Departamento de Engenharia Informática

Base de Dados e Análise de Informação

2024/2025

Mestrado em Engenharia Biomédica

Projeto Final - *Movies&Series*

Duarte Ferreira (2020235393)

Cristiana Azevedo (2020221121)

Coimbra, 5 de janeiro de 2025

Índice

Capítulo 1 - Introdução

Este Projeto Final da unidade curricular de Bases de Dados e Análise de Informação teve como objetivo o desenvolvimento de um website para a avaliação de filmes e séries (*Movies&Series*). O website envolve a apresentação de uma listagem de cinco séries e filmes, onde o utilizador pode consultar os detalhes de cada conteúdo, ler as respetivas *reviews* e adicionar novas *reviews*. Além disso, é também possível aceder a informações detalhadas sobre os géneros abordados nas séries e filmes, bem como sobre os atores e realizadores envolvidos.

Capítulo 2 - Avaliação

2.1 Realizações do Grupo e Tarefas Pendentes

O principal objetivo definido para este trabalho foi realizado com sucesso, uma vez que todas as funcionalidades planeadas foram implementadas proporcionando aos utilizadores interagir com a plataforma de forma eficiente e intuitiva. No entanto, considerando a vasta gama de funcionalidades que websites deste tipo podem oferecer, existem algumas melhorias que poderiam ser aplicadas para tornar a plataforma ainda mais valiosa. Entre os principais passos futuros a serem realizados, destacam-se:

→ Permitir que os utilizadores indiquem se já viram determinado filme;
→ Melhorar a plataforma com mais filmes, séries, atores e realizadores;
→ Adicionar mais informações aos filmes e séries, como fotos e trailers correspondentes;
→ Criar uma secção dedicada aos filmes e séries mais vistos ou com melhores avaliações;
→ Desenvolver uma página inicial com login de utilizador e opção de criação de conta. No login, o utilizador deverá inserir o seu nome de utilizador e palavra-passe. Na criação de conta, serão solicitados o email, nome de utilizador e palavra-passe.

2.2 Contribuições Individuais de Cada Participante do Grupo

Tarefa	Duarte	Cristiana
Construção da narrativa do projeto	X	X
Diagrama ER	X	X
Adicionar informação à base de dados	X	
Models.py	X	
Página de Perfis dos Utilizadores	X	
Homepage		X
Página de Filmes/Séries		X
Página dos Géneros/Realizadores/Atores	X	
Página do Histórico	X	
Página das *Reviews*		X
Inserção de novas *reviews*		X
Procura de filmes/séries na barra de pesquisa		X
Relatório	X	X

2.3 Horas de Esforço por Membro

A tabela seguinte ilustra o número aproximado de horas utilizadas por cada membro do grupo para a realização deste trabalho:

Nome do Membro	Horas de Esforço
Duarte Ferreira	27 h
Cristiana Azevedo	25 h

2.4 Autoavaliação do Individual e do Grupo

A autoavaliação individual de cada membro e a avaliação geral do grupo encontram-se na tabela seguinte:

Nome do Membro	Avaliação
Duarte Ferreira	85%
Cristiana Azevedo	85%
Grupo	85%

Capítulo 3 - Descrição do Trabalho

Este trabalho teve como objetivo desenvolver uma plataforma para exibir informações sobre filmes e séries. O projeto foi dividido em duas etapas principais: a criação de um Diagrama de Entidade-Relacionamento (ER) e a implementação de um website utilizando o framework Django.

Numa primeira instância, nós desenvolvemos o Diagrama ER que serviu como base conceitual para a estruturação da base de dados. Este diagrama teve como propósito modelar as entidades do sistema e definir o relacionamento entre elas. Esta abordagem permitiu obter uma base robusta para o desenvolvimento dos próximos passos do projeto, assegurando integridade e coerência da informação do sistema.

Com o Diagrama ER completo, partimos para a segunda etapa deste trabalho, que consistiu na implementação de um website com recurso ao Django. Nesta fase desenvolvemos um sistema web dinâmico, eficiente e em constante contacto com a base de dados. O website foi criado para exibir informações sobre filmes e séries e tem funcionalidades como:

- **Seleção de utilizadores:** Permite selecionar o utilizador pretendido, personalizando a experiência e garantindo acesso às informações e interações associadas ao mesmo.
- **Visualização de informações sobre atores:** Permite consultar detalhes sobre os atores que participaram nos filmes e séries disponíveis na plataforma.
- **Informações sobre diretores:** Os utilizadores podem visualizar dados sobre os diretores responsáveis pelos filmes e séries, incluindo as suas contribuições.

- **Géneros de filmes e séries:** Para cada ator ou diretor, é possível visualizar os géneros associados aos filmes e séries, acompanhados por uma breve descrição que contextualiza as suas características.

- **Catálogo completo de filmes e séries:** Os utilizadores têm acesso a uma listagem de todos os filmes e séries presentes na base de dados, organizada de forma intuitiva.

- **Detalhes de cada filme ou série:** É possível consultar informações detalhadas sobre qualquer filme ou série, incluindo título, sinopse, data de lançamento, duração para os filmes e o número de temporadas e episódios para as séries.

- **Publicação de uma *review*:** Cada filme e série têm uma secção onde é possível o utilizador avaliar o filme/série, classificando-o de 0-5 e escrevendo um comentário sobre o mesmo.

- **Histórico de visualizações:** A plataforma regista o histórico de visualizações dos utilizadores, indicando as datas em que cada filme ou série foi assistido.

- **Avaliações de filmes e séries:** Existe uma secção dedicada às *reviews* feitas pelo utilizador, permitindo que ele reveja as notas e comentários atribuídos a cada obra.

- **Barra de pesquisa:** Inclui uma funcionalidade de pesquisa que possibilita encontrar rapidamente um filme ou série específico, facilitando a navegação.

Capítulo 4 - Diagrama ER

Este capítulo apresenta o Diagrama de ER desenvolvido no contexto do projeto com o objetivo de estruturar e organizar as informações, permitindo uma visão clara das entidades desenvolvidas e do seu relacionamento.

4.1 Entidades e Atributos

Neste subcapítulo as entidades e os atributos são detalhados, com o intuito de esclarecer a estrutura e as informações que cada componente armazena.

4.1.1 Person

A entidade “Person” representa a pessoa que irá interagir com o modelo e tem como atributos:

- id (PK) (Int)- Identificador único de cada utilizador
- name (VChar) - Nome do utilizador
- email (VChar)l - Email do utilizador
- password (VChar)- Password do utilizador

4.1.2 Visualization

A entidade “Visualization” exibe todos os filmes e séries assistidos pelo usuário, com a respetiva data de visualização e possui os seguintes atributos:

- id (PK) (Int): Identificador único da visualização.
- view_date (Date): Data em que a filme/série foi visualizada.

4.1.3 Review

A entidade “Review” representa a avaliação fornecida pelo usuário para cada *media* (filme ou série) assistida, incluindo um comentário, uma nota e a data da avaliação. Os seus atributos são os seguintes:

- id (PK) (Int): Identificador único da avaliação.
- rev_date (Date): Data da avaliação.
- rating (Int): Nota atribuída à *media*.
- comment (VChar): Comentário associado à avaliação.

4.1.4 Media

A entidade “Media” representa as informações detalhadas de uma *media*, seja ela um filme ou uma série. Os atributos associados a esta entidade são os seguintes:

- id (PK): Identificador único de *media*.
- media_type (VChar): Tipo de *media* (*Film* ou *Serie*).
- title (VChar): Título de *media*.
- release_date (Date): Data de lançamento.
- synopsis (VChar): Sinopse de *media*.

4.1.5 Film

A entidade “Film” estabelece uma relação de herança com a entidade “Media”, representando informações específicas relacionadas a filmes, enquanto herda os atributos gerais de “Media”. Além dos atributos herdados, “Film” inclui:

- duration (VChar): Duração do filme.

4.1.6 Serie

A entidade “Serie”, à semelhança da entidade “Film”, também estabelece uma relação de herança com a entidade “Media”. Assim, “Serie” destaca as informações específicas das séries, enquanto herda os atributos gerais de “Media”. Os atributos específicos de “Serie” são:

- seasons (Int): Número de temporadas.
- episodes (Int): Número de episódios.

4.1.7 Genre

A entidade "Genre" é responsável por representar os diferentes géneros de *media*. Os seus atributos são os seguintes:

- id (PK) (Int): Identificador único do género.
- name (VChar): Nome do género.
- description (VChar): Descrição do género.

4.1.8 Actor

A entidade "Actor" representa os atores associados às produções de *media*. Os seus atributos são os seguintes:

- id (PK) (Int): Identificador único do ator.
- name (VChar): Nome do ator.
- birth_date (Date): Data de nascimento.
- nationality (VChar): Nacionalidade.

4.1.9 Director

A entidade "Director" representa os diretores responsáveis pela realização das produções de *media*. Os seus atributos são dados por:

- id (PK) (Int): Identificador único do diretor.
- name (VChar): Nome do diretor.
- birth_date (Date): Data de nascimento.
- nationality (VChar): Nacionalidade.

4.2 Relacionamento entre as entidades

Os relacionamentos entre as entidades modeladas no diagrama incluem:

- ➔ **Person & Visualization:** A relação entre Person e Visualization é de **1:1**, uma vez que uma filme/série está associado a apenas 1 utilizador.
- ➔ **Visualization & Person:** A relação entre Visualization e Person é de **0:n**, já que a entidade Visualization é a entidade fraca e é dependente da entidade Person. Ou seja, a visualização só ocorre se existir o utilizador que a assista.
- ➔ **Person & Review:** A relação entre Person e Review é de **1:1**, uma vez que uma avaliação está associada a apenas 1 utilizador.
- ➔ **Review & Person:** A relação entre Review e Person é de **0:n**, já que a entidade Review é a entidade fraca e é dependente da entidade Person.
- ➔ **Visualization & Media:** A relação entre Visualization e Media é de **0:n**, já que a entidade Visualization é a entidade fraca e é dependente da entidade Media. Ou seja, a visualização só ocorre se existir o filme/série.
- ➔ **Media & Visualization:** A relação entre Media e Visualization é de **1:1**, já que uma visualização só ser feita por um único *media*.
- ➔ **Review & Media:** A relação entre Review e Media é de **0:n**, já que a entidade Review é a entidade fraca e é dependente da entidade Media.

- ➔ **Media & Review:** A relação entre Media e Review é de **1:1**, pois uma avaliação só pode ser relativa a um único *media*.
- ➔ **Media & Genre:** A relação entre Media e Genre é de **1:n**, uma vez que uma *media* pode pertencer a um ou a vários géneros cinematográficos
- ➔ **Genre & Media:** A relação entre Genre e Media é de **1:n**, pois um género pode ser associado a várias *medias*.
- ➔ **Media & Actor:** A relação entre Media e Actor é de **1:n**, já que uma *media* pode ter um ou vários atores e um ator pode participar de várias *medias.*
- ➔ **Actor & Media:** A relação entre Actor e Media é de **1:n**, uma vez que um ator pode participar em apenas uma *media* ou em várias
- ➔ **Media & Director:** A relação entre Media e Director é de **1:n**, uma vez que uma *media* pode ter um ou vários diretores.
- ➔ **Director & Media:** A relação entre Director e Media é de **1:n**, uma vez que um diretor pode participar de uma ou várias *medias.*
- ➔ **Media & Film/Serie:** A relação entre Media e Film/Serie é caracterizada como uma **relação de herança**, onde Media atua como uma entidade genérica das entidades Film e Serie.

4.3 Diagrama Entidade-Relacionamento

O diagrama Entidade-Relacionamento efetuado para a elaboração deste trabalho final encontra-se representado abaixo:

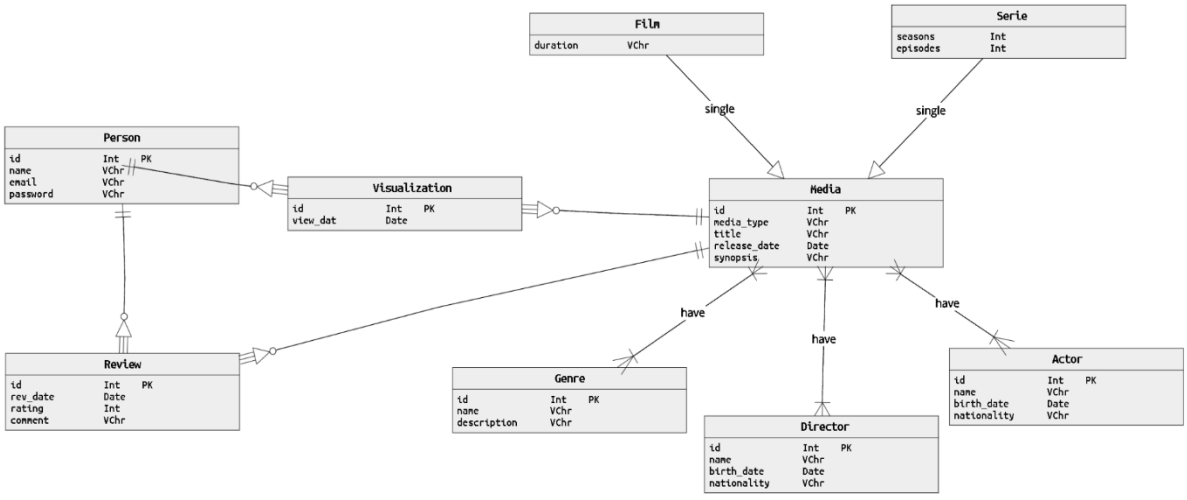

Figura 1 - Diagrama ER simplificado

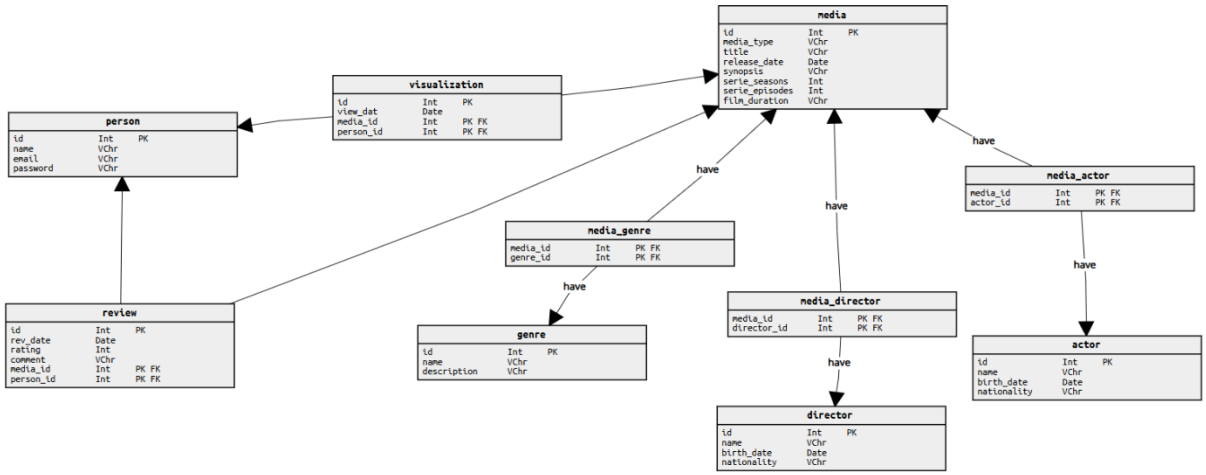

Figura 2 - Diagrama ER físico

Capítulo 5 - Website

A segunda fase deste projeto consiste na criação de um website utilizando o framework Django. Nesta fase, o primeiro passo foi criar os pastas do projeto, media e brMedia, através dos comandos:

- py -m django startproject media
- py manage.py startapp brMedia

De seguida, procedemos à elaboração do ficheiro *models.py* com base no ER desenvolvido anteriormente. Nesse processo cada entidade foi desenvolvida de forma independente, com o objetivo de modelar as diferentes partes do sistema e garantir a correta interação com a base de dados.

O ficheiro *models.py* do Django é onde definimos as classes que representam as tabelas da base de dados. Cada classe criada é automaticamente mapeada para uma tabela na base de dados, e cada atributo dentro da classe corresponde a uma coluna dessa tabela. Estas entidades são criadas como classes *Python*, recorrendo a dados específicos como *CharField*, *IntegerField*, *DateTimeField*, entre outros. A relação entre as entidades é feita recorrendo a comandos como o *ForeignKey*, *ManyToManyField* e *OneToOneField*. O *models.py* que nós criamos encontra-se representado nas **Figuras 3, 4 e 5**.

```python
from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=255, null=False, blank=False)
    email = models.EmailField(unique=True, max_length=255, blank=True)
    password = models.CharField(max_length=255, null=False, blank=True)

    def __str__(self):
        return f"Username: {self.name} for email: {self.email}"


class Genre(models.Model):
    name = models.CharField(max_length=255, null=True, blank=True)
    description = models.CharField(max_length=512, null=True, blank=True)

    def __str__(self):
        return f"Genre: {self.name}" if self.name else "Unnamed Genre"


class Director(models.Model):
    name = models.CharField(max_length=255, null=True, blank=True)
    birth_date = models.DateField(null=True, blank=True)
    nationality = models.CharField(max_length=255, null=True, blank=True)

    def __str__(self):
        return f"Director: {self.name}" if self.name else "Unnamed Director"


class Actor(models.Model):
    name = models.CharField(max_length=255, null=True, blank=True)
    birth_date = models.DateField(null=True, blank=True)
    nationality = models.CharField(max_length=100, null=True, blank=True)

    def __str__(self):
        return f"Actor:{self.name}" if self.name else "Unnamed Actor"

```

Figura 3 - Representação da classe Person, Genre, Director e Actor no *models.py*

```python
class Media(models.Model):
    title = models.CharField(max_length=255, null=False, blank=False)
    media_type = models.CharField(max_length=10, choices=[('serie', 'Serie'), ('film', 'Film')])
    release_date = models.DateField(null=True, blank=True)
    synopsis = models.CharField(max_length=512, null=False, blank=False)
    directors = models.ManyToManyField(Director, related_name='media_director', blank=True)
    actors = models.ManyToManyField(Actor, related_name='media_actor', blank=True)
    genre = models.ManyToManyField(Genre, related_name='media_genre', blank=True)

    class Meta:
        abstract = False

    def __str__(self):
        if self.title:
            return f"{self.media_type}: {self.title} released on {self.release_date}"
        return 'Untitled media'


class Film(Media):
    duration = models.CharField(max_length=255, null=True, blank=True)

    def __str__(self):
        return f"Film: {self.title}" if self.title else "Unnamed Film"


class Series(Media):
    seasons = models.IntegerField(null=True, blank=True)
    episodes = models.IntegerField(null=True, blank=True)

    def __str__(self):
        return f"Serie: {self.title}" if self.title else "Unnamed Series"

```

Figura 4- Representação da classe Film, Media e Series no *models.py*

```python
class Visualization(models.Model):
    person = models.ForeignKey(Person, on_delete=models.CASCADE)
    media = models.ForeignKey(Media, on_delete=models.CASCADE)
    view_date = models.DateField(null=True, blank=True)

    def __str__(self):
        if self.person and self.media:
            return f"{self.person.name} viewed {self.media.title} on {self.view_date}"
        return "Unnamed Visualization"


class Review(models.Model):
    person = models.ForeignKey(Person, on_delete=models.CASCADE)
    media = models.ForeignKey(Media, on_delete=models.CASCADE)
    rev_date = models.DateField(null=True, blank=True)
    rating = models.IntegerField(null=True, blank=True)
    comment = models.CharField(max_length=512, null=True, blank=True)

    def __str__(self):
        if self.person and self.media:
            return f"Review by {self.person.name} for {self.media.title} on {self.rev_date}"
        return "Unnamed Review"

```

Figura 5- Representação da classe Visualization e Review no *models.py*

Após a criação do ficheiro *models.py*, utilizámos os seguintes comandos para realizar as migrações para a base de dados 'brMedia', gerida através do *pgAdmin*.:

- python manage.py makemigrations brMedia
- python manage.py migrate

Como resultado deste processo, foram geradas as tabelas abaixo, às quais adicionamos informação recorrendo aos scripts disponibilizados no **Capítulo 7**.

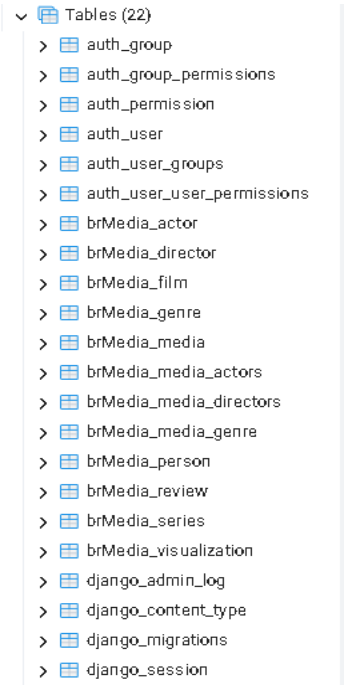

Figura 6- Tabelas criadas para a base de dados brMedia

Com a ligação à base de dados estabelecida, foi possível desenvolver as páginas do website. Para isso, foram considerados três tipos de ficheiros principais: *views.py*, *urls.py* e os ficheiros HTML.

No ficheiro *views.py*, foram definidas as funções responsáveis por obter dados específicos da base de dados e enviá-los para os templates HTML. Os ficheiros HTML foram criados para cada página do website, definindo a estrutura e o conteúdo visual que os utilizadores irão ver. No ficheiro *urls.py*, foram estabelecidas as associações entre as rotas (URLs) e as funções definidas em *views.py*, permitindo que a aplicação saiba qual view deve ser chamada para cada URL específico.

O ficheiro *urls.py* desenvolvido está apresentado na **Figura 7**. No total, foram desenvolvidos 9 ficheiros HTML: *actors.html, directors.html, genres.html, homepage.html, media.html, reviews.html, search.html, select_user.html, visualization.html*. Um exemplo destes ficheiros, *actors.html*, está apresentado no **Capítulo 7**.

No ficheiro *views.py* foram elaboradas as onze funções apresentadas no *urls.py*:

- **Função *select_profile***
  - URL associado (exemplo): http://127.0.0.1:8000/brMedia/profile/

  Esta função é responsável por recolher os dados úteis para a criação da página onde o utilizador pode selecionar o seu perfil. Primeiro, nesta função, são obtidos todos os registos de utilizadores na tabela 'Person' da base de dados. Depois, toda a informação recolhida é enviada para o template *select_user.html* - **Figura 8, 9**.

- **Função *user_profile***
  - URL associado (exemplo): http://127.0.0.1:8000/brMedia/user_profile/?user=2

  Auxilia na exibição do perfil do utilizador selecionado, onde são apresentadas as listas de todos os géneros, atores, diretores, filmes e séries disponíveis. Para isso, primeiro, obtém o id do utilizador a partir dos parâmetros GET. De seguida, recolhe todas as informações sobre géneros, atores e diretores disponíveis na base de dados para depois passar essas informações para o template *homepage.html* – **Figura 10, 11.**

- **Função *films*, Função *series***
  - URL associado (exemplo): http://127.0.0.1:8000/brMedia/films/?user=2; http://127.0.0.1:8000/brMedia/series/?user=2

  Esta função pretende recolher as informações sobre todos os filmes/séries disponíveis, como título, data de lançamento, atores, diretores, géneros e *reviews* associados. No caso dos filmes também é recolhido a duração do mesmo, e no caso das séries, o número de temporadas e episódios. Para isso, é obtido o id do utilizador e o tipo de *media* a ser mostrado a partir dos parâmetros GET. De seguida são recolhidos todos os filmes/séries, os dados das suas relações many-to-many: atores, diretores, géneros e ainda as reviews associadas a cada filme/série. No final, toda a informação recolhida é enviada para o template *media.html*, onde será apresentada de forma organizada ao utilizador – **Figura 12, 13, 14, 15.**

- **Função *genres*, Função *directors*, Função *actors***
  - URL associado (exemplo): http://127.0.0.1:8000/brMedia/genres/Action/?user=2; http://127.0.0.1:8000/brMedia/actors/Leonardo%20DiCaprio/?user=2; http://127.0.0.1:8000/brMedia/directors/Christopher%20Nolan/?user=2

Estas funções têm como objetivo obter o género/diretor/ator solicitado pelo utilizador. Para isso, cada função obtém o id do utilizador e as informações procuradas pelo mesmo a partir dos parâmetros GET. De seguida, recolhe a informação da base de dados respetiva ao género/diretor/ator procurado e redireciona-a para o template apropriado (*genres.html, directors.html, actors.htm*) - **Figura 16, 17, 18, 19, 20, 21.**

- **Função *visualization*, Função *reviews***
  - URL associado (exemplo): http://127.0.0.1:8000/brMedia/visualization?user=2; http://127.0.0.1:8000/brMedia/reviews?user=2;

Estas funções pretendem recuperar todas as visualizações/avaliações realizadas pelo utilizador. Para isso, cada função obtém o id do utilizador a partir dos parâmetros GET e recolhe da base de dados as informações da tabela 'Visualization'/'Review' associadas a esse utilizador. No final, redireciona a informação para o template *visualization.html/ reviews.html* para ser apresentada ao utilizador de forma estruturada – **Figura 22, 23, 24, 25.**

- **Função *submit_review***
  - URL associado (exemplo): http://127.0.0.1:8000/brMedia/submit-review?id=2&comment=Amazing%21&rating=5&media_title=Inception

Permite que o utilizador submeta uma nova avaliação para um *media* específico. Para isso, a função obtém o id do user, o comentário, a classificação e o título do *media* a partir dos parâmetros GET. De seguida, cria uma nova instância na tabela 'Review', associando a review ao utilizador e ao *media* correspondente – **Figura 26, 27.**

- **Função *search***
  - URL associado (exemplo): http://127.0.0.1:8000/brMedia/search?query=break&user=2

Permite a pesquisa de *media* (filmes ou séries) com base numa procura na barra de pesquisa do website por parte do utilizador. Para isso, a função obtém o id do user e a consulta de pesquisa (query) a partir dos parâmetros GET. Divide a consulta em palavras e procura no título de todos os *media* existentes. Para cada *media* correspondente, recolhe todas as suas informações (como atores, diretores, géneros, sinopse e reviews). No final, passa os resultados da pesquisa para o template *search.html*. – **Figura 28, 29.**

A seguir são apresentadas as Figuras que demonstram todas as funcionalidades do website, a par com o código nos *views.py*.

```python
from django.urls import path

from . import views

urlpatterns = [
    path('films/',views.films, name='films'),
    path('series/',views.series,name='series'),
    path('genres/<str:genre_name>/', views.genres, name='genres'),
    path('directors/<str:director_name>/', views.directors, name='directors'),
    path('actors/<str:actor_name>/', views.actors, name='actors'),
    path('profile/',views.select_profile, name='select_profile'),
    path('user_profile/', views.user_profile, name='user_profile'),
    path('visualization',views.visualization, name='visualization'),
    path('reviews',views.reviews,name='reviews'),
    path('submit-review', views.submit_review, name='submit_review'),
    path('search', views.search,name='search')
]
```

Figura 7 - Representação de urls.*py*

```python
def select_profile(request):
    template=loader.get_template('brMedia/select_user.html')
    people = Person.objects.all()
    context = {
        'users':people,
    }
    return HttpResponse(template.render(context, request))
```

Figura 8 - Representação da função *select_profile* no *views.py*

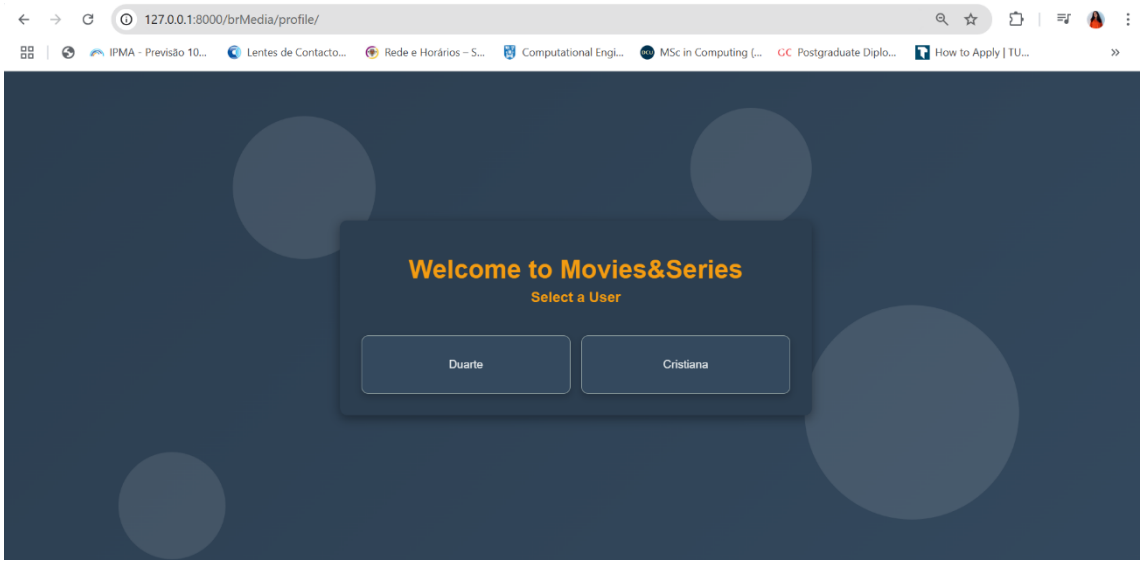

Figura 9 – Website relacionada com *select_profile* do *views.py*

```python
def user_profile(request):
    user_id = request.GET.get('user')
    user = Person.objects.get(id=user_id)
    all_genre = Genre.objects.all()
    all_actors = Actor.objects.all()
    all_directors = Director.objects.all()

    context = {
        'userID': user_id,
        'user_name':user.name,
        'all_genre': all_genre,
        'all_actors': all_actors,
        'all_directors': all_directors,
    }
    template=loader.get_template('brMedia/homepage.html')
    return HttpResponse(template.render(context, request))
```

Figura 10 - Representação da função *user_profile* no *views.py*

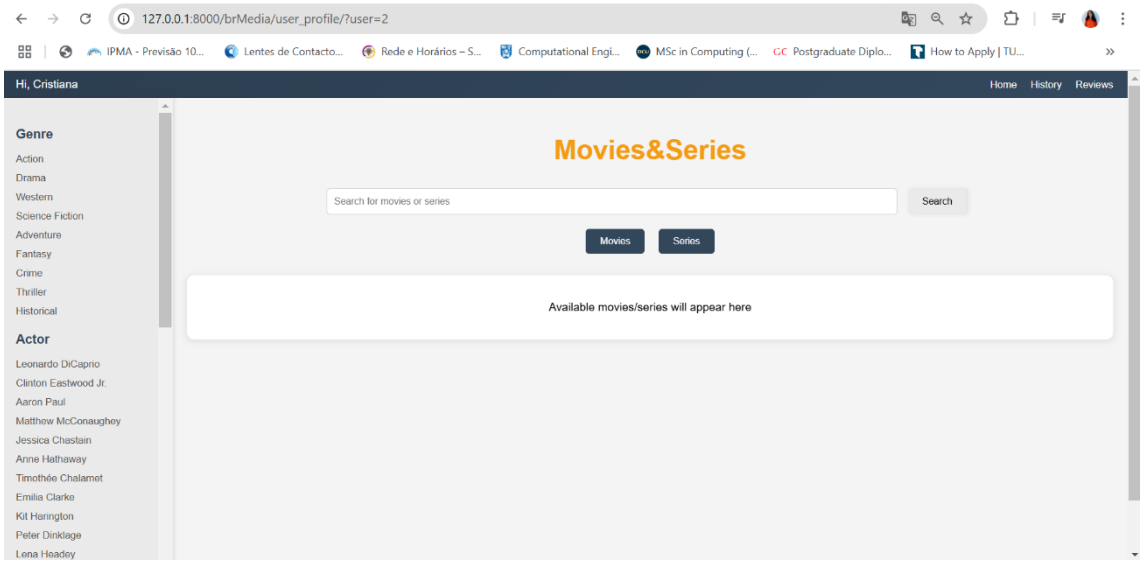

Figura 11 - Website relacionada com *user_profile* do *views.py*

```python
def films(request):
    user_id = request.GET.get('user')
    user = Person.objects.get(id=user_id)
    all_genre = Genre.objects.all()
    all_actors = Actor.objects.all()
    all_directors = Director.objects.all()


    # Filtra apenas os filmes (fazendo o join com a tabela de Media) e as suas respetivas relações many-to-many
    all_films = Film.objects.select_related('media_ptr').prefetch_related('actors', 'directors', 'genre').all()

    films_data = []
    for film in all_films:

        # Recolher as reviews associadas ao filme
        reviews = Review.objects.filter(media_id=film.media_ptr.id)
        data_reviews = [
                {
                    'review_date': str(review.rev_date),
                    'rating': review.rating,
                    'comment': review.comment,
                    'reviewer': Person.objects.get(id=review.person_id).name,
                }
                for review in reviews
            ]
        json_data_reviews = json.dumps(data_reviews)


        films_data.append({
            'title': film.media_ptr.title,
            'release_date': film.media_ptr.release_date,
            'duration': film.duration,
            'actors': ", ".join(actor.name for actor in film.actors.all()),
            'directors': ", ".join(director.name for director in film.directors.all()),
            'genres': ", ".join(genre.name for genre in film.genre.all()),
            'synopsis': film.synopsis,
            'reviews': json_data_reviews,
        })

    context = {
        'userID': user_id,
        'user_name':user.name,
        'all_genre': all_genre,
        'all_actors': all_actors,
        'all_directors': all_directors,
        'all_films': films_data,
        'type_media': 'film',
    }
    template=loader.get_template('brMedia/media.html')
    return HttpResponse(template.render(context, request))
```

Figura 12 - Representação da função *films* no *views.py*

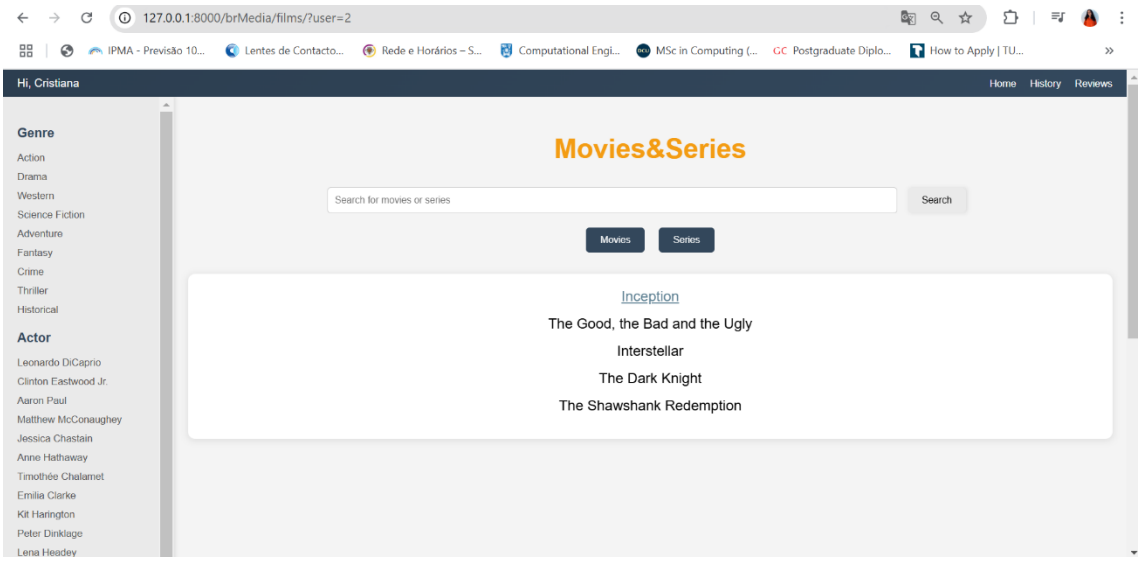

Figura 13 - Website relacionada com *films* do *views.py*

```python
def series(request):
    user_id = request.GET.get('user')
    user = Person.objects.get(id=user_id)
    all_genre = Genre.objects.all()
    all_actors = Actor.objects.all()
    all_directors = Director.objects.all()

    all_series = Series.objects.select_related('media_ptr').prefetch_related('actors','directors','genre').all()


    series_data = []
    for serie in all_series:

        # Recolher as reviews associadas a serie
        reviews = Review.objects.filter(media_id=serie.media_ptr.id)
        data_reviews = [
                {
                    'review_date': str(review.rev_date),
                    'rating': review.rating,
                    'comment': review.comment,
                    'reviewer': Person.objects.get(id=review.person_id).name,
                }
                for review in reviews
            ]
        json_data_reviews = json.dumps(data_reviews)


        series_data.append({
            'title': serie.media_ptr.title,
            'release_date': serie.media_ptr.release_date,
            'episodes': serie.episodes,
            'seasons': serie.seasons,
            'actors': ", ".join(actor.name for actor in serie.actors.all()),
            'directors': ", ".join(director.name for director in serie.directors.all()),
            'genres': ", ".join(genre.name for genre in serie.genre.all()),
            'synopsis': serie.synopsis,
            'reviews': json_data_reviews ,
        })


    context = {
        'userID': user_id,
        'user_name':user.name,
        'all_genre': all_genre,
        'all_actors': all_actors,
        'all_directors': all_directors,
        'all_series': series_data,
        'type_media': 'serie',
    }
    template=loader.get_template('brMedia/media.html')
    return HttpResponse(template.render(context, request))
```

Figura 14 - Representação da função *series* no *views.py*

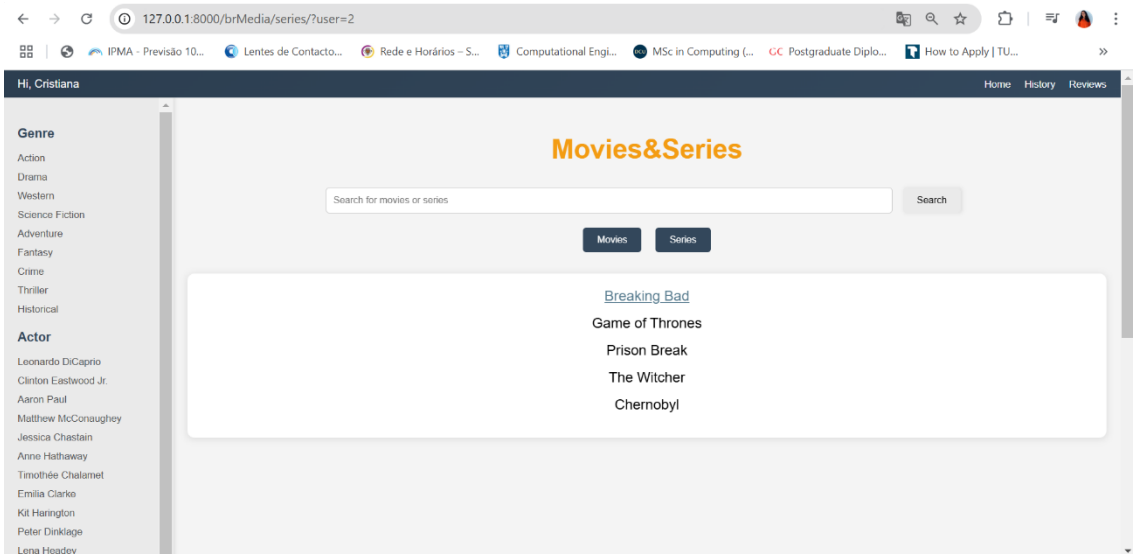

Figura 15 - Website relacionada com *series* do *views.py*

```python
def genres(request, genre_name):
    user_id = request.GET.get('user')
    genre = Genre.objects.get(name=genre_name)
    user = Person.objects.get(id=user_id)
    media= genre.media_genre.all()
    all_genre = Genre.objects.all()
    all_actors = Actor.objects.all()
    all_directors = Director.objects.all()
    context = {
        'genre': genre,
        'user_name':user.name,
        'userID':user_id,
        'media': media,
        'all_genre': all_genre,
        'all_actors': all_actors,
        'all_directors': all_directors,
    }

    template = loader.get_template('brMedia/genres.html')
    return HttpResponse(template.render(context, request))
```

Figura 16 - Representação da função *genres* no *views.py*

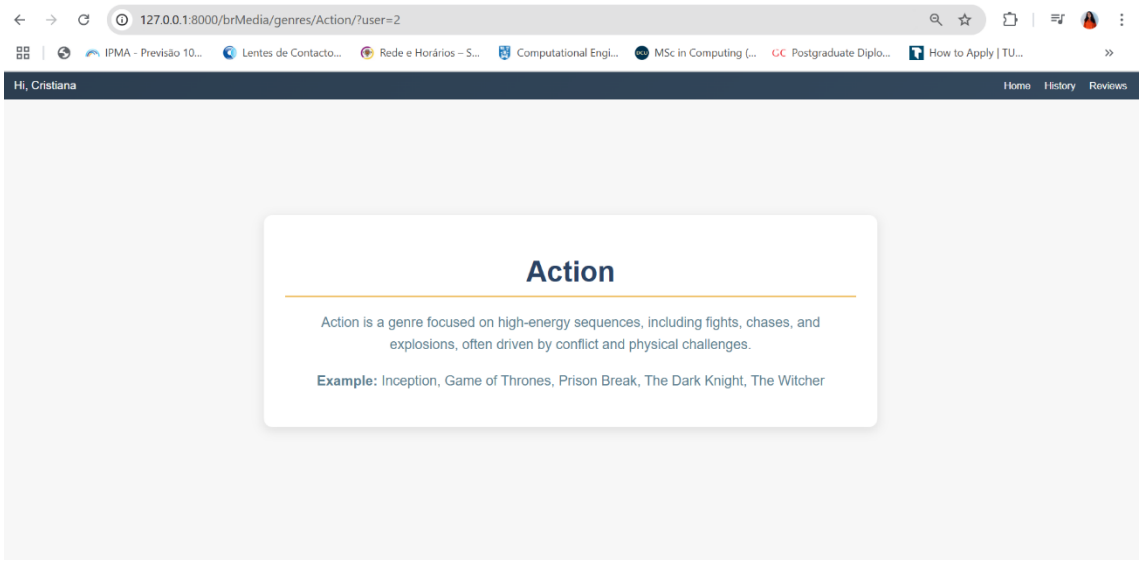

Figura 17 - Website relacionada com *genres* do *views.py*

```python
def actors(request, actor_name):
    user_id = request.GET.get('user')
    actor=Actor.objects.get(name=actor_name)
    user = Person.objects.get(id=user_id)
    media = actor.media_actor.all()
    all_genre = Genre.objects.all()
    all_actors = Actor.objects.all()
    all_directors = Director.objects.all()

    context = {
        'userID': user_id,
        'user_name':user.name,
        'actor': actor,
        'media': media,
        'all_genre': all_genre,
        'all_actors': all_actors,
        'all_directors': all_directors,
    }
    template=loader.get_template('brMedia/actors.html')
    return HttpResponse(template.render(context, request))
```

Figura 18 - Representação da função *actors* no *views.py*

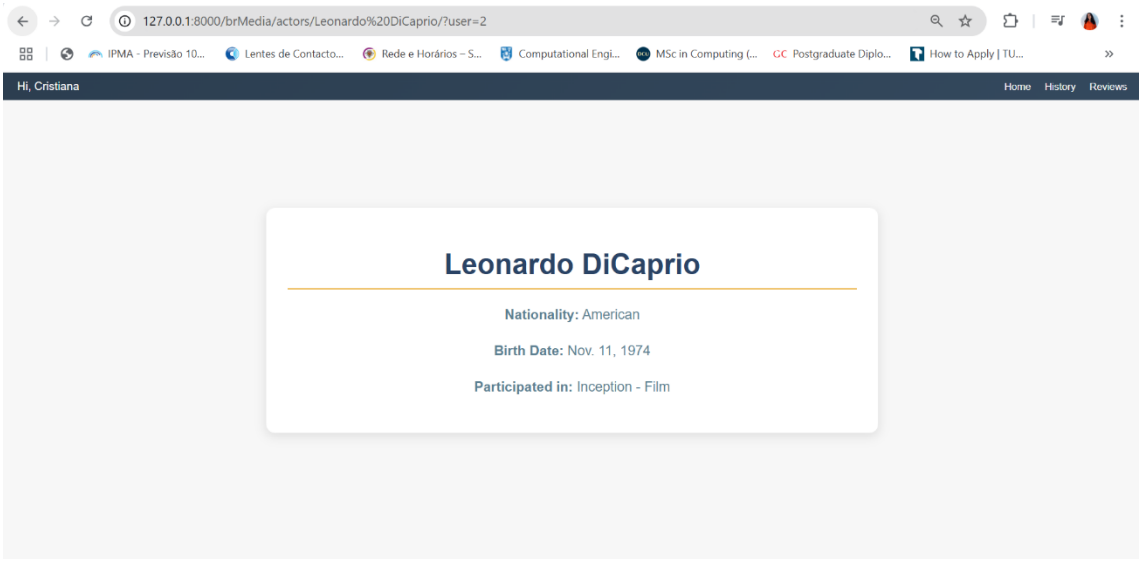

Figura 19 - Website relacionada com *actors* do *views.py*

```python
def directors(request, director_name):
    user_id = request.GET.get('user')
    director=Director.objects.get(name=director_name)
    user = Person.objects.get(id=user_id)
    media = Media.objects.all()
    all_genre = Genre.objects.all()
    all_actors = Actor.objects.all()
    all_directors = Director.objects.all()

    context = {
        'userID': user_id,
        'user_name':user.name,
        'director': director,
        'media':media,
        'all_genre': all_genre,
        'all_actors': all_actors,
        'all_directors': all_directors,
    }
    template=loader.get_template('brMedia/directors.html')
    return HttpResponse(template.render(context, request))
```

Figura 20 - Representação da função *directors* no *views.py*

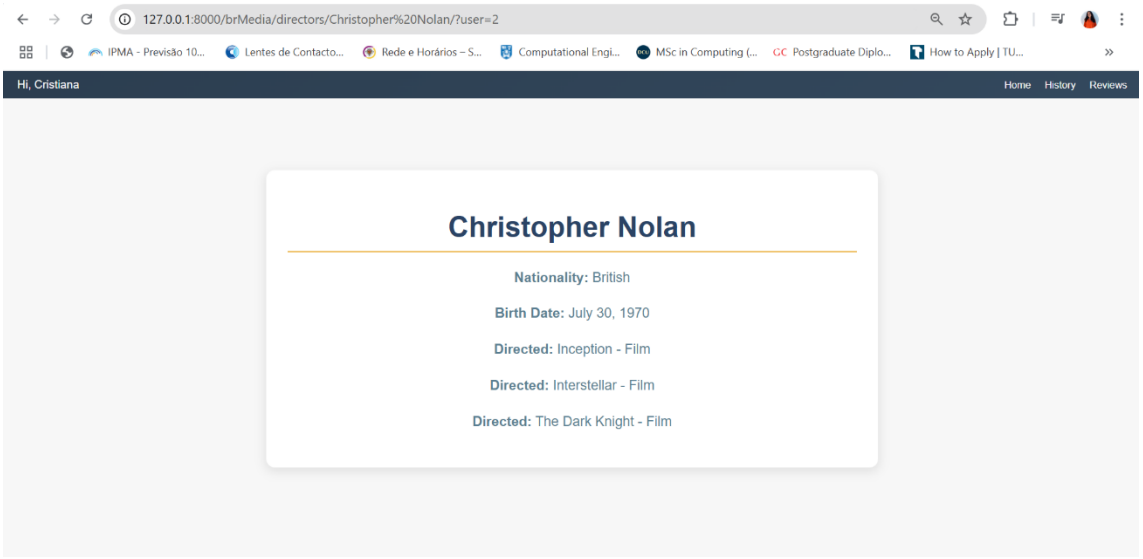

Figura 21 - Website relacionada com *actors* do *views.py*

```python
def visualization(request):
    user_id = request.GET.get('user')
    user = Person.objects.get(id=user_id)
    visualizations = Visualization.objects.filter(person=user).order_by('-view_date')
    all_genre = Genre.objects.all()
    all_actors = Actor.objects.all()
    all_directors = Director.objects.all()

    context = {
        'userID': user_id,
        'user_name': user.name,
        'visualizations': visualizations,
        'all_genre': all_genre,
        'all_actors': all_actors,
        'all_directors': all_directors,

    }
    template = loader.get_template('brMedia/visualization.html')
    return HttpResponse(template.render(context, request))
```

Figura 22 - Representação da função *visualization* no *views.py*

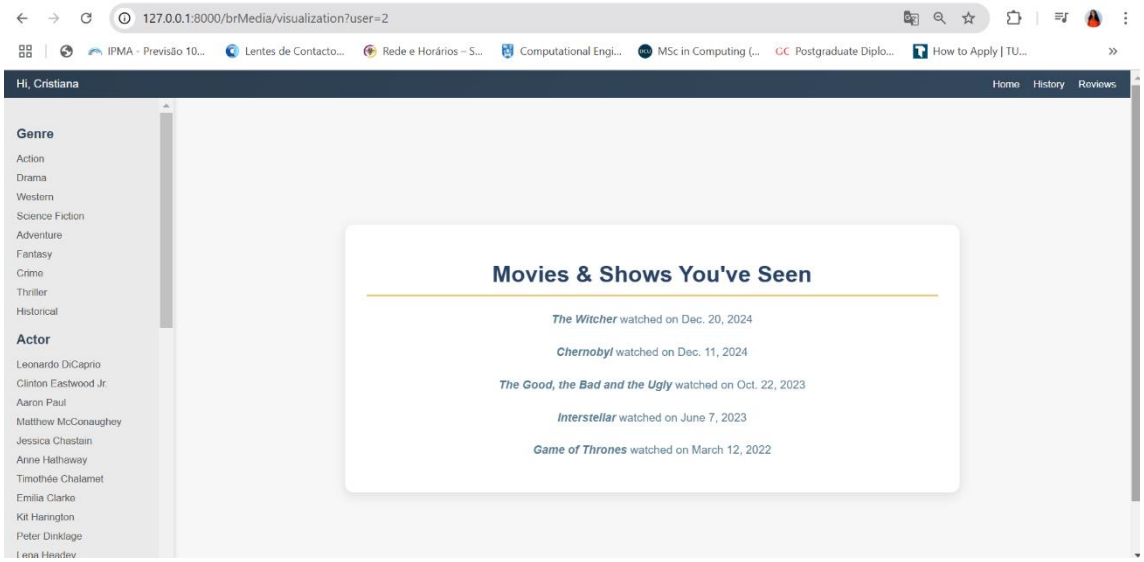

Figura 23 - Website relacionada com *visualization* do *views.py*

```python
def reviews(request):
    user_id = request.GET.get('user')
    user = Person.objects.get(id=user_id)
    all_genre = Genre.objects.all()
    all_media = Media.objects.all()
    all_actors = Actor.objects.all()
    all_directors = Director.objects.all()

    all_reviews = Review.objects.all()

    review_boolean = False
    data_reviews = []
    for review in all_reviews:
        if review.person_id == int(user_id):
            review_boolean = True
            actual_media = Media.objects.get(id=review.media_id)
            title = actual_media.title

            data_reviews.append({
                'title': title,
                'review_date': str(review.rev_date),
                'rating': review.rating,
                'comment': review.comment,
                'reviewer': Person.objects.get(id=review.person_id).name,
            })

    context = {
        'userID': user_id,
        'user_name':user.name,
        'all_genre': all_genre,
        'all_actors': all_actors,
        'all_directors': all_directors,
        'all_media': data_reviews,
        'review_boolean': review_boolean
    }
    template=loader.get_template('brMedia/reviews.html')
    return HttpResponse(template.render(context, request))
```

Figura 24 - Representação da função *reviews* no *views.py*

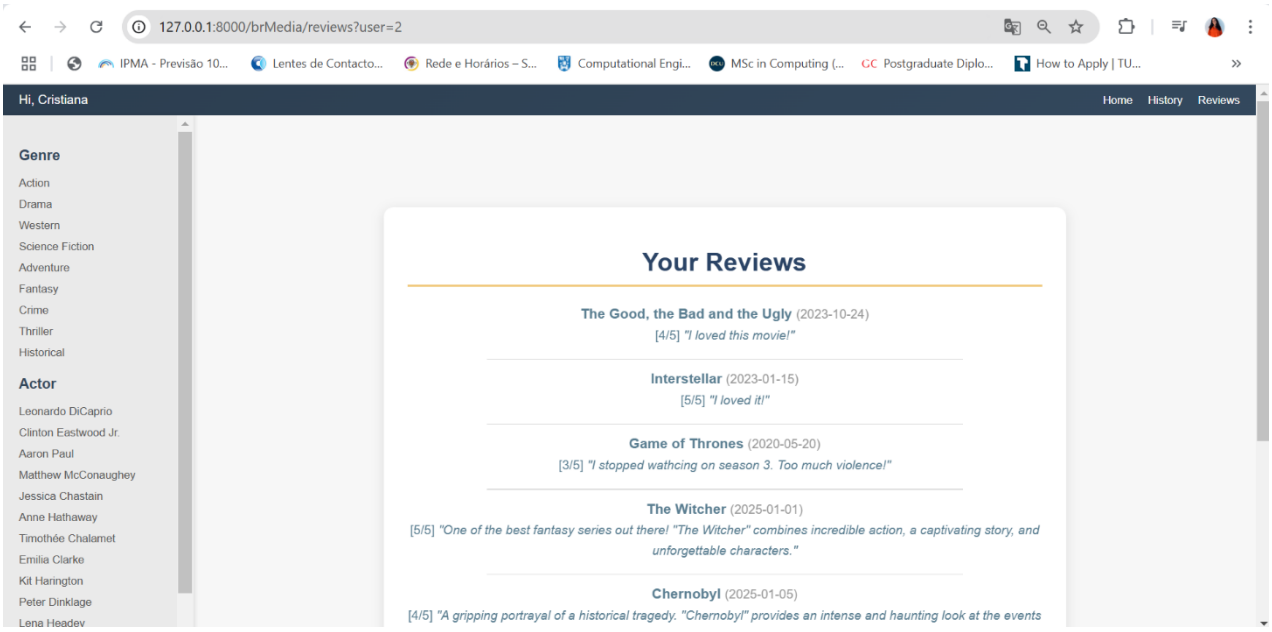

Figura 25 - Website relacionada com *reviews* do *views.py*

```python
def submit_review(request):

    user_id = request.GET.get('id')
    user = Person.objects.get(id=user_id)

    comment = request.GET.get('comment')
    rating = request.GET.get('rating')
    title1 = request.GET.get('media_title')
    print("AAAAAAAAA", comment, rating, title1)
    if not comment or not rating:
        print('error: Todos os campos são obrigatórios!')

    try:
        media = Media.objects.get(title=title1)
    except Media.DoesNotExist:
        return HttpResponse("Media not found", status=404)

    # Salvar no postgres
    review = Review(
        rev_date=datetime.date.today(),
        rating=int(rating),
        comment=comment,
        media_id=media.id,
        person_id=user_id
    )
    review.save()

    print('message:Review successfully saved')
    return redirect(request.META.get('HTTP_REFERER'))
```

Figura 26 - Representação da função *submit_review* no *views.py*

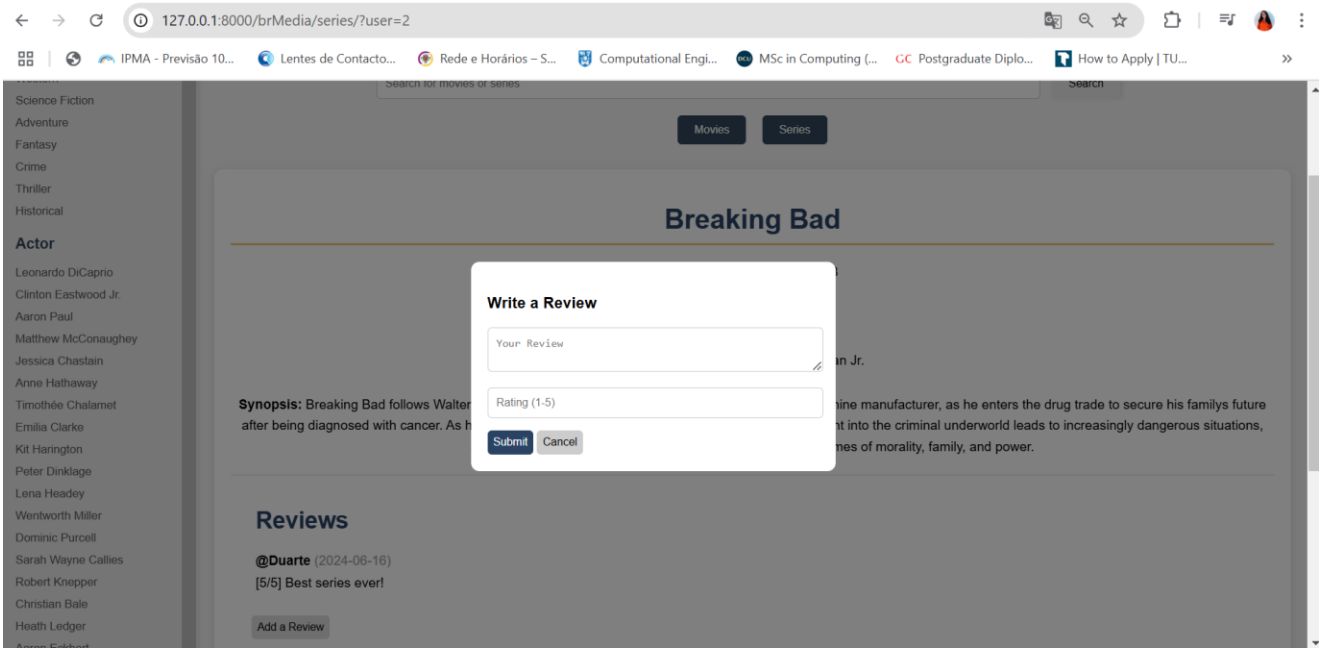

Figura 27 - Website relacionada com *submit_review* do *views.py*

```python
def search(request):
    user_id = request.GET.get('user')
    user = Person.objects.get(id=user_id)
    all_genre = Genre.objects.all()
    all_actors = Actor.objects.all()
    all_directors = Director.objects.all()

    query = request.GET.get('query')
    user_words = query.split()

    all_media = Media.objects.all()
    actual_media = []

    for media in all_media:
        for word in user_words:
            if word.lower() in media.title.lower():

                reviews = Review.objects.filter(media_id=media.id)
                data_reviews = [
                        {
                            'review_date': str(review.rev_date),
                            'rating': review.rating,
                            'comment': review.comment,
                            'reviewer': Person.objects.get(id=review.person_id).name,
                        }
                        for review in reviews
                    ]
                json_data_reviews = json.dumps(data_reviews)

                if media.media_type == 'Film':
                    film = Film.objects.select_related('media_ptr').prefetch_related('actors', 'directors', 'genre').all().get(media_ptr_id = media.id)

                    actual_media.append({
                        'type': 'film',
                        'title': media.title,
                        'release_date': media.release_date,
                        'duration': film.duration,
                        'actors': ", ".join(actor.name for actor in film.actors.all()),
                        'directors': ", ".join(director.name for director in film.directors.all()),
                        'genres': ", ".join(genre.name for genre in film.genre.all()),
                        'synopsis': film.synopsis,
                        'reviews': json_data_reviews,
                    })

                elif media.media_type == 'Serie':
                    serie = Series.objects.select_related('media_ptr').prefetch_related('actors', 'directors', 'genre').all().get(media_ptr_id = media.id)
                    actual_media.append({
                        'type': 'serie',
                        'title': media.title,
                        'release_date': media.release_date,
                        'episodes': serie.episodes,
                        'seasons': serie.seasons,
                        'actors': ", ".join(actor.name for actor in serie.actors.all()),
                        'directors': ", ".join(director.name for director in serie.directors.all()),
                        'genres': ", ".join(genre.name for genre in serie.genre.all()),
                        'synopsis': serie.synopsis,
                        'reviews': json_data_reviews,
                    })


    context = {
        'userID': user_id,
        'user_name':user.name,
        'all_genre': all_genre,
        'all_actors': all_actors,
        'all_directors': all_directors,
        'all_media': actual_media,
    }
    template=loader.get_template('brMedia/search.html')
    return HttpResponse(template.render(context, request))
```

Figura 28 - Representação da função *search* no *views.py*

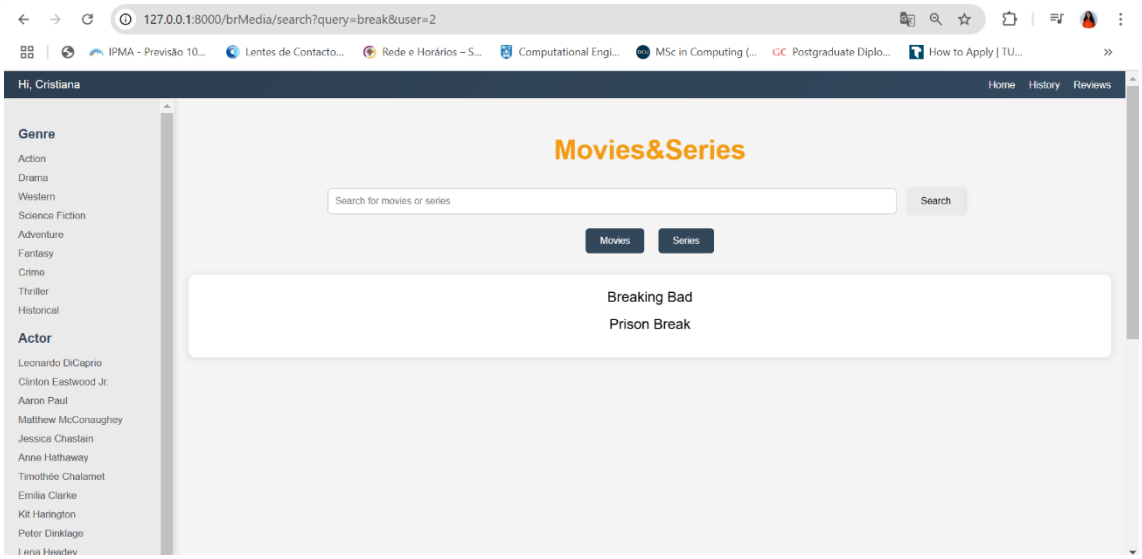

Figura 29 - Website relacionada com *search* do *views.py*

Capítulo 6 - Conclusão

O principal objetivo definido para este trabalho foi realizado com sucesso, uma vez que foi desenvolvido um website intuitivo e funcional, permitindo aos utilizadores interagirem com a plataforma de forma eficiente. Todas as funcionalidades planeadas, como a visualização de detalhes de filmes e séries, a leitura e adição de avaliações, e a consulta de informações sobre géneros, atores e realizadores, foram implementadas corretamente.

Em suma, o projeto atingiu os objetivos propostos, resultando numa plataforma para a avaliação de filmes e séries, que poderá ser expandida no futuro com novas funcionalidades e conteúdos.

Capítulo 7 - Scripts

INSERT INTO "brMedia_actor" (id, name, birth_date, nationality) VALUES

(1, 'Leonardo DiCaprio', '1974-11-11', 'American'),

(2, 'Clinton Eastwood Jr.', '1930-05-31', 'American'),

(3, 'Aaron Paul', '1979-08-27', 'American'),

(4,'Matthew McConaughey', ' 1969-11-4','American'),

(5, 'Jessica Chastain','1977-03-24','American'),

(6,'Anne Hathaway','1982-11-12','American'),

(7,'Timothée Chalamet','1995-12-27','American'),

(8, 'Emilia Clarke', '1986-10-23', 'British'),

(9, 'Kit Harington', '1986-12-26', 'British'),

(10, 'Peter Dinklage', '1969-06-11', 'American'),

(11, 'Lena Headey', '1973-10-03', 'British'),

(12, 'Wentworth Miller', '1972-06-02', 'American'),

(13, 'Dominic Purcell', '1970-02-17', 'Australian'),

(14, 'Sarah Wayne Callies', '1977-06-01', 'American'),

(15, 'Robert Knepper', '1959-07-08', 'American'),

(16, 'Christian Bale', '1974-01-30', 'British'),

(17, 'Heath Ledger', '1979-04-04', 'Australian'),

(18, 'Aaron Eckhart', '1968-03-12', 'American'),

(19, 'Maggie Gyllenhaal', '1977-11-16', 'American'),

(20, 'Tim Robbins', '1958-10-16', 'American'),

(21, 'Morgan Freeman', '1937-06-01', 'American'),

(22, 'Bob Gunton', '1945-11-15', 'American'),

(23, 'William Sadler', '1950-04-13', 'American'),

(24, 'Henry Cavill', '1983-05-05', 'British'),

(25, 'Anya Chalotra', '1996-07-21', 'British'),

(26, 'Freya Allan', '2001-09-06', 'British'),

(27, 'Joey Batey', '1989-09-01', 'British'),

(28, 'Jared Harris', '1961-08-24', 'British'),

(29, 'Stellan Skarsgård', '1951-06-13', 'Swedish'),

(30, 'Emily Watson', '1967-01-14', 'British'),

(31, 'Paul Ritter', '1966-12-05', 'British');

INSERT INTO "brMedia_director"(id,name,birth_date,nationality) VALUES

(1, 'Christopher Nolan', '1970-07-30', 'British'),

(2,'Sergio Leone','1929-01-03','Italian'),

(3,'George Vincent Gilligan Jr.','1967-02-10','American'),

(4, 'David Benioff', '1970-09-25', 'American'),

(5, 'D.B. Weiss', '1971-04-23', 'American'),

(6, 'Miguel Sapochnik', '1974-10-01', 'British'),

(7, 'Jeremy Podeswa', '1962-12-28', 'Canadian'),

(8, 'Mikael Håfström', '1960-03-01', 'Swedish'),

(9, 'Kevin Hooks', '1958-04-19', 'American'),

(10, 'Tori Garrett', '1977-07-28', 'Australian'),

(11, 'Vince Misiano', '1960-12-05', 'American'),

(12, 'Frank Darabont', '1959-01-28', 'American'),

(13, 'Alik Sakharov', '1964-07-19', 'Ukrainian-American'),

(14, 'Lauren Schmidt Hissrich', '1974-06-08', 'American'),

(15, 'Johan Renck', '1966-12-05', 'Swedish');

INSERT INTO "brMedia_genre(id,name,description) VALUES

(1,'Action','Action is a genre focused on high-energy sequences, including fights, chases, and explosions, often driven by conflict and physical challenges.'),

(2,'Western','A Western is a genre set in the American Old West, featuring cowboys, outlaws, and rugged landscapes, focusing on themes of justice and survival.'),

(3,'Drama','Drama is a genre that explores emotional, realistic stories with complex characters and intense conflicts.'),

(4,'Science Fiction', 'Science fiction is a genre of speculative fiction that explores imaginative and futuristic concepts, often grounded in science, technology, and the natural world. It examines the impact of advancements or phenomena on individuals, societies, and the universe.'),

(5,'Adventure','Adventure is a genre characterized by exciting, dynamic, and often risky journeys or quests. It focuses on exploration, action, and overcoming obstacles, transporting the audience to extraordinary settings or situations filled with danger and discovery.'),

(6, 'Fantasy', 'A genre that involves magic, imaginary worlds, and supernatural creatures. Often set in epic fantasy universes'),

(7, 'Crime', 'A genre focused on the investigation and resolution of criminal acts, often involving law enforcement or criminals. '),

(8, 'Thriller', 'A genre characterized by suspense, tension, and excitement, often involving high-stakes situations.'),

(9, 'Historical', 'A genre that depicts past events, often aiming for accuracy and educational value.');

INSERT INTO "brMedia_media"(id,title,media_type,release_date,synopsis) VALUES

(1,'Inception','film','2010-07-16', 'Inception follows Dom Cobb, a skilled thief who specializes in extracting secrets from within dreams. Given a chance to erase his criminal past, Cobb is tasked with performing the impossible: inception, planting an idea in someone's mind through shared dreaming. As his team dives deeper into the targets subconscious, they face dangerous challenges and unraveling realities, blurring the line between dreams and reality.'),

(2,'The Good, the Bad and the Ugly','film','1969-01-23','The Good, the Bad and the Ugly follows three gunmen—Blondie (the Good), Tuco (the Ugly), and Angel Eyes (the Bad)—as they search for a hidden cache of Confederate gold during the American Civil War. They form uneasy alliances and face deadly showdowns, culminating in a tense Mexican standoff at a cemetery. The film is famous for its iconic duels, memorable score by Ennio Morricone, and exploration of moral ambiguity.'),

(3,'Breaking Bad','serie','2008-01-20','Breaking Bad follows Walter White, a high school chemistry teacher turned methamphetamine manufacturer, as he enters the drug trade to secure his familys future after being diagnosed with cancer. As he partners with former student Jesse Pinkman, Walters descent into the criminal underworld leads to increasingly dangerous situations, altering his life and those around him. The series explores themes of morality, family, and power.'),

(4,'Interstellar','film','2014-11-6','In Interstellar, ex-NASA pilot Cooper joins a mission to find a new home for humanity as Earth becomes uninhabitable. Traveling through a wormhole, the team explores distant planets near a massive black hole. Facing time dilation and emotional sacrifice, Cooper fights to save his family and humanity, uncovering profound truths about love, survival, and the universe.'),

(5,'Game of Thrones','serie','2011-04-17','Game of Thrones is a fantasy drama set in the continents of Westeros and Essos. The story follows noble families, primarily the Starks, Lannisters, and Targaryens, as they vie for the Iron Throne. Central themes include power struggles, betrayal, and war. Meanwhile, supernatural forces like dragons and the undead grow in prominence. The series explores the complexity of politics, family, and survival, with shifting alliances and numerous character arcs that evolve throughout its eight seasons.'),

(6, 'Prison Break', 'serie', '2005-08-29', 'A man deliberately gets himself sent to prison to break his brother out, who has been wrongfully convicted of a crime. The show follows their journey and the challenges they face inside and outside the prison.'),

(7, 'The Dark Knight', 'film', '2008-07-18', 'Batman faces the Joker, a criminal mastermind who seeks to create chaos in Gotham City. A gripping tale of heroism and sacrifice, featuring iconic performances and action.'),

(8, 'The Shawshank Redemption', 'film', '1994-09-22', 'Two imprisoned men bond over a number of years, finding solace and eventual redemption through acts of common decency.'),

(9, 'The Witcher', 'serie', '2019-12-20', 'Geralt of Rivia, a monster hunter, navigates a world full of political intrigue, dangerous creatures, and powerful magic while trying to find his place in a chaotic world.'),

(10, 'Chernobyl', 'serie', '2019-05-06', 'A dramatization of the true story of the 1986 nuclear disaster in Chernobyl, showing the catastrophic events and the aftermath as experienced by those involved and affected.');

INSERT into "brMedia_media_actors"(id,media_id,actor_id) VALUES

(1,1,1),

(2,2,2),

(3,3,3),

(4,4,4),

(5,4,5),

(6,4,6),

(7,4,7),

(8,5,8),

(9,5,9),

(10,5,10),

(11,5,11),

(12,6,12),

(13,6,13),

(14,6,14),

(15,6,15),

(16,7,16),

(17,7,17),

(18,7,18),

(19,7,19),

(20,8,20),

(21,8,21),

(22,8,22),

(23,8,23),

(24,9,24),

(25,9,25),

(26,9,26),

(27,9,27),

(28,10,28),

(29,10,29),

(30,10,30),

(31,10,31);

INSERT INTO "brMedia_media_directors"(id,media_id,director_id) VALUES

(1,1,1),

(2,2,2),

(3,3,3),

(4,4,1),

(5,5,4),

(6,5,5),

(7,5,6),

(8,5,7),

(9,6,8),

(10,6,9),

(11,6,10),

(12,6,11),

(13,7,1),

(14,8,12),

(15,9,13),

(16,9,14),

(17,10,15);

```
INSERT INTO "brMedia_media_genre"(id,media_id,genre_id) VALUES
(1,1,1),
(2,2,2),
(3,3,3),
(4,4,3),
(5,4,4),
(6,4,5),
(7,5,1),
(8,5,3),
(9,5,6),
(10,5,5),
(11,6,1),
(12,6,3),
(13,6,8),
(14,6,7),
(15,7,1),
(16,7,7),
(17,7,8),
(18,8,3),
(19,8,7),
(20,9,6),
(21,9,1),
(22,9,5),
(23,10,9),
(24,10,8),
(25,10,3);


INSERT INTO "brMedia_person"(id,name,email,password) VALUES
(1,'Duarte','duarte1454@gmail.com','pass123'),
(2,'Cristiana','cris123@gmail.com','password987');
```

INSERT INTO "brMedia_review"(id,rev_date,rating,comment,media_id,person_id) VALUES

(1,'2024-12-15',5,'Amazing movie!',1,1),

(2,'2023-10-24',4,'I loved this movie!',2,2),

(3,'2024-06-16',5,'Best series ever!',3,1),

(4,'2023-01-15',5,'I loved it!',4,2),

(5,'2020-05-20',3,'I stopped wathcing on season 3. Too much violence!',5,2),

(6,'2021-02-20',5,'It is my favourite series ever!',6,1),

(7, '2025-01-03', 5, 'An absolutely thrilling experience! The action and suspense in The Dark Knight are unmatched. Heath Ledgers performance as the Joker is legendary.', 7, 1),

(8, '2025-01-02', 4, 'A beautifully written drama with outstanding performances. "The Shawshank Redemption" offers a poignant message of hope and redemption.', 8, 1),

(9, '2025-01-01', 5, 'One of the best fantasy series out there! "The Witcher" combines incredible action, a captivating story, and unforgettable characters.', 9, 2),

(10, '2025-01-05', 4, 'A gripping portrayal of a historical tragedy. "Chernobyl" provides an intense and haunting look at the events surrounding the nuclear disaster.', 10, 2);

INSERT INTO "brMedia_visualization"(id,view_date,media_id,person_id) VALUES

(1,'2024-12-10',1,1),

(2,'2023-10-22',2,2),

(3,'2024-06-10',3,1),

(4,'2023-06-07',4,2),

(5,'2022-03-12',5,2),

(6,'2021-01-15',6,1),

(7, '2024-12-23', 7, 1),

(8, '2024-11-03', 8, 1),

(9, '2024-12-20', 9, 2),

(10, '2024-12-11', 10, 2);

INSERT INTO "brMedia_film" (media_ptr_id,duration) VALUES

(1,'147 minutes'),

(2,'172 minutes'),

(4,'169 minutes'),

(7, '152 minutes'),

(8, '142 minutes');

INSERT INTO "brMedia_series" (media_ptr_id,seasons,episodes) VALUES

(3,5,62),

(5,8,73),

(6,5,90),

(9, 1, 8),

(10, 1, 5);

Código de *actors.html*

```
<!DOCTYPE html>
<html lang="pt">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{{ actor.name }} - Description</title>
    <style>
        :root {
            --blue: #2a4365;
            --gold: #f0c674;
            --light-blue: #5a7e91;
            --bg-color: #f7f7f7;
            --content-bg-color: #ffffff;
            --text-color: #333;
            --shadow: rgba(0, 0, 0, 0.1);
        }

        body {
            font-family: 'Arial', sans-serif;
            background-color: var(--bg-color);
            color: var(--text-color);
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction: column;
            min-height: 100vh;
        }

        header {
            display: flex;
            justify-content: space-between;
            align-items: center;
            background: linear-gradient(135deg, #2c3e50, #34495e);
            color: white;
            padding: 10px 20px;
            position: sticky;
            top: 0;
            z-index: 1000;
        }

        header .user-info {
            font-size: 16px;
            color: white;
        }
```

```
        header .links a {
            color: white;
            text-decoration: none;
            margin-left: 15px;
            font-size: 14px;
        }

        header .links a:hover {
            text-decoration: underline;
        }

        .center_container {
            display: flex;
            justify-content: center;
            align-items: center;
            height: 90vh;
        }

        .container {
            background-color: var(--content-bg-color);
            border-radius: 12px;
            box-shadow: 0 4px 15px var(--shadow);
            padding: 30px;
            width: 80%;
            max-width: 800px;
            text-align: center;
        }

        h1 {
            font-size: 2.5em;
            color: var(--blue);
            margin-bottom: 20px;
            border-bottom: 3px solid var(--gold);
            padding-bottom: 10px;
        }

        p {
            font-size: 1.2em;
            line-height: 1.6;
            color: var(--light-blue);
            margin-top: 20px;
        }

        footer {
            margin-top: 40px;
            font-size: 1em;
            color: var(--blue);
        }

        footer p {
            margin: 0;
        }

        .user-info {
            color: var(--light-blue);
        }

    </style>
</head>
<body>
    <header>
        <div class="user-info">Hi, {{ user_name }}</div>
        <div class="links">
            <a href="{% url 'user_profile' %}?user={{ userID }}">Home</a>
            <a href="{% url 'visualization' %}?user={{ userID }}">History</a>
            <a href="{% url 'reviews' %}?user={{ userID }}">Reviews</a>
        </div>
    </header>
    <div class="center_container">
        <div class="container">
            <h1>{{ actor.name }}</h1>

            {% if actor %}
                <p><strong>Nationality:</strong> {{ actor.nationality }}</p>
                <p><strong>Birth Date:</strong> {{ actor.birth_date }}</p>
                {% if actor.media_actor.all %}
                    {% for media in actor.media_actor.all %}
                        <p><strong>Participated in:</strong> {{ media.title }} -  {{ media.get_media_type_display }}</p>
                    {% endfor %}
                {% endif %}
            {% endif %}
        </div>
    </div>
</body>
</html>
```