Base de Dados - Projeto - GameShelf

- Grupo P2G3
- André Alexandre Nº114143 LECI
- Tiago Melo Nº113362 LECI

universidade de aveiro

Conceito e tema

- GameShelf é um site de avaliações para utilizadores catalogarem videojogos e partilharem críticas (semelhante ao LetterBoxd, mas para videojogos).

- Escolhemos esta ideia pois ambos os elementos do grupo gostam de jogar videojogos e tal como a indústria cinematográfica, a indústria dos videojogos tem se tornado cada vez mais numa experiência que merece ser classificada por cada jogador.

Funcionalidades

1. Fazer reviews de jogos
2. Comentar outras reviews
3. Criar conta e adicionar amigos
4. Criar Listas
5. Consultar as estatísticas de cada perfil
6. Filtros de pesquisa

Diagrama Entidade-Relacionamento

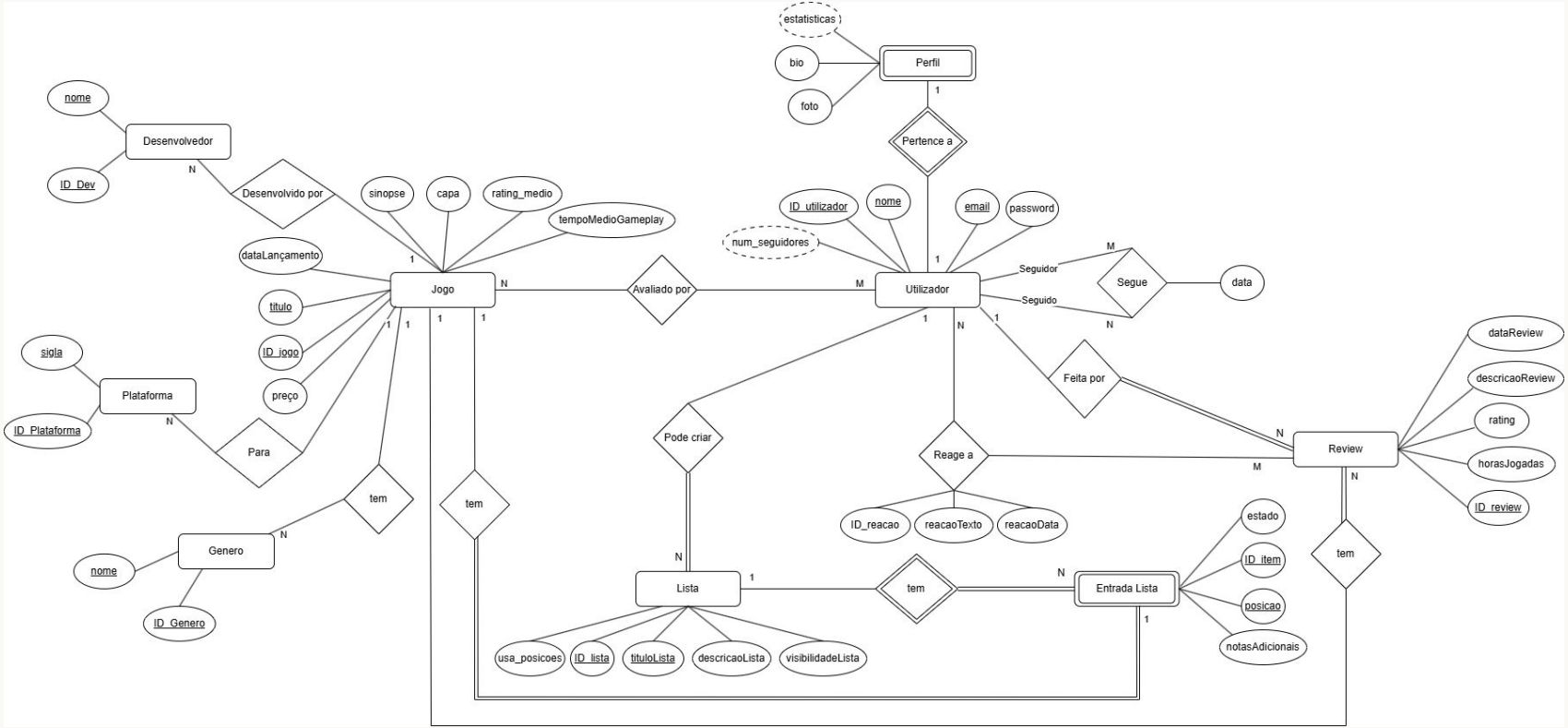
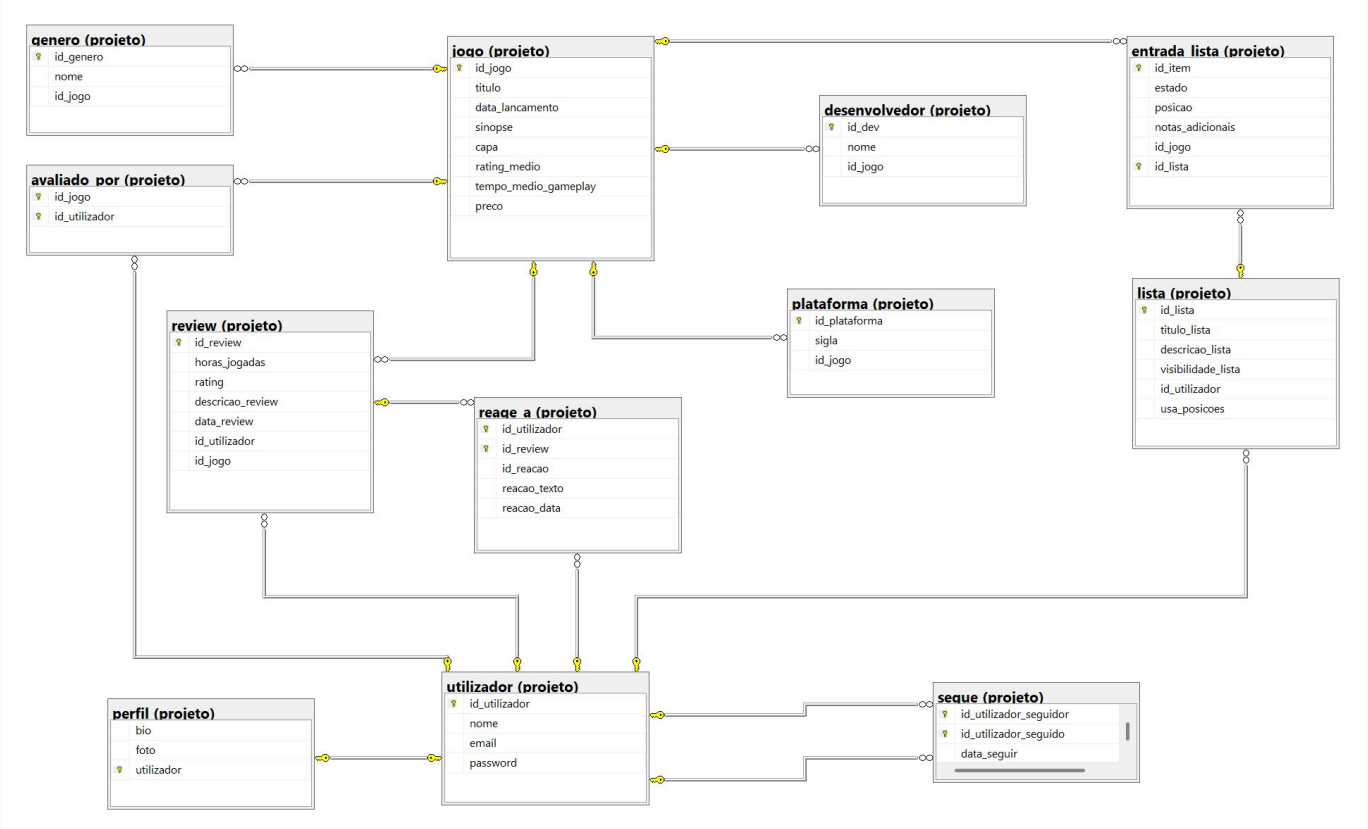


Diagrama Relacional



Stored Procedures

```sql
CREATE PROCEDURE projeto.sp_SearchLists
    @currentUserId VARCHAR(20),
    @searchText NVARCHAR(100) = NULL,
    @filterType NVARCHAR(20) = 'All' -- All, Friends, Mine, MadeByMods
AS
BEGIN
    -- Friends filter: Only public lists from users I follow
    IF @filterType = 'Friends'
    BEGIN
        SELECT DISTINCT l.id_lista, l.titulo_lista, u.nome AS criador
        FROM projeto.lista l
        JOIN projeto.utilizador u ON l.id_utilizador = u.id_utilizador
        JOIN projeto.segue s ON u.id_utilizador = s.id_utilizador_seguido
        WHERE s.id_utilizador_seguidor = @currentUserId
        AND l.visibilidade_lista = 'Publica'
        AND (@searchText IS NULL OR l.titulo_lista LIKE '%' + @searchText + '%')
        ORDER BY l.titulo_lista;
    END
    -- MadeByMods filter: All lists from admin
    ELSE IF @filterType = 'MadeByMods'
    BEGIN
        SELECT l.id_lista, l.titulo_lista, u.nome AS criador
        FROM projeto.lista l
        JOIN projeto.utilizador u ON l.id_utilizador = u.id_utilizador
        WHERE u.nome = 'adminmod'
        AND l.visibilidade_lista = 'Publica'
        AND (@searchText IS NULL OR l.titulo_lista LIKE '%' + @searchText + '%')
        ORDER BY l.titulo_lista;
    END
    -- Mine filter: Only my lists (both public and private)
    ELSE IF @filterType = 'Mine'
    BEGIN
        SELECT l.id_lista, l.titulo_lista, u.nome AS criador
        FROM projeto.lista l
        JOIN projeto.utilizador u ON l.id_utilizador = u.id_utilizador
        WHERE u.id_utilizador = @currentUserId
        AND (@searchText IS NULL OR l.titulo_lista LIKE '%' + @searchText + '%')
        ORDER BY l.titulo_lista;
    END
    -- Default All filter: All public lists + my private lists
    ELSE -- All
    BEGIN
        -- Public lists from all users
        SELECT l.id_lista, l.titulo_lista, u.nome AS criador
        FROM projeto.lista l
        JOIN projeto.utilizador u ON l.id_utilizador = u.id_utilizador
        WHERE l.visibilidade_lista = 'Publica'
        AND (@searchText IS NULL OR l.titulo_lista LIKE '%' + @searchText + '%')

        UNION

        -- My private lists
        SELECT l.id_lista, l.titulo_lista, u.nome AS criador
        FROM projeto.lista l
        JOIN projeto.utilizador u ON l.id_utilizador = u.id_utilizador
        WHERE u.id_utilizador = @currentUserId
        AND l.visibilidade_lista = 'Privada'
        AND (@searchText IS NULL OR l.titulo_lista LIKE '%' + @searchText + '%')

        ORDER BY titulo_lista;
    END
END
GO
```

```sql
--SP para eliminar uma review
CREATE PROCEDURE projeto.sp_DeleteReview
    @reviewId VARCHAR(20),
    @userId VARCHAR(20)
AS
BEGIN
    DECLARE @gameId VARCHAR(20);

    BEGIN TRANSACTION;

    BEGIN TRY
        -- Get game ID first
        SELECT @gameId = id_jogo
        FROM projeto.review
        WHERE id_review = @reviewId AND id_utilizador = @userId;

        IF @gameId IS NULL
        BEGIN
            RAISERROR('Review not found or not owned by user', 16, 1);
            ROLLBACK;
            RETURN;
        END

        -- Delete reactions first (triggers will handle the rest)
        DELETE FROM projeto.reage_a WHERE id_review = @reviewId;

        -- Delete the review (this will trigger tr_UpdateGameRating)
        DELETE FROM projeto.review WHERE id_review = @reviewId;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END
GO
```

UDFs

```sql
-- UDF para estatisticas do utilizador
CREATE FUNCTION projeto.fn_GetUserStats
(
    @userId VARCHAR(20)
)
RETURNS TABLE
AS
RETURN
(
    SELECT
        COUNT(DISTINCT r.id_jogo) as games_reviewed,
        COUNT(DISTINCT l.id_lista) as lists_created,
        COUNT(DISTINCT s1.id_utilizador_seguido) as following_count,
        COUNT(DISTINCT s2.id_utilizador_seguidor) as followers_count
    FROM projeto.utilizador u
    LEFT JOIN projeto.review r ON u.id_utilizador = r.id_utilizador
    LEFT JOIN projeto.lista l ON u.id_utilizador = l.id_utilizador
    LEFT JOIN projeto.segue s1 ON u.id_utilizador = s1.id_utilizador_seguidor
    LEFT JOIN projeto.segue s2 ON u.id_utilizador = s2.id_utilizador_seguido
    WHERE u.id_utilizador = @userId
);
GO
```

```sql
-- UDF para gerar IDs exclusivos
CREATE FUNCTION projeto.fn_GenerateEntryId
(
    @listId VARCHAR(20)
)
RETURNS VARCHAR(20)
AS
BEGIN
    DECLARE @count INT;
    DECLARE @newId VARCHAR(20);

    SELECT @count = COUNT(*) + 1
    FROM projeto.entrada_lista
    WHERE id_lista = @listId;

    SET @newId = 'E' + RIGHT('000' + CAST(@count AS VARCHAR(3)), 3);

    RETURN @newId;
END
GO
```

Triggers

```sql
---- MAIN PAGE ----
-- Trigger para atualizar rating dos jogos quando a review muda
CREATE TRIGGER projeto.tr_UpdateGameRating
ON projeto.review
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @affectedGames TABLE (id_jogo VARCHAR(20));

    INSERT INTO @affectedGames (id_jogo)
    SELECT DISTINCT id_jogo FROM inserted
    UNION
    SELECT DISTINCT id_jogo FROM deleted;

    UPDATE projeto.jogo
    SET rating_medio = (
        SELECT ISNULL(AVG(CAST(rating AS DECIMAL(3,2))), 0)
        FROM projeto.review r
        WHERE r.id_jogo = projeto.jogo.id_jogo
    )
    WHERE id_jogo IN (SELECT id_jogo FROM @affectedGames);
END
GO
```

```sql
-- Trigger para verificar numero maximo de jogos por lista (maximo: 20)
CREATE TRIGGER projeto.tr_VerificarMaximoJogosPorLista
ON projeto.entrada_lista
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @count_jogos INT;
    DECLARE @id_lista VARCHAR(20);
    DECLARE @MAX_JOGOS_POR_LISTA INT = 20;

    -- Verificar cada lista afetada pela inser��o
    -- CURSOR Funciona como um while com FETCH NEXT
    DECLARE lista_cursor CURSOR FOR
    SELECT DISTINCT id_lista FROM inserted;

    OPEN lista_cursor;
    FETCH NEXT FROM lista_cursor INTO @id_lista;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Contar quantos jogos existem na lista ap�s a inser��o
        SELECT @count_jogos = COUNT(*)
        FROM projeto.entrada_lista
        WHERE id_lista = @id_lista;

        -- Se exceder o limite, fazer rollback
        IF @count_jogos > @MAX_JOGOS_POR_LISTA
        BEGIN
            CLOSE lista_cursor;
            DEALLOCATE lista_cursor;

            RAISERROR('Erro: A lista j@ cont@m o n@mero m@ximo de jogos permitidos (%d). N@o @ poss@vel adicionar mais jogos.',
                      16, 1, @MAX_JOGOS_POR_LISTA);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        FETCH NEXT FROM lista_cursor INTO @id_lista;
    END

    CLOSE lista_cursor;
    DEALLOCATE lista_cursor;
END;
```

DEMO