



*Base de dados*  
*Aplicação de monitorização de filmes*  
*e séries*

LEIC - Grupo 1303

2024/2025

Catarina Bastos - [up202307631@fe.up.pt](mailto:up202307631@fe.up.pt)

Nuno Costa - [up202305503@fe.up.pt](mailto:up202305503@fe.up.pt)

Vasco Gonçalves - [up202305513@fe.up.pt](mailto:up202305513@fe.up.pt)

# Índice

1. [Descrição](#)
2. [Atributos](#)
3. [Integração de AI - Chat GPT](#)
4. [Diagrama Relacional](#)
  - 4.1. [Antes da integração de AI](#)
  - 4.2. [Integração de AI](#)
  - 4.3. [Após a integração de AI](#)
5. [Dependências Funcionais e Formas Normais](#)
  - 5.1. [Antes da integração de AI](#)
  - 5.2. [Integração de AI](#)
  - 5.3. [Após integração de AI](#)
6. [Restrições](#)
7. [SQLite \(integração de AI\)](#)
8. [Integração geral de AI](#)
9. [Diagrama de classes - UML](#)

## Descrição

O objetivo desta base de dados é gerir informações relacionadas com filmes, séries e atividades associadas aos utilizadores de uma plataforma de rastreamento de conteúdo audiovisual, baseando-se numa aplicação chamada Queue que é uma espécie de rede social para filmes e séries.

Efetivamente, a classe utilizador contém informações pessoais, como nome, bio, género, data de nascimento, email e password, mas também dados de engajamento, como número de seguidores, de utilizadores que segue, de pontos e a data de entrada na aplicação. Cada utilizador pode criar listas personalizadas para organizar os seus conteúdos, com a possibilidade de classificar listas como favoritas. Além disso, os utilizadores podem seguir-se uns aos outros e interagir com o sistema através de atividades, que consiste na avaliação de filmes, séries ou episódios, associando ratings, críticas, progresso e incluindo a data de início e de fim desta atividade.

O sistema distingue dois tipos principais de itens: filmes, e séries. Cada série está associada a temporadas que, por sua vez, contêm episódios específicos. Restrições, como duração positiva e datas de lançamento válidas, garantem a consistência dos dados. Cada item está também associado a determinadas faixas etárias e categorias, o que proporciona filtros úteis para os utilizadores.

A base de dados inclui também informações sobre os participantes, como atores ou diretores, e suas contribuições para o item em questão. Cada participante é associado a atributos como nome, data de nascimento, género, biografia, profissão, número de projetos em que participaram. Além disso, existe também a entidade plataforma que armazena dados sobre serviços de streaming, com custos associados.

Concluindo, esta base de dados garante que os utilizadores interajam com o conteúdo audiovisual de maneira personalizada, eficiente e organizada.

# Atributos

## Utilizador

- Nome;
- Bio;
- Género;
- Data de Nascimento;
- Data de Entrada;
- Número de Seguidores;
- Número de Seguindo;
- Número de Pontos;
- Email;
- UPassword;

## Lista

- Nome;
- Descrição;
- Número de Itens;
- Tipo de Lista;
- Número de Favoritos;

## Adição

- Data de Adição;

## Participação

- Data de Adição;

## Atividade

- Rating;
- Data de Início;
- Data de Fim;
- Progresso;
- Crítica;

## Participante

- Nome;
- Data de Nascimento;
- Género;
- Biografia;
- Número de projetos;

## Item

- Nome;
- Descrição;
- Classificação;
- Tipo;

### – Filme

- Data de Lançamento;
- Duração;
- Local de Filmagem;
- Linguagem;

### – Série

- Número de Temporadas;
- Data de Início;
- Data de Fim;

## Temporada

- Número de Episódios;
- Número;
- Descrição;

## Episódio

- Nome;
- Número;
- Descrição;
- Classificação;
- Duração;
- Local de Filmagem;
- Linguagem;
- Data de Lançamento;

## Faixa Etária

- Idade de Início;
- Idade de Fim;

## Plataforma

- Nome;
- Custo;

## Categoria

- Nome;

# Integração de AI – Chat GPT

Como era pretendido, utilizamos uma ferramenta de inteligência artificial (Chat GPT) para nos apoiar na realização deste projeto.

Algumas das alterações sugeridas foram:

## 1. Entidade "Lista":

- **Objetivo:** Representar listas de filmes/séries que o utilizador quer ver ou já viu.
- **Alteração sugerida:**
  - Acrescentar um campo `tipoLista` que indique se a lista é de filmes/séries que o utilizador já viu ou pretende ver.
  - O campo `numCoisas` pode ser renomeado para algo mais específico, como `numItens`.
  - Poderia ter também um campo `dataAdicao`, que indica quando o utilizador adicionou o filme/série à lista.

## 2. Detalhamento da Entidade "Assistiu":

- **Modificação sugerida:** A entidade `Assistiu` poderia ser renomeada para algo mais descritivo como `Histórico de Visualização` ou `Atividade`, e poderia incluir mais atributos como: data de início, data de conclusão, progresso (caso tenha parado no meio de uma série).
- **Motivo:** Isso adicionaria detalhes importantes sobre o que o utilizador já assistiu, dando mais flexibilidade à aplicação.

## 10. Atributos de Episódio e Temporada:

- **Modificação sugerida:** A classe `Episódio` poderia incluir atributos como `Data de Lançamento`, e `Temporada` poderia incluir `Sinopse` da temporada como um todo.
- **Motivo:** Detalha melhor o conteúdo e fornece uma navegação mais clara entre temporadas e episódios.

## Alterações e Sugestões Adicionais

1. **Atividade:** Incluí um campo `idItem` para conectar filmes, séries ou episódios diretamente. Isso evita a necessidade de "alterar o diagrama" para acomodar diferentes tipos de mídia.
2. **Adição:** A relação foi simplificada, associando itens diretamente à lista.
3. **Gostou/Sugestão:** Adicionei flexibilidade para que as sugestões e gostos possam ser relacionados a qualquer tipo de item (filme ou série).
4. **Segue:** Formalizei a relação entre seguidores e seguidos, vinculando-os diretamente ao utilizador.

Através destas mudanças foi possível tornar alguns nomes mais claros e semânticos, proporcionar mais flexibilidade na gestão do histórico de visualizações e progressos e, por fim permitir que utilizadores interajam entre si de uma forma mais rica.

# Diagrama Relacional

## Antes da integração de AI

*Utilizador*(id, nome, bio, género, dataNascimento, dataEntrada, nSeguidores, nPontos, nSeguindo, email, uPassword)

*Filme*(id, nome, descrição, classificação, tipo, dataLançamento, duração, localFilmagem, linguagem)

*Série*(id, nome, descrição, classificação, tipo, nTemporadas, dataInício, dataFim)

*Temporada*(id, nEpisódios, número, descrição, idSérie -> Série)

*Episódio*(id, nome, número, descrição, classificação, tipo, duração, localFilmagem, linguagem, dataLançamento, idTemporada -> Temporada)

*Lista*(id, nome, descrição, numItens, tipoLista, nFavoritos, idUtilizador -> Utilizador)

*Adição*(idLista -> Lista, idItem -> Item, dataAdição)

*FaixaEtária*(id, idadeInício, idadeFim)

*Atividade*(idUtilizador -> Utilizador, idItem -> Item, rating, dataInício, dataFim, progresso, crítica)

*Participante*(id, nome, dataNascimento, género, biografia, profissão, nProjetos)

*Plataforma*(id, nome, custo)

*Categoria*(id, nome)

*Seguir*(idSegue -> Utilizador, idSeguido -> Utilizador)

*ItemCategoria*(idItem -> Item, idCategoria -> Categoria)

*Favorito*(idLista -> Lista, idUtilizador -> Utilizador)

## Integração de AI

Após desenvolvermos o diagrama relacional utilizamos o apoio de ferramentas de IA (mais especificamente, o chatGPT) para promover a coerência e eficiência do mesmo. Incluímos algumas imagens que indicam algumas das suas sugestões que consideramos mais pertinentes.

podes indicar formas e sugestões que tornariam este diagrama relacional melhor e mais coerente?

### Relação mais robusta entre Faixa Etária e Itens

Atualmente, não existe uma relação clara entre `FaixaEtária` e os itens que possuem restrições de idade. Sugere-se incluir um relacionamento direto entre `FaixaEtária` e `Item`:

- Exemplo: Adicionar `idFaixaEtária->FaixaEtária` em `Item` para indicar a faixa etária aplicável a filmes, séries ou episódios.

### Melhor modelagem de Participantes

Os `Participante` (atores, diretores, etc.) não estão diretamente conectados a `Filme`, `Série` ou `Episódio`. Sugere-se adicionar uma relação intermediária chamada `Participação`:

- Exemplo:
  - Nova entidade `Participação`: `{idParticipante->Participante, idItem->Item, papel}`.
  - Atributo `papel` pode descrever a função do participante (ex.: "Ator", "Diretor", etc.).

Nova entidade `Item`: `{id, nome, descrição, classificação, tipo, dataLançamento, duração, localFilmagem, linguagem}`.

`Filme`, `Série` e `Episódio` herdam propriedades de `Item`.

## Após a integração de AI

*Utilizador*(id, nome, bio, género, dataNascimento, dataEntrada, nSeguidores, nPontos, nSeguindo, email, uPassword)

*Item*(id, nome, descrição, classificação, tipo, idFaixaEtária->FaixaEtária)

*Filme*(idItem->Item, dataLançamento, duração, localFilmagem, linguagem)

*Série*(idItem->Item, nTemporadas, dataInício, dataFim)

*Temporada*(id, nEpisódios, número, descrição, idSérie->Série)

*Episódio*(id, nome, número, descrição, classificação, duração, localFilmagem, linguagem, dataLançamento, idTemporada->Temporada)

*Lista*(id, nome, descrição, numItens, tipoLista, nFavoritos, idUtilizador->Utilizador)

*Adição*(idLista->Lista, idItem->Item, dataAdição)

*FaixaEtária*(id, idadeInício, idadeFim)

*Atividade*(idUtilizador->Utilizador, idItem->Item, rating, dataInício, dataFim, progresso, crítica)

*Participante*(id, nome, dataNascimento, género, biografia, nProjetos)

*Participação*(idParticipante->Participante, idItem->Item, profissão)

*Plataforma*(id, nome, custo)

*Categoria*(id, nome)

*Seguir*(idSegue->Utilizador, idSeguido->Utilizador)

*ItemCategoria*(idItem->Item, idCategoria->Categoria)

*Favorito*(idLista->Lista, idUtilizador->Utilizador)



# Dependências Funcionais e Formas Normais

## Antes da integração de AI

**Utilizador**(id, nome, bio, género, dataNascimento, dataEntrada, nSeguidores, nPontos, nSeguindo, email, uPassword)

1. {id} -> {nome, bio, género, dataNascimento, dataEntrada, nSeguidores, nPontos, nSeguindo, email, uPassword}
2. {email} -> {id, nome, bio, género, dataNascimento, dataEntrada, nSeguidores, nPontos, nSeguindo, uPassword}

**Item**(id, nome, descrição, classificação, tipo, idFaixaEtária->FaixaEtária)

1. {id} -> {nome, descrição, classificação, tipo, idFaixaEtária}

**Filme**(idItem->Item, dataLançamento, duração, localFilmagem, linguagem)

1. {idItem} -> {dataLançamento, duração, localFilmagem, linguagem}

**Série**(idItem->Item, nTemporadas, dataInício, dataFim)

1. {idItem} -> {nTemporadas, dataInício, dataFim}

**Temporada**(id, nEpisódios, número, descrição, idSérie->Série)

1. {id} -> {nEpisódios, número, descrição, idSérie}

**Episódio**(id, nome, número, descrição, classificação, duração, localFilmagem, linguagem, dataLançamento, idTemporada->Temporada)

1. {id} -> {nome, número, descrição, classificação, duração, localFilmagem, linguagem, dataLançamento, idTemporada}

**Lista**(id, nome, descrição, numItens, tipoLista, nFavoritos, idUtilizador->Utilizador)

1. {id} -> {nome, descrição, numItens, tipoLista, nFavoritos, idUtilizador}

**Adição**(idLista->Lista, idItem->Item, dataAdição)

1. {idLista, idItem} -> {dataAdição}

## Antes da integração de AI(continuação)

**FaixaEtária**(id, idadeInício, idadeFim)

1. {id} -> {idadeInício, idadeFim}

**Atividade**(idUtilizador->Utilizador, idItem->Item, rating, dataInício, dataFim, progresso, crítica)

1. {idUtilizador, idItem} -> {rating, dataInício, dataFim, progresso, crítica}

**Participante**(id, nome, dataNascimento, género, biografia, nProjetos)

1. {id} -> {nome, dataNascimento, género, biografia, nProjetos}

**Plataforma**(id, nome, custo)

1. {id} -> {nome, custo}
2. {nome} -> {id, custo}

**Categoria**(id, nome)

1. {id} -> {nome}
2. {nome} -> {id}

**Seguir**(idSegue->Utilizador, idSeguido->Utilizador)

1. {idSegue, idSeguido} -> {}

**ItemCategoria**(idItem->Item, idCategoria->Categoria)

1. {idItem, idCategoria} -> {}

**Favorito**(idLista->Lista, idUtilizador->Utilizador)

1. {idLista, idUtilizador} -> {}

Para garantir a 3ª Forma Normal é necessário assegurar que para todo  $A \rightarrow B$  não trivial  $A$  é uma superkey ou  $B$  consiste apenas em atributos primos (são membros de pelo menos uma chave da relação). Por outro lado, para cumprir a Forma Normal de Boyce-Codd basta apenas que para todo  $A \rightarrow B$  não trivial  $A$  é uma superkey.

Deste modo, podemos concluir que todas as relações se encontram na 3ª Forma Normal (3NF) ou na Forma Normal de Boyce-Codd (BCNF).

# Integração de AI

## Dependências Funcionais

o que consideras importante ter em conta no desenvolvimento das dependências funcionais deste modelo e que sugestões tens?

O par `nome, data` também é único (um filme não pode ter o mesmo nome e data que outro filme).

A combinação de `idSerie` (à qual pertence) e `numero` (da temporada) é única.

O par `idTemporada, nome` (um nome único dentro de uma temporada) também determina os outros atributos.

O par `idUtilizador, nome` também determina os atributos da lista (um utilizador não pode ter duas listas com o mesmo nome).

## Formas Normais

o que consideras importante ter em conta em relação à 3FN e BCNF deste modelo?

O modelo apresentado parece estar em conformidade tanto com a 3FN quanto com a BCNF. Ele não apresenta dependências transitivas ou parciais e todas as dependências funcionais têm a parte esquerda como uma superchave. No entanto, se algum atributo de uma tabela se mostrar desnecessário ou derivável de outras tabelas (como `nFavoritos` ou `nItens`), considerar removê-los para evitar redundância e otimizar ainda mais o design.

## Após a integração de AI

**Utilizador**(id, nome, bio, género, dataNascimento, dataEntrada, nSeguidores, nPontos, nSeguindo, email, uPassword)

1. {id} -> {nome, bio, género, dataNascimento, dataEntrada, nSeguidores, nPontos, nSeguindo, email, uPassword}
2. {email} -> {id, nome, bio, género, dataNascimento, dataEntrada, nSeguidores, nPontos, nSeguindo, uPassword}

**Item**(id, nome, descrição, classificação, tipo, idFaixaEtária->FaixaEtária)

1. {id} -> {nome, descrição, classificação, tipo, idFaixaEtária}

**Filme**(idItem->Item, dataLançamento, duração, localFilmagem, linguagem)

1. {idItem} -> {dataLançamento, duração, localFilmagem, linguagem}
2. {nome, dataLançamento} -> {idItem, duração, localFilmagem, linguagem}

**Série**(idItem->Item, nTemporadas, dataInício, dataFim)

1. {idItem} -> {nTemporadas, dataInício, dataFim}
2. {nome, dataInício, dataFim} -> {idItem, nTemporadas}

**Temporada**(id, nEpisódios, número, descrição, idSérie->Série)

1. {id} -> {nEpisódios, número, descrição, idSérie}
2. {idSérie, número} -> {id, nEpisódios, descrição}

**Episódio**(id, nome, número, descrição, classificação, duração, localFilmagem, linguagem, dataLançamento, idTemporada->Temporada)

1. {id} -> {nome, número, descrição, classificação, duração, localFilmagem, linguagem, dataLançamento, idTemporada}
2. {idTemporada, nome} -> {id, número, descrição, classificação, duração, localFilmagem, linguagem, dataLançamento}

**Lista**(id, nome, descrição, numItens, tipoLista, nFavoritos, idUtilizador->Utilizador)

1. {id} -> {nome, descrição, numItens, tipoLista, nFavoritos, idUtilizador}
2. {idUtilizador, nome} -> {id, descrição, numItens, tipoLista, nFavoritos}

## Após a integração de AI(continuação)

*Adição*(idLista->Lista, idItem->Item, dataAdição)

1. {idLista, idItem} -> {dataAdição}

*FaixaEtária*(id, idadeInício, idadeFim)

1. {id} -> {idadeInício, idadeFim}

*Atividade*(idUtilizador->Utilizador, idItem->Item, rating, dataInício, dataFim, progresso, crítica)

1. {idUtilizador, idItem} -> {rating, dataInício, dataFim, progresso, crítica}

*Participante*(id, nome, dataNascimento, género, biografia, nProjetos)

1. {id} -> {nome, dataNascimento, género, biografia, nProjetos}

*Plataforma*(id, nome, custo)

1. {id} -> {nome, custo}
2. {nome} -> {id, custo}

*Categoria*(id, nome)

1. {id} -> {nome}
2. {nome} -> {id}

*Seguir*(idSegue->Utilizador, idSeguido->Utilizador)

1. {idSegue, idSeguido} -> {}

*ItemCategoria*(idItem->Item, idCategoria->Categoria)

1. {idItem, idCategoria} -> {}

*Favorito*(idLista->Lista, idUtilizador->Utilizador)

1. {idLista, idUtilizador} -> {}

Por fim, concluímos que as relações estão de acordo com a 3FN e BCNF uma vez que não apresentam dependências parciais ou transitivas, ou seja, a parte esquerda de cada dependência é uma superkey que permite determinar todos os atributos da relação.

# Restrições

De forma a garantir uma manutenção eficiente da base de dados e oferecer mais segurança aos usuários, foram implementadas algumas restrições a determinadas classes, nomeadamente do tipo de chave, integridade referencial, CHECK e NOT NULL.

Deste modo, quando a restrição NOT NULL é aplicada a um atributo torna-se obrigatório a presença deste mesmo atributo para a definição mínima da classe. Praticamente todas as classes possuem pelo menos um atributo com a restrição NOT NULL (como se pode comprovar pelo UML). Alguns exemplos destes casos são:

- Os atributos nome, data de nascimento, data de entrada, email e password da classe Utilizador são considerados NOT NULL, visto que sem estes seria impossível definir o Utilizador enquanto que os restantes atributos, como a bio e género podem ser opcionais;
- O atributo idade de início da classe Faixa Etária é também fundamental para a definição desta classe uma vez que essa informação é necessária para determinar a faixa mínima de idade permitida para o conteúdo;
- Os atributos nome, número de temporadas e data de início da classe Série também se encontram com a restrição NOT NULL pois sem estes não se consegue definir uma Série e é através destes que se consegue distinguir esta das outras.

Aplicou-se a restrição UNIQUE aos atributos/grupos de atributos que precisam de ser distintos para garantir a integridade e evitar duplicações:

- O atributo email da classe Utilizador é único (não é possível existirem dois utilizadores com o mesmo email);
- O conjunto nome e data de lançamento da classe Filme é UNIQUE (não existem filmes que tenham igual nome e data de lançamento);
- O atributo nome da classe Categoria também tem a restrição UNIQUE (não existem duas categorias com o mesmo nome).

## Restrições (continuação)

Por sua vez, utilizou-se a restrição CHECK para garantir a validade dos dados inseridos, evitando inconsistências e valores fora das restrições estabelecidas para a base de dados. Aqui estão algumas dessas restrições aplicadas ao nosso modelo:


- De forma a garantir uma maior otimização, o atributo nome e password têm comprimentos limitados (nome deverá ter entre 1 a 50 caracteres e password deverá ter entre 8 a 50);
- O atributo data de início da classe Atividade deverá ter um valor inferior ou igual ao do atributo data de fim da mesma classe;
- O atributo duração da classe Filme e da classe Episódio deverá ser maior que 0 dado que não existem filmes ou episódios com duração igual a 0 unidades de tempo;
- Para certificar que o nosso modelo é coerente o atributo número de projetos da classe Participante deve também ser superior a 0 porque só assim se considera que este participante realmente esteve envolvido num determinado item.

Por fim, relativamente a restrições de integridade referencial, foram inseridas chaves estrangeiras a classes que se encontram relacionadas com outras.


- Na classe Lista, utiliza-se a chave estrangeira idUtilizador que corresponde à chave primária id na classe Utilizador;
- Para garantir que cada temporada está associada a uma série válida utiliza-se a chave estrangeira idSérie na classe Temporada que se refere ao id da série a que esta temporada pertence;
- Da mesma forma, na classe Episódio utiliza-se a chave estrangeira idTemporada que se refere ao id da temporada a que este episódio pertence, fazendo com que a exclusão de uma temporada resulte na exclusão automática dos seus episódios.

# SQLite (integração de AI)

## Create2.sql

 **create1.sql**  
File

What can i improve here

 Let me review your file for any improvements. I'll analyze the contents and provide feedback. One moment!

Your SQL script is quite comprehensive and well-structured, defining multiple tables with constraints, relationships, and checks. Here are a few points for improvement or verification:

**Atividade :**

- Validate `progresso` (e.g., ensure it's within an acceptable range, such as 0–100%).

**Filme and Episodio :**

- Add constraints to ensure `dataLancamento` is not in the future (if applicable).

**Referential Integrity:**

- Use `ON DELETE CASCADE` or `ON UPDATE CASCADE` where appropriate for foreign keys to maintain data integrity during updates or deletions.

**Blob Usage:**

- Storing `BLOB` (e.g., `fotoPerfil`) in the database can impact performance. Evaluate whether this is necessary or if it could be stored externally (e.g., in a file system or cloud storage).

## Populate2.sql

A AI foi utilizada para completar com mais informação a povoação da base de dados, como se pode ver através deste link:

<https://chatgpt.com/share/674cc5ae-a764-8005-b6c4-9776aec2334d>



# ***Integração geral de AI***

*A utilização do AI foi muito útil para a finalização do trabalho, uma vez que esta é uma ferramenta muito rápida e útil, especialmente quando se trata de SQL, apesar de algumas limitações, como por exemplo a dificuldade desta tecnologia para estar em sintonia com aquilo que é a ideia do projeto em si. Visto que já tínhamos utilizado AI no nosso UML, esta ferramenta acabou por não ser muito usada no desenvolvimento do ficheiro create2.sql.*

# Diagrama de classes – UML

