deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

GameVault

Trabalho realizado por:
Diogo Alexandre Rocha Domingues, nº114192

Marta Pereira Condeço, nº 112903

P8-G2

Bases de Dados

Prof. Joaquim Sousa Pinto

Ano Letivo 2023/2024

Índice

Introdução

A escolha do nosso projeto foi motivada pelo desenvolvimento do front end utilizado numa outra disciplina. Este sistema de informação tem como propósito permitir uma fácil pesquisa de informação relativamente a jogos. Permitir num só local apresentar os jogos existentes, as respetivas DLC's e reviews, apresentar também as plataformas de jogos existentes, lojas para os jogos em questão e preços associados. Achamos interessante também apresentar empresas relacionadas como editoras e desenvolvedoras, respetivas informações e jogos associados.

Para o desenvolvimento do projeto procuramos informações reais de jogos e empresas apesar das ligações entre eles terem sido completamente aleatórias.

Estrutura da Pasta do Projeto

Dentro do repositório BD-Projeto e do ficheiro .zip é possível encontrar 4 pastas "Análise de Requisitos + Diagramas" onde podemos encontrar a análise de requisitos feita no início do projeto bem como o diagrama DER e o esquema relacional. Já na pasta "FE" encontramos todo o código necessário para correr a interface. Na pasta SQL temos os vários ficheiros .sql divididos de acordo com o tipo de queries (DDL.sql, INSERTS.sql, sp.sql, trg.sql, udf.sql, views.sql). Já na pasta "Apresentação" temos a apresentação que foi feita no dia 31/05.

- Pasta SQL:
  - DDL.sql -> Script SQL que contém a definição de todas as tabelas necessárias ao correto funcionamento da base de dados, assim como as definições de chaves primárias e estrangeiras e algumas verificações para os atributos das tabelas;
  - INSERTS.sql -> Script SQL que contém os dados a ser inserdos em cada tabela;
  - sp.sql -> Script SQL que contém todos os *Stored Procedures;*
  - trg.sql -> Script SQL que contém todos os Triggers*;*
  - udf.sql -> Script SQL que contém todas as *User Defined Functions;*
  - views.sql -> Script SQL que contém todas as *Views;*

Requisitos Funcionais

No desenvolvimento deste projeto consideramos necessários os requisitos funcionais:

- Procurar informações de cada empresa, plataforma e loja de jogos
- Procurar informações dos jogos e respetivas reviews
- Permitir efetuar reviews, posts e responder aos posts e reagir às mesmas (upvotes e downvotes)
- Existência de contas para que o utilizador possa depois ver mais facilmente quais foram os posts, reviews e respostas que inseriu e reagiu para poder alterar mais tarde se assim quiser

Entidades

Utilizador – Representa um utilizador que tem conta no sistema. Possui um ID, um nome, um email, uma data de adesão, uma password, uma imagem, um Token e um TokenExpiration. Este tem vários reviews, vários posts, várias respostas ao post da comunidade, vários review votes, vários post votes e vários resposta votes.

Review – Representa uma classificação dada a um jogo. Possui um ID, um comentário, uma hora, um rating, uma data, número de upvotes e número de downvotes. Uma review tem vários review votes.

Jogo – Representa um jogo. Tem um ID, um nome, uma descrição e uma data de lançamento. Um jogo pode estar à venda em várias lojas, pode ter vários géneros, pode ter vários DLC's, pode ser jogado em várias plataformas, pode ser desenvolvido por várias desenvolvedoras e publicado por várias editoras.

Género - Representa um dos vários géneros de jogos que existem. Tem um ID e um nome. Um género pode estar associado a vários jogos.

DLC – Representa uma expansão de um jogo. Tem um ID, um nome um tipo e uma data de lançamento. Uma DLC pertence a um só jogo.

Empresa – Representa uma entidade que tem como atributos um ID, um nome, uma localização e uma data de criação.

Plataforma – Representa uma plataforma onde um jogo pode ser jogado. Tem um ID e um nome. Uma plataforma disponibiliza vários jogos.

Desenvolvedora - É uma empresa que desenvolve jogos. Uma empresa pode desenvolver vários jogos

Editora - É uma empresa que publica jogos. Ume editora pode publicar vários jogos.

Loja – Representa uma entidade que tem um jogo à venda. Tem um ID e um nome. Uma loja pode vender vários jogos.

Post da Comunidade – Representa um post da comunidade que pode ser uma questão, por exemplo. Tem um ID, um título e texto. Um post pode ter várias respostas e vários votos.

Resposta ao Post da Comunidade – Representa uma resposta a um certo post da comunidade. É composto por ID e texto. Uma resposta ao post da comunidade pode ter vários votos.

PostVotos – Representa os votos num dado post. Tem um ID e um voto.

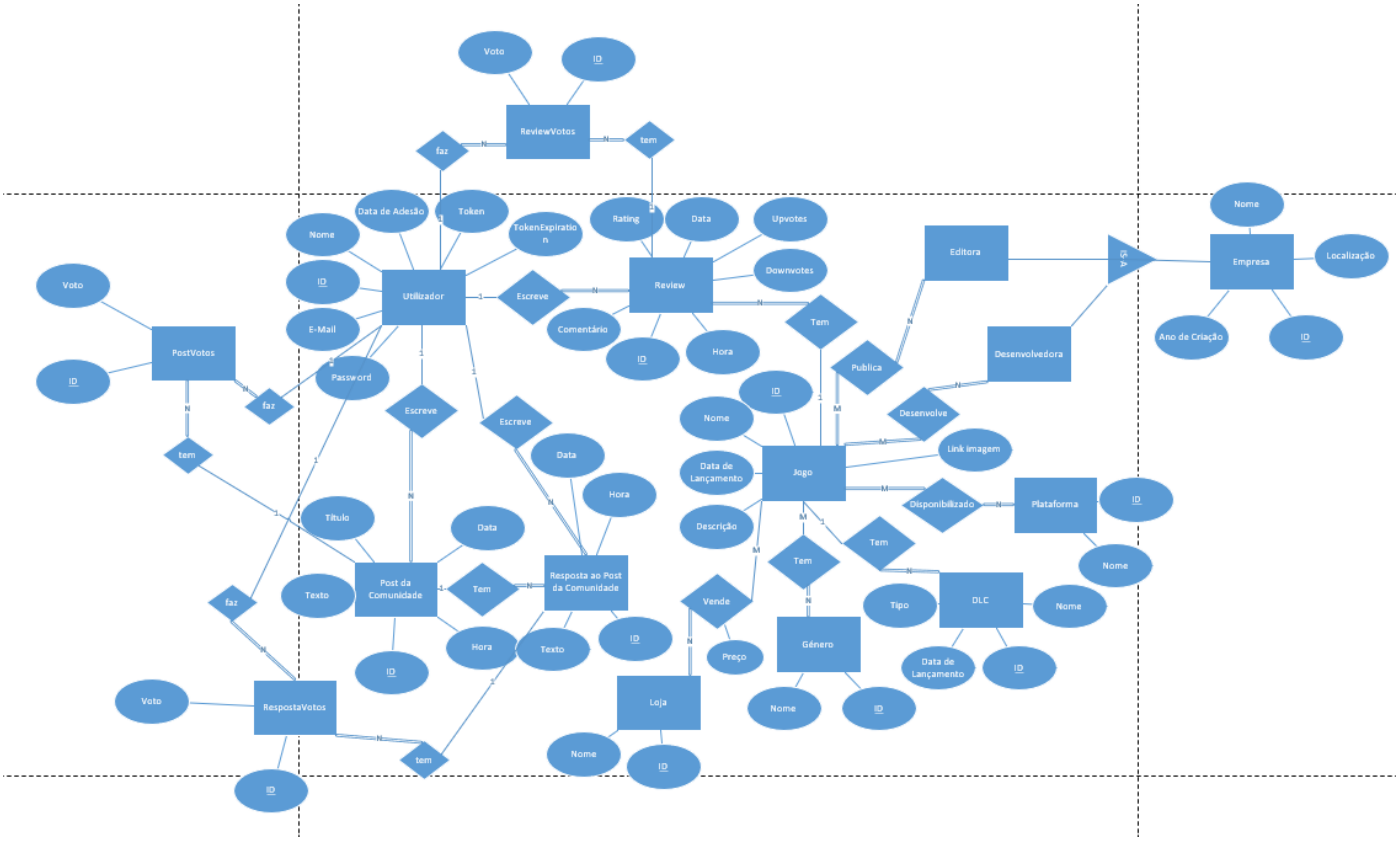
RespostaVotos – Representa os votos numa dado resposta de um post. Tem um ID e um voto.

ReviewVotos – Representa os votos numa dada review. É composto por um ID e por votos.

Diagramas

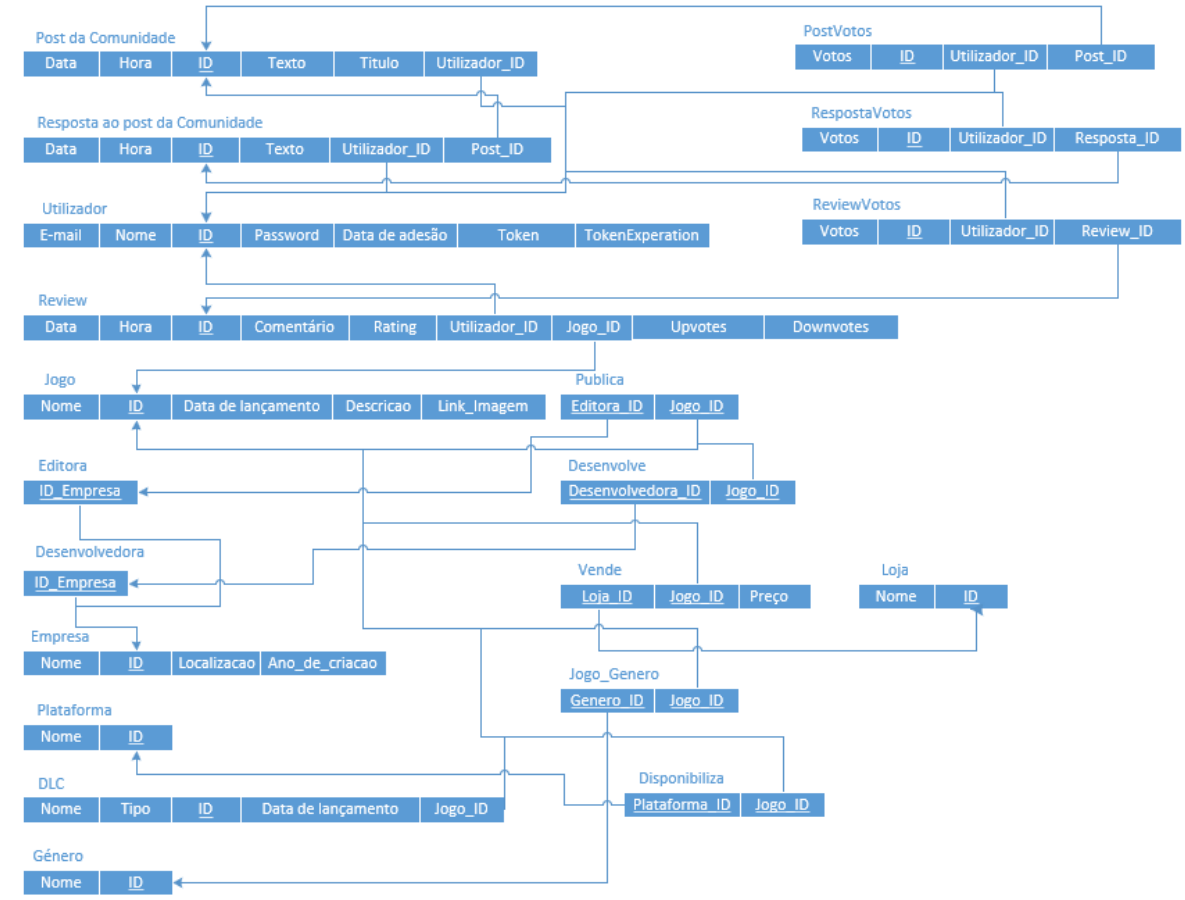
Diagrama Entidade-Relacionamento

Abaixo está representado o DER (diagrama entidade-relacionamento) correspondente às entidades referidas acima e relações entre as mesmas



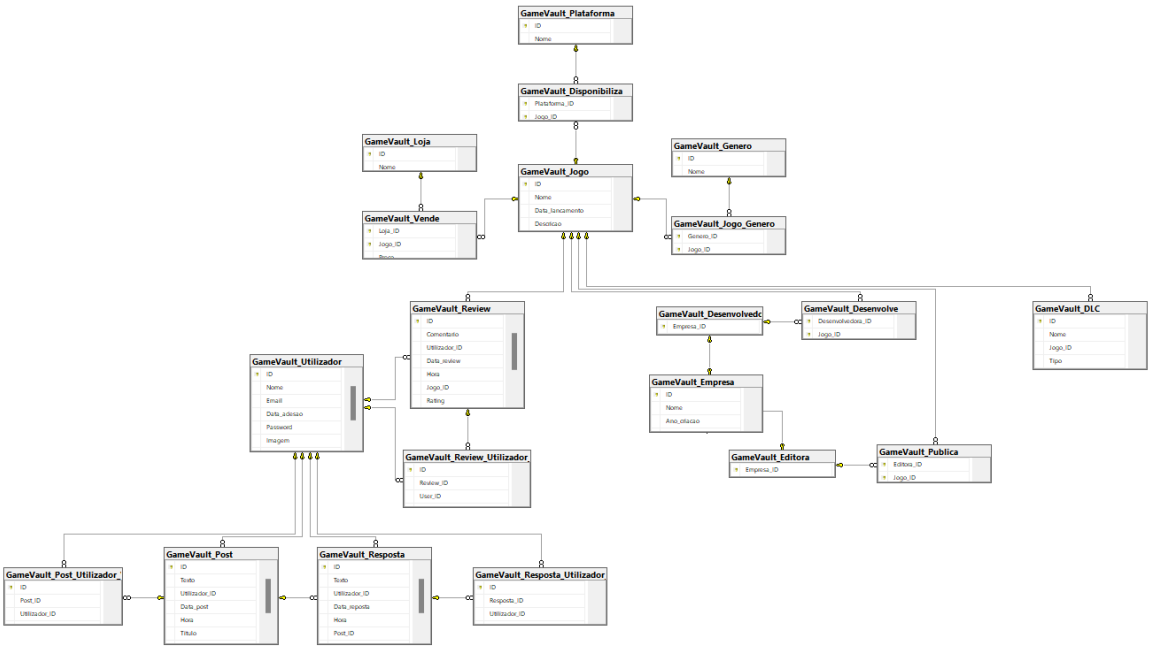
Esquema Relacional

Abaixo está ilustrado o esquema relacional desenvolvido de acordo com o DER.



Esquema Relacional (SGBD)

Abaixo está representado o esquema relacional feito pelo SGBD usado (SQL Server Management Studio)



Queries SQL

Data Defintion Language (DDL)

A DDL é uma parte importante da linguagem SQL, permite definir a estrutura dos dados na ser guadados na base de dados ao permitir definir as entidades, o esquema de cada relação, o domínio de valores associados com cada atributo e respetivas restrições, o conjunto de índices a manter para cada relação, as respetivas primary keys e foreign key, etc.

Não vamos apresentar aqui o nosso código DDL pela sua extensão, mas está presente na pasta envia no ficheiro DDL.sql

Data Manipulation Language (DML)

A DML é uma parte essencial da linguagem SQL pelo seu papel na inserção, eliminação e alteração de dados bem como na pesquisa simples e avançada de dados.

O código de inserção de dados iniciais encontra-se no ficheiro INSERTS.sql da pasta entregue.

Views

As views são relações vituais derivadas das relações presentes já na base de dados, não têm armazenamento físico dos dados. Podem ser usadas para facilitar a apresentação dos dados adaptadando o esquema da base de dados a diferentes aplicações.

No nosso trabalho utilizamos as views para mais facilmente apresentarmos os dados na interface, foram também usadas em UDF's (User Defined Functions) e SP's (Stored Procedures) para efetuar ordenações e filtrações dos dados.

Não será aqui apresentado o código todo mas deixamos um exemplo que é empresaInfo que relaciona 7 tabelas iniciais. Esta view é utilizada para apresentar dados diretamente na interface, para efetuar pesquisas e filtrações.

```sql
--view empresas + plataforma + lojas e numero de jogos associados + info
CREATE VIEW empresaInfo AS
    SELECT emp.Nome, emp.ID, emp.Ano_criacao, emp.Localizacao, COUNT(j.ID) AS numJogos, 'D' AS [type]
    FROM GameVault_Empresa AS emp
    JOIN GameVault_Desenvolve AS p
    ON emp.ID=p.Desenvolvedora_ID
    JOIN GameVault_Jogo AS j
    On p.Jogo_ID=j.ID
    GROUP BY emp.Nome,emp.ID, emp.Ano_criacao, emp.Localizacao
    UNION
    SELECT emp.Nome, emp.ID, emp.Ano_criacao, emp.Localizacao, COUNT(j.ID) AS numJogos, 'E' AS [type]
    FROM GameVault_Empresa AS emp
    JOIN GameVault_Publica AS p
    ON emp.ID=p.Editora_ID
    JOIN GameVault_Jogo AS j
    On p.Jogo_ID=j.ID
    GROUP BY emp.Nome,emp.ID, emp.Ano_criacao, emp.Localizacao
    UNION
    SELECT p.Nome, p.ID,NULL AS Ano_criacao, NULL AS Localizacao, COUNT(j.ID) AS numJogos, 'P' AS [type]
    FROM GameVault_Disponibiliza AS d
    JOIN GameVault_Plataforma AS p
    ON p.ID=d.Plataforma_ID
    JOIN GameVault_Jogo AS j
    On d.Jogo_ID=j.ID
    GROUP BY p.Nome,p.ID
    UNION
    SELECT l.Nome, l.ID,NULL AS Ano_criacao, NULL AS Localizacao, COUNT(j.ID) AS numJogos, 'L' AS [type]
    FROM GameVault_Loja AS l
    JOIN GameVault_Vende AS v
    ON l.ID=v.Loja_ID
    JOIN GameVault_Jogo AS j
    ON v.Jogo_ID=j.ID
    GROUP BY l.Nome,l.ID
GO
```

Indexes

Os indexes são estruturas de dados que oferecem uma forma de cesso aos dados mais rápida, aumentado o desempenho e a eficiência na busca dos dados. É possível indexar qualquer atributo da relação, criar múltiplos indexes e criar indexes com vários atributos.

Para o nosso trabalho, além dos indexes já existentes criados automaticamente nas primary keys, criámos indexes nos atributos que eram usados para efetuar pesquisas.

Este código está presente no ficheiro DDL.sql

```
CREATE INDEX Post_Titulo
ON GameVault_Post (Titulo);
GO

CREATE INDEX Empresa_Nome
ON GameVault_Empresa (Nome);
GO

CREATE INDEX Jogo_Nome
ON GameVault_Jogo (Nome);
GO
```

User Defined Functions (UDF's)

As UDF's permitem criar funções personalizadas adaptadas às necessidades e funcionalidades requeridas. Oferecem uma série de vantagens e utilidades que ajudam a simplificar consultas de dados, podem incorporar lógica complexa, podem ser usadas como finte de dados, etc. Neste projeto as UDF´s foram utilizadas para efetuar verificações de inputs do utilizador, efetuar filtrações e seleções mais especificas e complexas de dados.

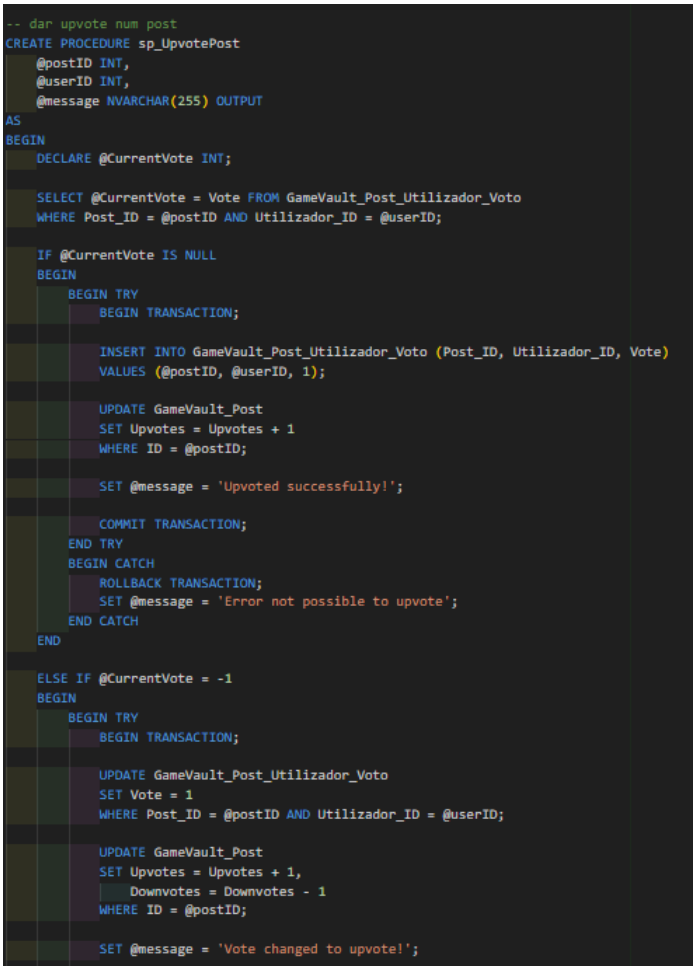
Não iremos apresentar aqui todas as UDF´s utilizadas no projeto mas apresentamos um exemplo que é uma UDF utilizada para selecionar as reviews de um jogo específico selecionado pelo utilizador na interface. O código relativo às restantes UDF´s podem encontra-se no ficheiro udf.sql

```sql
--escolher a reviews de um determinado jogo
CREATE FUNCTION udf_Reviews (@gameid INT)
RETURNS @table TABLE (
    ID INT,
    Utilizador_ID INT,
    Username VARCHAR(120),
    Comentario VARCHAR(MAX),
    Rating DECIMAL(2,1),
    Data_review DATE,
    Hora TIME,
    Upvotes INT,
    Downvotes INT
)
AS
BEGIN
    INSERT INTO @table
    SELECT
        r.ID,
        r.Utilizador_ID,
        u.Nome AS Username,
        r.Comentario,
        r.Rating,
        r.Data_review,
        r.Hora,
        r.Upvotes,
        r.Downvotes
    FROM
        GameVault_Review r
    JOIN
        GameVault_Utilizador u ON r.Utilizador_ID = u.ID
    WHERE
        r.Jogo_ID = @gameid;
    RETURN;
END;
GO
```

Stored Procedures (SP's)

Os SP's são essencialmente batches armazenados com um nome. Entre muitas vantagens a que mais se salienta é que o SQL Server não tem de recompilar o código cada vez que um procedimento é invocado, este é guardado na memória cache na primeira vez que são executados permitindo uma execução mais rápida. Neste trabalho foram utilizados para efetuar autenticações, validações, inserções, alterações e remoções de dados, ordenações, pesquisas e seleções.

No exemplo a seguir é apresentada um SP que é utilizada para adicionar upvotes a um post, tendo parâmetros de entrada, de saída e transactions. Dada a extensão do código não iremos colocar aqui todas as stored procedures criadas no projeto mas estas podem ser encontradas no ficheiro sp.sql



```
-- dar upvote num post
CREATE PROCEDURE sp_UpvotePost
    @postID INT,
    @userID INT,
    @message NVARCHAR(255) OUTPUT
AS
BEGIN
    DECLARE @CurrentVote INT;

    SELECT @CurrentVote = Vote FROM GameVault_Post_Utilizador_Voto
    WHERE Post_ID = @postID AND Utilizador_ID = @userID;

    IF @CurrentVote IS NULL
    BEGIN
        BEGIN TRY
            BEGIN TRANSACTION;

            INSERT INTO GameVault_Post_Utilizador_Voto (Post_ID, Utilizador_ID, Vote)
            VALUES (@postID, @userID, 1);

            UPDATE GameVault_Post
            SET Upvotes = Upvotes + 1
            WHERE ID = @postID;

            SET @message = 'Upvoted successfully!';

            COMMIT TRANSACTION;
        END TRY
        BEGIN CATCH
            ROLLBACK TRANSACTION;
            SET @message = 'Error not possible to upvote';
        END CATCH
    END

    ELSE IF @CurrentVote = -1
    BEGIN
        BEGIN TRY
            BEGIN TRANSACTION;

            UPDATE GameVault_Post_Utilizador_Voto
            SET Vote = 1
            WHERE Post_ID = @postID AND Utilizador_ID = @userID;

            UPDATE GameVault_Post
            SET Upvotes = Upvotes + 1,
                Downvotes = Downvotes - 1
            WHERE ID = @postID;

            SET @message = 'Vote changed to upvote!';
```

```
            COMMIT TRANSACTION;
        END TRY
        BEGIN CATCH
            ROLLBACK TRANSACTION;
            SET @message = 'Error not possible to change to upvote';
        END CATCH
    END

    ELSE
    BEGIN
        BEGIN TRY
            BEGIN TRANSACTION;

            DELETE FROM GameVault_Post_Utilizador_Voto
            WHERE Post_ID = @postID AND Utilizador_ID = @userID;

            UPDATE GameVault_Post
            SET Upvotes = Upvotes - 1
            WHERE ID = @postID;

            SET @message = 'Upvote removed!';

            COMMIT TRANSACTION;
        END TRY
        BEGIN CATCH
            ROLLBACK TRANSACTION;
            SET @message = 'Error not possible to remove upvote';
        END CATCH
    END
END
GO
```

Triggers

Os triggers são casos específicos de stored procedures que são executados automaticamente em determinadas circunstâncias como resposta à manipulação de dados. No presente projeto foi apenas criado um trigger relacionado com a eliminação de utilizadores para garantir a existência de conteúdo e a preservação deste, mesmo que não existam utilizadores.

```
--trigger para quando apagar um user e os seus posts nao serem apagados, passam para um user default
CREATE TRIGGER trg_DeleteUser
ON GameVault_Utilizador
INSTEAD OF DELETE
AS
BEGIN
    BEGIN TRANSACTION
        BEGIN TRY
            IF (SELECT ID FROM Deleted) IN (SELECT Utilizador_ID FROM GameVault_Resposta)
                UPDATE GameVault_Resposta
                SET Utilizador_ID = 1 WHERE Utilizador_ID = (SELECT ID FROM Deleted)

            IF (SELECT ID FROM Deleted) IN (SELECT Utilizador_ID FROM GameVault_Post)
                UPDATE GameVault_Post
                SET Utilizador_ID = 1 WHERE Utilizador_ID = (SELECT ID FROM Deleted)

            IF (SELECT ID FROM Deleted) IN (SELECT Utilizador_ID FROM GameVault_Review)
                UPDATE GameVault_Review
                SET Utilizador_ID = 1 WHERE Utilizador_ID = (SELECT ID FROM Deleted)

            DELETE FROM GameVault_Utilizador WHERE ID=(SELECT ID FROM Deleted)
            PRINT 'User deleted sucessfully';
        COMMIT TRANSACTION
    END TRY
    BEGIN CATCH
        PRINT 'Couldnt delete user';
        ROLLBACK TRANSACTION;
        RETURN;
    END CATCH
END;
GO
```

Interface

Para a interface de interação com a base de dados decidimos usar um projeto desenvolvido na cadeira de Interação Humano-Computador, que era um site de review de jogos, por isso encaixava-se perfeitamente para a nossa base de dados.

Na interface podemos ter um utilizador que tem uma conta ou que não tem uma conta. Os utilizadores com conta podem escrever reviews, dar feedback de outras reviews, escrever e responder a posts. Já os utilizadores sem conta apenas podem ver os jogos e reviews associadas, mas sem poder interagir com estas bem como ver os posts e respostas, mas sem poder interagir com estes. Poderia, também, ter sido feito a parte de um gestor que poderia apagar jogos, empresas e posts.

A interface possui 6 páginas terminadas a Home Page, a Search Page, a Game Page, a Companies Page, a Company Details Page e a Community Page.

Na Home Page podemos ver os 6 jogos com melhor classificação ordenados do maior número de reviews para o menor.

Na Search Page podemos observar 9 jogos selecionados aleatoriamente se nenhuma pesquisa ou filtragem for feita, ou podemos efetuar uma pesquisa por nome, filtrar os jogos com os critérios que pretendemos e ainda ordená-los de diferentes formas, de notar que estas funcionalidades podem ser todas aplicadas ao mesmo tempo.

Na Game Page, podemos observar os detalhes de um jogo selecionado bem como possíveis DLC's disponíveis e ver reviews de outros utilizadores com as quais podemos interagir para dizer se concordamos ou não com elas, e postar uma review.

Na Companies Page podemos pesquisar uma empresa pelo seu nome ou por um jogo que esteja associado a esta e filtrá-las por desenvolvedora, editora, vendedora ou uma plataforma.

Na Company Details Page podemos observar os dados de cada empresa e os jogos para os quais contribuiu e se quisermos visitar um desses jogos podemos clicar que seremos encaminhados para a página correta.

Na Comunity Page podemos ver posts de outros jogadores com informações diversas, podemos pesquisar os posts e ordená-los pelo número de respostas e daya.

É de notar que se uma ação efetuar uma alteração na base de dados essa a página onde foi feita essa ação é reiniciada para mostrar os resultados atualizados.

Ligação à base de dados - Relatório Complementar

Para efetuar a ligação entre a interface e a base de dados usamos um servidor local com a ajuda de diversos frameworks. Utilizamos o framework *Express* para lidar com as requisições HTTP juntamente com a biblioteca *cors* para permitir requisições de diferentes origens e *express.json* para converte os dados obtidos num json.

Para nos ligarmos à base de dados usamos a função *connectDB()* que nos irá fornecer uma pool de conexões que será usada para executar as consultas e procedimentos desejados e serão devolvidos os dados requisitados no formato *json*.

```
const pool = await connectDB();
if (!pool) {
    console.error('Failed to connect to the database. Exiting...');
    process.exit(1);
}
```

```
const connectDB = async () => {
    console.log('Connecting to database with the following config:', dbConfig);
    try {
        const pool = await sql.connect(dbConfig);
        console.log('Database connected successfully');
        return pool;
    } catch (err) {
        console.error('Database connection failed:', err);
    }
};
```

Depois definimos várias rotas para interagir com a base de dados usando o *express.Router*. As rotas são do tipo *get* se quisermos obter informação da base de dados tais como obter os jogos com melhor classificação, para obter nove jogos aleatórios ou para obter detalhes de um jogo já as rotas do tipo *post* permitem-nos modificar ou adicionar informação à base de dados como as rotas para registar um novo utilizador, para autenticar um utilizador existente, para adicionar uma review ou dar feedback numa review.

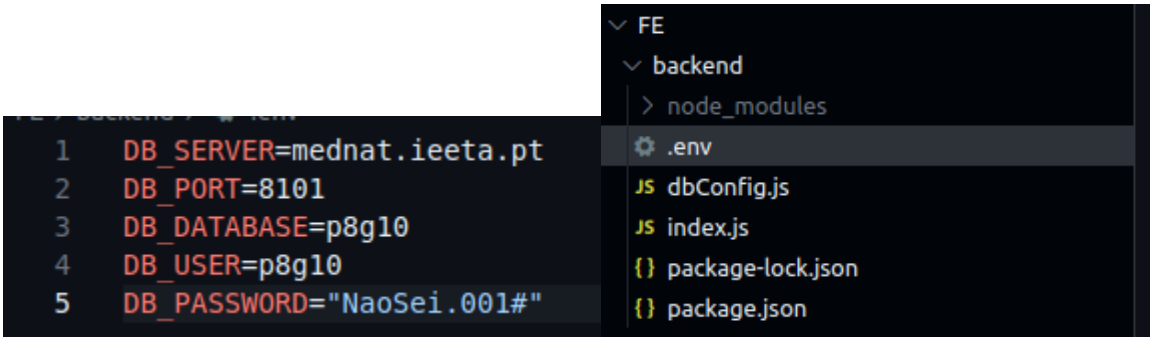
```
app.get('/api/topRatedGames', async (req, res) => {
    const { numberOfGames } = req.query;
    if (!numberOfGames || isNaN(numberOfGames)) {
        return res.status(400).send('NumberOfGames parameter is required and must be a number');
    }

    try {
        const result = await pool.request()
            .input('NumberOfGames', sql.Int, numberOfGames)
            .execute('sp_TopRatedGames');
        res.json(result.recordset);
    } catch (err) {
        console.error('Error fetching top rated games:', err);
        res.status(500).send('Server Error');
    }
});
```

```
app.post('/api/authenticate', async (req, res) => {
    const { email, password, staySignedIn } = req.body;
    try {
        const result = await pool.request()
            .input('Email', sql.VarChar, email)
            .input('Password', sql.VarChar, password)
            .input('StaySignedIn', sql.Bit, staySignedIn)
            .output('Token', sql.VarChar(255))
            .output('TokenExpiration', sql.DateTime)
            .output('ErrorMessage', sql.NVarChar(255))
            .output('Username', sql.VarChar(120))
            .output('ID', sql.Int)
            .execute('sp_AutenticarUtilizador');

        const errorMessage = result.output.ErrorMessage;
        const token = result.output.Token;
        const username = result.output.Username;
        const tokenExpiration = result.output.TokenExpiration;
        const id = result.output.ID;

        if (token) {
            res.json({ success: true, token, tokenExpiration, username, id });
        } else {
            res.json({ success: false, message: errorMessage });
        }
    } catch (err) {
        console.error('Error authenticating user:', err);
        res.status(500).send('Server Error');
    }
});
```

Depois no frontend as requisições são feitas para o *endpoint* desejado da API utilizando o *axios*.

```
const fetchGameDetails = async () => {
  try {
    const response = await axios.get(`http://localhost:5000/api/game/${id}`);
    if (response.data) {
      console.log('Fetched game data:', response.data); // Debugging step
      setGame(response.data);
    } else {
      console.error('Game data is undefined:', response.data);
    }
  } catch (error) {
    console.error('Error fetching game details:', error);
  }
};
```

Para alterar o utilizador que está a aceder à base de dados vamos ao ficheiro *.env* localizado em "FE/backend" e mudamos os dados.



Para iniciar o servidor bem como a interface, primeiramente vamos à pasta "FE/backend", abrimos um terminal e escrevemos *"npm install" e* esperamos a instalção das dependências. De seguida na pasta "FE" abrimos um terminal e escrevemos *"npm install"* após a instalação das dependências voltamos à pasta "backend" e escrevemos num terminal "node index.js", isto iniciará o servidor local que estabelecerá a ligação à base de dados, devemos manter este terminal aberto. Agora voltamos à pasta "FE" e escrevemos "*npm start", aguardamos* e em princípio abrirá uma página no browser com a Home Page da interface se isso não acontecer podemos aceder a "localhost:3000".

Trabalho Futuro

Para complementar o trabalho já realizado poderão ser acrescentadas as seguintes funcionalidades e medidas de segurança:

- Corrigir search bar da narvbar e permitir que esta efetue uma pesquisa geral
- Impedir SQL Injection
- Apenas aceitar passwords fortes no momento de criação de conta
- Implementar uma atualização de data quando reviews, posts e respostas são editados
- Criar uma página de perfil do utilizador onde possa ter uma foto, informações pertinentes, estatísticas do perfil e lista de jogos guardados
- Criar um administrador que possa adicionar e remover jogos, empresas, plataformas e lojas do site

Conclusão

O desenvolvimento deste projeto permitiu uma melhor compreensão da importância e utilidade de um Sistema de Gestão de Base de Dados (SGBD) e a aplicação e utilização da linguagem SQL (Structured Query Language) permitiu uma melhor aprendizagem e familiarização com a mesma.

Os objetivos iniciais do projeto foram atingidos e inclusivamente superados em alguns aspetos apesar de ainda existir muito espaço para melhorias e funcionalidades a ser exploradas e acrescentadas.

Com isto o projeto além de interessante foi gratificante de desenvolver e permitiu uma boa consolidação das aprendizagens da cadeira de Base de Dados.