Projeto do desenvolvimento de uma aplicação web voltada para visualização e busca de filmes

Matheus Sehnem de Oliveira[1]

[1]Centro de Ciências Tecnológicas – Universidade do Estado de Santa Catarina (UDESC)
Joinville, SC – Brazil

matheus.sdo@edu.udesc.br

Resumo. *Afim de facilitar a procura de filmes, o aplicativo que leva o nome de Popcorn Finds foi projetado com intuito de atender o usuário que deseja uma maneira simples e rápida de consultar e visualizar as informações de um filme antes de assistí-lo. Além disso, o usuário também poderá salvar os filmes e atores que mais lhe interessam, como também avaliar um filme, ajudando os demais usuários a decidirem sobre assistí-lo ou não. Portanto, neste projeto serão apresentados e explanados os objetivos dele, seus requisitos, sua estimativa de duração, sua modelagem, seu design, as ferramentas a serem utilizadas afim de atender o que será proposto e como foram desenvolvidos os testes unitários.*

1. Respositório do Projeto

Antes de iniciarmos o trabalho, o repositório pedido para o acesso aos arquivos do projeto estão em: `https://github.com/matheusrnk/trab-source-soft`

2. Descrição do Problema

A indústria do *streaming* nunca foi tão grande como em anos recentes e continua a crescer. Sabendo disso, temos diversos tipos de mídias sendo criadas nesta área, sendo uma delas, os filmes. Com isso, para as pessoas que não acompanham muito ou acompanham moderadamente a sétima arte, fica difícil saber quais filmes são bons, quais os populares e principalmente quais os que mais as agradam. Portanto, sabendo as dificuldades que o atual cenário demonstra para aqueles não são tão envolvidos na comunidade, o aplicativo Popcorn Finds está sendo projetado com o intuito de sanar estes problemas. Assim, as partes interessadas nesse projeto seriam os usuários de qualquer faixa-etária que possuem dificuldades em encontrar um filme para assistir em seu tempo livre e a comunidade de filmes num geral.

Afim de ajudá-los, o aplicativo foi projetado com o intuito de levar uma interface simples e direta ao usuário, onde se espera que o aplicativo entregue facilidade na hora de procurar um filme, possibilitando visualizar suas informações, salvá-lo, avaliá-lo, bem como procurar filmes por gêneros, colocações e popularidade, facilitando para àqueles que desejam uma busca mais direta.

Portanto, ao longo desta apresentação serão demonstrados os requisitos necessários para compor o que se deseja implementar, os objetivos e justificativas da escolha do tema de abordagem, a modelagem dos dados que estruturam a aplicação, o design escolhido para o desenvolvimento e as ferramentas necessárias para construir tal aplicação. No próximo tópico entraremos nos detalhes da escolha do tema e porque se acha relevante a construção de tal aplicativo.

3. Objetivos e Justificativa

Como escrito anteriormente, pelo fato da área de filmes ser vasta, é difícil encontrar um filme que se adeque às exigências da pessoa sem uma ferramenta para auxiliá-la. Portanto, o objetivo deste projeto é utilizar a API *Online Movie Database* para entregar uma aplicação simples e direta que possa usufruir das informações disponibilizadas pela API para facilitar a busca e visualização de filmes e atores. Além disso, se espera que, além de entregar um aplicativo capaz de facilitar a informação para as pessoas não tão adeptas ao mundo dos filmes, o aplicativo também ofereça a capacidade das mesmas se comunicarem através das avaliações dos filmes, elaborando críticas sobre eles e auxiliando o usuário a decidir com base no que está sendo dito.

Portanto, a justificativa para a criação de tal programa é, além do que já foi descrito, mostrar para as pessoas que não é necessário ficar buscando e acessando diferentes *sites* com listas de avaliações para encontrar o filme desejado. Ou seja, a ideia é diminuir o vasto campo de busca de uma ferramenta como o **Google**, para que a procura pelo filme seja mais direta, podendo também acessar tópicos pré-montados de filmes mais populares, melhores colocados e por gênero. A seguir veremos quais os requisitos necessários para construir tal aplicação.

4. Levantamento de requisitos

Os requisitos apresentados a baixo deverão compor as funcionalides do sistema, ou seja, tudo aquilo que é necessário para que o programa atinja as expectativas de implementação e que fazem sentido estarem como requisito de tal sistema. Além disso, também serão mostrados os requisitos não-funcionais nas tabelas a seguir:

	REQUISITOS FUNCIONAIS
RF1:	O software deve permitir a busca de filmes pela API Online Movie Database retornando id, nome, capa do filme, sinopse, rank, classificação indicativa, generos e ano, além de seu elenco principal, o qual é composto por atores, onde cada ator deve possuir seu id, nome, data de nascimento, sexo, sua mini biografia em ingles, lugar de nascimento, foto, altura e os filmes mais importantes.
RF2:	O software deve permitir o cadastro de um usuário contendo foto, nome de usuario, nome, email e senha.
RF3:	O software deve permitir a remoção, alteração e login do usuário.
RF4:	O software deve permitir que o usuário salve um filme ou um ator individualmente no banco de dados.
RF5:	O software deve permitir a exclusão de um filme ou ator salvo de um determinado usuario.
RF6:	O software deve permitir deve permitir que o usuário visualize os filmes e atores já salvos.
RF7:	O software deve permitir mostrar os filmes que estão para lançar, os filmes mais bem colocados, os mais populares, os mais populares por gênero e poder sortear um filme para ver as suas informações.
RF8:	O software deve permitir que o usuário faça uma avaliação e classificação (bom ou ruim) de um filme.
RF9:	O software deve permitir que o usuário vote em uma avaliação (bom ou ruim), destancando as avaliações mais bem votadas.
RF10:	O software deve permitir a remoção e visualização de uma avaliação através do usuário logado.

Tabela 1. Requisitos Funcionais

	REQUISITOS NÃO-FUNCIONAIS
RNF1:	O software deverá rodar em desktops, sejam eles Linux, Windows ou MacOS.
RNF2:	O software deverá ser desenvolvido na linguagem Python.
RNF3:	O software deverá apresentar segurança, utilizando-se das bibliotecas disponíveis em Python para implementá-las.
RNF4:	O software deverá se comunicar com o PostgreSQL.
RNF5:	O software deverá se comunicar com a API Online Movie Database.
RNF6:	O software deverá tentar reduzir o tempo de procura e apresentação dos filmes.
RNF7:	O software deverá prover consistência dos dados apresentados ao usuário.

Tabela 2. Requisitos Não-Funcionais

Com isto definido, podemos avançar e demonstrar o que foi projetado na parte de modelagem para satisfazer as necessidades da aplicação. Portanto, mostraremos dois diagramas e especificaremos o que ocorre em cada um.

5. Modelagem

Com o intuito de prover todas as funcionalidades citadas anteriormente, foram feitas duas modelagens: Uma delas sendo o diagrama de Entidade-Relacionamento Estendido, a qual mostra como vai ocorrer a navegação entre os dados, e a outra sendo o diagrama UML, que mostra como se pretende montar a aplicação.

- **Diagrama Entidade-Relacionamento Estendido**:

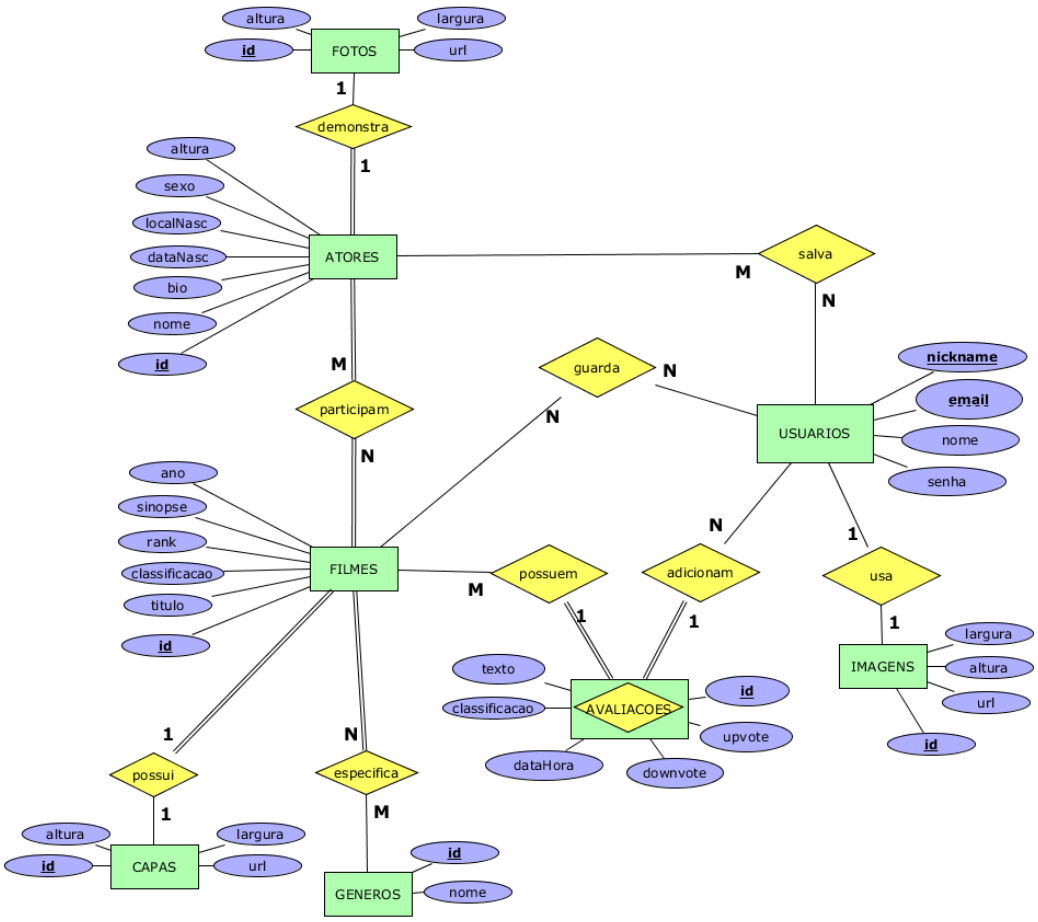

Figura 1. Diagrama E.R.E. da aplicação de filmes

- **Diagrama UML**:

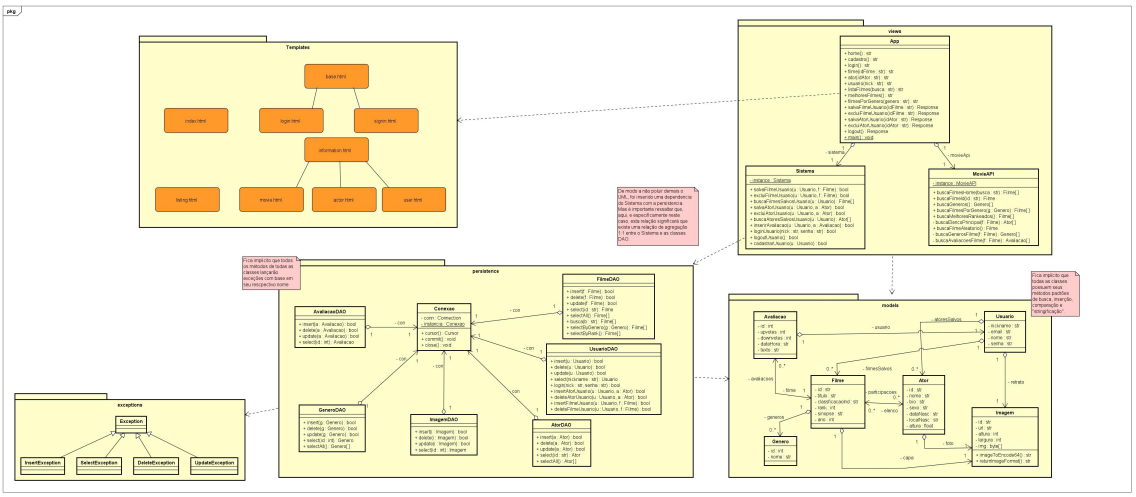

Figura 2. Diagrama UML da aplicação de filmes

Como é possível observar, o diagrama E.R.E. está completo e será utilizado da maneira que foi projetado. Em contrapartida, o diagrama UML ainda poderá passar por ajustes, uma vez que não foram especificados os métodos necessários para cômpor as classes. Isto se deve à dois motivos: O primeiro está relacionado a possível grande quantidade de métodos a serem implementados, não ficando muito legível a ideia que o diagrama quer passar. E a segunda seria o fato de que ainda não se sabe, em sua totalidade, o que será implementado em classes mais especificas como a que faz a conexão com a API. Portanto, se tem uma vaga ideia do que será implementado nestas classes, mas falta refinar um pouco o que é necessário de fato implementar.

Com isto definido, podemos avançar e estimar quanto tempo levará para o projeto ficar pronto, onde teremos uma ideia básica do quão complexo será sua montagem e desenvolvimento.

6. Estimativa de duração do projeto

Afim de obter uma estimativa mais próxima à realidade, se optou por utilizar o método de estimativa paramétrica **COCOMO + APF**, onde podemos de fato ter uma noção de quantas pessoas serão necessárias e qual a duração do projeto. Portanto, o primeiro passo é definir qual a complexidade do projeto e, logo em seguida, contar os elementos do software para podermos calcular os pontos de função do sistema e dar prosseguimento aos demais cálculos.

Assim, podemos perceber que, devido à natureza do problema e o ambiente em que ela será implementada (faculdade), é possível dizer que este é um projeto de fácil entedimento e de uma equipe pequena, uma vez que o objetivo do projeto é apenas uma aplicação para visualização e busca de filmes com apenas uma pessoa a implementando. Portanto, podemos assumir que:

$$Esforco = 2,4 * KLOC^{1,05} \tag{1}$$

$$Duracao = 2,5 * Esforco^{0,38} \tag{2}$$

Feito isso, devemos enumerar os elementos do software em EE (Entradas Externas), SE (Saídas Externas), CE (Consultas Externas), ALI (Arquivos Lógicos Internos) e AIE (Arquivos de Interface Externos). Então, segue:

- **AIE**:
    1. Interface externa para interagir com a API Online Movie Database.
- **ALI**:
    1. Arquivo para a tabela usuários;
    2. Arquivo para a tabela filmes;
    3. Arquivo para a tabela atores;
    4. Arquivo para a tabela avaliações;
    5. Arquivo para a tabela gêneros;
    6. Arquivo para a tabela imagens;
    7. Arquivo para a tabela capas;
    8. Arquivo para a tabela fotos;
- **EE**:
    1. Cadastro do usuário;
    2. Entrada do login do usuário;
    3. Cadastro de avaliação;
- **SE**:
    1. Mensagem de confirmação de cadastro;
    2. Mensagem de confirmação de entrada de login;
    3. Mensagem de confirmação de criação de avaliação;
    4. Mensagem de erro de cadastro;
    5. Mensagem de erro de entrada de login;
    6. Mensagem de erro de criação de avaliação;
    7. Mensagem de erro de busca de filme;
    8. Mensagem de erro de busca de ator;
    9. Lista de filmes por página;
    10. Lista de atores por página;
- **CE**:
    1. Busca de filme por título;
    2. Busca de filme por ”Em breve”;
    3. Busca de filme por posição geral;
    4. Busca de filme por popularidade;
    5. Busca de filme por gênero;
    6. Busca de perfil de usuário;
    7. Busca de avaliação de filme;
    8. Busca de ator por filme;
    9. Busca de filme por ator;
    10. Busca de filme por filmes salvos;
    11. Busca de ator por atores salvos;

Agora que temos os elementos enumerados, devemos definir a complexidade de um, atribuindo um peso e fornecendo uma breve explicação do porquê de sua pontuação. Segue:

	Complexidade dos elementos
EE	**3 pontos**. O motivo seria devido à facilidade na implementação destes componentes, uma vez que eles necessitam apenas de um redirecionamento de informação e verificações.
SE	**4 pontos**. O motivo seria devido à necessidade de outros componentes fazerem alguma ação para que estes sejam utilizados. Assim, sendo necessário que se elabore fluxo um pouco mais estruturado para eles.
CE	**4 pontos**. O motivo seria a necessidade de uma atenção maior na hora da montagem dos dados e de sua busca do que em relação ao elemento **EE**.
ALI	**7 pontos**. O motivo seria por causa da criticidade dessa parte, onde é fundamental que ela esteja corretamente funcionando para o que sistema não quebre ao longo do caminho ou possua limitantes.
AIE	**10 pontos**. A mesma de idea de **ALI** ocorre aqui, mas recebeu 3 pontos a mais pelo fato de não fazer parte do sistema original, sendo necessário um entendimento extra de como funciona a API para poder utilizá-la corretamente.

Tabela 3. Pesos dos elementos

Terminado esta parte, podemos prosseguir com os cálculos dos pontos de função não ajustados e responder algumas perguntas sobre o ajuste destes pontos. Segue os cálculos e as perguntas/respostas:

$$PFNA = 3 * 3 + 10 * 4 + 11 * 4 + 8 * 7 + 1 * 10 \tag{3}$$

$$PFNA = 159 \tag{4}$$

1. **Necessita de backup?** - Nota 4;
2. **Necessita de mecanismos especializados de comunicação?** - Nota 2;
3. **Tem processamento distribuído?** - Nota 2;
4. **Precisa de alto desempenho?** - Nota 1;
5. **Terá grande número de usuários em paralelo?** - Nota 2;
6. **Precisará de entrada de dados on-line?** - Nota 3;
7. **No caso de entradas on-line, existirão múltiplas telas?** - Nota 3;
8. **A atualização das entidades será feita on-line?** - Nota 3;
9. **As entradas e saídas de dados serão complexas?** - Nota 3;
10. **O processamento interno será complexo?** - Nota 4;
11. **O código será projetado para ser reutilizado?** - Nota 4;
12. **Migração e instalação estarão incluídos?** - Nota 1;
13. **O sistema será instalado em diversas organizações?** - Nota 0;
14. **O projeto pretende facilitar mudanças e operação do usuário?** - Nota 3;

Cálculado o **PFNA** e respondido às questões de ajuste, agora podemos obter os pontos de função ajustados e convertê-los em KLOCs, que é o que desejamos para calcular a duração do projeto. Portanto, seguem os cálculos:

$$PF = PFNA * (0,65 + 0,01 * \sum Respostas) \tag{5}$$

$$PF = 159 * (0,65 + 0,01 * 35) \tag{6}$$

$$PF = 159 \tag{7}$$

Assumindo que 1 PF é igual a 27 LOC em Python, temos:

$$LOC = 27 * 159 \tag{8}$$

$$LOC = 4293 \tag{9}$$

Como o COCOMO pede que as linhas de código estejam em KLOC, devemos pegar LOC e dividir por mil, uma vez que a medição está sendo feita a cada mil linhas de código. Portanto, sejam os cálculos a seguir:

$$Esforco = 2,4 * 4,293^{1,05} \tag{10}$$

$$Esforco = 11,082 \tag{11}$$

$$Duracao = 2,5 * 11,082^{0,38} \tag{12}$$

$$Duracao = 6,236 \tag{13}$$

Portanto, dados os cálculos acima e arredondando para o inteiro mais próximo, necessitaríamos de **11 pessoas-mês** para projeto e leveríamos **6 meses** para concluí-lo. Com isso, é possível dizer que, apesar de ser um projeto simples, é algo que leverá um tempo considerável para ficar pronto da maneira desejada e atender à todas as funcionalidades.

Desenvolvido esta parte, podemos falar um pouco mais sobre como o será o design da interface do usuário e o que se espera dela, sendo o próximo tópico abordado.

7. Design

Com uma interface simples e direta, os designs a serem apresentados a seguir refletem o que se espera implementar na aplicação. Eles foram pensados para manterem a maior simplicidade ao usuário e facilitar a visualização dos conteúdos desejados. Além disso, foram escolhidos tons de vermelho, cinza e preto para compor a coloração do design do site, uma vez que se acredita que estas cores trabalham bem juntas.

Com isso, foram montadas oito páginas de exemplo, ou seja, todas as páginas futuramente montadas se basearão em uma das oito, as quais foram criadas com a ajuda da ferramenta **Canva**. Portanto, será possível perceber que há páginas faltando a partir da visualização do gráfico de fluxo entre páginas que será mostrado a seguir, mas será possível perceber a qual cada uma pertence, respectivamente.

- **Gráfico de fluxo entre páginas**:

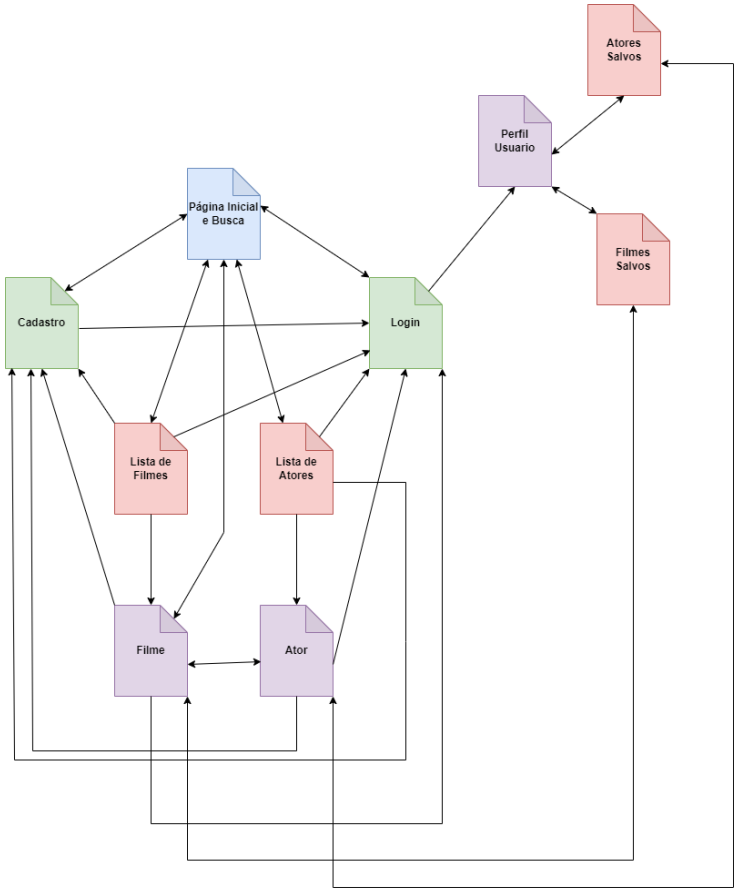

Figura 3. Gráfico de fluxo entre páginas da aplicação de filmes

- **Tela Inicial**:

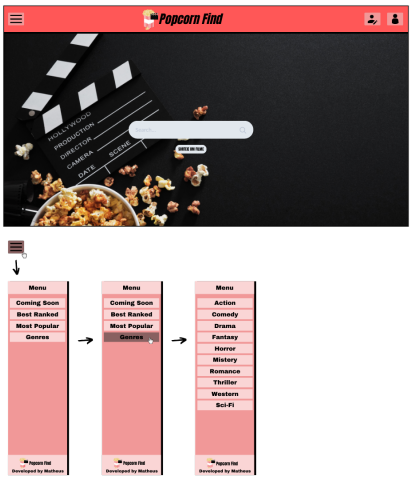

Figura 4. Tela inicial da aplicação de filmes

- **Tela de Cadastro e Login**:

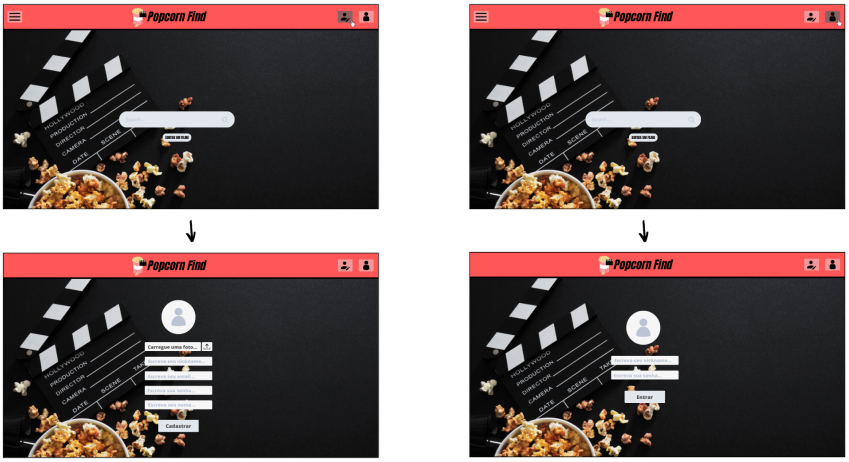

Figura 5. Tela de cadastro e login da aplicação de filmes

- **Tela de visualização de filmes, atores e usuários**:

Figura 6. Tela de visualização de filmes, atores e usuários da aplicação de filmes

- **Tela de lista de filmes e atores**:

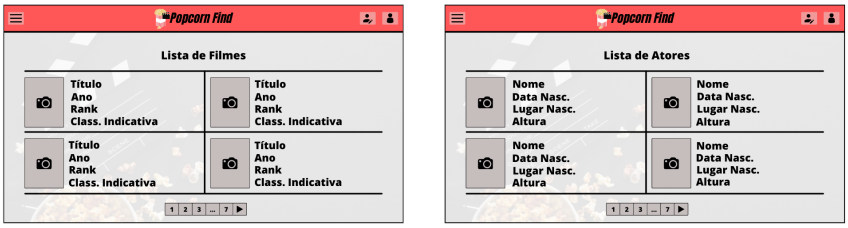

Figura 7. Tela de lista de filmes e atores da aplicação de filmes

Eventualmente, as páginas podem vir a sofrer alterações, já que os exemplos foram montados a partir de uma ferramenta que não leva a consideração a possiblidade de implementação ou não do design que está sendo proposto. Entretanto, a implementação tentará se manter o mais fiel possível ao que foi demonstrado. A seguir, será discursado sobre quais ferramentas serão utilizadas e porquê de utilizá-las.

8. Ferramentas

A princípio, seguindo a ideia do primeiro trabalho apresentado, a aplicação será montada utilizando o framework Flask. Com isso, é possível dizer que algumas das ferramentas utilizadas serão: Python, para construir o back-end; HTML+CSS+Python para construir o front-end; E o PostgreSQL para fazer a persistência dos dados; Por fim, será utilizado a biblioteca *unittest* para fazer os testes unitários da aplicação. Vale ressaltar que também serão utilizadas as ferramentas Jinja2 e Werkzeug, as quais são oferecidas pelo framework Flask. Além disso, como é sabido, este framework não oferece um padrão de trabalho, portanto, será utilizado o MVT (Model-View-Template, oferecido pelo Django), mas com algumas modificações nos Models e nas Views. Tais ferramentas foram escolhidas devido ao fato da familiaridade do autor com a linguagem e do desejo de entender se é viável e produtivo a utilização de tais ferramentas para construção de um website.

9. Testes Unitários

Nesta seção, o assunto discorrido será sobre a criação dos testes unitários, onde será feito uma breve introdução de cada classe feita para os testes, porquê estes testes foram escolhidos e porquê da ausência de outros testes que poderiam estar presentes nele. Além disso, será comentado o porquê da classe ***App*** não estar presente como uma das classes de teste.

Começando pelo módulo de **models**, temos todas as classes que representam alguma entidade núcleo da aplicação, como: Filme, Ator, Usuario etc. Sabendo que estas classes não provém muitos métodos para serem testados e que não possuem muitos desvios de fluxo, todos os testes criados para eles apenas testam se ao inserir uma informação para algum atributo, o valor retornado ao acessá-lo é o mesmo que foi fornecido. Portanto, um exemplo seriam os testes para a classe Genero, que são:

```
import unittest

from main.models.genero import Genero


class GeneroTest(unittest.TestCase):

    g = None

    def setUp(self) -> None:
        self.g = Genero(id=1, nome="Um genero")

    def testa_busca_id(self):
        self.assertEqual(1, self.g.id)

    def testa_busca_nome(self):
        self.assertEqual("Um genero", self.g.nome)
```

Onde se pode perceber que está se fazendo exatamente isso, no qual cada teste está verificando se o retorno do atributo condiz com o que foi inserido. Nada além disso é interessante de ser testado, uma vez que somente isto será utilizado destas classes.

Indo para o módulo de **exceptions**, temos que cada classe representa uma exceção para um possível erro no banco de dados. Como estas classes são apenas *templates* da classe Exception e serão somente utilizadas para repassar um possível erro ocorrido, não há porquê testá-las, uma vez que elas não implementarão métodos e não possuírão fluxo a ser testado. Portanto, estes foram os motivos para não ter sido criado um módulo de teste para estas classes.

Passando para o módulo de **persistence** e **views**, o primeiro representa todas as classes que farão a conexão com o banco de dados e o CRUD para a entidade que leva o nome no início do arquivo (Exemplo: GeneroDAO - Genero); Já o outro representa 3 classes muito importantes para o sistema, são elas: Sistema (Possui todos os métodos que conversarão com funcionalidades relacionadas ao usuário), MovieAPI (Possui todos os métodos que conversão com a API Online Movie Database e transformará seus dados em informações utilizáveis) e App (Possui toda a lógica de implementação da interface com usuário na web em união com os arquivos html residentes em **templates**). Optou-se por falar de ambos juntos por um motivo, que são os testes serem muito parecidos. Será possível perceber logo abaixo que os testes para ambos são muito similares, já que, devido ao fato de não se possuir implementação para os métodos, mas sim apenas seu esqueleto, ambos foram testados verificando somente o sucesso e o erro dos métodos, não implementando nenhum teste mais específico, como o da classe **Conexao**. Portanto, segue um exemplo da classe **GeneroDAO**:

```
import unittest

from main.models.genero import Genero
from main.persistence.GeneroDAO import GeneroDAO


class GeneroDAOTest(unittest.TestCase):

    gd = None
```

```
        g1 = None
        g2 = None

        def setUp(self) -> None:
            self.gd = GeneroDAO()
            self.g1 = Genero(id=1, nome="Genero")

        def testa_sucesso_insercao(self):
            self.assertTrue(self.gd.insert(self.g1))

        def testa_erro_insercao(self):
            self.assertFalse(self.gd.insert(self.g2))

        def testa_sucesso_delecao(self):
            self.assertTrue(self.gd.delete(self.g1))

        def testa_erro_delecao(self):
            self.assertFalse(self.gd.delete(self.g2))

        def testa_sucesso_atualizacao(self):
            self.assertTrue(self.gd.update(self.g1))

        def testa_erro_atualizacao(self):
            self.assertFalse(self.gd.update(self.g2))

        def testa_sucesso_selecao(self):
            self.assertIsNotNone(self.gd.select(id_genero=1))

        def testa_erro_selecao(self):
            self.assertIsNone(self.gd.select(id_genero=0))

        def testa_sucesso_selecao_todos(self):
            self.assertIsNotNone(self.gd.select_all())
```

E agora o exemplo da classe **Sistema**:

```
import unittest

from main.models.ator import Ator
from main.models.avaliacao import Avaliacao
from main.models.filme import Filme
from main.models.usuario import Usuario
from main.views.Sistema import Sistema


class SistemaTest(unittest.TestCase):
    sis = None
    ats = None
    fs = None
    u = None

    def setUp(self) -> None:
        self.sis = Sistema()
        self.ats = [Ator(id="nm00000"), Ator(id="nm00001")]
        self.fs = [Filme(id="tt00000"), Filme(id="tt00001")]
        self.u = Usuario(nickname="nick", email="email@email.com",
```

```
                                nome="nome", senha="12345",
                                atores_salvos=self.ats,
                                filmes_salvos=self.fs)

    def testa_unicidade_sistema(self):
        self.assertEqual(self.sis, Sistema())

    def testa_cadastrar_usuario(self):
        self.assertTrue(self.sis.cadastrar_usuario(self.u))

    def testa_login_usuario(self):
        self.assertTrue(self.sis.login_usuario(self.u.nickname,
                        self.u.senha))

    def testa_salvar_filme_usuario(self):
        self.assertTrue(self.sis.salva_filme_usuario(self.u,
                        Filme(id="tt00002")))

    def testa_excluir_filme_usuario(self):
        self.assertTrue(self.sis.exclui_filme_usuario(self.u,
                        Filme(id="tt00002")))

    def testa_buscar_filmes_salvos_usuario(self):
        self.assertIsNotNone(self.sis.busca_filmes_salvos_usuario(self.u))

    def testa_salvar_ator_usuario(self):
        self.assertTrue(self.sis.salva_ator_usuario(self.u,
                        Ator(id="nm00002")))

    def testa_excluir_ator_usuario(self):
        self.assertTrue(self.sis.exclui_ator_usuario(self.u,
                        Ator(id="nm00002")))

    def testa_buscar_atores_salvos_usuario(self):
        self.assertIsNotNone(self.sis.busca_atores_salvos_usuario(self.u))

    def testa_inserir_avaliacao_pelo_usuario(self):
        self.assertTrue(self.sis.insere_avaliacao_filme(self.u,
                        Avaliacao(id=1, data_hora="11/02/2023 12:10",
                        texto="Um filme muito bom...")))

    def testa_logout_usuario(self):
        self.assertFalse(self.sis.logout_usuario())
```

Com isso, é possível perceber que ambos testam condições em que o método tem sucesso e quando não tem sucesso (residência da similaridade), sendo possível utilizá-los de maneira melhor quando os seus métodos forem corretamente implementados. É importante ressaltar também que mais testes poderão ser necessários conforme estas classes forem crescendo, já que esta aplicação apresentada pode crescer bastante em especificação.

Por fim, é importante explicar o porque de não possuirmos testes para a classe App, apesar dela ser muito importante para o funcionamento do software. Devido à maneira com a qual se é implementado um software web utilizando o *framework* Flask, toda a lógica de funcionamento da interface web fica em um único módulo (App.py),

sendo ela a responsável não só por redirecionar chamadas de url, mas também carregar as páginas html. Com isso, apesar delas possuírem um retorno (do tipo str ou Response), não é possível verificar de maneira precisa o que está sendo retornado, uma vez que estas páginas dependem de entradas do usuário para tal. Assim, estes testes para esta classe se configuraríam mais como testes de integração do que testes unitários, fugindo da ideia inicial do trabalho.

10. Considerações Finais

Após esta apresentação, foi possível observar a proposta e justificativa para a implementação de tal aplicação web, contendo as informações necessárias para dar partida a implementação, como: levantamento de requisitos, estimativa de duração do projeto, modelagem de dados, designs, testes unitários e ferramentas a serem utilizadas. Com isso, se espera que a aplicação possa de fato se basear no que foi descrito, atingindo a meta de implementação e produzindo uma aplicação coesa com que foi proposto.