Como usar o GitHub

No diretório RAIZ do projeto:

1) Git init
2) Git add .
3) Git config - -global user.email “email@dominio.com”
4) Git config - -global user.name “Nome usuário Github”
5) Git commit -m “first commit” (-am = força todos os arquivos)
6) Git remote add origin “path do repositório no github”
7) Git push - - set-upstream origin main ou master

Para atualizar os projetos:

1) Git add .
2) Git commit -m “nome da alteração realizada”
3) Git push - - set

Criando ambiente do projeto

1) Criar a pasta do projeto
2) Criar o ambiente virtual (python -m venv venv)
3) **Ativar o ambiente (.\venv\Scripts\activate)**
4) Instalar o Django (pip install Django)

Criando / Executando projeto Django

1) Criando o projeto (Django-admin startproject nome_principal_do_projeto **.**) **(Atenção** - Digite o ponto para o projeto ser criado na mesma pasta) - Usaremos o nome **core** para o projeto das aulas.
2) Executando o projeto (python **manage.py** runserver)

Estrutura/Arquivos do Projeto

1) Manage.py -> gerencia todas as ações do projeto (testes, migrates, execução)
2) Settings.py -> todas as configurações do projeto (app, bd, timezone etc)
3) __init__.py -> informa para o compilador que o projeto é um sistema python
4) Urls.py -> contém todas as rotas do projeto (www.filmes.com, admin etc)

Divisões das APPs dentro do projeto Django

1) Cada APP gerencia uma parte do projeto (gerenciar livros, leitores, autores etc, CRUD)

Criando App no Django

1) Python manage.py startapp filmes (nome do app que será criado)
2) Inserir a app criada no arquivo setttings.py na seção **INSTALLED_APPS**

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'filmes',
]
```

Criando o banco de dados (Migrations)

1) Criando o script com a estrutura da base de dados (python manage.py makemigrations) – Busca em todas as app além da estrutura padrão do Admin do Django
2) Executar o script para realmente gerar a estrutura do BD (python manage.py migrate)

Criando Super Usuário

1) Python manage.py createsuperuser
   Username: seu_Usuario
   Email adress: seu_email
   Password: senha que vc não vai esquecer!!!!
   Password(again):

Criando o models.py (Filmes)

Criar a classe dos campos da tabela que será criada no banco de dados

```python
class Filme(models.Model):
    id = models.AutoField(primary_key=True)
    titulo = models.CharField(max_length=100)
    sinopse = models.TextField()
    duracao = models.IntegerField()
    ano = models.IntegerField()
    genero = models.CharField(max_length=100)
    diretor = models.CharField(max_length=100)
    atores = models.CharField(max_length=100)
    # poster = models.ImageField(upload_to='posters/')

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

1) python manage.py makemigrations (cria o arquivo 0001 initial.py na pasta migrations)
2) python manage.py migrate
3) Para que a tabela apareça no ADMIN temos que registrar o modelo no **admin.py,** criando a classe FilmeAdmin, mostrando quais os campos serão mostrados na tela. (importar o modelo filme)

```python
from django.contrib import admin
from filmes.models import Filme

class FilmeAdmin(admin.ModelAdmin):
    list_display = ('titulo', 'ano', 'genero', 'diretor', 'atores')
    search_fields = ('titulo', 'ano', 'genero', 'diretor', 'atores')

admin.site.register(Filme, FilmeAdmin)
```

OBS: Configurações do arquivo **settings.py:**

1) Alterar LANGUAGE_CODE = 'pt-br'
2) Alterar TIME_ZONE = 'America/São_Paulo'

No terminal do VSCode digite: **python manage.py runserver**

Isso iniciará o servidor local, pressione a tecla **CTRL** e clique no link http://127.0.0.1:8000 para iniciar o projeto.
No navegador, digite a url para chamar a tela de admin: http://127.0.0.1:8000/**admin**

OBS.: Cadastre alguns filmes.

Criando chave estrangeira FK (models.py)

ATENÇÃO: Antes de realizar as alterações, excluir todos os registros da tabela filme.
Como iremos criar uma chave estrangera (FK), teremos um problema com a integridade referencial da tabela filmes com a tabela genero (Teremos outra solução em outra etapa do projeto).

Criar a classe de gênero acima da classe Filme

```
3  class Genero(models.Model):
4      id = models.AutoField(primary_key=True)
5      nome = models.CharField(max_length=100)
6
7      def __str__(self):
8          return self.nome
```

Alterar o campo gênero na classe Filme

```
genero = models.ForeignKey(Genero, on_delete=models.PROTECT, related_name='filmes')
```

Criar a migrations:

Python manage.py makemigrations

Python manage.py migrate

Alterar o arquivo Admin.py, incluindo a classe de gênero.

```
class GeneroAdmin(admin.ModelAdmin):
    list_display = ('nome')
    search_fields = ('nome')

admin.site.register(Genero, GeneroAdmin)
```

Inserir um registro de foto no Filme (models.py)

Incluir o campo poster do tipo **ImageField**

```
poster = models.ImageField(upload_to='posters/', blank=True, null=True)
```

ATENÇÃO: Instalar a biblioteca pillow (pip install pillow)

**** Como alteramos os campos de uma tabela no models.py temos que executar os comandos para criar a nova migration. Isso será executado toda vez que criarmos uma tabela no models ou realizar alguma alteração em campos ****

Python manage.py makemigrations

Python manage.py migrate

Criar a pasta **media/posters,** onde será salva as imagens do cadastro de filmes

Configura o arquivo **setttings.py** o caminho para armazenar imagens.

```
13   import os
14   from pathlib import Path
```

```
117  MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
118  MEDIA_URL = '/media/'
```

Alterar o arquivo **urls.py** para localizar a caminho para salvar as imagens

```
17  from django.contrib import admin
18  from django.urls import path
19  from django.conf import settings
20  from django.conf.urls.static import static
21
22  urlpatterns = [
23      path('admin/', admin.site.urls),
24  ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

No terminal do VSCode digite: **python manage.py runserver**

Isso iniciará o servidor local, pressione a tecla **CTRL** e clique no link http://127.0.0.1:8000 para iniciar o projeto.
No navegador, digite a url para chamar a tela de admin: http://127.0.0.1:8000/**admin**

Cadastro alguns gêneros de filme e alguns filmes. Agora teremos 2 tabelas no ADMIN.

VIEW / URLS

Exemplo sem o uso de templates (*somente para fins de esclarecimento das url/views*)

1) Na pasta principal do projeto acessar o arquivo **urls.py**
   a. Incluir uma url
   b. Configura a view que será chamada para esse url.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('filmes', filmes_view),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

```

2) Criar a view no arquivo **views.py** da app filmes

```
from django.shortcuts import render
from django.http import HttpResponse

def filmes_view(request):
    html = '''
    <html>
    <head>
        <title>Filmes</title>
    </head>
    <body>
        <h1>Filmes</h1>
        <ul>
            <li>Matrix</li>
            <li>Star Wars</li>
            <li>De volta para o futuro</li>
            <li>Harry Potter</li>
        </ul>
    </body>
    </html>
    '''
    return HttpResponse(html)

```

OBS: Visão do administrador(admin) e do usuário comum do site

No terminal do VSCode digite: **python manage.py runserver**

Isso iniciará o servidor local, pressione a tecla **CTRL** e clique no link http://127.0.0.1:8000 para iniciar o projeto.
No navegador, digite a url para chamar a tela de admin: http://127.0.0.1:8000/**filmes**

TEMPLATES

1) Dentro da app filmes criar uma pasta com nome **'templates',** dentro dessa pasta crie um arquivo com nome **filmes.html.**
2) Alterar o arquivo **view.py** para acessar o template correspondente

```python
from django.shortcuts import render
from filmes.models import Filme

def filmes_view(request):
    filmes = Filme.objects.all()
    return render(request, 'filmes.html', {'filmes': filmes})

```

Django template language

1) No arquivo **filmes.html** incluir a tag de bloco de conteúdo {% block contente %} e {% endblock %}
2) Importar o model filmes no arquivo **view.py** (from filmes.models import Filme)
3) Alterar o arquivo **filmes.html** para incluir os dados da tabela filmes

```
</head>
{% block content %}

<body>
    <h1>Filmes</h1>
        {% for filme in filmes %}
            <H3>  {{ filme.titulo }} | {{ filme.genero}} | {{ filme.diretor }} </H3>
        {% endfor %}
</body>
{% endblock %}
```

***** Teste novamente o projeto com a nova view. *****

No terminal do VSCode digite: **python manage.py runserver**

Isso iniciará o servidor local, pressione a tecla **CTRL** e clique no link http://127.0.0.1:8000 para iniciar o projeto.
No navegador, digite a url para chamar a tela de admin: http://127.0.0.1:8000/**filmes**

ORM (Object Relation Mapping)

1) Incluir um filtro para pesquisas no arquivo **views.py**, alterar o objects de ***all*** para ***filter*** (**projeto Aula 3 e 4).**

```
def filmes_view(request):
    filmes = Filme.objects.filter(titulo='Star Wars' )
```

Neste exemplo estamos pesquisado exatamente o nome do filme, então onde digitemos 'Star Wars' digite o nome do seu filme que deseja localizar.

```
4   def filmes_view(request):
5       # filmes = Filme.objects.filter(titulo='Star Wars' )  # busca por titulo
6       filmes = Filme.objects.filter(genero__nome='ficção' ) # busca por genero
7
8       return render(request, 'filmes.html' , {'filmes': filmes})
```

2) **Model__contains** (faz a pesquisa contendo somente a string) (**Atenção:** Utilizar **dois underlines (_ _).**

```
4   def filmes_view(request):
5       filmes = Filme.objects.filter(titulo__contains='Wars' )
6
7       return render(request, 'filmes.html' , {'filmes': filmes})
```

No terminal do VSCode digite: **python manage.py runserver**

Isso iniciará o servidor local, pressione a tecla **CTRL** e clique no link http://127.0.0.1:8000 para iniciar o projeto.
No navegador, digite a url para chamar a tela de admin: http://127.0.0.1:8000/**filmes**

Objeto request (views)

1) Por meio do **request** o usuário pode passar **parâmetros** para explicar o que realmente ele quer ver na pesquisa

   Ex. na url digitar:

   127.0.0.1:8000/filmes?search=Star Wars (Digite o nome do seu filme)

   No arquivo **views.py** incluir o comando **print(request)** para a ver o resultado da requisição no terminal de comando

2) Com o request podemos capturar o valor que o usuário quer pesquisar.

```
4  def filmes_view(request):
5      filmes = Filme.objects.all()
6      search = request.GET.get('search')
7      if search:
8          filmes = Filme.objects.filter(titulo__contains=search)
9      return render(request, 'filmes.html' , {'filmes': filmes})
10
```

3) Para ordenar a lista, após o filtro pode usar o **order_by:**
   a. Filme.objects.all()order_by('titulo')

Melhorando os templates e views

1) Alterar o arquivo **urls.py** para pesquisa encontrar a rota/url para pesquisar o filme por título

```
8   urlpatterns = [
9       path('admin/', admin.site.urls),
10      path('filmes/', filmes_view, name='filmes_list'),
11  ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Incluir o CSS do arquivo filmes.hml (github aulas 3 e 4)

Obs.: Lembrando, sou da época do monitor mono cromático, preto e branco e **NÃO** sou especialista em Front-end (CSS, bootstrap etc), vc pode incluir o seu CSS se preferir.

Criando o templates base

1) Na pasta principal do projeto **(core)** criar a pasta **templetes** copiar o arquivo **base.html** do Github das aulas 3 e 4.
2) No arquivo **settings.py** alterar o bloco de templates incluindo a nova pasta de templates em DIRS

```
45  TEMPLATES = [
46      {
47          'BACKEND': 'django.template.backends.django.DjangoTemplates',
48          'DIRS': ['core/templates'], # adicionei essa linha
49          'APP_DIRS': True,
50          'OPTIONS': {
51              'context_processors': [
52                  'django.template.context_processors.debug',
53                  'django.template.context_processors.request',
54                  'django.contrib.auth.context_processors.auth',
55                  'django.contrib.messages.context_processors.messages',
56              ],
57          },
58      },
59  ]
```

3) No arquivo **filmes.html** incluir o **extends** na primeira linha para poder usar o templates base.

```
1  {% extends "base.html" %}
2
3  {% block content %}
4      <style>
```

Forms no Django

1) Vamos começar incluindo uma nova rota no arquivo **urls.py** para direcionar para o formulário de cadastro de filmes.

```
10  urlpatterns = [
11      path('admin/', admin.site.urls),
12      path('filmes/', filmes_view, name='filmes_list'),
13      path('novo_livro/', novo_filmes_view, name='novo_filmes'),
14  ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
15
```

2) No arquivo **views.py** de filmes, criar a função que será a view que usará o novo formulário de cadastro de filmes. (Incluir o import no **url.py**)

```
11
12  def novo_filmes_view(request):
13      return 'novo filme'
14
```

```
7   from filmes.views import novo_filmes_view
```

3) Criar o template referente ao form de cadastro do novo filme. Na pasta templates de filmes criar o arquivo **novo_filme.html** (conforme exemplo do github)

```
1   {% extends "base.html" %}
2
3   {% block content %}
4       <form method="POST" enctype="multipart/form-data">
5           {% csrf_token %}
6
7           {{ novo_filme_form }}
8
9       </form>
10  {% endblock %}
```

4) Na pasta de filmes criar o arquivo **forms.py** (conforme exemplo do github)

```python
from django import forms
from filmes.models import Genero

class FilmeForm(forms.Form):
    titulo = forms.CharField(max_length=100)
    sinopse = forms.CharField(widget=forms.Textarea)
    duracao = forms.IntegerField()
    ano = forms.IntegerField()
    genero = forms.ModelChoiceField(Genero.objects.all())
    diretor = forms.CharField(max_length=100)
    atores = forms.CharField(max_length=100)
    poster = forms.ImageField()
```

5) Alterar o arquivo **views.py** para inserir os dados do formulário (importar o FilmeForm do arquivo **forms.py**)

```python
from filmes.forms import FilmeForm
```

```python
def novo_filmes_view(request):
    novo_filme_form = FilmeForm()
    return render(request, 'novo_filme.html', {'novo_filme_form': novo_filme_form})
```

No terminal do VSCode digite: **python manage.py runserver**

No navegador, digite a url para chamar a tela de admin: http://127.0.0.1:8000/**novo_filme/**

Melhorias no Forms Django

No arquivo **novo_filme.html,** podemos fazer algumas melhorias no html junto com o Django template language

Ex.

```
{{ novo_filme_form.as_p }}
```

O **.as_p** formata como parágrafo.

```
9         <table>
10            {{ novo_filme_form.as_table }}
11        </table>
12
```

O .**as_table** cria uma tabela com os dados do formulário.

Incluir o botão salvar

1) Continuando no arquivo **novo_filme.html** incluir o input do tipo submit para que os dados possam ser enviados para o servidor (POST)

```
<input type="submit" value="Adicionar Filme" class="btn btn-primary">
```

2) No arquivo **views.py** alterar a função **novo_vilmes_view** para verificar se o método usado é POST e enviar os dados para servidor, validando as informações e incluir o método **save()** para confirmar a inclusão no bando de dados. (Obs.: Import redirect)

```
13  def novo_filmes_view(request):
14      if request.method == 'POST':
15          novo_filme_form = FilmeForm(request.POST, request.FILES)
16          if novo_filme_form.is_valid():
17              novo_filme_form.save()
18              return redirect('filmes_list')
19      else:
20          novo_filme_form = FilmeForm()
21      return render(request, 'novo_filme.html', {'novo_filme_form': novo_filme_form})
22
```

Git

3) Teremos que subscrever a função **save()** no arquivo **forms.py** para o Django identificar a tabela onde serão incluídas as informações (Teremos outras maneiras para facilitar esse processo)

```
13
14      def save(self):
15          filme = Filme(
16              titulo=self.cleaned_data['titulo'],
17              sinopse=self.cleaned_data['sinopse'],
18              duracao=self.cleaned_data['duracao'],
19              ano=self.cleaned_data['ano'],
20              genero=self.cleaned_data['genero'],
21              diretor=self.cleaned_data['diretor'],
22              atores=self.cleaned_data['atores'],
23              poster=self.cleaned_data['poster']
24          )
25          filme.save()
26          return filme
```

4) Incluir um botão para cadastrar novo filme no **base.html.**

```
72  <header>
73    <nav>
74      <ul>
75        <li><a href="{% url 'filmes_list' %}">Listar Filmes</a></li>
76        <li><a href="{% url 'novo_filmes' %}">Cadasrar Filmes</a></li>
77      </ul>
78    </nav>
79  </header>
```

No terminal do VSCode digite: **python manage.py runserver**

No navegador, digite a url para chamar a tela de admin: http://127.0.0.1:8000/**novo_filme**/

Aperfeiçoando os Forms com ModelForm

1) No mesmo arquivo **forms.py** vamos criar uma classe herdando um **ModelForm** que buscara os campos que estão na tabela que deseja criar o formulário (no nosso exemplo será Filme).

```
28  class FilmesModelForm(forms.ModelForm):
29      class Meta:
30          model = Filme
31          fields = '__all__'
32
```

2) Agora teremos que alterar o arquivo **views.py** para buscar essa nova classe.

```
3  from filmes.forms import FilmesModelForm
```

```
13  def novo_filmes_view(request):
14      if request.method == 'POST':
15          novo_filme_form = FilmesModelForm(request.POST, request.FILES)
16          if novo_filme_form.is_valid():
17              novo_filme_form.save()
18              return redirect('filmes_list')
19      else:
20          novo_filme_form = FilmesModelForm()
21      return render(request, 'novo_filme.html', {'novo_filme_form': novo_filme_form})
```

Criando Validações dos Forms

O Dajngo possui várias regras prontas para validar os campos que estão sendo usados em cada cadastro.

1) No arquivo **forms.py**, vamos criar uma função para validar o mínimo de caracteres aceitos no campo sinopse.

```
33      def clean_sinopse(self):
34          sinopse = self.cleaned_data.get('sinopse')
35          if len(sinopse) < 20:
36              self.add_error('sinopse', 'A sinopse deve ter pelo menos 20 caracteres.')
37          return sinopse
38
```

Obs.: Toda função de validação, necessariamente tem que começar com **clean_.**

2) Para reforçar, vamos criar uma validação para o ano de lançamento dos filmes ficar entre 1900 e 2100.

```
39      def clean_ano(self):
40          ano = self.cleaned_data.get('ano')
41          if ano < 1900 or ano > 2100:
42              self.add_error('ano', 'O ano deve estar entre 1900 e 2100.')
43          return ano
```

Autenticação de Usuários

O Django, assim que criamos um projeto já é criado toda a estrutura de cadastro de usuário, onde podemos dar permissão de acesso aos cadastros e pesquisas do nosso projeto. Agora vamos validar se o usuário tem permissão para cadastrar um filme ou não.

1) Vamos criar uma **App** para controlar os usuários que poderão realizar o cadastro.
   a. Python manage.py startapp usuarios
   b. Incluir no arquivo **settings.py** na seção **INSTALLED_APPS** a nova app.

```
23  INSTALLED_APPS = [
24      'django.contrib.admin',
25      'django.contrib.auth',
26      'django.contrib.contenttypes',
27      'django.contrib.sessions',
28      'django.contrib.messages',
29      'django.contrib.staticfiles',
30      'filmes',
31      'usuarios',
32  ]
```

2) Teremos de criar uma rota para acessar o cadastro de usuários. No arquivo **urls.py** incluir a rota para acessar a **view** que terá o *form* para cadastro de usuário.

```
10  urlpatterns = [
11      path('',filmes_view, name='filmes_list'),
12      path('users/', usuario_view, name='usuario'),
13      path('filmes/', filmes_view, name='filmes_list'),
14      path('admin/', admin.site.urls),
15      path('filmes/', filmes_view, name='filmes_list'),
16      path('novo_filme/', novo_filmes_view, name='novo_filmes'),
17  ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
18
```

Obs.: O erro apresentando em **usuario_view,** é porque ainda não criamos a view responsável para isso no arquivo **views.py.**

3) No arquivo **views.py** da **app usuarios**, incluir a view para inclusão de usuários.

```
4  def usuario_view(request):
5      user_form = UserCreationForm()
6      return render(request, 'usuario.html', {'user_form': user_form})
```

4) Observe que estamos usando o UserCreationForm, que o Django já fornece pronta. Para usar basta fazer o import.

```
2    from django.contrib.auth.forms import UserCreationForm
```

e no arquivo **urls.py** fazer o import da nova view.

```
8    from usuarios.views import usuario_view
```

5) Criar a pasta **templates** na pasta da app **usuario**, dentro da pasta templates criar um arquivo **usuario.html.**

```
1   {% extends "base.html" %}
2
3   {% block content %}
4
5       <form method="POST">
6
7           {% csrf_token %}
8           {{ user_form.as_p }}
9
10          <input type="submit" value="Cadastrar Usuário" class="btn btn-primary">
11
12      </form>
13
14  {% endblock %}
```

6) Vamos incluir no arquivo **views.py** (do usuario) a validação do request se for **POST** com os dados do usuário redireciona para a tela de login, se for **GET** e redirecionar para o form vazio de cadastro de login.

```
1   from django.shortcuts import render, redirect
2   from django.contrib.auth.forms import UserCreationForm
3
4   def usuario_view(request):
5       if request.method == 'POST':
6           user_form = UserCreationForm(request.POST)
7           if user_form.is_valid():
8               user_form.save()
9               return redirect('login')
10
11      else:
12          user_form = UserCreationForm()
13
14      return render(request, 'usuario.html', {'user_form': user_form})
15
```

Obs.: importar o **redirect**

Ajustando o formulário de cadastro de usuários: incluir no arquivo **usuario.html** a tag ** ** com a class “form-group” para organizar os campos na tela.

```
{% extends "base.html" %}

{% block content %}

    <form method="POST">

        {% csrf_token %}
        <div class="form-group">
            {{ user_form.username.errors }}
            <label>Nome de Usuário:</label>
            {{ user_form.username }}<br>
        </div>
        <div class="form-group">
            {{ user_form.password1.errors }}
            <label>Senha:</label>
            {{ user_form.password1 }}<br>
        </div>
        <div class="form-group">
            {{ user_form.password2.errors }}
            <label>Confirmação de Senha:</label>
            {{ user_form.password2 }}<br>
        </div>

        <input type="submit" value="Cadastrar Usuário" class="btn btn-primary">

    </form>

{% endblock %}
```

Obs.: Como ainda não temos o form de login, quando incluir um usuário a Django irá redirecionar para a tela de login que não existe e veremos um erro.

7) Criar no arquivo **urls.py** a rota para o login.

```
17      path('login/', login_view, name='login')
```

8) No arquivo **views.py** do usuario, criar a função para permitir o login do usuário. (Nesse caso não iremos fazer do zero, vamos utilizar a estrutura que o Django disponibiliza).

```
def login_view(request):
    login_form = AuthenticationForm()
    return render(request, 'login.html', {'login_form': login_form})
```

Obs.: importar o AuthenticatinForm

```
from django.contrib.auth.forms import UserCreationForm, AuthenticationForm
```

Importar a view de login no arquivo **urls.py**:

```
from usuarios.views import usuario_view, login_view
```

9) Na pasta templates de usuarios, criar o arquivo **login.html.**

```
{% extends "base.html" %}

{% block content %}

    <form method="POST">
        {% csrf_token %}
        <table>
            {{ login_form.as_table }}
        </table>
        <input type="submit" value="Login" class="btn btn-primary">
    </form>

{% endblock %}
```

10) Vamos alterar o arquivo **views.py** para verificar se o usuário e a senha dígitos são validos. Na função login_view vamos inserir o método **authenticate.**

```
def login_view(request):
    if request.method == 'POST':
        usuario = request.POST['username']
        senha = request.POST['password']
        user = authenticate(request, username=usuario, password=senha)
        if user is not None:
            login(request, user)
            return redirect('filmes_list')
        else:
            login_form = AuthenticationForm()

    else:
        login_form = AuthenticationForm()
    return render(request, 'login.html', {'login_form': login_form})
```

Obs.: Importar o authenticate e login.

```
from django.contrib.auth import authenticate, login
```

11) Vamos criar a função de logout para finalizar a etapa de autenticação de usuários, para isso vamos incluir uma rota no arquivo **urls.py** para chamar a view de logout.

```
    path('logout/', logout_view, name='logout')
```

12) No arquivo views.py de usuarios, incluir a função de logout.

```
31 def logout_view(request):
32     logout(request)
33     return redirect('filmes_list')
```

Obs.: Importar a função de logout.

```
3   from django.contrib.auth import authenticate, login, logout
```

Melhorando a navegação no Site

No arquivo **base.html**, vamos fazer vairas alterações utilizando o **Django template language**, para verificar se o usuário já está logado, se o botão CADASTRAR deve ficar visível ou não, se o botão SAIR estará disponível etc.

```
74      <ul>
75      |
76      |  <li><a href="{% url 'filmes_list' %}">Filmes</a></li>
77      |
78      |  {% if user.is_authenticated %}
79      |    <li> Olá, {{ user.username }}</li>
80      |    <li><a href="{% url 'novo_filmes' %}">Cadasrar Filmes</a></li>
81      |    <li><a href="{% url 'logout' %}">Logout</a></li>
82      |  {% else %}
83      |    <li><a href="{% url 'login' %}">Login</a></li>
84      |    <li><a href="{% url 'usuario' %}">Registre</a></li>
85      |  {% endif %}
86      |
87      </ul>
```

Class Based Views

Vamos reescrever as views de uma forma mais clara e organizada, utilizando esse novo recurso poderemos ter códigos sem a necessidade de incluir muitas validações como IFs e utilizaremos recurso prontos do framework Django.

A documentação do Django é bem completa, acesse o site https://docs.djangoproject.com/pt-br/5.2/ref/class-based-views/

1) No arquivo **views.py** de filmes vamos criar uma classe para substituir a function view, que ainda será a classe básica, teremos várias melhorias.

```
14  class FilmesView(View):
15      def get(self, request):
16          filmes = Filme.objects.all().order_by('titulo')
17          search = request.GET.get('search')
18          if search:
19              filmes = Filme.objects.filter(titulo__contains=search)
20
21          return render(request, 'filmes.html', {'filmes': filmes})
```

Obs.: Import o Django.view

```
4   from django.views import View
```

2) No arquivo **urls.py**, vamos alterar a maneira de chamar a view e importar a nova classe FilmesView.

```
15      # path('filmes/', filmes_view, name='filmes_list'),
16      path('filmes/', FilmesView.as_view(), name='filmes_list'),
```

```
7   from filmes.views import novo_filmes_view, FilmesView
```

3) Reescrever a views para cadastrar o novo filme.

```
38      def post(self, request):
39          novo_filme_form = FilmesModelForm(request.POST, request.FILES)
40          if novo_filme_form.is_valid():
41              novo_filme_form.save()
42              return redirect('filmes_list')
43          return render(request, 'novo_filme.html', {'novo_filme_form': novo_filme_form})
```

4) Altear no arquivo **urls.py** a rota para a novo filme com a nova view.

```
15      path('novo_filme/', NovoFilmesView.as_view(), name='novo_filmes'),
```

```
6   from filmes.views import FilmesView, NovoFilmesView
```

Class ListView - Generic views

Agora, iremos usar a classe ListView como mais recursos para criar uma lista.

1) Novamente no arquivo **views.py** de filmes, vamos criar uma outra view com a classe listview (Começamos do modo mais básico com mais código digitado e vamos evoluindo reduzindo a quantidade de código e usando as classes mais complexas).

```
24  class FilmesListView(ListView):
25      model = Filme
26      template_name = 'filmes.html'
27      context_object_name = 'filmes'
28
29      def get_queryset(self):
30          # super() acessa o queryset padrão da classe ListView (classe pai)
31          filmes = super().get_queryset().order_by('titulo')
32          search = self.request.GET.get('search')
33          if search:
34              filmes = filmes.filter(titulo__contains=search)
35          return filmes
```

Class CreateView – Generic views

1) Continuando no arquivo **views.py** vamos criar a classe NovoFilmes usando a nova classe CreateView (Agora veremos a economia de código e um código mais limpo).

```
60  class NovoFilmeCreateView(CreateView):
61      model = FilmesListView
62      form_class = FilmesModelForm
63      template_name = 'novo_filme.html'
64      success_url = '/filmes_list/'
```

2) Agora estamos usando os métodos e propriedades da classe CreateView. Para que o formulário de cadastro de filmes continue funcionando, teremos que alterar o arquivo **novo_filme.html** para acessar os dados de **form.as_table.**

```
 7          <table>
 8
 9              {{ form.as_table }}
10
11          </table>
```

Usando essa nova classe não temos que se preocupar com qual métodos está sendo usado, se é POST ou GET, isso tudo ficou automático.

Class DetailView - Generic views

Para que o usuário posso alterar ou excluir um filme, vamos criar um view com os detalhes do registro selecionado e mostrar um botão de alterar ou excluir.

1) No arquivo **views.py** incluir a nova classe de detalhes.

```
66  class CarDetailView(DetailView):
67      model = Filme
68      template_name = 'filme_detail.html'
```

2) Na pasta **templates**, criar o arquivo **filme_detail.html** (utilizar o arquivo disponível no Github ou criar o próprio estilo CSS). Abaixo o código necessário para que o form detalhes funcione.

```
83  <div class="filme-card">
84    {% if filme.poster %}
85        <img src="{{ filme.poster.url }}" alt="{{ filme.titulo }} - {{ filme.ano }}">
86    {% else %}
87        <img src="/media/posters/default.jpg" alt="{{ filme.titulo }} - {{ filme.ano }}">
88    {% endif %}
89      <h2>{{ object.titulo }} {{ object.ano }}</h2>
90      <p><strong>Sinopse:</strong> {{ filme.sinopse }}</p>
91      <p><strong>Duração:</strong> {{ filme.duracao }}</p>
92      <p><strong>Diretor:</strong> {{ filme.diretor }}</p>
93      <p><strong>Atores:</strong> {{ filme.atores }}</p>
94      <p><strong>Gênero:</strong> {{ filme.genero }}</p>
95  </div>
```

3) Incluir a rota para buscar o registro selecionado e que envie a ID do filme que deseja alterar ou excluir.

```
16      path('filme/<int:pk>/', FilmeDetailView.as_view(), name='filme_detail'),
```

```
6   from filmes.views import FilmesListView, NovoFilmeCreateView, FilmeDetailView
```

4) Vamos transformar o CARD de cada filme em um LINK para acessar o form de detalhes. (Incluir somente a tag <a href= ....>)

```
<div class="filme-grid">
  {% if filmes %}
    {% for filme in filmes %}
      <a href="{% url 'filme_detail' filme.id %}">
       <div class="filme-card">
         {% if filme.poster %}
           <img src="{{ filme.poster.url }}" alt="{{ filme.titulo }}">
         {% else %}
           <img src="/media/posters/default.jpg" alt="{{ filme.titulo }}">
         {% endif %}
         <h2>{{ filme.titulo }} </h2>
         <p>{{ filme.ano }}</p>
       </div>
       </a>
    {% endfor %}
  {% else %}
    <p class="no-results">Nenhum filme encontrado.</p>
  {% endif %}
</div>
```

Class UpdateView- Generic views

1) No mesmo arquivo **views.py** vamos criar a classe para fazer o update (alteração) do registro selecionado.

```
class FilmeUpdateView(UpdateView):
    model = Filme
    form_class = FilmesModelForm
    template_name = 'filme_update.html'
    success_url = '/filmes/'
```

2) No arquivo **urls.py**, incluir a rota para a view de update. Nessa rota também vamos enviar o campo ID para que o Django possa localizar o registro desejado.

```
    path('filme/<int:pk>/update/', FilmeUpdateView.as_view(), name='filme_update'),
```

3) Na pasta filmes/templates criar o arquivo **filme_update.html** onde os registros serão alterados. (Disponível no guithub pelicanojr)

Class DeleteView- Generic views

1) Continuando no arquivo **views.py** vamos usar agora a **deleteview.**

```
76  class DeleteView(DeleteView):
77      model = Filme
78      template_name = 'filme_delete.html'
79      success_url = '/filmes/'
```

2) No arquivo **urls.py**, incluir a rota para a view de delete. Nessa rota também vamos enviar o campo ID para que o Django possa localizar o registro desejado.

```
18      path('filme/<int:pk>/delete/', FilmeDeleteView.as_view(), name='filme_delete'),
```

3) Na pasta filmes/templates criar o arquivo **filme_delete.html** onde os registros serão alterados. (Disponível no guithub pelicanojr)
4) No arquivo **filme.detail.html**, vamos incluir o botão **deletar.**

```
<a href="{% url 'filme_delete' pk=object.pk %}" class="btn btn-danger">Deletar</a>
```

Obs.: Para alterar a rota de retorno após a alterações. (Opcional)

Vamos fazer um exemplo de subscrever um método. O Dev pode querer personalizar qualquer método disponível no Django. No nosso exemplo vamos subscrever o **get_success_url** no arquivo **views.py** para o retorno ser na view de detalhes e não para a lista de filmes.

```
4   from django.urls import reverse_lazy
```

```
70  class FilmeUpdateView(UpdateView):
71      model = Filme
72      form_class = FilmesModelForm
73      template_name = 'filme_update.html'
74
75      def get_success_url(self):
76          return reverse_lazy('filme_detail', kwargs={'pk': self.object.pk})
77
```

Para fazer uma verificação do usuário para não mostrar os botões de alterar e deletar se o usuário não estiver logado. No filmes_deltail.html, incluir **% if %** na que conténs os botões.

```
96  {%if user.is_authenticated%}
97    <div class="buttons-container">
98      <a href="{% url 'filme_update' pk=object.pk %}" class="btn btn-primary">Editar</a>
99      <a href="{% url 'filme_delete' pk=object.pk %}" class="btn btn-danger">Deletar</a>
100   </div>
101 {% endif %}
```

Autorização das Views

Autorização de usuário no Django define **o que cada usuário pode fazer** na base de dados — como **visualizar, criar, editar ou excluir registros** — com base em **permissões** e **grupos**. Isso é gerenciado pelo sistema de autenticação do Django (User, Permission, Group).

Obs.: Usaremos os ***decorators*** que em Python (e no Django) são funções que **modificam o comportamento de outras funções** sem alterar seu código original.

1) Para isso, no arquivo **views.py**, vamos fazer os *imports* de *login_required* e *method_decorator*.

```
5   from django.contrib.auth.decorators import login_required
6   from django.utils.decorators import method_decorator
```

2) Vamos incluir os *decorators* para validar se o usuário está logado em cada view.

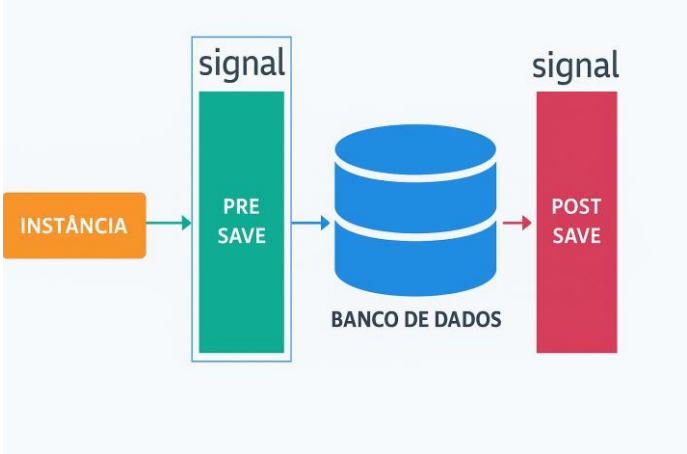
```
22  @method_decorator(login_required(login_url='login'), name='dispatch')
23  # dispatch é o método que processa a requisição
24  class NovoFilmeCreateView(CreateView):
```

```
34  @method_decorator(login_required(login_url='login'), name='dispatch')
35  class FilmeUpdateView(UpdateView):
```

```
43  @method_decorator(login_required(login_url='login'), name='dispatch')
44  class FilmeDeleteView(DeleteView):
```

Signals

Signals são uma forma do Django avisar automaticamente quando algo importante acontece no sistema, como a criação, atualização ou exclusão de um registro.



Eles permitem que você dispare ações automaticamente antes ou depois desses eventos sem precisar alterar diretamente a lógica principal da aplicação.

Exemplo de uso:

- Quando um usuário é criado, enviar um e-mail de boas-vindas.
- Quando um filme é apagado, deletar também suas fotos.

1) Para incluir *signals* em nosso projeto, vamos criar na app **filmes** a arquivo **signals.py** e adicionar a função ***ready*** no arquivo **apps.py.**

```
8       def ready(self):
9           import filmes.signals
```

2) Agora no arquivo **signals.py** vamos criar as funções para ativar os *signals*.

```
1   from django.db.models.signals import pre_save, pre_delete, post_save, post_delete
2   from django.dispatch import receiver
3   from filmes.models import Filme
4
5   # O decorator @receiver(receptor) conecta o sinal a uma função
6   # que será chamada quando o sinal for enviado sender é o
7   # modelo que está enviando o sinal, instance é a
8   # instância do modelo que está sendo salvo ou deletado
9   @receiver(pre_save, sender=Filme)
10  def filme_pre_save(sender, instance, **kwargs):
11      print('Filme pre save')
12      print(instance.titulo)
```

3) Execute o projeto digitando **python manage.py runsever,** cadastre um filme e no terminar verifique a mensagem (print) que foi enviada antes da inclusão do registro no banco de dados.

Para um exemplo mais prático vamos criar uma função para salvar um texto no campo de sinopse do filme caso o usuário não digite nenhuma informação. (**ATENÇÃO** – Vou deixar como desafio para vocês alterarem as validações que temos para o campo sinopse. **Forms.py** e **Models.py**)

```
14  @receiver(pre_save, sender=Filme)
15  def filme_pre_save(sender, instance, **kwargs):
16      if not instance.sinopse:
17          instance.sinopse = 'Sinopse não informada que em breve será atualizada utilizando o Django Signals.'
18
```

Integrando Django com a API da OpenAI

Vamos melhorar nosso projeto incluindo funcionalidades para utilizar os modelos de IA, no caso vamos usar o ChatGPT como exemplo. Para isso teremos que criar uma conta e gerar uma chave para acessar o modelo. Isso tem um custo que OpenAI cobra por cada requisição. Até o momento estavam disponibilizando $5,00 para teste.

1) Crie a conta no OpenAI https://platform.openai.com/
2) Na opção no menu esquerdo entre em API Keys e click no botão no canto direto **+Create new secret key**
3) Instalar a lib da OpenAI: **pip install openai**
4) Na raiz do projeto (filmes) criar uma pasta (no nosso exemplo será **API_openai**).
5) Dentro desta pasta criar o arquivo **client.py**

6) Alterar o arquivo **signals.py** para usar a função da OpenAI

Roteiro como inicializar o servidor AWS