/?description=material&quantity=
```

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

```
[
  {
    "id": 1,
    "description": "Aula de Inglês com material didático",
    "quantity": 1
  },
  {
    "id": 2,
    "description": "Material didático de Introdução a Física nível Ensino Superior",
    "quantity": 2
  }
]
```



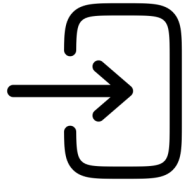
Alguns exemplos de filtros em (lookup_expressions)

LOOKUPS	SQL	DEFINIÇÃO
<code></></code> unaccent__icontains	<code></></code> WHERE unaccent(column_name) ILIKE '%value%'	<code>T</code> (LIKE sem acentos e case-insensitive)
<code></></code> exact	<code></></code> WHERE column_name = 'value'	<code>T</code> (igual a um valor específico)
<code></></code> in	<code></></code> WHERE column_name IN ('value1', 'value2', 'value3')	<code>T</code> (verifica se o valor está em uma lista de valores)
<code></></code> lt	<code></></code> WHERE column_name < 'value'	<code>T</code> (menor que)
<code></></code> gt	<code></></code> WHERE column_name > 'value'	<code>T</code> (maior que)
<code></></code> lte	<code></></code> WHERE column_name <= 'value'	<code>T</code> (menor ou igual a)
<code></></code> gte	<code></></code> WHERE column_name >= 'value'	<code>T</code> (maior ou igual a)

Veja mais em <https://medium.com/dajngo/lookup-expressions-in-django-61715708dd6f>



Para melhorar nosso projeto, agora vamos adicionar:



Login (Permissões)

- Create superuser
- Adicionar permissões nos viewsets



Paginação

- Adicionar variável “REST_FRAMEWORK” no settings.py
- LimitOffset



Filtros

- Instalar biblioteca django-filter (atualizar o settings.py)
- Criar arquivo “filters.py” na aplicação escola e fazer os filtros
- Adicionar os filtros criados no arquivo “views.py”
- Lookup expressions



Obrigado!



    @fpftech.educacional