

# Roteiro de Complementação da Primeira Aula Interativa

---

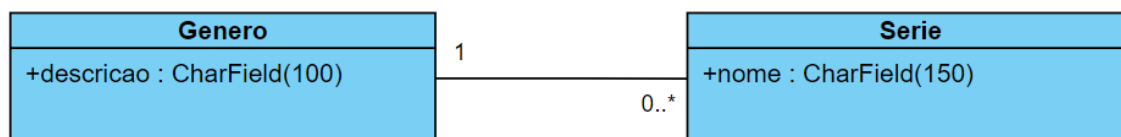
Módulo Python para Aplicações Web

Prof. Carlos Rodrigo Dias

## Sumário

Diagrama de Classes da Aplicação .....	2
Passo a Passo para a Criação do CRUD para a Classe Série.....	3
1. Criar o novo módulo usando o managy.py startapp .....	3
2. Adicionar o novo módulo ao settings.py do projeto .....	4
3. Criar e registrar no projeto o arquivo urls.py do novo módulo .....	4
4. Criar o método cadastrar, retornando apenas “Hello World” .....	4
5. Editar o arquivo urls.py do novo módulo e criar a rota para o método cadastrar.py ....	5
6. Editar o menu.html na página principal para acrescentar a chamada ao módulo, usando a rota criada.....	5
7. Testar o acesso ao novo módulo (ex. “Hello World”) .....	6
8. Criar o template para o novo módulo e indicar a renderização na view .....	6
9. Criar o novo modelo Serie .....	7
10. Criar a migration com o manage.py makemigrations .....	8
11. Executar a migration com o manage.py migrate .....	8
12. Registrar o modelo no admin.py do módulo.....	9
13. Criar o form para o novo módulo .....	10
14. Alterar o template do novo módulo (ex. serie.html) .....	11
15. Alterar o método cadastrar .....	14
16. Criar novo método para deletar .....	16
17. Criar nova página para atualizar (serie_upd.html).....	17
18. Criar novo método para atualizar .....	18
19. Registrar as URLs para deletar e atualizar. ....	20
Material Extra.....	20
Criação de uma Rota Raiz da Aplicação .....	20
Inclusão de Legenda Personalizada no Form Automático .....	22
Projeto Final Disponível no Github.....	23
Sugestão para Manutenção Evolutiva na Aplicação .....	23

## Diagrama de Classes da Aplicação



## Passo a Passo para a Criação do CRUD para a Classe Série

Inicialmente, completar o projeto até a aula PAW 24 ou baixar o projeto base em <https://github.com/rodrigo-igti/paw-interativa1-inicial> e descompactar os arquivos em um diretório chamado criado com o nome igtiflixweb.

Como alternativa, o projeto pode ser clonado em um repositório local chamado igtiflixweb através do comando

```
git clone https://github.com/rodrigo-igti/paw-interativa1-inicial.git igtiflixweb
```

Iniciar o servidor web do Django através do comando:

```
python manage.py runserver
```

Testar a aplicação no navegador, acessando a url

<http://127.0.0.1:8000/principal/>

A tela será exibida como mostrada a seguir.



Depois de ter os arquivos do nosso projeto inicial copiados localmente, abrir o diretório do projeto em um ambiente de desenvolvimento para Python, como o PyCharm ou o VS Code.

O chamado “terminal” é o prompt de comandos ou interpretador de comandos, que no Windows, por exemplo é executado através do comando cmd.exe. Pode-se também usar o terminal disponível nos ambiente de desenvolvimento escolhido. É importante lembrar de utilizar sempre um *virtual environment* quando trabalhar com projetos Python.

Para continuar, o pacote do Django já deve estar instalado. Caso contrário, com o *virtual environment* já ativado previamente, executar o comando:

```
python -m pip install django
```

A seguir, são apresentados os passos que devem executados para que o CRUD “Serie” seja completamente criado, envolvendo a criação do modelo, form e templates, bem como as configurações para colocá-lo funcional. Importante ressaltar que os passos devem ser executados estritamente na ordem apresentada, para evitar bugs durante testes de execução.

### 1. Criar o novo módulo usando o managy.py startapp

No terminal executar o comando:

```
python manage.py startapp serie
```

## 2. Adicionar o novo módulo ao settings.py do projeto

Editar o arquivo `igtflixweb\igtflix\settings.py`

No final da linha 42, incluir uma vírgula e incluir a linha seguinte com o nome do novo módulo 'serie'.

```
34 INSTALLED_APPS = [  
35     'django.contrib.admin',  
36     'django.contrib.auth',  
37     'django.contrib.contenttypes',  
38     'django.contrib.sessions',  
39     'django.contrib.messages',  
40     'django.contrib.staticfiles',  
41     'principal',  
42     'genero',  
43     'serie',  
44 ]
```

## 3. Criar e registrar no projeto o arquivo urls.py do novo módulo

Editar o arquivo `igtflixweb\igtflix\urls.py`

No final da linha 22 acrescentar uma vírgula e incluir a linha 23 com a nova rota para a url a ser informada no navegador (`http://127.0.0.1:8000/serie/`)

```
19 urlpatterns = [  
20     path('admin/', admin.site.urls),  
21     path(route='principal/', view=include('principal.urls')),  
22     path(route='genero/', view=include('genero.urls')),  
23     path(route='serie/', view=include('serie.urls'))  
24 ]
```

O include da linha 23, que foi acrescentada, indica que, para uma rota que inicie com `http://127.0.0.1:8000/serie/`, o restante da rota deverá ser tratado pelo arquivo `igtflixweb\serie\urls.py`, que será criado no Passo 5.

## 4. Criar o método cadastrar, retornando apenas "Hello World"

Os métodos do novo módulo (serie) ficam dentro do arquivo `views.py` do módulo, que foi criado através do comando executado no Passo 1. Cada método do arquivo das views será chamado através da criação de rota específica, como, por exemplo, a que será criada no Passo 5.

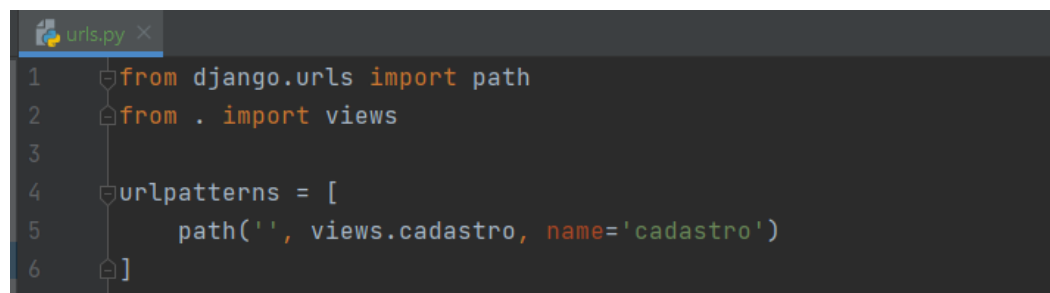
Portanto, neste Passo 4, temos que editar o arquivo `igtflixweb\serie\views.py` cujo conteúdo no momento ficará como mostrado a seguir.

```
1 from django.shortcuts import render  
2 from django.http import HttpResponse  
3  
4  
5 def cadastro(request):  
6     return HttpResponse('Hello world')
```

O método cadastro criado acima retorna (return) um objeto instanciado da classe `HttpResponse`, importada do pacote `django.http`, que é uma resposta ao navegador que fez o request. Esta resposta será apenas uma string de texto "Hello world", que o navegador simplesmente irá exibir. Mas até o momento, não há como o navegador acessar este método, pois é necessário que seja definida a rota para isto, o que será feito no Passo 5.

## 5. Editar o arquivo `urls.py` do novo módulo e criar a rota para o método `cadastro.py`

Para criar a rota para nosso novo método cadastro, que vai exibir o "Hello world" no navegador, temos que inicialmente criar o arquivo `igtiflixweb\serie\urls.py` e incluir o seguinte código:



```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.cadastro, name='cadastro')
6 ]
```

Para poder referenciar o método cadastro de views, tivemos que importar a views do diretório corrente (.), que é o diretório `serie`, mesmo diretório do arquivo `urls.py` criado.

Importante ressaltar que o arquivo `urls.py` foi criado por você pois ele não é criado automaticamente quando o módulo é criado.

## 6. Editar o `menu.html` na página principal para acrescentar a chamada ao módulo, usando a rota criada

O arquivo a ser editado é o `igtiflixweb\templates\menu.html` que contém o nosso menu superior que é incorporados nos templates das demais páginas da nossa aplicação.

A edição necessária no arquivo é a alteração da linha 7, onde é definida a referência para o link **Séries**. Anteriormente estava como `/principal` (e abrindo a página principal), mas a nova referência (href) para o link **Séries** do menu (identificado pela marcação de âncora, `<a>`) será a URL base da aplicação acrescentada de `/serie`.



```
1 <!DOCTYPE html>
2 <header>
3     <nav>
4         <ul>
5             <li><a href= "/principal">Principal</a> </li>
6             <li><a href= "/genero">Gêneros</a> </li>
7             <li><a href= "/serie">Séries</a></li>
8         </ul>
9     </nav>
10 </header>
11
12 </html>
```

## 7. Testar o acesso ao novo módulo (ex. “Hello World”)

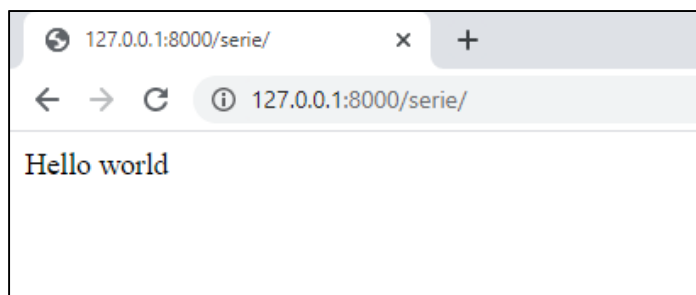
Para testar a aplicação, iniciar o servidor no terminal, executando o comando:

```
python manage.py runserver
```

Em seguida abrir o navegador e digitar a URL

<http://127.0.0.1:8000/serie/>

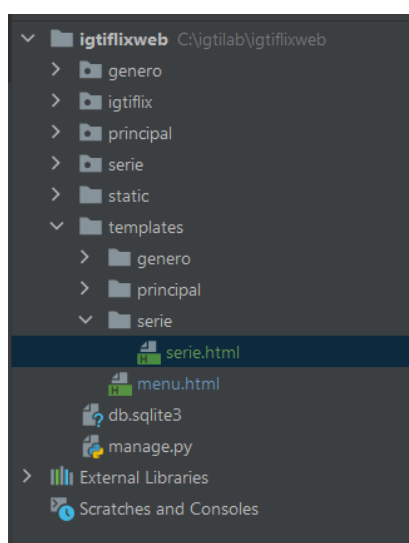
O resultado exibido será:



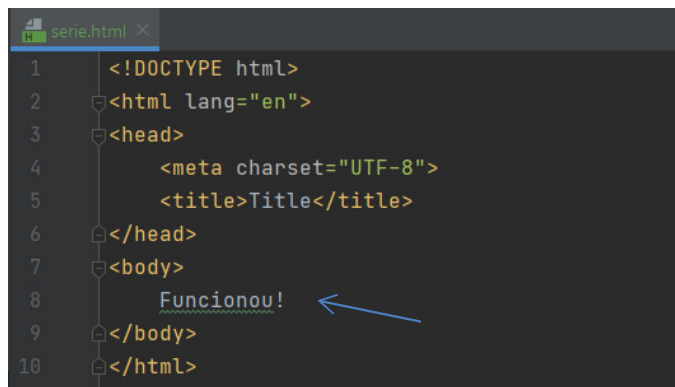
## 8. Criar o template para o novo módulo e indicar a renderização na view

Inicialmente, criar o diretório `serie` dentro de `igtiflixweb/templates`. Para criar um diretório, clique com o botão direito sobre `templates`, e, no menu de atalho, selecione `New > Directory`. Dê o nome (`serie`) e tecla ENTER.

Em seguida, criar o arquivo html, `serie.html`, dentro de `igtiflixweb/templates/serie`.



Abra o arquivo de template criado (`igtiflixweb/templates/serie/serie.html`), e inclua algum texto para testar seu funcionamento. Por exemplo, inclua o conteúdo “Funcionou!” entre as marcações `<body>` e `</body>` no arquivo recém-criado `serie.html`, como mostrado abaixo:



```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Title</title>
6 </head>
7 <body>
8   Funcionou!
9 </body>
10 </html>

```

Agora é necessário alterar o método da view para renderizar este novo template.

Edite o arquivo `igtflixweb\serie\views.py`, substituindo a linha 6, que retorna uma resposta http, para que passe a retornar a renderização do template criado. Vamos usar então o método `render`, já importado na primeira linha do arquivo. O novo código do método cadastro da `views.py` ficará como a seguir.



```

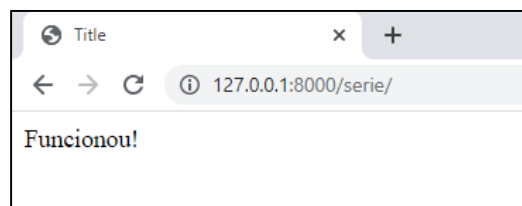
1 from django.shortcuts import render
2 from django.http import HttpResponse
3
4
5 def cadastro(request):
6   return render(request, 'serie/serie.html')

```

Com a alteração, o Django vai encontrar o template `serie.html` e retorná-lo ao navegador que fez o request. Observe que além da referência ao template, é passado também o próprio request recebido do navegador, que poderia ser acessado pelo template, caso necessário.

Posteriormente iremos alterar o template, no Passo 14, para ele exibir os dados da série e incluir um formulário para o cadastro de séries.

Teste novamente o acesso à url do módulo `serie` e veja o resultado.



## 9. Criar o novo modelo Serie

Para criar o novo modelo de dados que será persistido no banco de dados para guardar as informações das séries, vamos abrir o arquivo `igtflixweb\serie\models.py` e trabalhar nele.

Por padrão o `models.py` já foi criado quando criamos o módulo `serie`, e nele já vem a importação do pacote `models` do django. Vamos ter que utilizar alguns métodos deste pacote.

Como apresentado no diagrama de classes, teremos que criar uma classe `Serie`. O código do arquivo `models.py` ficará como o seguinte.

```

models.py
1  from django.db import models
2
3
4  class Serie(models.Model):
5      nome = models.CharField(max_length=150)
6      idGenero = models.ForeignKey('genero.Genero', on_delete=models.PROTECT)
7
8  def __str__(self):
9      return self.nome

```

Esta nova classe Serie herda a classe Model do pacote models.

Teremos então o atributo `nome`, que será do tipo `CharField` com tamanho máximo de 150 caracteres.

E teremos o atributo `idGenero`, que é uma chave estrangeira (`ForeignKey`) que referencia a chave primária da classe `Genero` do modelo `genero`, conforme passado como primeiro parâmetro do método `models.ForeignKey()`.

O segundo parâmetro informado indica o que deve ser feito quando um registro da tabela referenciada é excluído (`on_delete`). No nosso caso, informamos que se tiver alguma série com o valor `idGenero` referenciando este registro a ser excluído, o registro não poderá ser excluído de `Gênero`. A constante `models.PROTECT` indica exatamente isto.

Importante ressaltar que não é necessário informar a chave primária a ser criada para o novo modelo. O Django irá criar automaticamente um campo `id`, auto-incremental, e ele será a chave primária da tabela.

Finalmente, precisamos sobrescrever o método `__str__()` da classe `Models` herdada, pois ela permite criarmos uma representação em string do objeto instanciado. No caso, uma exibição do objeto como string irá apresentar o nome da série. Um exemplo de utilização deste tipo de representação do objeto é na lista de séries exibida no `/admin`.

## 10. Criar a migration com o `manage.py makemigrations`

Após a definição da classe `Serie` no `models`, é necessário preparar a migração, que irá criar a tabela no banco de dados.

Antes de continuar, interrompa a execução do nosso servidor web no terminal, através do pressionamento das teclas `CTRL + C`.

Depois de interromper o servidor web, o comando a ser executado para a preparação da migração é:

```
python manage.py makemigrations serie
```

Como retorno da execução com sucesso, o Django indica que o modelo foi criado:

```

(djangoenv) C:\igtilab\igtiflixweb>python manage.py makemigrations serie
Migrations for 'serie':
  serie\migrations\0001_initial.py
  - Create model Serie

```

## 11. Executar a migration com o `manage.py migrate`

Para efetivar a migração deve ser executado o comando no terminal



```
python manage.py migrate
```

Se a migração finalizar com sucesso, o Django exibe OK, conforme mostrado a seguir:

```
(djangoenv) C:\igtilab\igtiflixweb>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, genero, serie, sessions
Running migrations:
  Applying serie.0001_initial... OK
```

Isto significa que está tudo certo e que podemos então rodar novamente o nosso servidor, executando o seguinte comando no terminal:

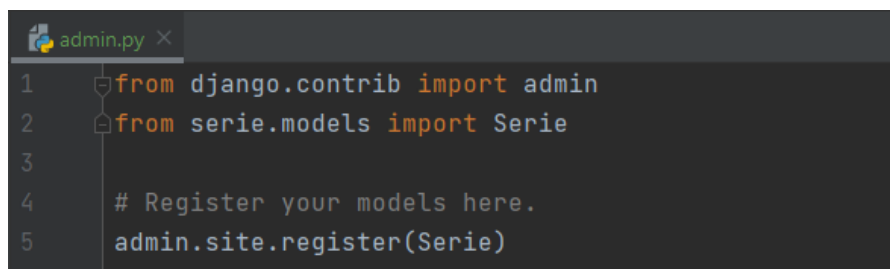
```
python manage.py runserver
```

## 12. Registrar o modelo no admin.py do módulo

A página /admin do Django, acessada através da URL <http://127.0.0.1:8000/admin/> permite acessar algumas informações da aplicação que são guardadas nas tabelas do banco de dados SQLite, controlada pelo Django.

Assim, as tabelas criadas para nossos models também podem ser gerenciadas pelo /admin. Para isto, precisamos registrar os models de cada módulo no arquivo admin.py do respectivo módulo (ou app).

Vamos então registrar nosso novo modelo Serie no módulo serie. Para isto, edite o arquivo igtiflixweb\serie\admin.py e informe seu conteúdo da seguinte forma.

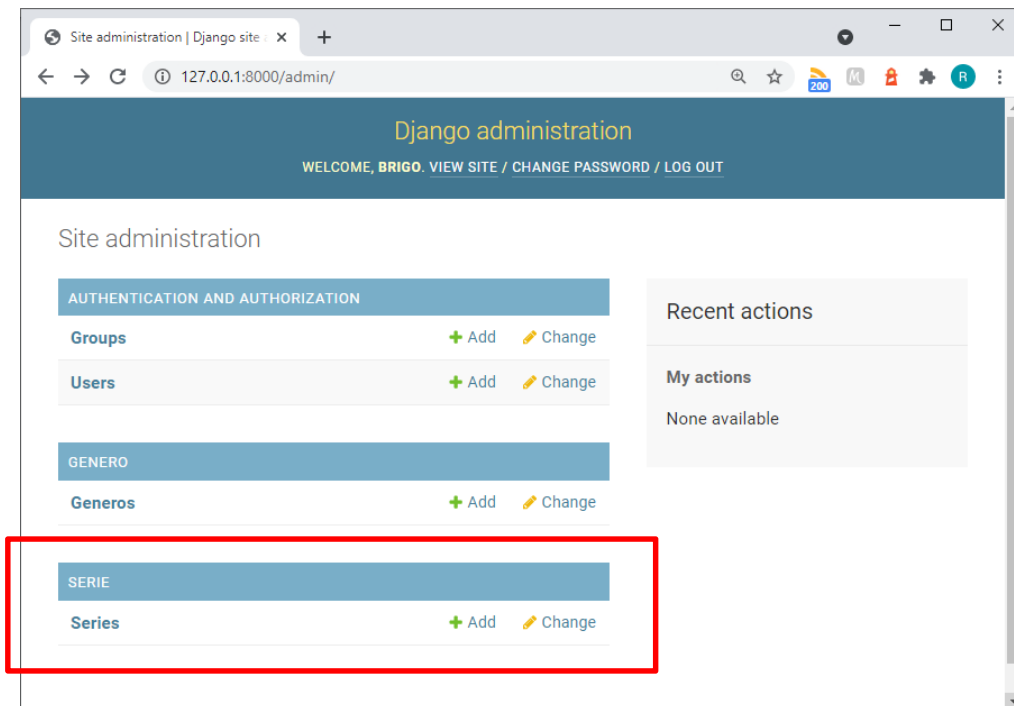


```
admin.py
1 from django.contrib import admin
2 from serie.models import Serie
3
4 # Register your models here.
5 admin.site.register(Serie)
```

Inicialmente é importada a biblioteca admin, a partir da qual será executado o método `register()` que é responsável por fazer com que o admin “enxergue” o nosso novo modelo. Para passar o modelo Serie (nossa classe Serie) para o método `register()`, é necessário importá-lo também, o que é feito na linha 2.

Para testar isto, verifique se o servidor web ainda está em execução. Caso ele indique algum erro, verifique as informações do erro, e em qual arquivo, e corrija-as, retornando aos passos anteriores e checando cada linha de código.

Se o servidor web estiver executando normalmente, vá ao navegador e acesse a URL <http://127.0.0.1:8000/admin/>. Você verá que o nosso novo modelo Serie estará devidamente registrado e podemos interagir com ele, incluindo novos registros de séries, e também executando ações de alteração e exclusão.



### 13. Criar o form para o novo módulo

Agora que o banco de dados já está preparado para tratar o modelo Serie, e já é possível renderizar a nova view (conforme visto no Passo 8, será necessário criar o formulário para que o usuário possa interagir com a aplicação, incluindo uma nova série ao banco de dados.

Da forma que estamos fazendo no nosso projeto, estamos aproveitando os recursos que o Django nos fornece para automatizar ao máximo a criação do form.

Para criar o form, primeiro é necessário criar o arquivo forms.py no nosso módulo. Clique com o botão direito sobre o nome do módulo serie, e selecione New > Python file. Informe o nome forms. No arquivo criado, igtiflixweb\serie\forms.py, insira o seguinte código:

```
1 from django import forms
2 from .models import Serie
3
4
5 class SerieForm(forms.ModelForm):
6
7     class Meta:
8         model = Serie
9         fields = '__all__'
```

Na linha 1 é importado o pacote forms do Django, que tem a classe ModelForm que será herdada para a nossa nova classe SerieForm. Temos que importar também o nosso modelo Serie, que está no arquivo models dentro do mesmo diretório do arquivo forms.py que estamos editando (por isto indicamos .models).

Na class Meta, faz parte ModelForm, e portanto fará parte da nossa classe SerieForm, é necessário indicar que o formulário será criado baseado no model Serie (confirme linha 8) e que serão utilizados todos os atributos (ou campos) do modelo (conforme linha 9). Ressalto que o

“\_\_all\_\_” indica todos os campos, mas seria possível também selecionar apenas alguns campos do modelo, caso fosse desejado.

O formulário está pronto para ser incluído na nossa página de cadastro de série, o que será feito através do template da serie.

#### 14. Alterar o template do novo módulo (ex. serie.html)

Antes de realizar esta alteração, vamos ajusta o nome do arquivo de folhas de estilo (.css) criado para o módulo genero, com o objetivo de que ele possa ser usado também para o módulo série, ou outros módulos que forem criados posteriormente. Atualmente o arquivo é o `igtiflixweb\static\css\genero.css`. Vamos alterar o nome dele para `crud.css`, porque a ideia é que ele seja aproveitado em todos os CRUDs que forem criados.

Em seguida, altere a referência ao antigo arquivo css para o novo, no template do gênero. Edite o arquivo `igtiflixweb\templates\genero\genero.html`, alterando a linha 9 conforme figura a seguir:



```
1 <!DOCTYPE html>
2 {% load static %}
3 <html>
4 <head>
5 <meta charset="utf-8">
6 <title>IGTIFlix</title>
7 <link href="{% static 'css/reset.css' %}" rel="stylesheet" type="text/css">
8 <link href="{% static 'css/menu.css' %}" rel="stylesheet" type="text/css">
9 <link href="{% static 'css/crud.css' %}" rel="stylesheet" type="text/css">
10
11
12 </head>
```

O mesmo deverá ser feito para o template usado para o *update* do Genero. Para isto, edite o arquivo `igtiflixweb\templates\genero\genero_upd.html`, alterando a linha 9 da mesma forma como foi feito para o `genero.html`, mostrado na figura anterior.

Agora sim, para ajustar o arquivo do template do novo módulo serie (`serie.html`), que já foi criado mas que atualmente apenas exibe a palavra “Funcionou!” (conforme feito no Passo 8), uma forma mais rápida é copiar e colar o conteúdo do template do módulo genero e, posteriormente, realizar as devidas alterações. É isto que será feito, porém é necessária muita atenção para evitar erros que possam interromper a execução do servidor. Caso o servidor web seja interrompido por, por exemplo, um erro de sintaxe, basta corrigir o erro e executar novamente o servidor web com o comando:

```
python manage.py runserver
```

Então, como foi dito, abra o arquivo `igtiflixweb\templates\genero\genero.html` e copie todo o seu conteúdo e cole no arquivo `igtiflixweb\templates\serie\serie.html`, substituindo qualquer conteúdo que já existia neste último arquivo.

Em seguida, devem ser realizadas todas as alterações, substituindo as referências ao modulo genero, ao modelo Genero e também a atributo descricao do modelo Genero, para o módulo serie, modelo Serie e atributo nome, respectivamente.

A figura a seguir mostra um bloco de linhas que devem ser alteradas no arquivo `igtflixweb\templates\serie\serie.html`.



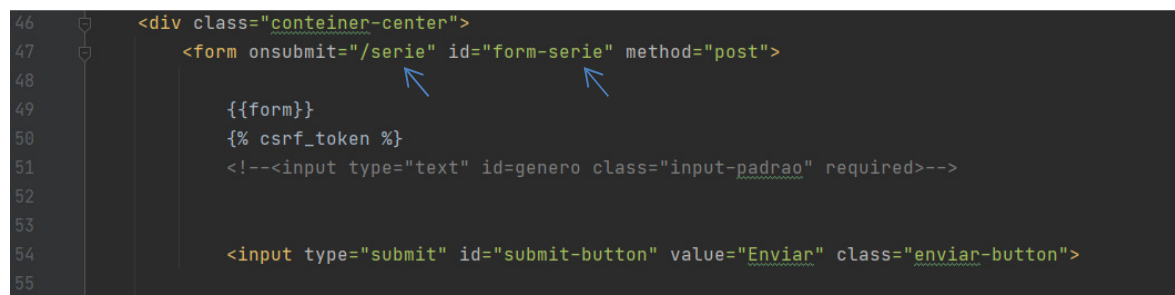
```
13 <body>
14 {% include "../menu.html" %}
15 <main>
16 <div class="container-center">
17 <span>
18     {% if serie_records %}
19     <table>
20     <thead>
21     <tr>
22         <th>Serie</th>
23         <th>Clique para alterar</th>
24         <th>Clique para excluir</th>
25     </tr>
26     </thead>
27
28     <tbody>
29     {% for serie in serie_records %}
30     <tr>
31         <td>{{serie.nome}}</td>
32         <td><button class="list-button editar-button" onclick="updateItem('{{serie.id}}')">Alterar</button></td>
33         <td><button class="list-button apagar-button" onclick="deleteItem('{{serie.id}}')">Excluir</button></td>
34     </tr>
35     </tbody>
36     </table>
37     </span>
38 </div>
39 </main>
40 </body>
```

As alterações são:

- linha 18: a lista `serie_records` será recebida do dicionário enviado no render da view, que será alterada no Passo 18. Esta lista conterá a relação de series armazenadas no banco de dados, e que será exibida na tabela. Esta linha corresponde a um `if` (se), que só vai exibir o bloco seguinte se existir algum elemento na lista. O `endif` deste bloco está na linha 42 (não mostrada nesta figura).
- linha 22: é o cabeçalho da tabela referente à coluna onde será mostrado o nome da série.
- linha 29: este “for” permite acessar cada um dos elementos da lista `serie_records` que já foi checada na linha 18. Dentro do `for` cada elemento da lista será referenciado como `serie`.
- linha 32: corresponde ao botão de edição de uma série específica. Assim, quando a página html do template for montada, o botão deverá chamar a função javascript `updateItem()`, passando o `id` da `serie` a que ele se refere. Por isto, devemos indicar aqui que será o atributo `serie.id`.
- linha 33: semelhante à linha 32, aqui o `id` é usado para referenciar a série que será excluída através da chamada da função javascript `deleteItem()`.

Vale ressaltar que as funções javascript indicadas nas linhas 32 e 33 estão criadas no próprio template `serie.html`, entre as linhas 66 e 89, e também serão alterados.

Na próxima figura é apresentado outro trecho do template que deve ser alterado.



```
46 <div class="container-center">
47 <form onsubmit="/serie" id="form-serie" method="post">
48
49     {{form}}
50     {% csrf_token %}
51     <!--<input type="text" id=genero class="input-padrao" required-->
52
53
54     <input type="submit" id="submit-button" value="Enviar" class="enviar-button">
55 </form>
```

- linha 47: nesta linha temos a marcação de início do formulário, na qual devemos informar o que será chamado ao clicar no botão `submit` Enviar (definido na linha 54).

Ao clicar no botão será chamada a página que inicia com a URL base, seguida de `/serie`. O identificador deste elemento form do html será “form-serie”.

Importante destacar que a linha 49 indica que será exibido aqui o formulário chamado `form`, que irá ser trazido através do dicionário de dados (ainda não criado) informado no `render` da nossa view do módulo `serie`.

Finalmente, na imagem a seguir, é apresentado o código javascript presente no template, que também terá pequenas alterações. Neste trecho está o código das funções `deleteItem()`, responsável por chamar o método `delete` da nossa view, e o `updateItem()`, responsável por chamar o método `update` da nossa view. Nestas funções, é indicada a rota a ser utilizada na URL, complementando com o `id` da série a ser tratada.

Importante ressaltar que o código javascript é executado no cliente, ou seja, no navegador, e não no servidor. Assim, ele faz o request desejado (excluir ou atualizar) e recebe a resposta do servidor.



```
66 <script>
67   function deleteItem(id) {
68     const csrftoken = document.querySelector('[name=csrfmiddlewaretoken]').value;
69     const request = new Request(
70       "/serie/delete/"+id,
71       {headers: {'X-CSRFToken': csrftoken}}
72     );
73     fetch(request, {
74       method: 'DELETE',
75       mode: 'same-origin'
76     }).then(function(response) {
77       if(response.status===500){
78         alert("Não foi possível excluir.");
79       } else{
80         location.replace("/serie")
81       }
82     })
83   }
84
85   function updateItem(id) {
86     location.replace("/serie/update/"+id)
87   }
88 }
89 </script>
```

- linha 70: a função javascript `deleteItem()` cria um request para a rota `/serie/delete`, e a complementa com o `id` que foi informado como parâmetro da função (lá na linha 33), que irá indicar qual série deve ser excluída.
- linha 78: simplesmente foi excluído o texto “Gênero em uso”, que só faz sentido quando excluimos um gênero que tem uma série relacionada a ele. Para série isto não faz sentido, pois não há nenhuma outra tabela que fizesse referência. (entretanto isto pode mudar, caso você expanda a sua aplicação, criando, por exemplo, uma nova classe chamada Episódio, que fará referência à classe Série).
- depois que a função javascript realiza o HTTP request, usando o método “DELETE” com a URL para o delete da nossa view (linhas 73 a 75), ele verifica o retorno do servidor (linhas 76 e 77). Se o retorno foi 500, indicando um erro, ele exibe a mensagem na linha 78, caso contrário, ele acessa a url raiz do módulo `serie` (linha 80).
- linha 87: esta é a única linha da função `updateItem()`, que é responsável por montar a URL para a rota de atualização (`update`) incluindo ao final o `id` da série a ser alterada, e que foi passada para a função conforme visto na linha 32.

Com o template pronto, precisamos alterar os métodos da view do módulo serie. Se você executar a aplicação agora, você não vai conseguir visualizar o formulário, pois ainda não foi alterada a view para que ele possa ser renderizado juntamente com o template. Isto é o que será feito no Passo 15.

Observe que as alterações do template não acarretam uma recarga do servidor web, pois elas se referem apenas às marcações html e javascript, que irão rodar no cliente.

## 15. Alterar o método cadastrar

Neste passo 15, vamos alterar o arquivo da nossa view para renderizar o template, incluindo um dicionário de dados com o form criado e lista com as séries a serem exibidas, que serão mesclados ao template nos locais que vimos no Passo 14.

Para facilitar, vamos abrir o arquivo `igtflixweb\genero\views.py`, copiar o conteúdo referente à importação dos pacotes e ao código do método `cadastro()`, linhas 1 a 20, e colar este conteúdo no arquivo `igtflixweb\serie\views.py`. E seguida realize as alterações indicadas na figura seguinte (não se esqueça de ter certeza que está alterando o arquivo `views.py` correto, do módulo `serie`, e não do módulo `genero`).

```
1 from django.http import HttpResponseRedirect
2 from django.shortcuts import render
3
4 from . import forms
5 from . import models
6
7
8 def cadastro(request):
9     form = forms.SerieForm()
10
11     if request.method == 'POST':
12         form = forms.SerieForm(request.POST)
13         if form.is_valid():
14             print("Saving")
15             form.save(commit=True)
16         else:
17             print("ERROR")
18             series_list = models.Serie.objects.order_by('nome')
19             data_dict = {"serie_records": series_list, 'form': form}
20             return render(request, 'serie/serie.html', data_dict)
21
```

Nas linhas 4 e 5 foram importados os arquivos que contêm o nosso form e o nosso modelo, e que estão no mesmo diretório do arquivo `views`. Observe que na importação em Python não é necessário informar a extensão `.py` dos arquivos.

As alterações realizadas no método `cadastro()` foram:

- linha 9: instanciamos um objeto chamado `form` a partir da classe `SerieForm` que está no nosso arquivo `forms`, que foi importado na linha 4
- linha 11: se o método HTTP que chamou a função for o POST, ele repassa o request para o nosso `SerieForm`, para que os dados sejam gravados no banco de dados, o que ocorrerá apenas se o form estiver com dados válidos.
- linha 17: é criada a lista `series_list` como uma lista de objetos da classe `Serie`, ordenado alfabeticamente pelo atributo `nome`.
- linha 18: a lista criada na linha 17 é incluída, juntamente com o nosso `form` instanciado, em um dicionário de dados chamado `data_dict`. Dentro do template a lista será referenciada como `serie_records`.

- linha 20: a função `cadastro()` retorna a renderização do nosso template “serie/serie.html”, mesclado com os dados do `data_dict` criado na linha 18.

As linhas 13 e 16 apenas exibem no terminam uma depuração do código. Em uma aplicação em produção isto deve ser convertido em um alerta na janela do navegador para o usuário.

Agora, verifique se o servidor web está executando normalmente. Neste caso, como a rota para o cadastro já foi criada, acesse no navegador a URL <http://127.0.0.1:8000/serie/> (ou clique no menu Séries se a aplicação já estiver aberta no navegador). Será mostrado o conteúdo da página de cadastro de séries, conforme ilustrado na figura a seguir.

Nenhum dado cadastrado

Nome:

IdGenero:

Enviar

Nesta página, utilize o formulário para cadastrar algumas séries, informando o nome e selecionando em `IdGenero` o gênero desejado, da lista de gêneros já incluídos na página de Gêneros. Após clicar no botão Enviar (o nosso submit), a nova série será criada e exibida na própria janela.

Após incluir uma série de exemplo, o resultado será como o mostrado na figura a seguir.

Serie	Clique para alterar	Clique para excluir
Breaking Bad	Alterar	Excluir

Nome:

IdGenero:

Enviar

Observe que na listagem de séries cadastradas não é exibido o gênero da série, apesar desta informação fazer parte do nosso modelo `Serie`. Isto ocorre porque esta informação não foi incluída no nosso template, para ser exibida.

Vamos ajustar isto alterando o conteúdo do arquivo com o nosso template: `igtiflixweb\templates\serie\serie.html`.



```

19 <table>
20 <thead>
21 <tr>
22 <th>Serie</th>
23 <th>Gênero</th>
24 <th>Clique para alterar</th>
25 <th>Clique para excluir</th>
26 </tr>
27 </thead>
28
29 <tbody>
30 {% for serie in serie_records %}
31 <tr>
32 <td>{{serie.nome}}</td>
33 <td>{{serie.idGenero}}</td>
34 <td><button class="list-button editar-button" onclick="updateItem({{serie.id}})">Alterar</button></td>
35 <td><button class="list-button apagar-button" onclick="deleteItem({{serie.id}})">Excluir</button></td>
36 </tr>
37 {% endfor %}
38 </tbody>
39 </table>
40
41

```

Foram incluídas as linhas 23, que é a inclusão de uma nova coluna com cabeçalho “Gênero”, e a linha 33, que corresponde ao conteúdo da nova coluna, que é o `idGenero`.

Dê um *refresh* (F5) na página `serie` e você verá que a coluna foi incluída corretamente. Observe também que o que foi exibida a descrição do Gênero na tabela `Genero`, e não o valor do campo ID. Por padrão o Django exibe o conteúdo do primeiro atributo descrito no modelo da classe relacionada (`Genero`), e não o id, que é o que está armazenado na tabela `Serie`.

Serie	Gênero	Clique para alterar	Clique para excluir
Breaking Bad	Crime	Alterar	Excluir

Nome:

IdGenero:

Observe que os botões “Alterar” e “Excluir” já estão criados. Porém eles ainda não estão funcionando corretamente. Isto porque nós os incluímos no template, que gerou a página que visualizamos, mas os métodos que devem ser executados ainda não foram criados. Isto será feito nos próximos passos.

## 16. Criar novo método para deletar

Como foi visto no Passo 14, o template já está com o botão para excluir uma série e já é chamada a função javascript que elabora a rota necessária. Entretanto, a rota ainda não responde pois o respectivo método não foi criado na view.

Para criar o método delete, também iremos copiar o código existente para `Genero`. Assim, abra o arquivo `igtiflixweb\genero\views.py`, e copie o conteúdo referente ao código do método `delete()`, **linhas 23 a 32**, e cole este conteúdo no arquivo `igtiflixweb\serie\views.py`. E seguida realize as alterações indicadas na figura seguinte (não se esqueça de ter certeza que está alterando o arquivo `views.py` correto, do módulo `serie`, e não do módulo `genero`).



```

23 def delete(request, id):
24     try:
25         models.Serie.objects.filter(id=id).delete()
26         form = forms.SerieForm()
27         series_list = models.Serie.objects.order_by('nome')
28         data_dict = {"serie_records": series_list, 'form': form}
29         return render(request, 'serie/serie.html', data_dict)
30     except:
31
32         return HttpResponseNotAllowed();

```

O método `delete()` recebe o `id` a ser excluído. As instruções referentes à exclusão do objeto referenciado pelo `id` estão dentro de um bloco `try:` (linhas 24 a 29). Assim, se por algum motivo a exclusão não for possível de ser realizada, caindo no bloco `except:` o método irá retornar um erro de “Not Allowed” para o navegador cliente requisitante.

Na linha 25 é realizada a exclusão a partir do método `delete()` do objeto instanciado do modelo `Serie` obtido a partir da filtragem pelo `id`. Tudo isto realizado em uma linha apenas.

Em seguida um novo `form` é instanciado da nossa classe `SerieForm` importada do arquivo `forms.py`.

Na linha 27 é criada a lista `series_list` como uma lista de objetos da classe `Serie`, ordenado alfabeticamente pelo atributo `nome`.

Na linha 28 a lista criada na linha 27 é incluída, juntamente com o nosso `form` instanciado, em um dicionário de dados chamado `data_dict`. Dentro do template a lista será referenciada como `serie_records`.

Na linha 29: a função `delete()` retorna a renderização do nosso template “`serie/serie.html`”, mesclado com os dados do `data_dict` criado na linha 28.

Com isto, o método `delete()` da view já está pronto. Entretanto, ele ainda não está funcionando na aplicação, pois a respectiva rota ainda não foi definida, o que será feito no Passo 19.

## 17. Criar nova página para atualizar (`serie_upd.html`)

Antes de criar o método `update()` da view, é necessário criar um template que será usado para carregar o formulário com os dados da série selecionada pelo `id`. E este formulário poderá ser alterado e, em seguida, realizada a submissão e posterior atualização do banco de dados.

Crie o arquivo `serie_upd.htm` dentro de `igtiflixweb\templates\serie`.

Será realizada também uma cópia do template criado para atualização do `Genero`. Assim, abra o arquivo `igtiflixweb\templates\genero\genero_upd.html` e copie todo o seu conteúdo e cole no arquivo `igtiflixweb\templates\serie\serie_upd.html`, substituindo qualquer conteúdo que já existia neste último arquivo.

Em seguida, devem ser realizadas todas as alterações, substituindo as referências ao módulo `genero`, ao modelo `Genero` e também a atributo `descricao` do modelo `Genero`, para o módulo `serie`, modelo `Serie` e atributo `nome`, respectivamente.

A figura a seguir mostra um bloco de linhas que devem ser alteradas no arquivo `igtiflixweb\templates\serie\serie_upd.html`.

```

16 <div class="container-center">
17 <form onsubmit="/serie" id="form-serie" method="post">
18     {{form}}
19     {% csrf_token %}
20     <input type="submit" id="submit-button" value="Enviar" class="enviar-button">
21
22 </form>
23
24 </div>
25

```

Na linha 17, alterar a rota que será utilizada após a submissão do formulário. Ao clicar no botão Enviar, definido na linha 21, o form irá salvar os dados automaticamente e retornar para a rota “/serie”, que é a tela inicial do cadastro.

## 18. Criar novo método para atualizar

Agora que temos o template, podemos criar o método atualizar na view de serie.

Como foi visto no Passo 14, o template já está com o botão para atualizar cada série e já é chamada a função javascript que elabora a rota necessária. Entretanto, a rota ainda não responde, pois o respectivo método não foi criado na view.

Para criar o método `update()`, também iremos copiar o código já existente para Genero. Assim, abra o arquivo `igtflixweb\genero\views.py`, e copie o conteúdo referente ao código do método `update()`, **linhas 35 a 47**, e cole este conteúdo no arquivo `igtflixweb\serie\views.py`. Em seguida, realize as alterações indicadas na figura seguinte (não se esqueça de ter certeza que está alterando o arquivo `views.py` correto, do módulo `serie`, e não do módulo `genero`).

```

36 def update(request, id):
37     item = models.Serie.objects.get(id=id)
38     if request.method == "GET":
39         form = forms.SerieForm(initial={'nome': item.nome, 'idGenero': item.idGenero})
40         data_dict = {'form': form}
41         return render(request, 'serie/serie_upd.html', data_dict)
42     else:
43         form = forms.SerieForm(request.POST)
44         item.nome = form.data['nome']
45         item.idGenero_id = form.data['idGenero']
46         item.save()
47         form = forms.SerieForm()
48         series_list = models.Serie.objects.order_by('nome')
49         data_dict = {'serie_records': series_list, 'form': form}
50         return render(request, 'serie/serie.html', data_dict)
51

```

Na linha 37 é obtido o objeto da classe `Serie` que corresponde ao `id` fornecido para o método `update()` e que terá seus atributos atualizados. Lembrando que quem forneceu este `id` foi o cliente, o navegador, através da função javascript presente de um dos templates criados para a `serie`.

Na linha 38 é verificado se o template é o `serie.html`, que é o que envia o HTTP request com o método GET. Se for um GET, então deverá ser exibido o template `serie_upd.html`, que irá exibir os dados da série selecionada. Para isto na linha 39 é instanciado um `form` da classe

`SerieForm`. Os valores dos campos do formulário serão preenchidos com o campo `nome` do objeto selecionado e será acrescentado também o campo `idGenero`, que permitirá alterar também o gênero da série. Por padrão do Django em chaves estrangeiras, o campo `idGenero` não exibirá o `id` e, no nosso caso, irá exibir o valor do atributo `descricao` do Gênero.

Em seguida é montado o dicionário com apenas o formulário, devidamente preenchido com os valores e na linha 42 é executado o método de renderização do template `serie_upd.html`, mesclando-o com o dicionário que contém `form` criado. Então o formulário será exibido e permitirá a edição dos valores dos atributos, e submeter estas alterações, conforme foi criado no template no Passo 17.

A partir da linha 42 o bloco `else:` será executado se o método do HTTP request não for o GET, o que ocorrerá quando a requisição vier do template `serie_upd.html`, pois o método do HTTP request do botão `submit` do `form` é definido como “post”. Observe também que no `pitpr` template, `serie.html`, não foi especificado o método, e, por isto, por padrão é assumido o método GET.

Na linha 43 é instanciado o objeto `form` a partir da classe `SerieForm`, passando o conteúdo do request, que irá conter os dados informados no formulário. Assim, estes dados poderão ser obtidos a partir do objeto `form`, e, nas linhas 44 e 45 estes valores irão atualizar os atributos do objeto instanciado na linha 37 e que corresponde à série a ser atualizada. A linha 46 é a responsável salvar esta atualização no banco de dados.

Uma observação importante em relação à linha 45 é que ela foi acrescentada para atualizar os dados do atributo `idGenero`. Entretanto, como vimos, quando trabalhamos com um objeto instanciado da classe `Serie`, o Django não utiliza o `id`, mas sim o atributo `descricao`. Por outro lado, apesar do `form` exibir na combobox `idGenero` a descrição do gênero, o valor que ele armazena internamente é o `id`. Assim, irá ocorrer um erro se tentarmos fazer a atribuição:

```
item.idGenero = form.data['idGenero']
```

Para resolver isto, nós conseguiremos acessar o valor `id` de uma chave estrangeira acrescentando “\_id” ao final do nome do atributo chave estrangeira. O correto então ficará como:

```
item.idGenero_id = form.data['idGenero']
```

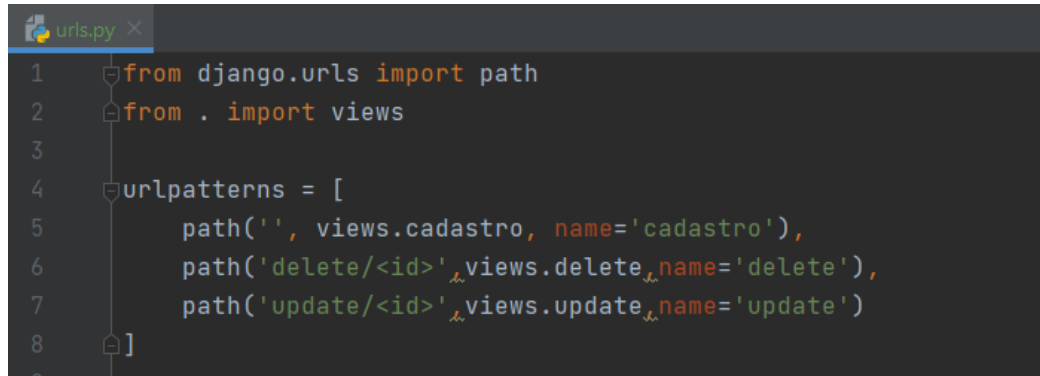
Na linha 47 foi acrescentada uma nova linha, que instancia um novo `form` vazio da classe `SerieForm`. Isto irá permitir que, ao renderizar novamente o template `serie.html`, o que será feito na linha 50, o formulário de cadastro irá ser exibido com os campos vazios, e não com os dados trazidos do formulário do template `serie_upd.html`. Atenção! Considerando isto, observe que ao cadastrar uma nova série os valores digitados são mantidos no formulário. Agora você já sabe como resolver isto, bastando incluir uma linha idêntica à linha 47 no método `cadastro`, antes do retorno da renderização. Sugiro também alterar o arquivo `views` do módulo `genero`, para limpar os campos depois da inclusão e alteração.

Finalmente, nas linhas seguintes é criada a lista de series, é montado o dicionário com esta lista e com o último `form` criado, e é renderizado o template `serie.html` juntamente com o dicionário.

## 19. Registrar as URLs para deletar e atualizar.

Para que o delete e update funcionem corretamente, só está faltando indicar que os métodos deverão ser acionados nas rotas específicas.

Isto é feito editando-se o arquivo `igtflixweb/serie/urls.py`, conforme mostrado na figura a seguir.



```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.cadastro, name='cadastro'),
6     path('delete/<id>', views.delete, name='delete'),
7     path('update/<id>', views.update, name='update')
8 ]
```

Foram acrescentadas as linhas 6 e 7. É importante ressaltar que na rota foi indicado que será informado o id da série a ser excluída ou alterada, da mesma forma como foi implementado para o módulo genero.

Agora pode ser realizado o teste da aplicação. Antes, verificar se o servidor web está funcionando corretamente, sem erros.

Com este último passo, ficam implementados os CRUDs para as classes genero e serie, e o projeto proposto de ser implementado em Django foi concluído.

## Material Extra

A seguir apresento algumas sugestões adicionais de melhorias no projeto que podem ser implementadas rapidamente.

### Criação de uma Rota Raiz da Aplicação

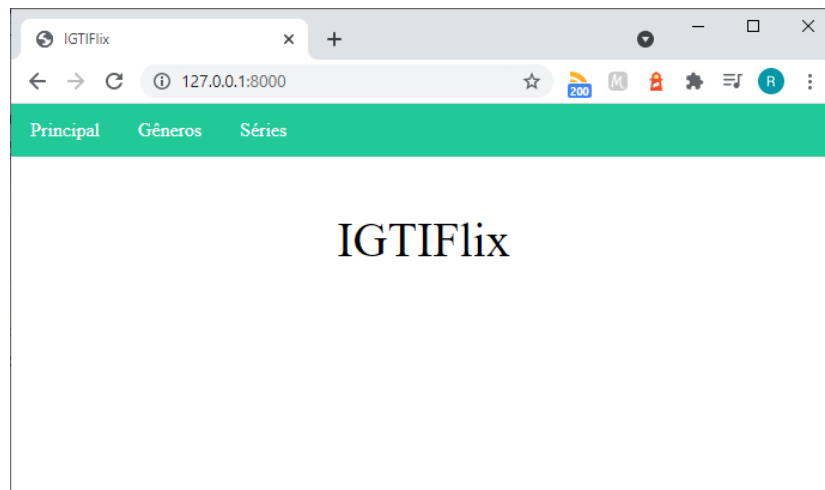
O nosso projeto finalizou sem a definição de uma rota raiz. Assim, quando é acessada a URL <http://127.0.0.1:8000> é apresentado um erro.

Para resolver isto, pode-se simplesmente incluir uma rota “” no arquivo `\igtflixweb\igtflix\urls.py` apontando para a página do módulo principal, como mostrado na figura abaixo,



```
19 urlpatterns = [
20     path('', view=include('principal.urls')),
21     path('admin/', admin.site.urls),
22     path(route='principal/', view=include('principal.urls')),
23     path(route='genero/', view=include('genero.urls')),
24     path(route='serie/', view=include('serie.urls'))
25 ]
```

Agora é possível acessar <http://127.0.0.1:8000> sem erros, e o resultado é mostrado na figura a seguir.

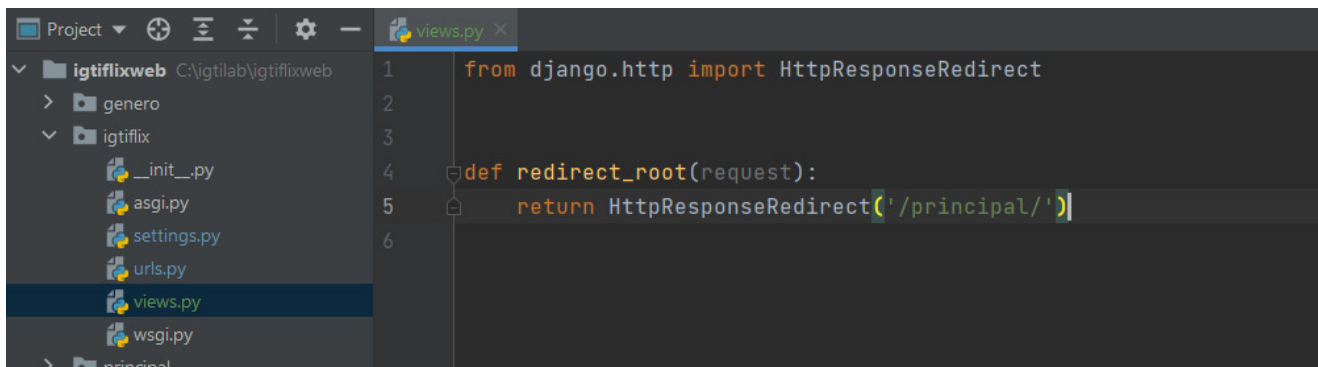


Apesar de funcionar, temos aqui algo que não é aconselhado pelas melhores práticas. O que temos com esta solução é duas URLs apontando para um mesmo recurso e uma URL deve sempre se referir a um recurso único.

A prática mais aconselhável aqui é usar o redirecionamento, ou seja, criar uma página para ser acessada pela rota raiz, e que irá redirecionar o navegador para a página identificada pela URL /principal.

Para fazer isto, vamos criar uma nova view no diretório do projeto. Assim, crie e edite o arquivo `\igtiflixweb\igtiflix\views.py`.

Criar o arquivo `views.py` no diretório da aplicação (mesmo diretório do arquivo de configurações `settings.py`) com o código apresentado na imagem a seguir.



Nesta view, assim que o método `redirect_root()` é acionado, ele retorna o redirecionamento para a rota /principal, acionando esta URL da aplicação. No navegador a URL exibida será então <http://127.0.0.1:8000/principal/>

Para o novo método ser acionado temos então que alterar o arquivo `\igtiflixweb\igtiflix\urls.py` conforme imagem a seguir.

```

16 from django.contrib import admin
17 from django.urls import path, include
18 from .views import redirect_root
19
20 urlpatterns = [
21     path('', redirect_root),
22     path('admin/', admin.site.urls),
23     path(route='principal/', view=include('principal.urls')),
24     path(route='genero/', view=include('genero.urls')),
25     path(route='serie/', view=include('serie.urls'))
26 ]
27

```

Na linha 18 foi incluída a importação do método `redirect_root` da nova view e na linha 21 é criada a rota "" que irá executar o método `redirect_root` que, como vimos, irá redirecionar o navegador para a URL `/principal`.

### Inclusão de Legenda Personalizada no Form Automático

Como vimos, as legendas dos campos nos formulários criados corresponde aos nomes dos respectivos atributos da Classe.

É possível personalizar as legendas, colocando acentuação ou alterando o texto para um texto mais representativo.

Para isto, basta colocar uma string contendo o texto da legenda no primeiro parâmetro da instanciação de cada atributo no modelo.

Em chaves estrangeiras o primeiro parâmetro é o nome da classe referenciada. Assim, será necessário explicitar o parâmetro da legenda com `verbose_name = "texto da legenda"`

Por exemplo, a legenda para os atributo `nome` e `idGenero` do modelo `Serie` poderá ser criada alterando o campo `nome` no arquivo `\igtflixweb\serie\models.py` como mostrado a seguir:

```

1 from django.db import models
2
3
4 class Serie(models.Model):
5     nome = models.CharField("Título da Série", max_length=150)
6     idGenero = models.ForeignKey('genero.Genero', verbose_name="Gênero", on_delete=models.PROTECT)
7
8     def __str__(self):
9         return self.nome
10

```

Com estas alterações no nosso modelo `Serie`, ao acessar nossa aplicação, o formulário CRUD da `Série` terá suas legendas alteradas, como mostrado na figura abaixo.

Como sugestão adicional, faça as alterações das legendas também para a Classe Genero.

## Projeto Final Disponível no Github

O projeto final completo, incluindo todas as manutenções realizadas neste documento, está disponibilizado no repositório do github que pode ser acessado através da URL:

<https://github.com/rodrigo-igti/paw-interativa1-final.git>

## Sugestão para Manutenção Evolutiva na Aplicação

Para praticar os conhecimentos obtidos sobre o desenvolvimento de aplicações Django, deixo como sugestão um ajuste na classe Serie e a implementação de uma nova classe, Episodio, conforme mostrado no diagrama de classes abaixo.

