Trilha 7 - Desenvolvimento Back-End (Python)

> Exercício Prático 3: Exposição Virtual de Coleções, Autenticação e Testes Automatizados

> Esta prática é uma continuação da atividade desenvolvida na `Unidade 6` da trilha.

Contexto

A biblioteca está expandindo seus serviços para incluir uma exposição virtual de coleções de livros. Essa nova funcionalidade permite que os usuários criem e gerenciem suas próprias coleções, apresentando suas obras favoritas em um formato compartilhável. Somente o colecionador, ou seja, o usuário que cadastrou a coleção, poderá gerenciar os livros de sua coleção, mantendo o controle sobre as edições e exclusões.

Objetivo

1. Implementar um modelo de coleção de livros associado a um usuário (colecionador).
2. Adicionar autenticação baseada em Token e permissões para garantir que apenas o colecionador possa gerenciar sua coleção.
3. Documentar a API com drf-spectacular.
4. Desenvolver testes automatizados para a funcionalidade de coleções.

Passo 1: Implementação da Funcionalidade de Coleção de Livros com Colecionador

1. Definição do Modelo de Coleção.

Modifique ou crie o modelo `Colecao` em `models.py` para incluir um relacionamento com a classe `User` do Django:

```
from django.db import models
from django.contrib.auth.models import User
from .models import Livro

class Colecao(models.Model):
    nome = models.CharField(max_length=100, unique=True)
    descricao = models.TextField(blank=True)
    livros = models.ManyToManyField(Livro, related_name="colecoes")
    colecionador = models.ForeignKey(User, on_delete=models.CASCADE,
related_name="colecoes")

    def __str__(self):
        return f"{self.nome} - {self.colecionador.username}"
```

2. Crie e aplique as migrações para refletir as alterações no banco de dados.

3. Crie um serializador chamado `ColecaoSerializer` e lembre-se de incluir todos os campos do modelo `Colecao`.

4. Implemente *views* que possibilitem a listagem, recuperação, cadastro, edição e exclusão de coleções. Assegure que apenas o colecionador possa modificar dados das suas coleções. Os demais usuários devem podem visualizar e listar coleções de terceiros.

   Sugestão de nomes:

   - ColecaoListCreate
   - ColecaoDetail

5. Crie um arquivo chamado `custom_permissions.py` e defina a permissão para que apenas o colecionador possa modificar sua coleção.

Passo 2: Autenticação e Permissões

- Implemente autenticação baseada em Token utilizando o mecanismo de token padrão do DRF. Realize a migração e garanta que somente usuários autenticados possam acessar e gerenciar coleções.

Passo 3: Documentação da API

- Utilize a biblioteca drf-spectacular para documentar a API, configurando os endpoints para o schema e a documentação da API no arquivo `urls.py`.

Passo 4: Desenvolvimento de Testes Automatizados

Desenvolva testes automatizados para a funcionalidade de coleções de livros. Assegure que os testes verifiquem:

1. Criação de uma nova coleção e associação correta ao usuário autenticado;
2. Permissões de acesso:
   - Apenas o colecionador pode editar ou deletar sua coleção;
   - Usuários não autenticados não podem criar, atualizar ou deletar coleções
3. Listagem de coleções visíveis para usuários autenticados

Passo 5: Cobertura de testes

- Para garantir a efetividade dos testes desenvolvidos e ter uma visão clara da cobertura de código, você deve utilizar a ferramenta coverage.py. Se certifique que os testes desenvolvidos cobrem todas as funcionalidades de coleções.

- Não é recomendado o versionamento do report gerado pelo coverage.py, como a pasta `htmlcov`. No entanto, **para facilitar o processo de correção, a pasta deve ser versionada junto ao projeto**.

Passo 6: Configuração de CORS

Para permitir que clientes externos consumam a API, é necessário configurar o *Cross-Origin Resource Sharing* (CORS). Isso pode ser feito utilizando a biblioteca `django-cors-headers`. Essa biblioteca ajuda a controlar quais origens têm permissão para acessar recursos da API em seu projeto Django.

1. Instale a biblioteca

2. As instruções de configuração podem ser obtidas em: https://github.com/adamchainz/django-cors-headers
3. Adicione as configurações de CORS para permitir o acesso de um IP hipotético que fará uso dos seus serviços:
    - IP: 192.168.1.100

Passo 7: 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 8: Criação/Atualização de Repositório Público no GitHub

> Você pode optar por utilizar o repositório criado no exercício anterior. Se for essa a sua opção, desconsidere as instruções 1, 2 e 3.

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`.
    - Marque a opção "Public" e clique em "Create repository".
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.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`).
   - Acesse a área de atividades da trilha e cole o link para a submissão.