@fpftech.educacional

FPFtech
Escola Tecnológica

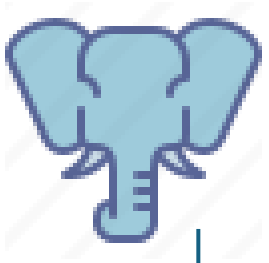
Ciclo de Requests

FPFtech
Escola Tecnológica

GET https://dummyjson.com/users/1

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

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



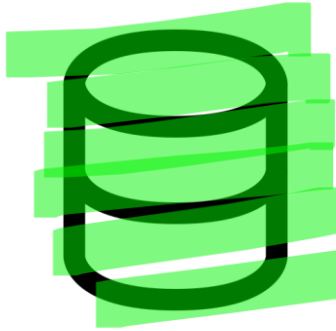
request

endpoint
urls

viewsets

Serializers
JSON=Django

models



database

```python
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)),
]
```

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

```python
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
```

Melhorando nosso projeto

FPFtech
Escola Tecnológica

Para melhorar nosso projeto, agora vamos adicionar:

Login (Permissões)

Paginação

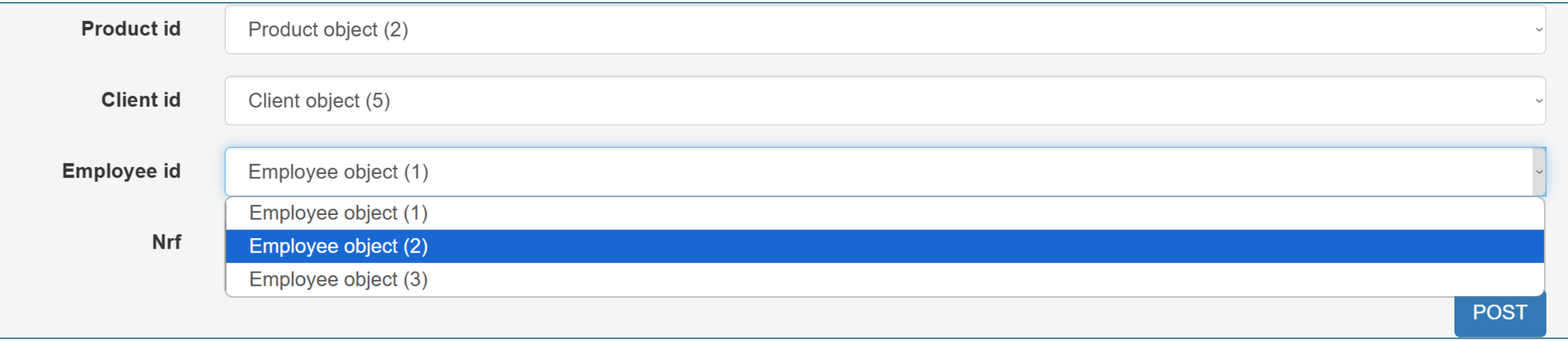
Filtros

Mas primeiro vamos arrumar isso aqui...

FPFtech
Escola Tecnológica

Rode seu projeto e acesse:

http:localhost:8000/sales/



Vá para o seu MODELS.py e adicione:

```python
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}"
```

```python
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}"
```

```python
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}"
```

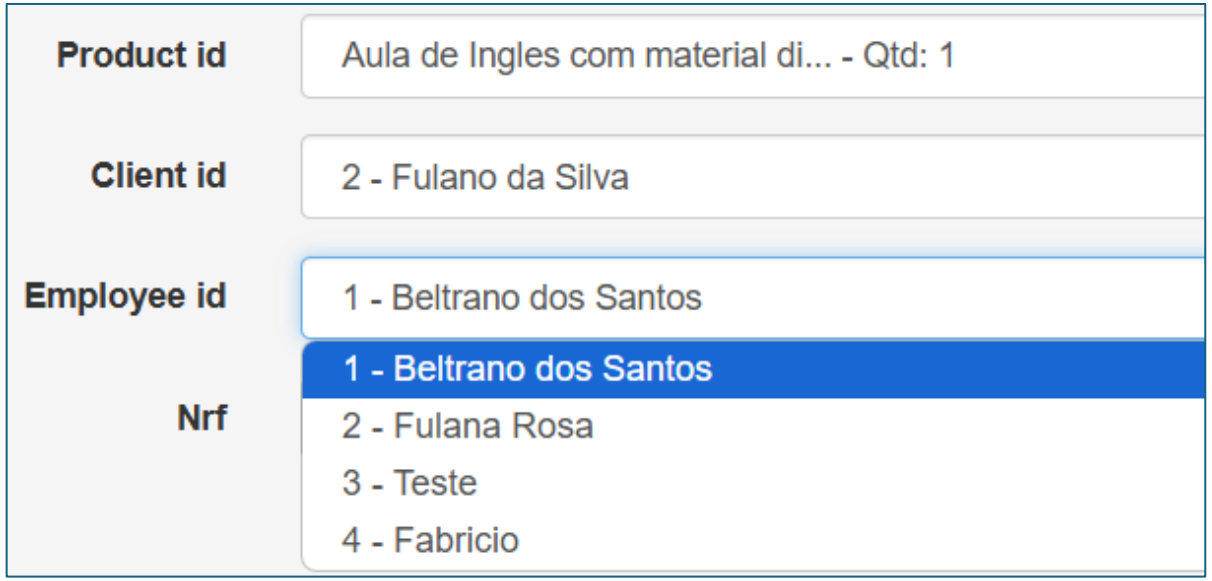
```python
class Sale(ModelBase):
    product = models.ForeignKey(
        Product,
        db_column='id_product',
        null=False,
        on_delete=models.DO_NOTHING
    )
    client = models.ForeignKey(
        Client,
        db_column='id_client',
        null=False,
        on_delete=models.DO_NOTHING
    )
    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}")
```

FPFtech
Escola Tecnológica

Depois de adicionar eles, pode rodar seu projeto:

Acesse http://localhost/sales/



Melhorando nosso projeto

FPFtech
Escola Tecnológica

Para melhorar nosso projeto, agora vamos adicionar:

Login (Permissões)

Paginação

Filtros

Login (Permissões)

FPFtech
Escola Tecnológica

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)

FPFtech
Escola Tecnológica

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

```
(.venv) PS C:\Users\jonatas.lopes\PycharmProjects\djangoProject1> py
Username (leave blank to use 'jonatas.lopes'): jon
Email address: jon@gmail.com
Password:
Password (again):
The password is too similar to the username.
This password is too short. It must contain at least 8 characters.
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
```

<- aqui eu digitei: jon (por questões de segurança, a senha não vai aparecer

<- aqui eu digitei novamente: jon

<- validação: nome e senha iguais.

<- validação: senha maior que 8, e senha comum.

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

```
Superuser created successfully.
```

Ver o usuário criado no postgres

FPFtech
Escola Tecnológica

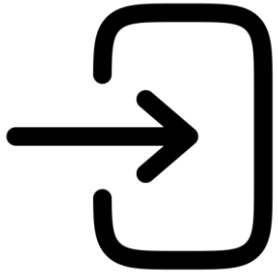
- escola 3
  - information_schema
  - pg_catalog
  - public
    - tables 14
      - auth_group
      - auth_group_permissions
      - auth_permission
      - auth_user

WHERE ORDER BY

	id	password	last_login	is_superuser	username	first_name	last_name	email	is_staff	is_act
1	1	pbkdf2_sha256$1000000$X3gM…	<null>	true	jon			jon@gmail.com	true	true

<- aqui a senha “jon” foi encriptada

viewsets



Depois de adicionar os amarelos...

Rode o Django novamente

```python
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]
```

FPFtech
Escola Tecnológica

Faça o Login com seu superuser

Django REST framework

Log in

Api Root / Client List

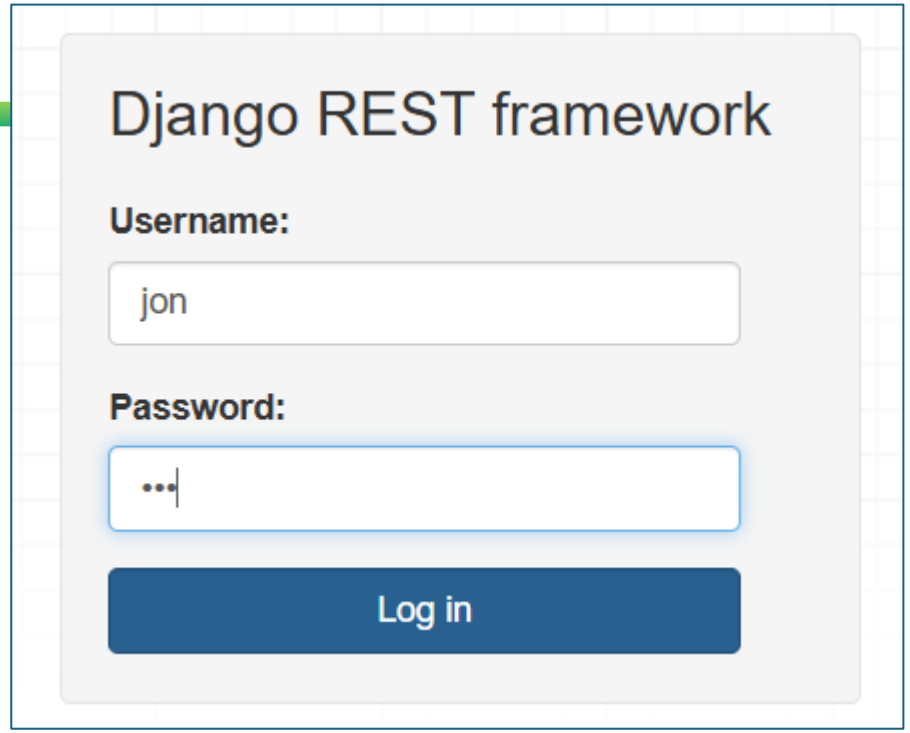
Client List

GET

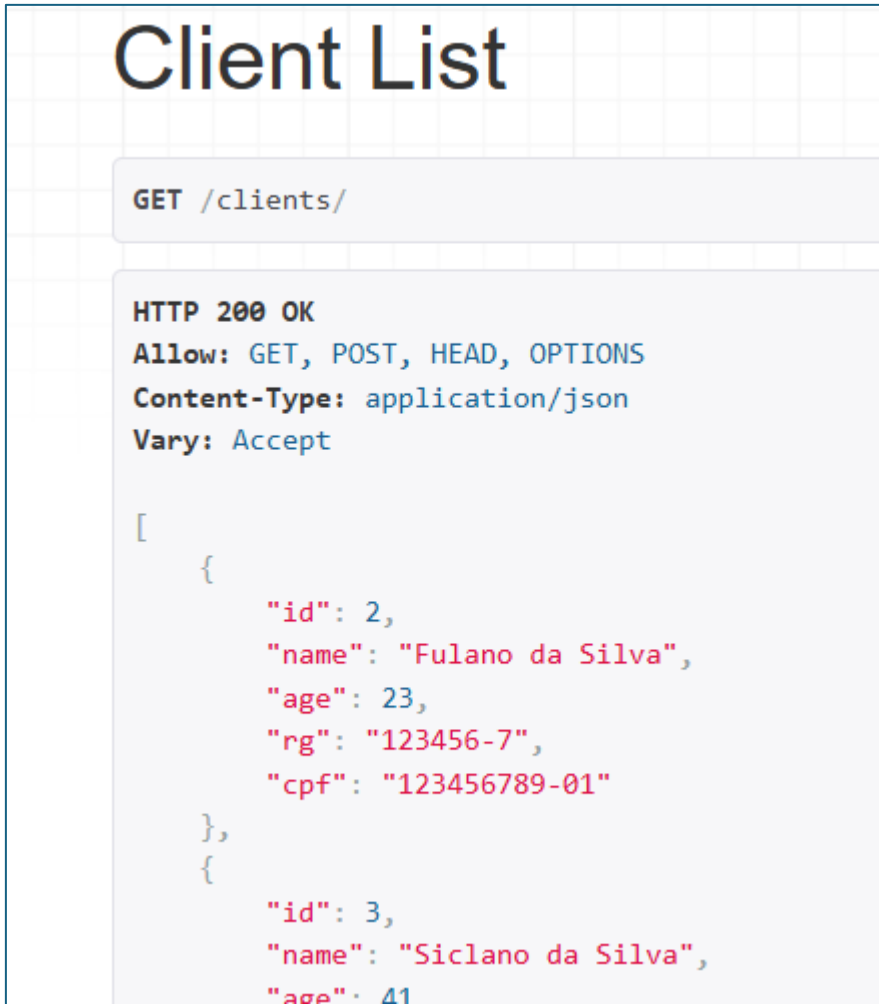
```
GET /clients/
```

```
HTTP 403 Forbidden
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

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



Agora sim...



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)

Melhorando nosso projeto

FPFtech
Escola Tecnológica

Para melhorar nosso projeto, agora vamos adicionar:

Login (Permissões)

- Create superuser
- Adicionar permissões nos viewsets

Paginação

Filtros

Paginação

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

- **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=

```
# https://docs.djangoproject.com/en/5.2/ref/settings/#default_auto_field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

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

- 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= 2 e Offset 1

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

Limit= 6 e Offset 3

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

Limit 4, Offset 5

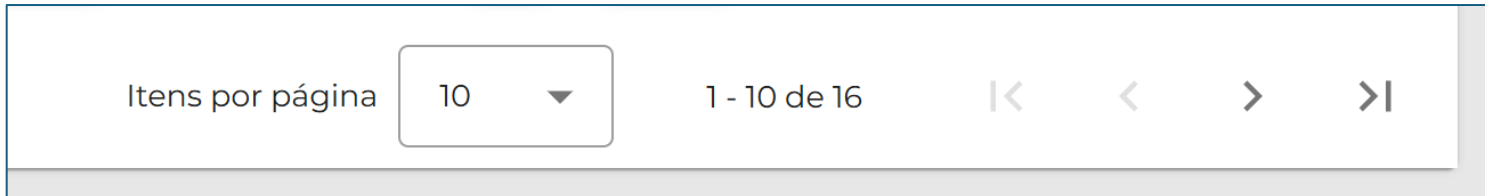
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

Limit 6, Offset 2

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

Limit 3, Offset 9

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



Rodando o proj.

FPFtech
Escola Tecnológica

```
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"
        }
    ]
}
```

Melhorando nosso projeto

FPFtech
Escola Tecnológica

Para melhorar nosso projeto, agora vamos adicionar:

Login (Permissões)

- Create superuser
- Adicionar permissões nos viewsets

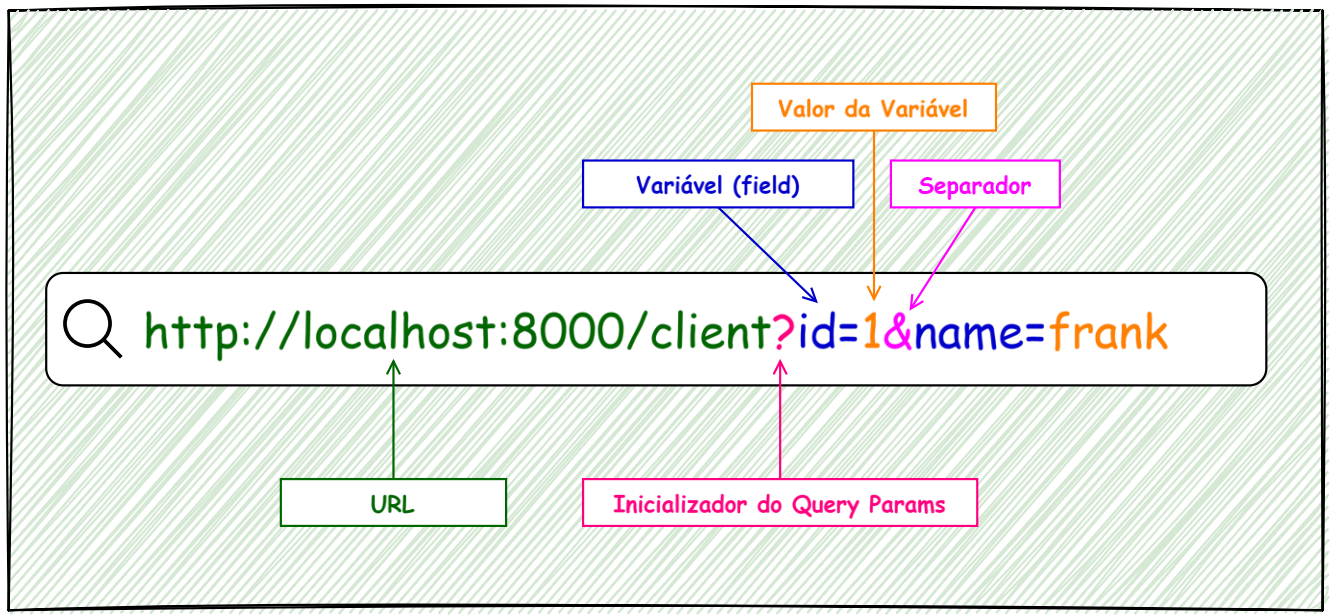
Paginação

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

Filtros

Filtro

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




```
roject1> pip install django-filter
```

```
ect1> pip freeze > requirements.txt
```

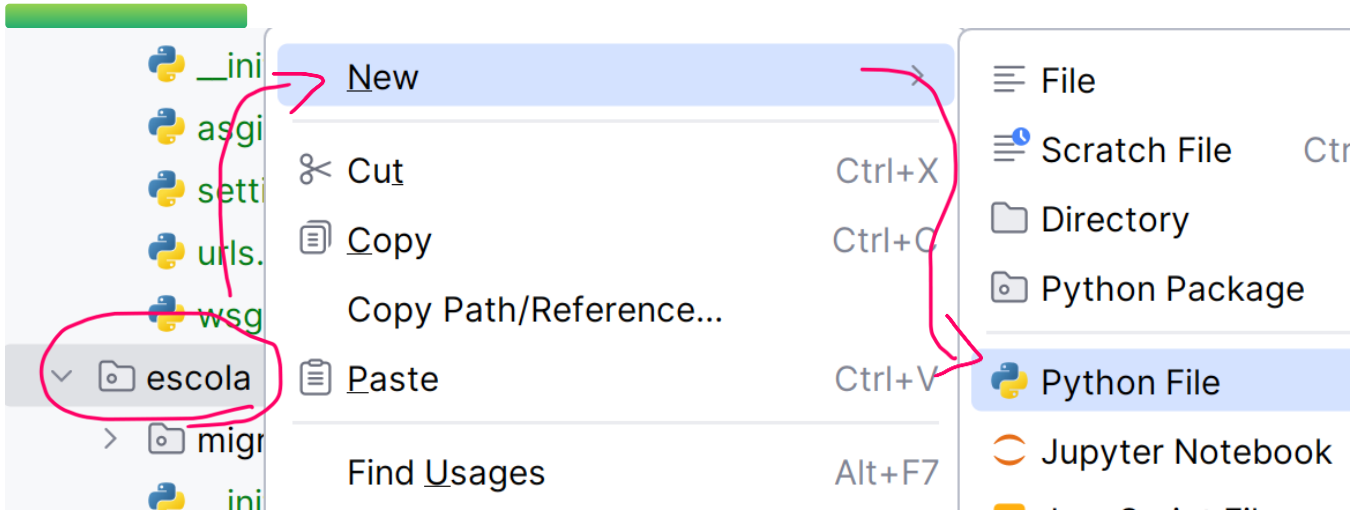
```
asgiref==3.8.1
Django==5.2.1
django-filter==25.1
djangorestframework==3.16.0
psycopg2-binary==2.9.10
sqlparse==0.5.3
tzdata==2025.2
```

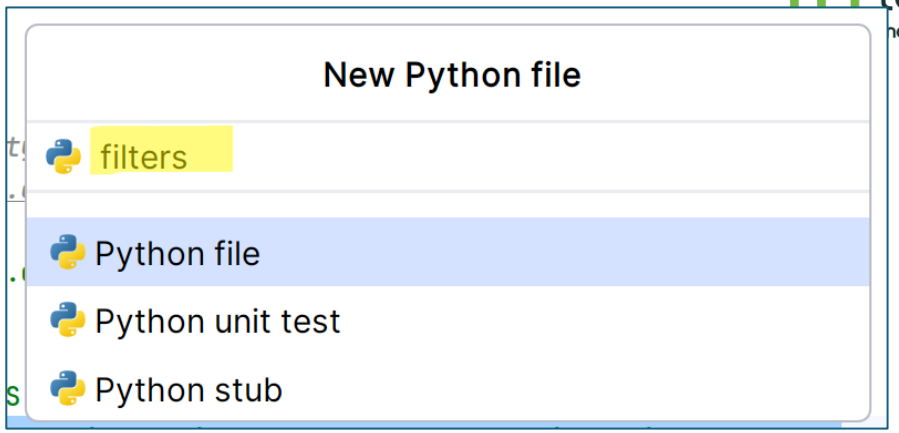
Filtro, Atualizando o settings.py

```python
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'escola.apps.EscolaConfig',
    'rest_framework',
    'django_filters'
]
```

```python
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






```python
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

```python
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

Api Root / Product List

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 Ingles 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)

FPFtech
Escola Tecnológica

LOOKUPS	SQL	DEFINIÇÃO
unaccent__icontains	WHERE unaccent(column_name) ILIKE '%value%'	(LIKE sem acentos e case-insensitive)
exact	WHERE column_name = 'value'	(igual a um valor específico)
in	WHERE column_name IN ('value1', 'value2', 'value3')	(verifica se o valor está em uma lista de valores)
lt	WHERE column_name < 'value'	(menor que)
gt	WHERE column_name > 'value'	(maior que)
lte	WHERE column_name <= 'value'	(menor ou igual a)
gte	WHERE column_name >= 'value'	(maior ou igual a)

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

Melhorando nosso projeto

FPFtech
Escola Tecnológica

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

FPFtech
Escola Tecnológica

Obrigado!

FPFtech
Escola Tecnológica

@fpftech.educacional