Trilha 7 - Desenvolvimento Back-End (Python)

> Exercício Prático 2: Class-Based Views, Paginação, Ordenação e Busca

Contexto

Você está desenvolvendo um sistema de gerenciamento de biblioteca que permite a administração de livros, autores e categorias. Neste exercício, você vai evoluir o exercício anterior, reimplementando as funcionalidades utilizando class-based views em vez de function-based views. Além disso, serão adicionadas novas funcionalidades, como recursos de paginação de resultados, ordenação e busca de termos utilizando a biblioteca django-filter.

Objetivo

1. Reimplementar as views utilizando class-based views.
2. Adicionar recursos de paginação de resultados, ordenação e busca de termos.
3. Criar um repositório público no GitHub para a submissão do exercício.

Passo 1: Reimplementação das Views

1. No arquivo `views.py` do aplicativo `core`, reimplemente as views utilizando class-based views em vez de function-based views.

Serão fornecidos exemplos para o modelo `Livro`, os exemplos devem ser replicados também para `Categoria` e `Autor`.

```
from rest_framework import generics
from .models import Livro
from .serializers import LivroSerializer

class LivroList(generics.ListCreateAPIView):
    queryset = Livro.objects.all()
    serializer_class = LivroSerializer
    name = "livro-list"

class LivroDetail(generics.RetrieveUpdateDestroyAPIView):
    queryset = Livro.objects.all()
    serializer_class = LivroSerializer
    name = "livro-detail"
```

> Atenção: propague as mudanças, com as devidas adequações, para as views das demais classes de modelo.

Passo 2: Adição de Recursos de Paginação, Ordenação e Busca

1. Instale a biblioteca django-filter:

```
pip install django-filter
```

2. Configure a biblioteca django-filter no projeto.
    1. Incluir a biblioteca em `INSTALLED_APPS`.
    2. Realizar configuração global dos filtros no projeto. Adicionar a configuração `DEFAULT_FILTER_BACKENDS`, incluindo `DjangoFilterBackend`, `OrderingFilter` e `SearchFilter`.
3. Realize a configuração global de paginação no projeto.
    1. Configurar `DEFAULT_PAGINATION_CLASS` para trabalhar com `LimitOffsetPagination`.
    2. Configurar `PAGE_SIZE` para o tamanho `5`.
4. Crie um filtro personalizado, importe a classe `FilterSet` do django-filter e crie filtros para a classe `Livro` de acordo com código abaixo:

```
from django_filters import rest_framework as filters
from .models import Livro

class LivroFilter(filters.FilterSet):
    titulo = filters.CharFilter(lookup_expr='icontains')
    autor = filters.CharFilter(field_name='autor__nome',
lookup_expr='icontains')
    categoria = filters.AllValuesFilter(field_name='categoria__nome')

    class Meta:
        model = Livro
        fields = ['titulo', 'autor', 'categoria']
```

4. Atualize a view para utilizar o filtro criado:

```
class LivroList(generics.ListCreateAPIView):
    queryset = Livro.objects.all()
    serializer_class = LivroSerializer
    filterset_class = LivroFilter
```

> Atenção: Implemente filtros de busca para o atributo `nome`, utilizando o prefixo `^ (inicia com)`, para `Categoria` e `Livro`.

5. Adicione recursos de ordenação nas classes de views:

A view `LivroList` deve permitir a ordenação por `titulo`, `autor`, `categoria`, `publicado_em`.

```
ordering_fields = ['titulo', 'autor', 'categoria', 'publicado_em']
```

> Atenção: Possibilite a ordenação a partir do atributo `nome` para `Categoria` e `Livro`.

Passo 3: Configuração das URLs

1. Atualize as rotas no `urls.py` do aplicativo `core` para utilizar as novas classes de views:

```
from django.urls import path
from . import views

urlpatterns = [
    path('livros/', views.LivroList.as_view(), name='livros-list'),
    path('livros/<int:pk>/', views.LivroDetail.as_view(), name='livro-
detail'),
]
```

> Atenção: propague as mudanças, com as devidas adequações, para as views das demais classes de modelo.

Passo 4: Produção de dados de teste

Dentro da aplicação `core` crie as pastas `management/commands`. A partir da pasta `commands`, você deve criar dois novos arquivos: `___init__.py` e `populate_db.py`.

O arquivo `__init__.py` em um diretório de um projeto Python tem a função principal de indicar que o diretório deve ser tratado como um pacote. Isso permite que os módulos e subpacotes dentro desse diretório sejam importados.

Adicione o seguinte conteúdo ao arquivo `populate_db.py`:

```
from django.core.management.base import BaseCommand
from core.models import Categoria, Autor, Livro


class Command(BaseCommand):
    help = "Cria registros de exemplo no banco de dados"

    def handle(self, *args, **options):
        categoria_misterio = Categoria.objects.create(nome="Mistério")
        categoria_ficcao = Categoria.objects.create(nome="Ficção")
        categoria_fantasia = Categoria.objects.create(nome="Fantasia")
        categoria_romance = Categoria.objects.create(nome="Romance")

        autor_agatha_christie = Autor.objects.create(nome="Agatha
Christie")
        autor_arthur_c_clarke = Autor.objects.create(nome="Arthur C.
Clarke")
        autor_arthur_conan_doyle = Autor.objects.create(nome="Arthur Conan
Doyle")
        autor_cs_lewis = Autor.objects.create(nome="C.S. Lewis")
        autor_emily_bronte = Autor.objects.create(nome="Emily Brontë")
```

```
        autor_george_rr_martin = Autor.objects.create(nome="George R.R.
Martin")
        autor_isaac_asimov = Autor.objects.create(nome="Isaac Asimov")
        autor_jrr_tolkien = Autor.objects.create(nome="J.R.R. Tolkien")

        Livro.objects.create(
            titulo="Assassinato no Expresso do Oriente",
            autor=autor_agatha_christie,
            categoria=categoria_misterio,
            publicado_em="1934-01-01",
        )
        Livro.objects.create(
            titulo="Morte no Nilo",
            autor=autor_agatha_christie,
            categoria=categoria_misterio,
            publicado_em="1937-11-01",
        )
        Livro.objects.create(
            titulo="2001: Uma Odisseia no Espaço",
            autor=autor_arthur_c_clarke,
            categoria=categoria_ficcao,
            publicado_em="1968-06-16",
        )
        Livro.objects.create(
            titulo="Encontro com Rama",
            autor=autor_arthur_c_clarke,
            categoria=categoria_ficcao,
            publicado_em="1973-06-01",
        )
        Livro.objects.create(
            titulo="O Cão dos Baskervilles",
            autor=autor_arthur_conan_doyle,
            categoria=categoria_misterio,
            publicado_em="1902-04-01",
        )
        Livro.objects.create(
            titulo="Um Estudo em Vermelho",
            autor=autor_arthur_conan_doyle,
            categoria=categoria_misterio,
            publicado_em="1887-11-01",
        )
        Livro.objects.create(
            titulo="As Crônicas de Nárnia",
            autor=autor_cs_lewis,
            categoria=categoria_fantasia,
            publicado_em="1950-10-16",
        )
        Livro.objects.create(
            titulo="O Leão, a Feiticeira e o Guarda-Roupa",
            autor=autor_cs_lewis,
            categoria=categoria_fantasia,
            publicado_em="1950-10-16",
        )
        Livro.objects.create(
```

```
            titulo="O Morro dos Ventos Uivantes",
            autor=autor_emily_bronte,
            categoria=categoria_romance,
            publicado_em="1847-12-01",
        )
        Livro.objects.create(
            titulo="A Guerra dos Tronos",
            autor=autor_george_rr_martin,
            categoria=categoria_fantasia,
            publicado_em="1996-08-06",
        )
        Livro.objects.create(
            titulo="A Fúria dos Reis",
            autor=autor_george_rr_martin,
            categoria=categoria_fantasia,
            publicado_em="1998-11-16",
        )
        Livro.objects.create(
            titulo="Fundação",
            autor=autor_isaac_asimov,
            categoria=categoria_ficcao,
            publicado_em="1951-06-01",
        )
        Livro.objects.create(
            titulo="Eu, Robô",
            autor=autor_isaac_asimov,
            categoria=categoria_ficcao,
            publicado_em="1950-12-02",
        )
        Livro.objects.create(
            titulo="O Senhor dos Anéis",
            autor=autor_jrr_tolkien,
            categoria=categoria_fantasia,
            publicado_em="1954-07-29",
        )
        Livro.objects.create(
            titulo="O Hobbit",
            autor=autor_jrr_tolkien,
            categoria=categoria_fantasia,
            publicado_em="1937-09-21",
        )
```

O código Python acima cria registros de exemplo em um banco de dados usando o Django e o recurso Command.

Dentro do método handle, o código cria várias instâncias das classes Categoria, Autor e Livro, preenchendo-os com dados fictícios.

Assumindo que o banco de dados SQLite da aplicação já foi criado seguindo as orientações do exercício anterior, aplique o comando abaixo para a inserção dos registros para testes:

```
python manage.py populate_db
```

> Atenção: a execução do comando deve ser realizada apenas uma vez, caso contrário os registros serão inseridos em duplicidade no banco de dados.

Passo 5: Teste da API

1. Inicie o servidor Django:

   ```
   python manage.py runserver
   ```

2. Utilize o Postman para testar as seguintes operações na API, utilizando os recursos de paginação, ordenação e busca:

   - **Listar todos os livros com paginação (GET `/livros/?limit=<limite>&offset=<deslocamento>`)**
   - **Ordenar os livros por título (GET `/livros/?ordering=titulo`)**
   - **Buscar livros por título, autor ou categoria (GET `/livros/?titulo=<termo_busca>&autor=<termo_busca>&categoria=<termo_busca>`)**
   - **Criar um novo livro (POST `/livros/create/`)**
   - **Obter detalhes de um livro específico (GET `/livros/<id>/`)**
   - **Atualizar um livro (PUT `/livros/<id>/`)**
   - **Deletar um livro (DELETE `/livros/<id>/`)**

> Atenção: realize testes também nas funcionalidades desenvolvidas paras as demais classes de modelo.

Passo 6: Criação do arquivo requirements.txt

A boa prática de gerar o arquivo requirements.txt em projetos Python visa garantir que todas as dependências necessárias para o projeto sejam claramente listadas com suas versões exatas. Isso facilita a reprodução do ambiente por outros desenvolvedores ou sistemas automatizados, garantindo que o projeto funcione com as mesmas bibliotecas e versões utilizadas no desenvolvimento.

A partir do diretório raiz, utilize o comando pip freeze para gerar o arquivo requirements.txt:

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

> Após o comando, o novo arquivo será gerado e deve ser vesionado junto ao código projeto.

Passo 7: Criação de Repositório Público no GitHub

1. **Crie um repositório público na sua conta pessoal do GitHub**:
   - Acesse GitHub e faça login.
   - Clique em "New" (para criar um novo repositório).

- Dê um nome ao repositório, por exemplo, `biblioteca-django-v2`.
- Marque a opção "Public" e clique em "Create repository".

> Você pode também optar por utilizar o repositório criado no exercício anterior. Se for essa a sua opção, desconsidere as instruções 2 e 3.

2. **Inclua o arquivo `.gitignore` no projeto**:

   - Crie um arquivo chamado `.gitignore` na raiz do projeto.
   - Adicione as seguintes linhas ao arquivo `.gitignore`:

   ```
   # Byte-compiled
   __pycache__/
   *.py[cod]
   *$py.class

   # virtualenv
   venv/
   .venv/

   # Django:
   *.log
   *.pot
   db.sqlite3
   media/

   # IDEs
   .idea/
   .vscode/
   ```

   O arquivo `.gitignore` é usado para especificar quais arquivos e diretórios devem ser ignorados pelo Git. É importante incluí-lo no seu projeto para evitar que arquivos sensíveis ou desnecessários sejam adicionados aos commits e enviados para o repositório. Não é considerado uma boa prática versionar arquivos binários.

3. **Adicione o repositório remoto ao projeto local**:

   - No terminal, na raiz do projeto Django, inicialize o repositório git:

   ```
   git init
   ```

   - Adicione todos os arquivos ao commit:

   ```
   git add .
   ```

   - Faça um commit:

```
git commit -m "Initial commit"
```

- Adicione o repositório remoto:

```
git remote add origin https://github.com/username/biblioteca-django-v2.git
```

(Substitua `username` pelo seu nome de usuário no GitHub)

4. **Faça push dos arquivos para o repositório remoto**:

```
git push -u origin main
```

5. **Envie o link do repositório como solução da atividade**:
   - Copie a URL do repositório (por exemplo, `https://github.com/username/biblioteca-django-v2`).
   - Acesse a área de atividades da trilha e cole o link para a submissão.