PYTHON ACADEMY

FILTROS, BUSCA E ORDENAÇÃO NO DJANGO REST FRAMEWORK

Esse ebook mostra como estender as funcionalidades de sua API, adicionando Filtros, Busca Textual e Ordenação às APIs desenvolvidas com o poderoso Django REST Framework

PYTHONACADEMY.COM.BR

Crie ebooks profissionais incríveis em minutos com IA

Chega de formatar texto no Google Docs, Word ou ferramentas que só te fazem perder tempo...

E deixe que nossa IA faça o trabalho pesado!

- Capas gerados por IA
- Infográficos feitos por IA
- Edite em Markdown em Tempo Real
- Adicione Banners Promocionais

TESTE AGORA

PRIMEIRO CAPÍTULO 100% GRÁTIS

Faaaaala **DEV**!

Estamos juntos para mais um passeio no parque do nosso querido **Django REST Framework**, ou **DRF**!

Vamos mostrar nesse post como podemos estender nossas APIs para adicionar as funcionalidades de Filtro, Busca Textual e Ordenação.

Essas funcionalidades são muito importantes e veremos aqui como são simples de implementar no DRF.

Ahhh, caso não tenha visto nosso post sobre DRF: 👇

> *Esse post faz parte da construção de APIs de Fundos Imobiliários iniciada no nosso post sobre Django REST Framework! Para entender as referências e configurações do projeto, sugiro começar por ele!*

Agora que está craque em DRF, **#VamosNessa**! 👊

Introdução

O DRF nos possibilita adicionar as funcionalidades de ordenação de resultados, busca textual e filtragem de dados de maneira **extremamente simples**!

Para nos auxiliar, vamos começar instalando o pacote `django-filter`.

Na raíz do seu projeto, com o ambiente virtual instalado, execute:

```
pip install django-filter
```

Em seguida, adicione `django_filters` à variável `INSTALLED_APPS` do arquivo `settings.py` do projeto.

Feito isso, podemos começar!

Filtro

Suponha que seja necessário adicionar filtros à nossa API para que os usuários possam fazer, por exemplo:

```
http://localhost:8000/api/v1/fundos?setor=hibrido
```

E nossa API retorne apenas os Fundos em que `setor=hibrido`.

Para utilizar filtros em nossa API, adicionamos os atributos `filter_backends` e `filterset_fields`:

- Em `filter_backends` colocamos o *Backend* que irá processar os filtros;
- Em `filterset_fields` adicionamos quais campos queremos expôr para filtragem.

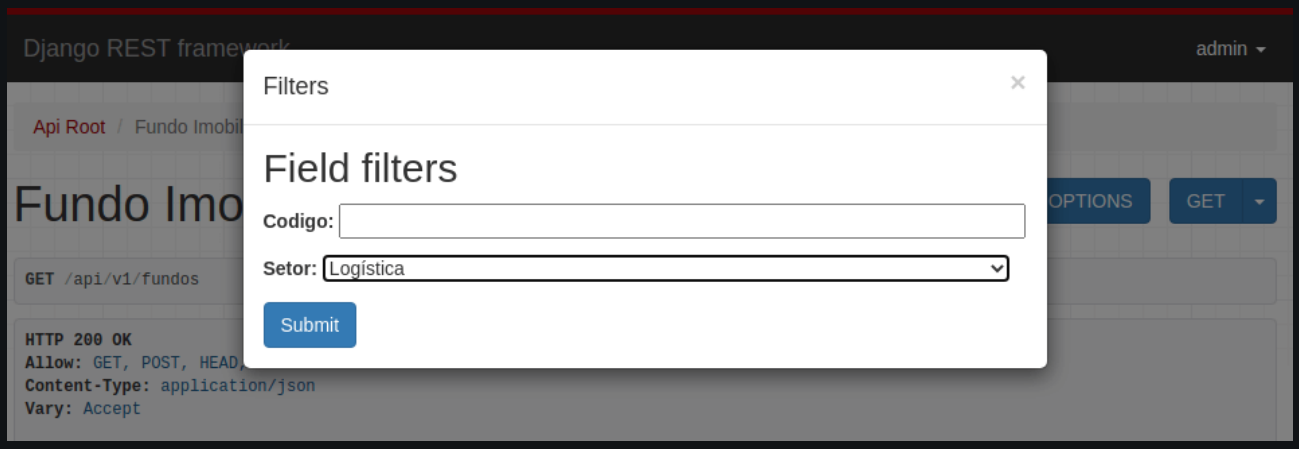
Com isso, nossa `ViewSet` fica assim:

```python
from django_filters.rest_framework import DjangoFilterBackend
from api.serializers import FundoImobiliarioSerializer
from rest_framework import viewsets, permissions
from api.models import FundoImobiliario

class FundoImobiliarioViewSet(viewsets.ModelViewSet):
  queryset = FundoImobiliario.objects.all()
  serializer_class = FundoImobiliarioSerializer
  permission_classes = [permissions.IsAuthenticated]
  filter_backends = [DjangoFilterBackend]
  filterset_fields = ['codigo', 'setor']
```

Dessa forma podemos filtrar Fundos Imobiliários pelo `código` e `setor`!

Agora, abra a interface navegável e veja que o DRF adicionou um botão *"Filters"*: nele podemos testar os filtros!



Teste a URL `http://localhost:8000/api/v1/fundos?setor=hibrido` com o Postman e veja que **show**!

Busca Textual

A busca textual serve para adicionar a funcionalidade de realizar buscas dentro de determinados valores de texto armazenados na base de dados.

Contudo a busca só funciona para campos de texto, como `CharField` e `TextField`.

Para utilizar a busca textual, devemos promover duas alterações em nossa `ViewSet`:

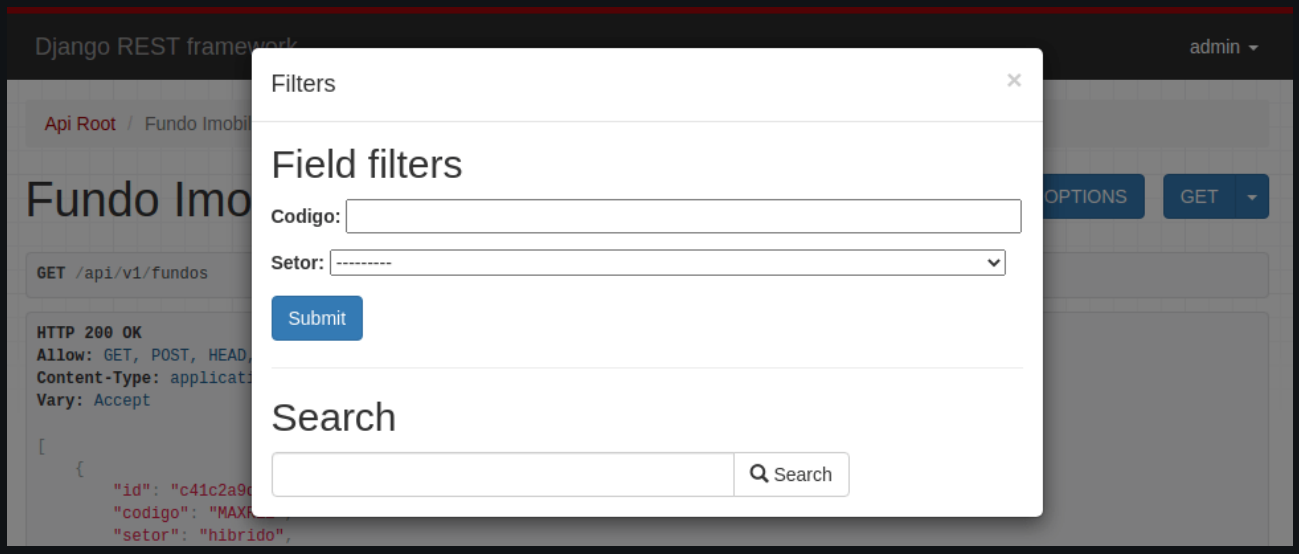
- Novamente alterar o atributo `filter_backends`, adicionando o *Backend* `SearchFilter` que irá processar a busca; e
- Adicionar o atributo `search_fields`, contendo os campos que permitirão a busca.

Assim:

```
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework.filters import SearchFilter
from api.serializers import FundoImobiliarioSerializer
from rest_framework import viewsets, permissions
from api.models import FundoImobiliario

class FundoImobiliarioViewSet(viewsets.ModelViewSet):
  queryset = FundoImobiliario.objects.all()
  serializer_class = FundoImobiliarioSerializer
  permission_classes = [permissions.IsAuthenticated]
  filter_backends = [DjangoFilterBackend, SearchFilter]
  filterset_fields = ['codigo', 'setor']
  search_fields = ['codigo', 'setor']
```

Na interface navegável, clique novamente em *Filters* e veja que foi adicionada o campo de busca ***Search***!



Nele você pode realizar pode buscar por valores contidos em `codigo` (ex: ALZR11, HGLG11, etc) e `setor` (ex: hibrido, hospital, residencial, etc).

DEMAIS!

Ordenação

E por último, e não menos importante: a **Ordenação**.

Essa funcionalidade serve para que possamos definir a ordem que os resultados devem ser apresentados.

A funcionalidade de Ordenação é configurada de forma semelhante aos Filtros e à Busca Textual. Assim:

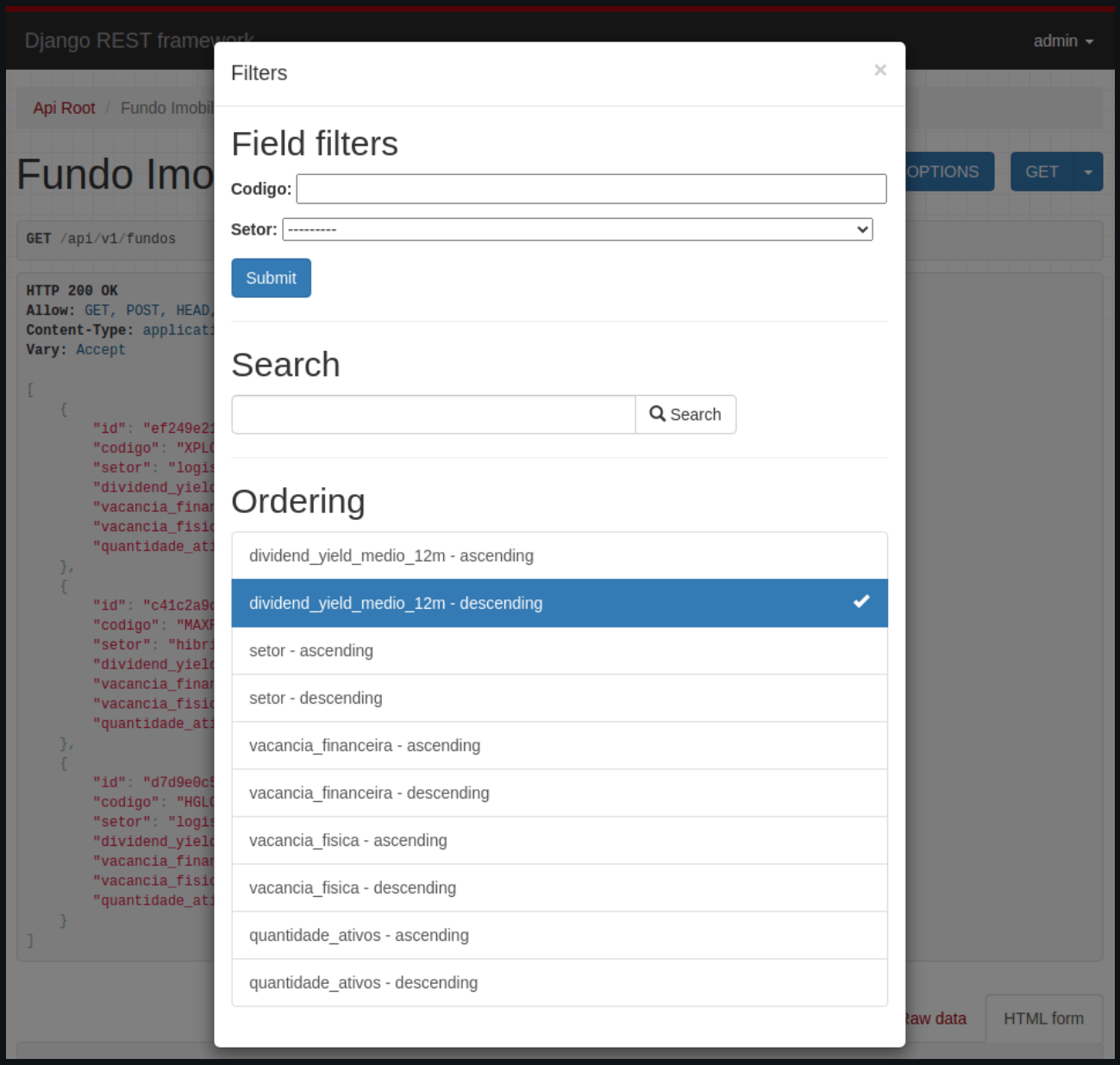
- Adicionar o *Backend* `OrderingFilter` ao atributo `filter_backends`;
- Adicionar o atributo `ordering_fields`, que são quais campos poderão ser ordenados
- Adicionar opcionalmente o atributo `ordering` que configura a ordenação que será aplicada por padrão ao chamar endpoints que retornem mais de um resultado.

Portanto, nossa versão final da nossa `ViewSet` de Fundos Imobiliários deverá ficar assim:

```
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework.filters import SearchFilter, OrderingFilter
from api.serializers import FundoImobiliarioSerializer
from rest_framework import viewsets, permissions
from api.models import FundoImobiliario

class FundoImobiliarioViewSet(viewsets.ModelViewSet):
    queryset = FundoImobiliario.objects.all()
    serializer_class = FundoImobiliarioSerializer
    permission_classes = [permissions.IsAuthenticated]
    filter_backends = [DjangoFilterBackend, SearchFilter,
        OrderingFilter]
    filterset_fields = ['codigo', 'setor']
    search_fields = ['codigo', 'setor']
    ordering = ['-dividend_yield_medio_12m']
    ordering_fields = [
        'dividend_yield_medio_12m',
        'setor',
        'vacancia_financeira',
        'vacancia_fisica',
        'quantidade_ativos']
```

Veja que a Interface Navegável já foi atualizada e adicionou a seção *Ordering*, com os campos configurados:



✨ *WOW!!!* ✨

> 💡 *Criei o **Ebookr.ai**, uma plataforma que usa IA para gerar ebooks profissionais sobre qualquer tema — com capa gerada por IA, infográficos automáticos e exportação em PDF. Confere!*

Conclusão

Como sempre, o Django sendo o **MELHOR** *framework* para Desenvolvimento Web!

Vimos nesse post como é simples estender e adicionar as funcionalidades de Filtros, Busca Textual e Ordenação à nossa API construída com o DRF!

> *E você, o que achou? Curte o DRF? Tem sugestão de conteúdo?*

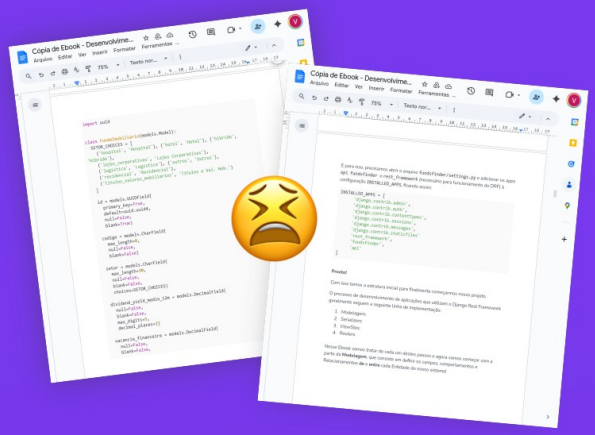
Conta aqui embaixo pra gente! 😉

Até mais!!!

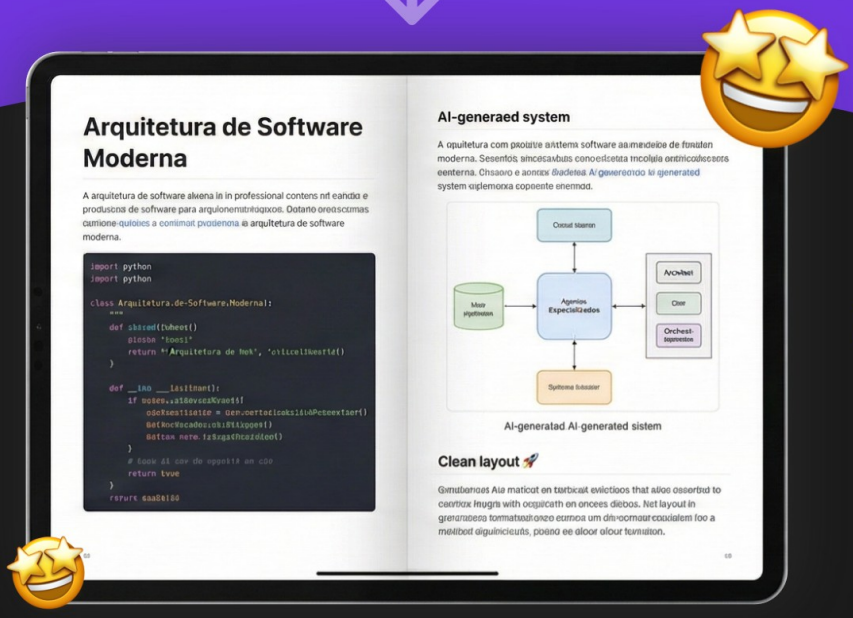
Não se esqueça de conferir!

Ebookr

Crie Ebooks profissionais em minutos com IA



Chega de formatar código no Google Docs ou Word



Capas gerados por IA

Infográficos feitos por IA

Adicione Banners Promocionais

Deixe que nossa IA faça o trabalho pesado

Edite em Markdown em Tempo Real

TESTE AGORA

PRIMEIRO CAPÍTULO 100% GRÁTIS