
**Escola Superior de Tecnologia e Gestão**

**Licenciatura em Engenharia Informática**

**Desenvolvimento de Aplicações Web**

**Projeto – Fase de Implementação**

# My Anime Collection

**Martinho José Novo Caeiro**

**Paulo António Tavares Abade**


**Beja, janeiro de 2025**

**Instituto Politécnico de Beja**
**Escola Superior de Tecnologia e Gestão**
**Licenciatura em Engenharia Informática**

**Desenvolvimento de Aplicações Web**

**Projeto – Fase de Implementação**

# My Anime Collection

**Martinho José Novo Caeiro**
**Paulo António Tavares Abade**

**DOCENTE**
**Professor Luís Carlos Bruno**

Trabalho realizado no âmbito da unidade curricular Desenvolvimento de Aplicações Web (Bruno, 2025)

**Beja, janeiro de 2025**

# Índice


**1 Introdução** **3**

**2 Análise do Sistema** **3**

**3 Caracterização dos Atores** **3**

3.1 Persona Nº1 - Miguel Silva . . . . . 4

3.2 Persona Nº2 - Catarina Silvestre . . . . . 4

**4 Diagrama de Casos de Uso** **5**

4.1 Caso de Uso Nº1 - Consultar Animes . . . . . 6

4.2 Caso de Uso Nº2 - Adicionar Animes a uma Lista . . . . . 7

4.3 Caso de Uso Nº3 - Consultar o Perfil de outro Utilizador . . . . . 7

4.4 Caso de Uso Nº4 - Avaliar uma Lista de Animes de outro Utilizador 8

4.5 Caso de Uso Nº5 (Partilhado) - Visualizar os animes com melhor avaliação . . . . . 8

**5 Desenho do Sistema** **9**

5.1 Modelação da Base de Dados . . . . . 9

5.2 Diagrama E/R . . . . . 9

5.3 Modelo Relacional . . . . . 10

5.4 Modelo Físico . . . . . 11

**6 Modelação de Interfaces Gráficas com o Utilizador** **12**

6.1 Storyboard(s) . . . . . 12

6.2 Interfaces do Caso de Uso 1 . . . . . 13

6.3 Interfaces do Caso de Uso 2 . . . . . 13

6.4 Interfaces do Caso de Uso 3 . . . . . 14

6.5 Interfaces do Caso de Uso 4 . . . . . 14

6.6 Interfaces do Caso de Uso 5 . . . . . 15

6.7 Protótipo de Média Fidelidade . . . . . 16

**7 Melhorias efetuadas na Análise e Desenho do Sistema** **17**

**8 Implementação** **18**

8.1 Arquitetura do Sistema . . . . . 18

8.2 Tecnologias Usadas . . . . . 19

8.3 Desenvolvimento da API . . . . . 19

8.4 Especificação da Interface . . . . . 20

8.5 Decisões de Implementação . . . . . 20
8.6 Principais Casos Relevantes de Programação . . . . . 21
**9 Desenvolvimento da App Frontend/MVC** **21**
9.1 Decisões de Implementação . . . . . 21
9.2 Principais Casos Relevantes de Programação . . . . . 22
Martinho José Novo Caeiro . . . . . 23
Avaliar um Anime Especifíco . . . . . 23
Visualizar o perfil de Outro Utilizador . . . . . 23
Avaliar uma Lista de Animes de Outro Utilizador . . . . . 24
9.3 Paulo António Tavares Abade . . . . . 24
Consultar Animes . . . . . 24
Adicionar Animes a uma Lista . . . . . 25
**10 Conclusão e Trabalho Futuro** **28**
**Bibliografia** **29**

# Índice de Figuras

1 Diagrama de Casos de Uso . . . . . 5
2 Modelo Físico . . . . . 11
3 Interface do Caso de Uso 1 . . . . . 13
4 Interface do Caso de Uso 2 . . . . . 13
5 Interface do Caso de Uso 3 . . . . . 14
6 Interface do Caso de Uso 4 . . . . . 14
7 Interface do Caso de Uso 5 . . . . . 15
8 Protótipo de Média Fidelidade . . . . . 16
9 Arquitetura do Sistema . . . . . 18

Índice de Tabelas

# 1 Introdução

Neste relatório iremos fazer a Implementação de uma aplicação de gestão de media, que tem como objetivo ajudar a organizar o conteúdo dos fãs de anime, ao ser possível categorizar os animes entre: Por visualizar, a visualizar, visualizados e também ser possível dar uma classificação aos mesmos ou às listas de outros utilizadores. Irá ser feita uma API que irá conter todos os animes e os seus dados relacionados, que será acompanhada de uma base de dados que irá guardar as informações dos utilizadores. Neste relatório será dividido em três grandes partes, sendo estas a Análise e Desenho do Sistema cujas quais já foram desenvolvidas no relatório anterior, seguida da parte de Implementação onde serão referidos os metodos de construção utilizados para fazer os casos de usos referidos na Analise e Desenho.

# 2 Análise do Sistema

Para analisar o sistema, é necessário caracterizar os atores que realizarão as tarefas neste sistema, e quais serão as tarefas que serão realizadas pelos mesmos, para isso serão utilizados os Casos de Uso, considerando a notação UML.

# 3 Caracterização dos Atores

Neste sistema o mesmo utilizador terá a posição de dois atores diferentes, sendo estes: **Visualizador** e **Gestor de Listas**.
Inicialmente, todos os utilizadores assumem o papel de "Visualizador" e depois irão assumir temporariamente o papel de gestor de listas. Para explicar como funciona, será feito o uso de Personas, que são personagens fictícios para representar o futuro utilizador deste sistema.

## 3.1 Persona Nº1 - Miguel Silva

O Miguel tem 18 anos e entrou no MyAnimeCollection para descobrir novos animes. Ao visualizar o anime "One Piece" e ter-se interessado pelo mesmo, adicionou este anime à sua lista "A Visualizar". Esta lista foi criada por ele, e é uma das listas que ele tem disponíveis no seu perfil. Após visualizar o anime "One Piece", avaliou-o com 4 estrelas.

## 3.2 Persona Nº2 - Catarina Silvestre

A Catarina tem 19 anos e descobriu o MyAnimeCollection pelo Miguel, e então quis verificar o perfil dele para ver que animes ele tem nas suas listas. Ao aceder o perfil do Miguel, viu as listas que ele tinha disponíveis no seu perfil. Após ver a lista de "A Visualizar" do Miguel, avaliou-a com 5 estrelas.

# 4 Diagrama de Casos de Uso

Para criar o diagrama de casos de uso, foi utilizado a plataforma draw.io, respeitando a notação de casos de uso do UML.

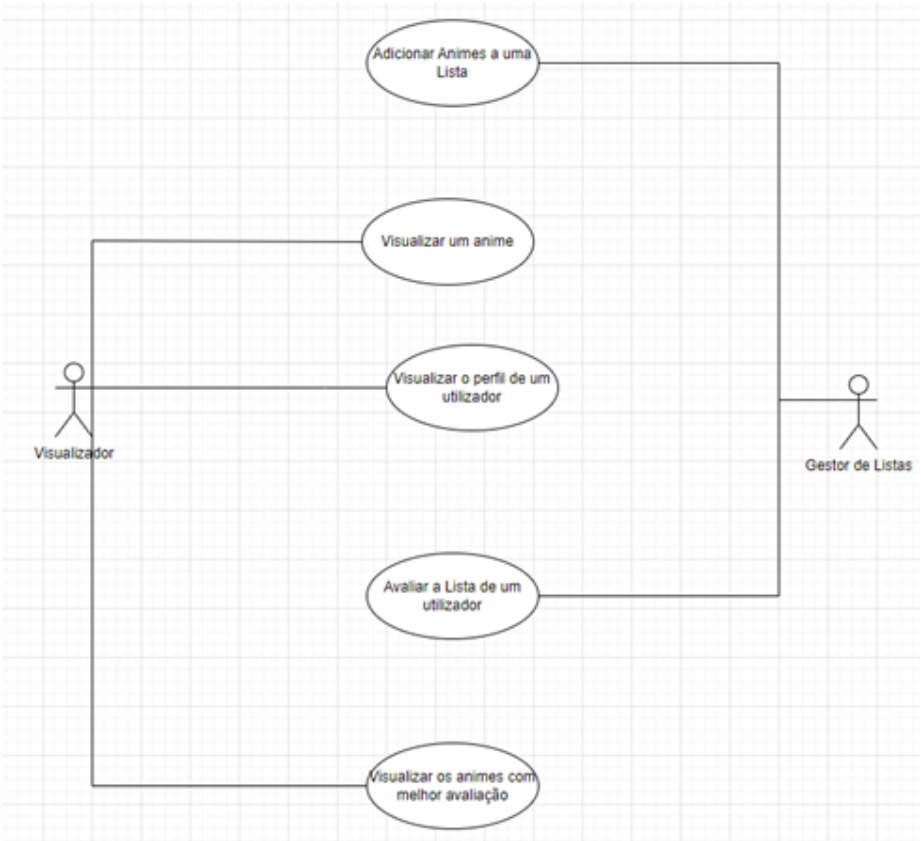



Figura 1: Diagrama de Casos de Uso

## 4.1 Caso de Uso Nº1 - Consultar Animes

| | |
|---|---|
| **Descrição** | Permite ao utilizador pesquisar informações sobre um anime específico, como sinopse, episódios, etc. |
| **Atores** | Visualizador |
| **Pré-Condições** | O utilizador deve estar conectado à internet e estar com a sessão iniciada |
| **Pós-Condições** | O anime é exibido com detalhes como sinopse, episódios, etc. |
| **Cenário Principal** | O Visualizador pesquisa um anime pelo nome, e o sistema retorna as informações detalhadas |
| **Situação de Falha** | Mostra um erro que identifica se o anime não foi encontrado ou foi um erro de conexão |

Tabela 1: Caso de Uso Nº1

## 4.2 Caso de Uso Nº2 - Adicionar Animes a uma Lista

| | |
|---|---|
| **Descrição** | Permite ao utilizador adicionar um anime à sua lista pessoal de animes para organizar seu progresso de visualização |
| **Atores** | Gestor de Lista |
| **Pré-Condições** | O utilizador deve estar conectado à internet e estar com a sessão iniciada |
| **Pós-Condições** | O anime é adicionado à lista com sucesso |
| **Cenário Principal** | O Gestor de Lista navega até o anime desejado e seleciona a opção "Adicionar à lista", categorizando-o (Por Visualizar, A Visualizar, Visualizados, ou uma criada pelo utilizador) |
| **Situação de Falha** | Mostra um erro que identifica uma falha na adição à lista por erro de sistema ou da conexão |

Tabela 2: Caso de Uso Nº2

## 4.3 Caso de Uso Nº3 - Consultar o Perfil de outro Utilizador

| | |
|---|---|
| **Descrição** | Permite visualizar o perfil e listas de outros utilizadores |
| **Atores** | Visualizador |
| **Pré-Condições** | O perfil do outro utilizador deve ser acessível. |
| **Pós-Condições** | As informações do perfil e listas são exibidas |
| **Cenário Principal** | O Visualizador busca pelo nome de outro utilizador e, se o perfil for acessível, o sistema exibe as informações do perfil, como listas e avaliações |
| **Situação de Falha** | Mostra um erro que identifica se a falha foi por inexistência do utilizador ou falha de conexão |

Tabela 3: Caso de Uso Nº3

## 4.4 Caso de Uso Nº4 - Avaliar uma Lista de Animes de outro Utilizador

| | |
|---|---|
| **Descrição** | Permite ao utilizador avaliar as listas de animes de outros utilizadores |
| **Atores** | Gestor de Lista |
| **Pré-Condições** | O utilizador deve estar conectado e estar com a sessão iniciada e a lista de outro utilizador deve ser acessível |
| **Pós-Condições** | A avaliação é submetida e exibida |
| **CenárioPrincipal** | O utilizador acessa a lista de outro utilizador, avalia e submete a avaliação, que é registada com sucesso |
| **Situação de Falha** | Mostra um erro que identifica uma falha ao enviar a avaliação devido a erro no sistema ou na conexão |

Tabela 4: Caso de Uso Nº4

## 4.5 Caso de Uso Nº5 (Partilhado) - Visualizar os animes com melhor avaliação

| | |
|---|---|
| **Descrição** | Exibe uma lista dos animes mais bem avaliados com base nas classificações de todos os utilizadores |
| **Atores** | Visualizador (não precisa estar com sessão iniciada) |
| **Pré-Condições** | Deve haver dados suficientes de avaliações para gerar uma lista |
| **Pós-Condições** | Lista dos animes mais bem avaliados é exibida |
| **Cenário Principal** | O Visualizador acessa a página de rankings, e o sistema exibe os animes com as melhores avaliações |
| **Situação de Falha** | Mostra um erro que identifica um erro de sistema |

Tabela 5: Caso de Uso Nº5

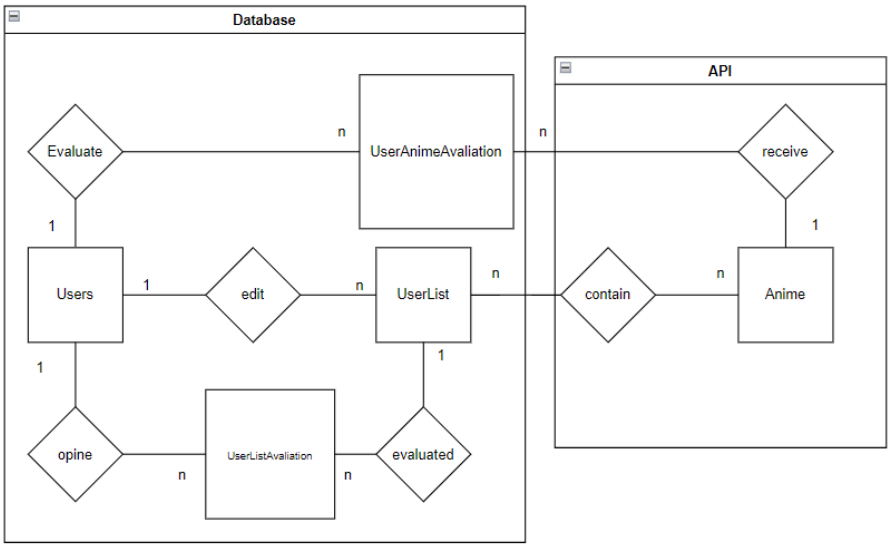
# 5 Desenho do Sistema

## 5.1 Modelação da Base de Dados

Nesta parte será mostrada como será o sistema, considerando as suas tabelas e relações entre elas, e depois será mostrado o modelo físico que indica as relações entre as tabelas, os atributos e o tipo dos mesmos. Neste sistema, existe a particularidade da convivência da API com a base de dados, visto que ambas necessitam um da outra.

## 5.2 Diagrama E/R

Este é o diagrama de entidade-relação utilizado pelo sistema do MyAnimeCollection:



## 5.3 Modelo Relacional

- Users(user_id, email, password, name, age, biography)
- UserList(userlist_id, user_id, name, description, anime_ids)
- UserListAvaliation (userlistavaliation_id, user_id, id_listaUtilizador, avaliation, datacreated)
- UserAnimeAvaliation (useranimelistavaliation_id, user_id, anime_id, avaliation, datacreated)
- Anime(anime_id, synopsis, genres, n_episodes, n_seasons)

## 5.4 Modelo Físico

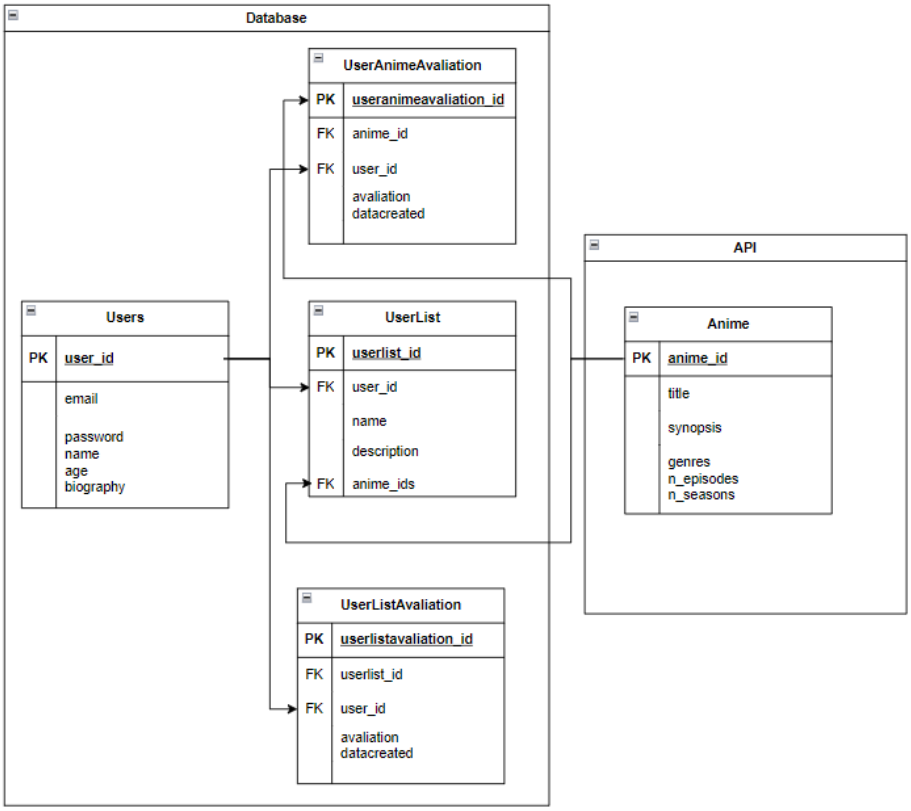
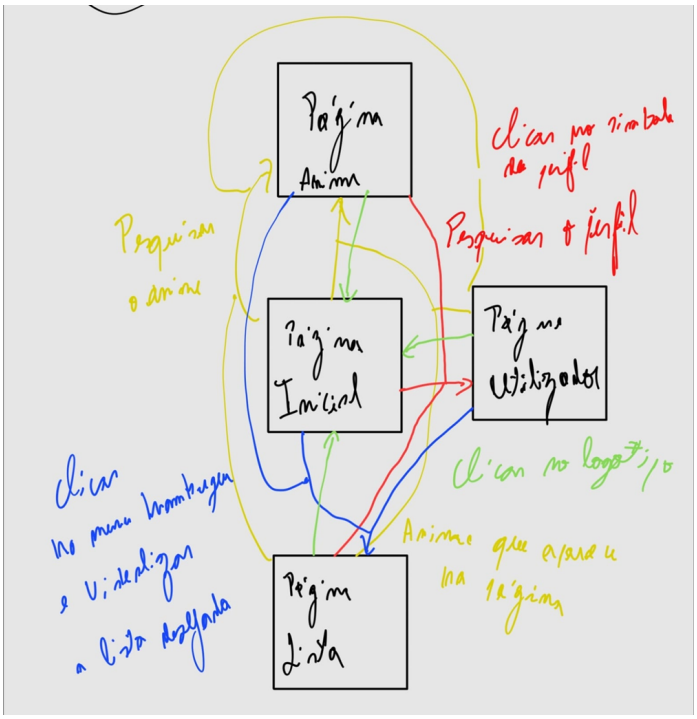



Figura 2: Modelo Físico

# 6 Modelação de Interfaces Gráficas com o Utilizador

## 6.1 Storyboard(s)

Neste Storyboard podemos visualizar as movimentações entre páginas principais. Existem mais opções que podem ser feitas nas páginas, porém serão apenas exploradas posteriormente.



## 6.2 Interfaces do Caso de Uso 1

Este é o caso de uso de consultar animes, onde o utilizador pode pesquisar informações sobre um anime específico, como sinopse, episódios, etc.

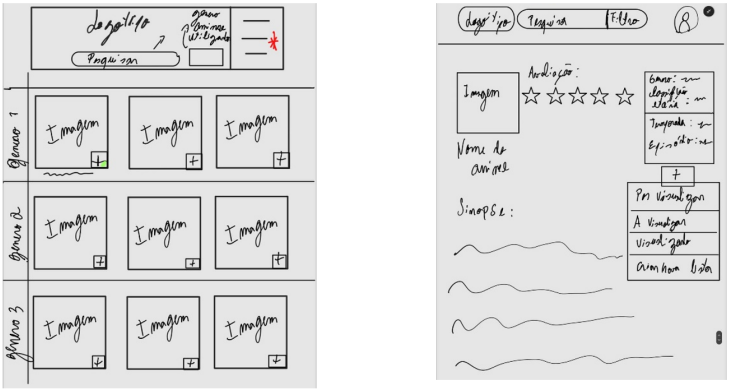


Figura 3: Interface do Caso de Uso 1

## 6.3 Interfaces do Caso de Uso 2

Este é o caso de uso de adicionar animes a uma lista, onde apenas o dono da lista pode adicionar animes à mesma.

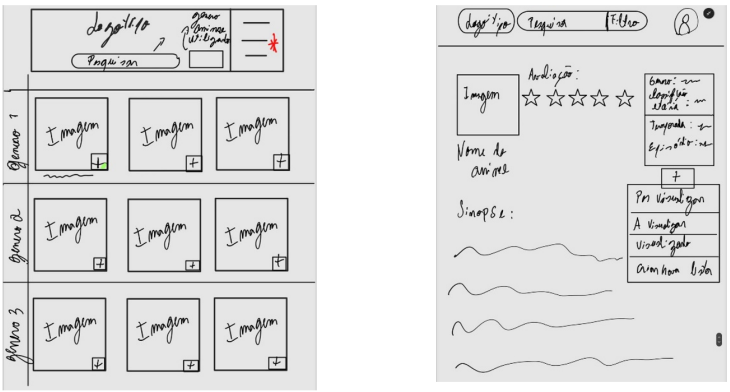


Figura 4: Interface do Caso de Uso 2

## 6.4 Interfaces do Caso de Uso 3

Este é o caso de uso de consultar o perfil de outro utilizador, onde o utilizador pode visualizar o perfil e listas de outros utilizadores.

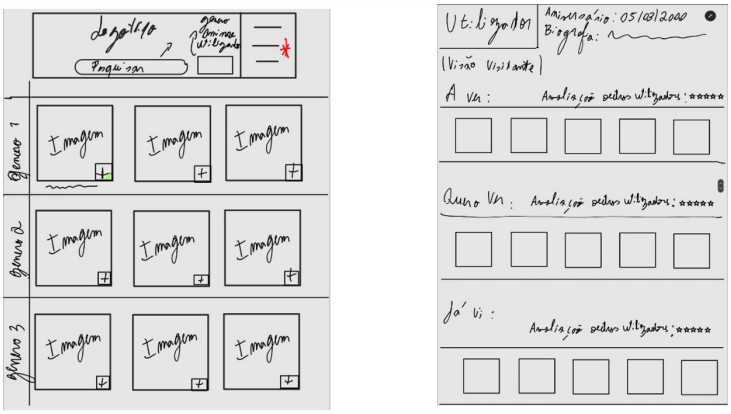


Figura 5: Interface do Caso de Uso 3

## 6.5 Interfaces do Caso de Uso 4

Este é o caso de uso de avaliar uma lista de animes de outro utilizador, onde o utilizador pode avaliar as listas de animes de outros utilizadores.

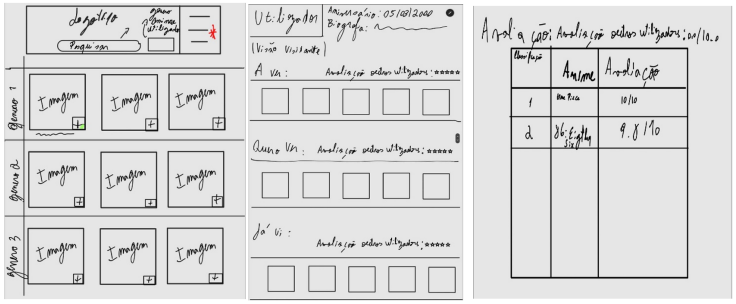


Figura 6: Interface do Caso de Uso 4

## 6.6 Interfaces do Caso de Uso 5

Este é o caso de uso de visualizar os animes com melhor avaliação, onde o utilizador pode visualizar os animes mais bem avaliados com base nas classificações de todos os utilizadores. Este caso de uso vai sofrer uma melhoria na fase de implementação.

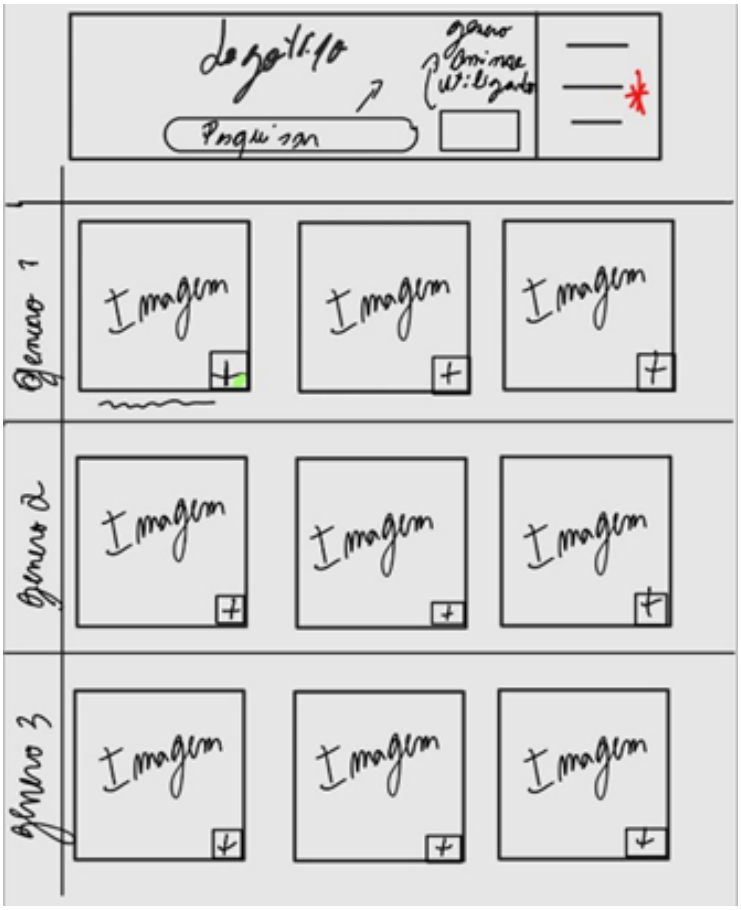



Figura 7: Interface do Caso de Uso 5

## 6.7 Protótipo de Média Fidelidade

Estes é o protótipo de média fidelidade do sistema, onde é possível visualizar as páginas principais do sistema. Este protótipo foi feito com recurso ao Balsamiq, que permite criar protótipos de média fidelidade de forma rápida e eficiente, e permite ainda que os utilizadores trabalhem em conjunto para criar protótipos de forma colaborativa.

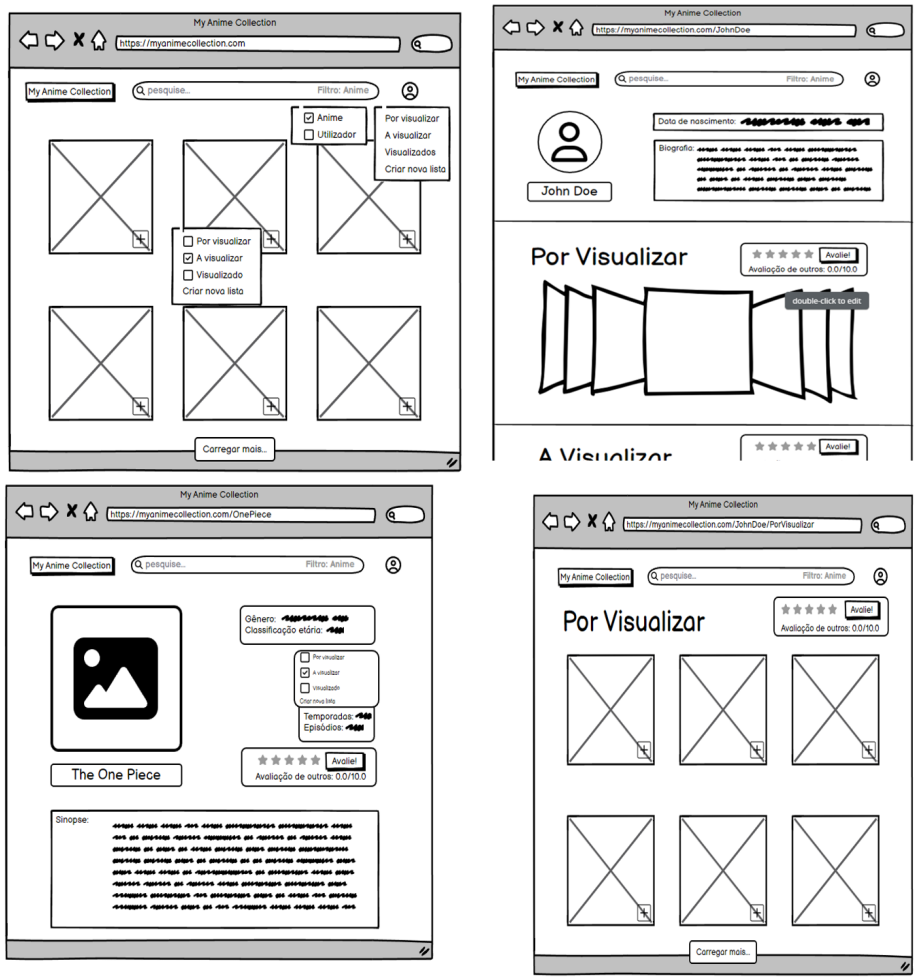



Figura 8: Protótipo de Média Fidelidade

# 7 Melhorias efetuadas na Análise e Desenho do Sistema

O Caso de Uso 5 foi aprofundado, agora sendo possivel visualizar animes por uma data especifica em vez de um periodo predefinido. Esta melhoria foi feita para que o utilizador possa ter uma maior flexibilidade na pesquisa de animes com melhor avaliação. Esta implementação torna o sistema mais completo e com mais funcionalidades, podendo assim ser utilizado como uma aplicação de Sistemas de Informação, já que possui uma funcionalidade CUBO que permite ao utilizador visualizar os dados de uma forma mais eficiente, na questão de tempo. É limitado a animes, porém pode ser expandido para outras especificações, como por exemplo, o género do anime, ou o estúdio que o produziu com melhor avaliação num determinado periodo de tempo.

(Inserir imagem da funcionalidade do ecrã de visualização de animes com melhor avaliação entre datas)

# 8 Implementação

## 8.1 Arquitetura do Sistema

Para o bom funcionamento do sistema, foi necessário criar uma arquitetura que permitisse a comunicação entre o Frontend e o Backend, e entre o Backend e a Base de Dados. A arquitetura do sistema foi feita com recurso ao padrão MVC, que permite separar o Frontend do Backend, e o Backend da Base de Dados. A aplicação comunica com a API e a Base de Dados e funciona como uma ponte entre os dois, permitindo ao utilizador interagir com o sistema. A arquitetura do sistema é a seguinte:

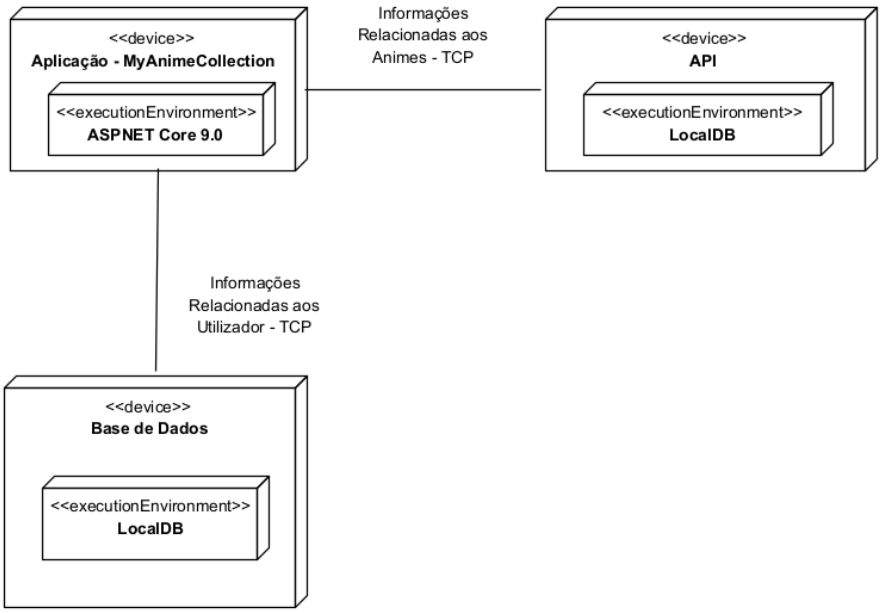



Figura 9: Arquitetura do Sistema

A aplicação representa o Frontend/Computador do utilizador do sistema, que se baseia no ASPNET Core 9.0, que é a aplicação que o utilizador interage. A aplicação comunica com a API, que possui a sua base de dados em LocalD e a Base de Dados do Sistema, possuiu outra base de dados em LocalDB.

## 8.2 Tecnologias Usadas

As tecnologias utilizadas para a implementação do sistema foram:

- ASP.NET Core 9.0 (Microsoft, 2025a)
- Entity Framework Core (Microsoft, 2025b)
- SQL Server (Microsoft, 2024b)
- Razor, HTML Helpers (Microsoft, 2024a)
- HTML, CSS, JavaScript (Mozilla, 2025)
- Swagger (Swagger, 2025)
- Bootstrap (Bootstrap, 2023)

Este sistema inicialmente foi desenvolvido em React, no âmbito da Unidade Curricular de Tecnologias Web e Desenvolvimento de Aplicações Móveis, porém, para implementar a API e a base de dados, o grupo decidiu mudar para ASP.NET Core 9.0, devido à facilidade de implementação da API e da base de dados, que foi feita em SQL Server, porém com uma metodologia de Code First. O Frontend foi desenvolvido em Razor, utilizando HTML, CSS, Bootstrap e HTML Helpers. O Backend foi desenvolvido em C#.

## 8.3 Desenvolvimento da API

A API foi desenvolvida em ASP.NET Core 9.0, utilizando Entity Framework Core para a comunicação com a base de dados em LocalDB. A API foi desenvolvida com o objetivo de fornecer os dados dos animes para o Frontend. A API foi alimentada com a API da Jinkan, que fornece dados de animes, como sinopse, número de episódios, número de temporadas, entre outros. Foi decido limitar os dados fornecidos pela API da Jinkan, para que a API do MyAnimeCollection fosse mais fácil de implementar e mais leve. A API foi desenvolvida com os seguintes endpoints:

- GET /api/animes - Retorna todos os animes
- GET /api/animes/id - Retorna um anime específico
- POST /api/animes - Adiciona um anime
- PUT /api/animes/id - Atualiza um anime
- DELETE /api/animes/id - Apaga um anime

## 8.4 Especificação da Interface

Para facilitar a compreensão do utilizador, a interface foi desenvolvida com o apoio do Swagger, que permite visualizar os endpoints da API e testá-los. A interface foi desenvolvida com o objetivo de ser simples e intuitiva, para que o utilizador possa facilmente perceber como utilizar a API. Ao utilizar o Swagger, o programador está a fazer alterações diretas na base de dados, da API. Como a base de dados é em LocalDB, as alterações são feitas diretamente na base de dados local, e não na base de dados da API, que pode um dia ser implementada num servidor.

## 8.5 Decisões de Implementação

Para criar a base de dados utilizada pela API, utilizamos a metodologia Code First, que permite criar a base de dados a partir do código. A base de dados foi criada depois com recurso ao Entity Framework Core, mais especificamente a parte das Migrations. Foi necessário eliminar a tabela dos géneros da base de dados, para permitir a implementação do sistema, porém numa futura melhoria, é possível tentar voltar a adicionar essa tabela, para que o sistema seja mais completo e tenha mais funcionalidades e torne-se mais útil para o utilizador. As rotas da API foram criadas de maneira a serem intuitivas e fáceis de perceber, para que o programador que venha a trabalhar na API no futuro possa facilmente perceber como a API funciona. Por exemplo, a rota */api/animes* permite ao utilizador ver todos os animes, enquanto a rota */api/animes/id* permite ao

utilizador ver um anime específico. Os controladores foram criados com base nas operações CRUD, e não foi necessário criar mais controladores, visto que o sistema é simples.

### 8.6 Principais Casos Relevantes de Programação

Sobre a implementação da API, apenas destacamos a implementação da API da Jinkan, que fornece os dados dos animes, como sinopse, número de episódios, número de temporadas, entre outros. A API da Jinkan foi implementada com recurso a um serviço, sendo que apenas insere dados na primeira vez que é chamada e a apenas se a tabela de animes estiver vazia. Isto foi feito para termos dados de teste de forma mais rápida e sem ter preocupações, ao modificar a API e a Base de Dados. As outras operações, permanecem funcionais.

## 9 Desenvolvimento da App Frontend/MVC

### 9.1 Decisões de Implementação

Para criar a base de dados utilizada pelo sistema, foi utilizada novamente a metodologia Code First. A base de dados foi criada depois com recurso ao Entity Framework Core, mais especificamente a parte das Migrations.
Foi necessário eliminar a tabela de AnimeList, para permitir a implementação do sistema, porém numa futura melhoria, é possível adicionar essa tabela numa futura atualização, para que o sistema seja mais completo e tenha mais funcionalidades e torne-se mais útil para o utilizador, como por exemplo, adicionar uma tabela de géneros, que permita ao utilizador pesquisar animes por género.
As rotas do sistema foram criadas de maneira a ser intuitivas e fáceis de perceber, para que o programador que venha a trabalhar no sistema no futuro possa facilmente perceber como o sistema funciona. Por exemplo, a rota */user/id* permite ao utilizador ver o perfil de um utilizador, enquanto a rota */user/id/id_lista*

permite ao utilizador ver a lista de um utilizador.
Os controladores foram criados com base nos casos de uso, para que o programador possa facilmente perceber como o sistema funciona. Por exemplo, o controlador *UserController* possui todos os métodos relacionados com o utilizador, como *Register*, *Login*, *Logout*, *Profile*, entre outros.
Na questão de segurança, foi implementado um sistema de autenticação, que permite ao utilizador registar-se e iniciar. O sistema de autenticação foi feito com recurso a Cookies, que permitem ao utilizador manter a sessão iniciada e reconhecer qual o utilizador que está a utilizar o sistema, esta funcionalidade foi essencial para a implementação do sistema, visto que o utilizador pode adicionar animes à sua lista, e avaliar listas de outros utilizadores.
Foi implementado um sistema de autorização, que permite ao utilizador aceder a certas páginas apenas se estiver autenticado. Por exemplo, o utilizador só pode aceder à página de perfil se estiver autenticado, caso contrário, é redirecionado para a página de login. Ainda foi implementado um sistema de autorização que permite apenas ao dono de uma lista gerir a mesma, ou seja, adicionar animes à lista ou remover animes da lista.

## 9.2 Principais Casos Relevantes de Programação

Antes de referir os principais casos de programação é necessário referir um acontecimento que, apesar de não ser um caso de programação, é essencial para o funcionamento do sistema. Este acontecimento é ao ligar a API, é necessário ligar a mesma duas vezes. A primeira vez é dá um erro, porém a segunda vez é bem sucedida. Os membros do grupo, acreditam que isto deve-se à base de dados, já foi possível arranjar uma solução para o caminho relativo dos ficheiros .mdf e .log da base de dados ligada à API.
Iniciando agora os principais casos de implementação, focando na parte de cada membro do grupo.

**Martinho José Novo Caeiro**

### Avaliar um Anime Especifíco

Este não é um caso de uso do sistema, porém é uma funcionalidade que foi implementada no sistema, para que seja possível fazer o caso de uso 5, partilhado entre os dois membros do grupo. Para avaliar um anime específico, foi necessário criar um método que permitisse essa funcionalidade ao utilizador. Este método foi criado no controlador *AnimeController*, que permite ao utilizador avaliar esse anime, com base em estrelas, que vão de 1 a 5. Essa informação é guardada na base de dados, e é possível visualizar a média da avaliação do anime na página do mesmo.

### Visualizar o perfil de Outro Utilizador

Neste caso de uso, foi necessário criar um método que permitisse ao utilizador visualizar o perfil de outro utilizador, sem que o utilizador autenticado tenha permissão de alterar qualquer coisa no perfil do utilizador que está a ser visualizado, isto foi feito com recurso a verificações nas views *Profile* e *UserList*, onde o utilizador apenas pode ver, se não pode o dono da lista ou do perfil. como é possível ver a seguir:

```
@if (userId == Model.UserId)
{
    <div>
        <button class="btn btn-danger btn-sm remove-anime-btn"
        data-anime-id="@anime.AnimeId">
            -
        </button>
    </div>
}
```

#### Avaliar uma Lista de Animes de Outro Utilizador

Para avaliar uma lista de animes de outro utilizador, foi necessário criar um método que permitisse essa funcionalidade ao utilizador. Este método foi criado no controlador *UserListController*, que permite ao utilizador avaliar a lista de outro utilizador, com base em estrelas, que vão de 1 a 5. Essa informação é guardada na base de dados, e é possível visualizar a média da avaliação da lista na página do perfil do utilizador e nos detalhes da lista.

```
var ratings = await _context.UserListAvaliations
    .Where(r => r.UserListId == id_lista)
    .ToListAsync();

var averageRating = ratings.Any() ? ratings.Average(r =>
r.Avaliation / 2.0) : 0;
ViewBag.AverageRating = averageRating;
```

### 9.3 Paulo António Tavares Abade

#### Consultar Animes

Para consultar animes, foi necessário criar um método que permitisse essa funcionalidade ao utilizador. Este método foi criado no controlador *AnimeController*, com o excerto de código visto a seguir, que permite ao utilizador pesquisar um anime pelo nome, e o sistema retorna as informações detalhadas do anime. Existe ainda a funcionalidade que permite ao utilizador ver todos os animes disponíveis na base de dados, sendo que esta fica na página principal do sistema, e é realizada pelo controlador *HomeController*.

```
public async Task<IActionResult> Details(int id)
{
    var anime = await _animeApiService.GetAnimeAsync(id);
    if (anime == null)
    {
        return NotFound();
    }

    // Calcular a média das avaliações
    var reviews = await _context.UserAnimeAvaliations.Where(r =>
    r.AnimeId == id).ToListAsync();
    var averageRating = reviews.Any() ? reviews.Average(r =>
    r.Avaliation / 2.0) : 0;
    ViewBag.AverageRating = averageRating;

    return View(anime);
}
```

### Adicionar Animes a uma Lista

Para adicionar animes a uma lista, foi necessário criar um método que permitisse essa funcionalidade ao utilizador. Este método foi criado no controlador *UserListController* que permite ao utilizador adicionar um anime às suas listas pessoais de animes para organizar seu progresso de visualização, ou apenas para separar os animes de acordo com o seu gosto, como por exemplo, separar os animes por aqueles mais nostáligicos, ou por aqueles que que são mais emotivos para aquele utilizador.
Esta foi a funcionalidade mais desafiadora, visto que surgiram vários problemas, como por exemplo, erros relacionados à inserção dos dados na base de dados, e erros relacionados à comunicação entre o Frontend e o Backend, que foram resolvidos com recurso a pesquisas na internet e a ajuda de colegas de curso.

Os erros associados à inserção de dados na base de dados, necessitou de eliminar a tabela intermediria que guardava os animes das listas dos utilizadores, e armazenar os ids dos animes diretamente na tabela de listas dos utilizadores, como um array, como tinha sido feito anteriormente no Projeto de TWAM, porém desta vez, cada lista, tinha o seu proprio array de animes, o que permitiu ao utilizador adicionar animes a uma lista especifica, sem que todas as listas tivessem que passar por um processo de atualização. Porém, não foi possível implementar a funcionalidade que permitiria trabalhar de forma mais eficiente com os dados, sendo essa a tabela intermediária. Esta funcionalidade será implementada numa futura melhoria do sistema. É possível ver isto na no modelo de lista a seguir

```
public class UserListModel {

    [Key]
    public int UserListId { get; set; }

    [ForeignKey("UserId")]
    public int UserId { get; set; }

    public UserModel? User { get; set; }

    [Required]
    [MaxLength(100)]
    public string Name { get; set; }

    [MaxLength(500)]
    public string Description { get; set; }

    // Lista de IDs de Animes
    public List<int> AnimeIds { get; set; } = new List<int>();
}
```

Os erros associados à comunição incorreta entre o Frontend e o Backend, que necessitaram de utilizar alguns métodos novos para nós, para solucionar os problemas, como a criação de uma classe interna no controlador que permitisse obter corretamente, o id do utilizador e o id do anime, para que o utilizador conseguisse remover com sucesso o anime da lista.

```
[HttpPost]
public async Task<IActionResult> RemoveAnimeFromList(
    [FromBody] RemoveAnimeRequest request)
{
    Console.WriteLine($"ID da lista recebido: {request.ListId}");
    Console.WriteLine($"ID do anime recebido: {request.AnimeId}");

    var userList = await _context.UserLists.FindAsync(request.ListId);

    (...)
}

public class RemoveAnimeRequest
{
    public int AnimeId { get; set; }
    public int ListId { get; set; }
}
```

# 10 Conclusão e Trabalho Futuro

Nesta etapa do projeto, obtivemos resultados que consideramos bastante satisfatórios e acreditamos que este projeto apresenta um grande potencial para o futuro. A utilização do ASP.NET, em vez do Laravel como inicialmente planeado, proporcionou-nos maior facilidade na implementação da API, uma vez que o ASP.NET nos permitiu criar uma API e um projeto MVC num único projeto, simplificando assim a comunicação entre a API e o Frontend. Estamos confiantes de que a aplicação tem um maior nivel de fidelidade em relação à versão feita no semestre anterior na Unidade Curricular de Tecnologias Web e Desenvolvimento de Aplicações Móveis, onde encontramos dificuldades significativas devido às limitações da RestDB, que possuía um limite que prejudicava imenso devido à nossa extrema necessidade para garantir ao utilizador algumas funcionalidades que tornavam a páginas mais intuitiva, como por exemplo, mostrar se o anime já estava adicionado a alguma lista.

# Bibliografia


Bootstrap. (2023). *Bootstrap*. Obtido setembro 13, 2023, de https://getbootstrap.com/

Bruno, L. (2025). *Página da Unidade Curricular de Desenvolvimento de Aplicações Web*. Obtido janeiro 17, 2025, de https://cms.ipbeja.pt/course/view.php?id=1451

Microsoft. (2024a). *Razor*. Obtido setembro 27, 2024, de https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-9.0

Microsoft. (2024b). *SQL Server*. Obtido julho 22, 2024, de https://learn.microsoft.com/en-us/sql/database-engine/configure-windows/sql-server-express-localdb?view=sql-server-ver16

Microsoft. (2025a). *ASP.NET*. Obtido janeiro 17, 2025, de https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-9.0

Microsoft. (2025b). *Entity Framework Core*. Obtido janeiro 17, 2025, de https://learn.microsoft.com/en-us/ef/

Mozilla. (2025). *JavaScript*. https://developer.mozilla.org/en-US/docs/Web/JavaScript

Swagger. (2025). *Swagger*. Obtido janeiro 17, 2025, de https://swagger.io/docs/