Trilha 7 - Desenvolvimento Back-End (Python)

> Exercício Prático: ORM Django, Serializadores e Function-Based Views

Contexto

Você está desenvolvendo um sistema de gerenciamento de biblioteca que permite a administração de livros, autores e categorias. Neste exercício, você vai praticar o uso do Django ORM para realizar operações de CRUD, além de criar serializers manualmente usando a classe Serializer. As views serão implementadas usando function-based views (FBVs).

Objetivo

1. Criar modelos para **Livros**, **Autores** e **Categorias**.
2. Praticar operações de CRUD usando o Django ORM no shell.
3. Criar serializers manuais com a classe `Serializer`.
4. Implementar function-based views para uma API RESTful.
5. Criar um repositório público no GitHub para a submissão do exercício.

Passo 1: Criação do Projeto

1. Crie um novo projeto Django:

```
django-admin startproject biblioteca
cd biblioteca
```

2. Crie um novo aplicativo dentro do projeto:

```
python manage.py startapp core
```

3. Adicione o aplicativo `core` ao `INSTALLED_APPS` no arquivo `settings.py`.

4. Verifique a configuração do banco de dados SQLite (padrão) no `settings.py`. Modifique o nome do banco para `bib.sqlite3`.

Passo 2: Criação dos Modelos

1. No arquivo `models.py` do aplicativo `core`, crie os seguintes modelos:

```
from django.db import models

class Categoria(models.Model):
    nome = models.CharField(max_length=100)
```

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

class Autor(models.Model):
    nome = models.CharField(max_length=100)

    def __str__(self):
        return self.nome

class Livro(models.Model):
    titulo = models.CharField(max_length=200)
    autor = models.ForeignKey(Autor, on_delete=models.CASCADE)
    categoria = models.ForeignKey(Categoria, on_delete=models.CASCADE)
    publicado_em = models.DateField()

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

2. Crie as migrações e aplique-as:

```
python manage.py makemigrations
python manage.py migrate
```

Passo 3: Praticando com o Django Shell

1. Abra o shell do Django:

```
python manage.py shell
```

2. **Inserção de dados**: crie instâncias para `Categoria`, `Autor` e `Livro`.

```
from core.models import Categoria, Autor, Livro
from datetime import date

# Criando categorias
ficcao = Categoria.objects.create(nome='Ficção')
ciencia = Categoria.objects.create(nome='Ciência')

# Criando autores
asimov = Autor.objects.create(nome='Isaac Asimov')
sagan = Autor.objects.create(nome='Carl Sagan')

# Criando livros
Livro.objects.create(titulo='Fundação', autor=asimov,
categoria=ficcao, publicado_em=date(1951, 6, 1))
Livro.objects.create(titulo='Cosmos', autor=sagan, categoria=ciencia,
publicado_em=date(1980, 10, 1))
```

3. **Consulta de dados**: liste todos os livros de uma determinada categoria.

```
ficcao_livros = Livro.objects.filter(categoria__nome='Ficção')
for livro in ficcao_livros:
    print(livro.titulo)
```

4. **Atualização de dados**: atualize o título de um livro.

```
fundacao = Livro.objects.get(titulo='Fundação')
fundacao.titulo = 'Fundação - Edição Revisada'
fundacao.save()
```

5. **Exclusão de dados**: exclua um livro do banco de dados.

```
cosmos = Livro.objects.get(titulo='Cosmos')
cosmos.delete()
```

Passo 4: Criação dos serializadores

1. Crie um arquivo `serializers.py` no aplicativo `core` e defina os serializers usando a classe `Serializer`:

```
from rest_framework import serializers
from .models import Categoria, Autor, Livro

class CategoriaSerializer(serializers.Serializer):
    id = serializers.IntegerField(read_only=True)
    nome = serializers.CharField(max_length=100)

    def create(self, validated_data):
        return Categoria.objects.create(**validated_data)

    def update(self, instance, validated_data):
        instance.nome = validated_data.get('nome', instance.nome)
        instance.save()
        return instance

class AutorSerializer(serializers.Serializer):
    id = serializers.IntegerField(read_only=True)
    nome = serializers.CharField(max_length=100)

    def create(self, validated_data):
        return Autor.objects.create(**validated_data)
```

```
    def update(self, instance, validated_data):
        instance.nome = validated_data.get('nome', instance.nome)
        instance.save()
        return instance

class LivroSerializer(serializers.Serializer):
    id = serializers.IntegerField(read_only=True)
    titulo = serializers.CharField(max_length=200)
    autor =
serializers.PrimaryKeyRelatedField(queryset=Autor.objects.all())
    categoria =
serializers.PrimaryKeyRelatedField(queryset=Categoria.objects.all())
    publicado_em = serializers.DateField()

    def create(self, validated_data):
        return Livro.objects.create(**validated_data)

    def update(self, instance, validated_data):
        instance.titulo = validated_data.get('titulo', instance.titulo)
        instance.autor = validated_data.get('autor', instance.autor)
        instance.categoria = validated_data.get('categoria',
instance.categoria)
        instance.publicado_em = validated_data.get('publicado_em',
instance.publicado_em)
        instance.save()
        return instance
```

Passo 5: Criação das Views Baseadas em Funções (FBVs)

1. Crie views para listar, criar, atualizar e deletar livros, utilizando os serializers:

```
from django.views.decorators.csrf import csrf_exempt
from rest_framework.response import Response
from rest_framework import status
from .models import Livro
from .serializers import LivroSerializer

@csrf_exempt
def livro_list_create(request):
    if request.method == 'GET':
        livros = Livro.objects.all()
        serializer = LivroSerializer(livros, many=True)
        return Response(serializer.data)

    if request.method == 'POST':
        serializer = LivroSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data,
status=status.HTTP_201_CREATED)
        return Response(serializer.errors,
```

```
status=status.HTTP_400_BAD_REQUEST)

@csrf_exempt
def livro_detail(request, pk):
    livro = Livro.objects.get(pk=pk)

    if request.method == 'GET':
        serializer = LivroSerializer(livro)
        return Response(serializer.data)

    if request.method == 'PUT':
        serializer = LivroSerializer(livro, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

    if request.method == 'DELETE':
        livro.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
```

Passo 6: Configuração das URLs

1. Adicione as rotas no `urls.py` do aplicativo `core`:

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

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

Passo 7: Teste da API

1. Inicie o servidor Django:

   ```
   python manage.py runserver
   ```

2. Utilize o Postman para testar as seguintes operações na API:

   - **Listar todos os livros (GET `/livros/`)**
   - **Criar um novo livro (POST `/livros/`)**
   - **Obter detalhes de um livro específico (GET `/livros/<id>/`)**
   - **Atualizar um livro (PUT `/livros/<id>/`)**
   - **Deletar um livro (DELETE `/livros/<id>/`)**

Passo 8: 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`.
   - Marque a opção "Public" e clique em "Create repository".

2. **Adicione o repositório remoto ao projeto local**:
   - No terminal, na raiz do projeto Django, inicialize o repositório git:

     ```
     git init
     ```

   - Adicione o repositório remoto:

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

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

3. **Faça commit e push dos arquivos**:
   - Adicione todos os arquivos e faça um commit:

     ```
     git add .
     git commit -m "Initial commit"
     ```

   - Envie os arquivos para o GitHub:

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

4. **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.