

# Relatório Final - BD



MIECT - Base de Dados - Trabalho Prático Final

## **Alunos:**

- Pedro Silva - [pedro.mfsilva@ua.pt](mailto:pedro.mfsilva@ua.pt) - 72645;
- Gonalo Grilo Alexandre - [grilogoncalo31@ua.pt](mailto:grilogoncalo31@ua.pt) -72608;

**Docente:** - Joaquim Sousa Pinto;

**Data:** 07/06/2017

# Índice

- Introdução
- Análise de Requisitos
- Diagrama de Entidade Relação
- Esquema Relacional
- Normalização
- Stored Procedures
- User-Defined Functions
- Views
- Indices
- Triggers
- Descrição da aplicação
- Notas
- Conclusão
- Referências

# Introdução

Como tema de projecto da unidade curricular de Base de Dados propusemos fazer uma plataforma de gestão jogos online.

O objectivo desta aplicação é permitir a um utilizador gerir uma base de dados de gestão de jogos contendo todos os dados relativos aos utilizadores da plataforma online assim como todos os registos de compra de jogos, podendo fazer perguntas em formato de “queries” personalizadas que devolvam dados de interesse ao utilizador que está a gerir a base de dados.

Esta aplicação foi desenvolvida em WPF/C# e é suportada por uma base de dados SQL Server que vai agregar grandes quantidades de dados das diversas entidades do SGBD. Esta base de dados deve fornecer um conjunto de funcionalidades básicas ao gestor da base de dados, tais como criar, apagar, alterar e consultar essa base de dados de forma eficiente, robusta e segura.

Este projecto foi desenvolvido com base na matéria lecionada na unidade curricular, o que permitiu um desenvolvimento faseado na seguinte estrutura: - Análise de requisitos; -Desenho Conceptual; -Desenho do Esquema Lógico; -Desenho do Esquema Físico e Administração;

Este relatório vai reflectir todo o trabalho envolvido no desenvolvimento desta aplicação, desde o protótipo em papel, as mudanças que foram feitas até a versão final da nossa aplicação.

# Análise de Requisitos

Esta foi uma das partes mais importantes, senão a parte mais importante, deste projecto visto que sem uma boa análise de requisitos não seria possível ter uma visão clara e objectiva das necessidades do nosso utilizador alvo que é o gestor da plataforma, e dos seus processos de pesquisa mais frequentes.

Com base em outras plataformas de jogos online(e.x: Steam, Origin) foi possível conceber as entidades principais da nossa base de dados, assim como a lista de atributos de cada uma destas entidades.

A entidade **Pessoa** é identificada unicamente pelo seu nome e possui como atributos não chave a sua data de nascimento e o seu endereço.

A entidade **Tipo Utilizador** é identificada unicamente pelo seu ID e possui como atributos não chave uma descrição da sua função assim como um atributo bit “Is\_Admin” que indica se o utilizador têm ou não privilégios de administrador.

A entidade **Utilizador** é identificada pelo seu Email que é a sua chave primária e é também identificada por duas chave estrangeiras: uma em que é identificada pelo nome do utilizador que herda da entidade **Pessoa** e outra que é identificada pelo seu typeID que herda da entidade **Tipo Utilizador**.

O **Utilizador** cria então a sua **Conta** e esta entidade possui como chave primária o seu ID e como chave estrangeira o email do utilizador. Esta entidade tem um NickName do utilizador, o método de pagamento à escolha do utilizador, um ranking da conta, uma password e um atributo bit “Is Disabled” que está ativo caso o utilizador tenha sido banido e a sua conta tenha sido desactivada.

Tendo criado a sua **Conta**, o utilizador terá acesso a três novas entidades: a entidade **Loja** em que o utilizador procura pelo **Jogo** que deseja comprar, ou adicionar esse **Jogo** à sua wishlist, ou ver os jogos que estão em promoção; outra entidade **Comunidade** em que o utilizador cria o sua própria comunidade e convida uma lista de amigos para jogar online; e finalmente uma entidade **Biblioteca** em que vão ser guardados os seus jogos.

A entidade **Jogo** têm um papel fundamental na nossa base de dados e esta entidade é fornecida por outra entidade que é o **Fornecedor** à qual não está muito presente no esquema da aplicação.

O jogo têm então como chave primária o seu número de registo e como chave estrangeira o nome do seu fornecedor. Este jogo têm como atributos não chave o nome desse jogo, a sua categoria e o seu preço.

O utilizador pode também deixar uma **Review** no **Jogo** tendo assim como chave primária desta entidade o ID dessa review e chaves estrangeiras o email do **Utilizador** e número de registo do **Jogo**.

Esta aplicação, no princípio, era para ter mais entidades e outras funcionalidades mas decidiu-se focar mais nas entidades principais e deixar algumas entidades e as outras funcionalidades de fora da nossa base de dados.

No entanto, com a nossa aplicação, o utilizador que vai gerir a plataforma será capaz de adicionar/apagar/alterar/pesquisar dados das diversas entidades anteriormente referidas e de ter uma boa performance na gestão dos dados.

# Diagrama Entidade Relação

Tal como foi dito anteriormente, o nosso projecto tinha muitas relações e muitas entidades e muitas outras funcionalidades o que traria também algumas dependências funcionais assim como algumas redundâncias relativamente ao funcionamento do nosso modelo de dados:

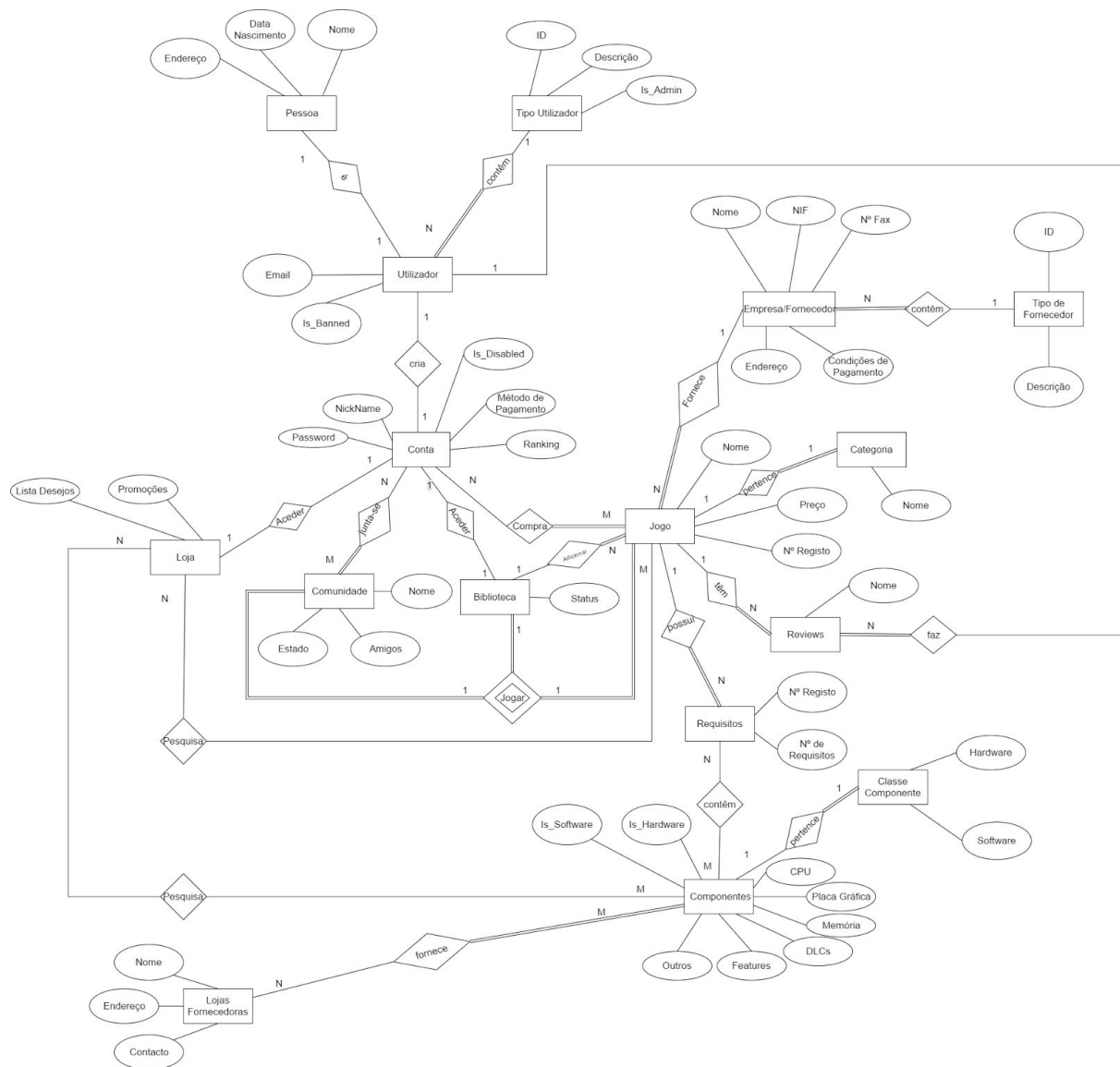


Fig.1 : Diagrama Anterior

Por essa razão, foi então decidido focar-nos mais nos aspectos principais que achamos necessários a ter na nossa aplicação:

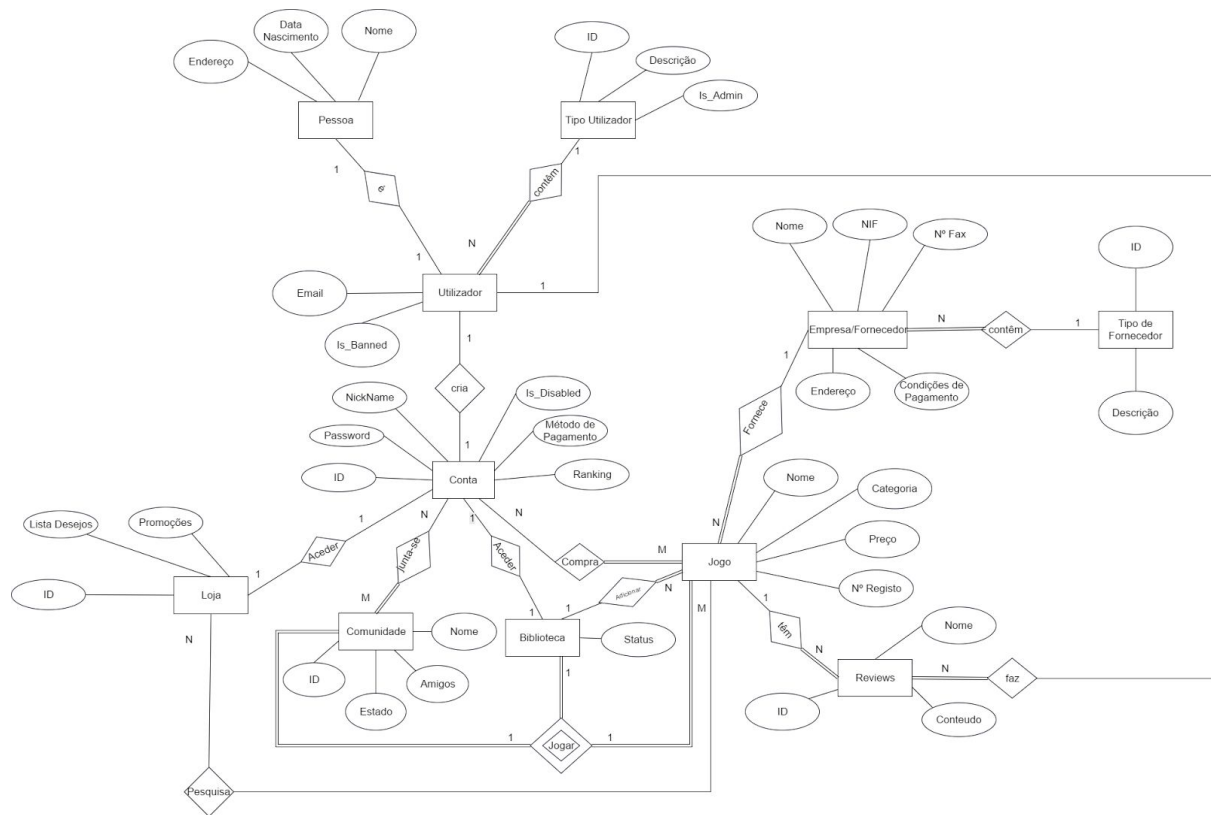


Fig.2 : Diagrama Final

## Esquema Relacional

Após a elaboração do Diagrama Entidade Relação(DER), procedeu-se de seguida à elaboração do Esquema Relacional(ER) da nossa Base de Dados. No entanto, tal como foi dito anteriormente, nós tínhamos um diagrama com muitas entidades e muitas dependências e algumas redundâncias nos atributos chaves do nosso modelo de dados:

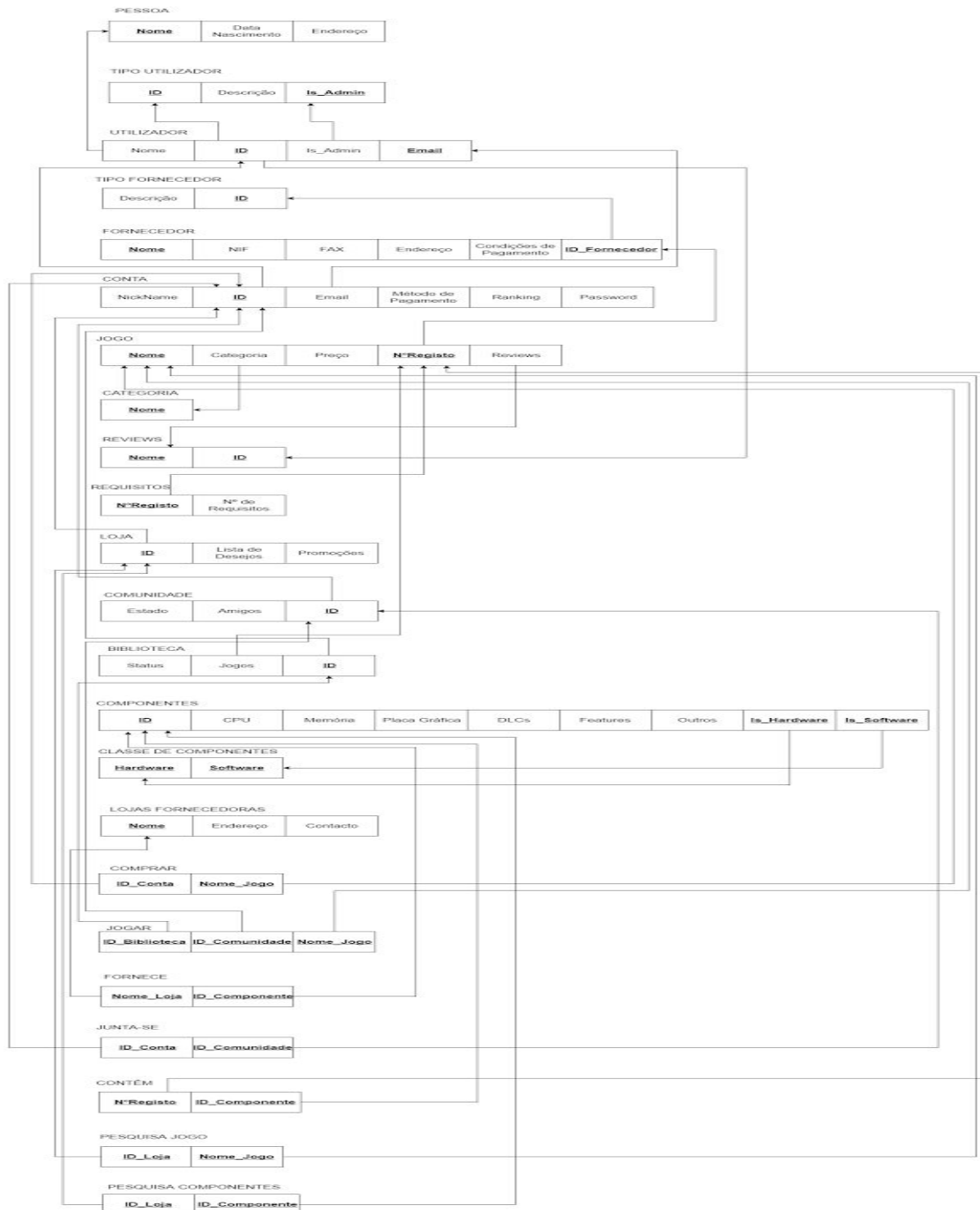


Fig.3 : Esquema Relacional Anterior

No entanto, com recurso a regras de normalização juntamente com a remoção de algumas entidades o esquema relacional foi possível tornar este esquema relacional mais simples e de interpretação mais clara:



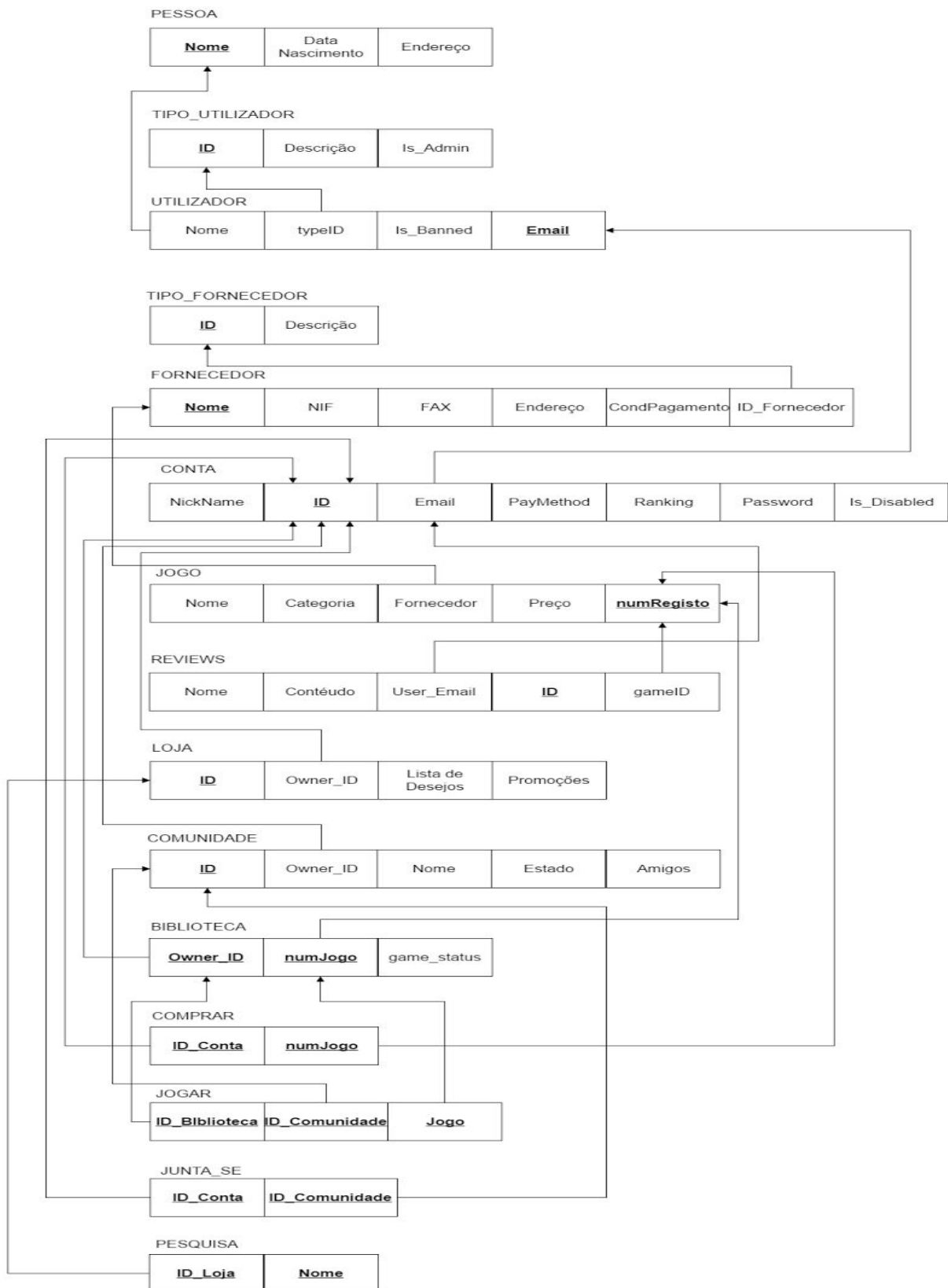


Fig.2: Esquema Relacional Final

# Normalização

Tal como foi referido anteriormente na parte do **Esquema Relacional**, foi necessário utilizar algumas regras de normalização para resolver as inconsistências do esquema anterior.

Com a normalização conseguimos resolver o problema de termos muitas entidades, o problema das redundâncias nos atributos chaves da entidade assim como alguns problemas nas dependências funcionais no modelo de dados. Alguns exemplos foram as entidades **Utilizador** e **Conta**.

Após a análise de todas as relações do nosso projecto, é possível concluir que se encontram na forma 3FN, ou seja não existem dependências transitivas onde não existem dependências funcionais entre elementos não chave.

## Stored Procedures

Foram utilizados os **Stored procedures** permitem criar uma camada de abstração entre o modelo de dados e a camada aplicacional. Desta forma, foi possível encapsular e proteger a base de dados, garantindo uma boa performance e mantendo a integridade dos dados. Outra grande vantagem foi o facto de não ser preciso ser recompilados cada vez que são invocados devido ao seu armazenamento em cache, o que permite uma maior rapidez no acesso aos dados o que beneficia a performance do nosso modelo de dados.

Sendo assim, foram utilizados os **Stored Procedures** para criar métodos que nos permitem inserir, eliminar e alterar os dados. Também foram utilizados alguns stored procedures para banir/desbanir utilizadores e para fazer disable/enable a contas. É de salientar que estes Stored Procedures contêm Transactions para garantir a integridade e consistência do nosso modelo de dados quando ocorrem alterações nos dados.

```
566 GO
567 CREATE PROCEDURE sp_createAccount
568     @NickName VARCHAR(50),
569     @Email VARCHAR(50),
570     @pass VARCHAR(50)
571 AS
572 DECLARE @count1 INT
573 SELECT @count1 = COUNT(Email) FROM Game_Masters.Utilizador WHERE Game_Masters.Utilizador.Email = @Email;
574 IF @count1 = 0
575 BEGIN
576     RAISERROR ('The user do not exist!', 14, 1)
577     RETURN
578 END
579
580 BEGIN TRANSACTION;
581 BEGIN TRY
582     INSERT INTO Game_Masters.Conta([NickName],[Email], [pass])
583     VALUES (@NickName, @Email, @pass)
584     COMMIT TRANSACTION;
585 END TRY
586 BEGIN CATCH
587     RAISERROR ('An error has occurred when creating the account!', 14, 1)
588     ROLLBACK TRANSACTION;
589 END CATCH;
590 GO
591
592 GO
593 CREATE PROCEDURE sp_removeAccount
594     @ID int
595 AS
596 IF @ID IS NULL
597 BEGIN
598     PRINT 'The ID of the account cannot be null!'
599     RETURN
600 END
601
602 BEGIN TRANSACTION;
603 BEGIN TRY
604     DELETE FROM Game_Masters.Conta WHERE ID = @ID;
605     COMMIT TRANSACTION;
606 END TRY
607 BEGIN CATCH
608     RAISERROR ('An error has occurred when trying to delete the account!', 14, 1)
609     ROLLBACK TRANSACTION;
610 END CATCH;
611 GO
```

```
613 GO
614 CREATE PROCEDURE sp_updateAccount
615     @NickName VARCHAR(50),
616     @ID int,
617     @Email VARCHAR(50),
618     @PayMethod VARCHAR(50),
619     @Ranking int,
620     @pass VARCHAR(50)
621 AS
622 IF @ID IS NULL
623 BEGIN
624     PRINT 'The account ID cannot be null!'
625     RETURN
626 END
627
628 DECLARE @count INT
629
630 --check if the Account ID exists
631 SELECT @count = COUNT(ID) FROM Game_Masters.Conta WHERE Game_Masters.Conta.ID = @ID;
632
633 IF @count = 0
634 BEGIN
635     RAISERROR ('The ID that you provided does not exist!', 14, 1)
636     RETURN
637 END
638
639 BEGIN TRANSACTION;
640 BEGIN TRY
641     UPDATE Game_Masters.Conta SET
642         NickName = @NickName,
643         Email = @Email,
644         PayMethod = @PayMethod,
645         Ranking = @Ranking,
646         pass = @pass
647     WHERE Game_Masters.Conta.ID = @ID;
648     COMMIT TRANSACTION;
649 END TRY
650 BEGIN CATCH
651     RAISERROR ('An error has occurred when trying to update the account!', 14, 1)
652     ROLLBACK TRANSACTION;
653 END CATCH;
654 GO
```

Fig.4 : Stored Procedures

# User-Defined Functions

O uso de **User-Defined Functions(UDF)** no nosso modelo de dados foi fundamental para tornar possível a visualização de dados na nossa aplicação.

Como as **UDFs** permitem o uso de lógica complexa nas consultas de dados, decidimos utilizar essas functions a fim de obter relações entre as entidades, obter os dados de uma determinada entidade e apresentá-los na forma aplicacional ao utilizador, e para procurar dados de uma dada entidade a partir de parâmetros fornecidos pelo utilizador

```

1533 GO
1534 CREATE FUNCTION udf_show_user()
1535 RETURNS TABLE
1536 AS
1537 RETURN (SELECT Game_Masters.Utilizador.Nome as 'User Name', Is_Admin as 'User Admin', Is_Banned as 'User Banned', Game_Masters.Utilizador.Email as 'User Email',
1538          COUNT(distinct Game_Masters.Conta.ID) as 'User Account',
1539          COUNT(distinct Game_Masters.Reviews.ID) as 'User Reviews'
1540          FROM ((Game_Masters.Utilizador INNER JOIN Game_Masters.Tipo_Utilizador ON Utilizador.TipoID = Tipo_Utilizador.ID) INNER JOIN Game_Masters.Conta ON Game_Masters.Utilizador.Email = Game_Masters.Conta.Email)
1541          INNER JOIN Game_Masters.Reviews ON Game_Masters.Utilizador.Email = Game_Masters.Reviews.user_email
1542          GROUP BY Game_Masters.Utilizador.Nome, Is_Admin, Is_Banned, Game_Masters.Utilizador.Email);
1543 GO
1544
1545 GO
1546 CREATE FUNCTION udf_getUser_Name(@Name VARCHAR(50))
1547 RETURNS TABLE
1548 AS
1549 RETURN (SELECT Nome as 'User Name', Email as 'User Email' FROM Game_Masters.Utilizador WHERE Nome like '%'+@Name+'%');
1550 GO
1551
1552 GO
1553 CREATE FUNCTION udf_showBannedUsers()
1554 RETURNS TABLE
1555 AS
1556 RETURN (SELECT Game_Masters.Utilizador.Nome as 'User Name', Is_Banned as 'User Banned', Game_Masters.Utilizador.Email as 'User Email'
1557          FROM Game_Masters.Utilizador
1558          WHERE Game_Masters.Utilizador.Is_Banned = '1'
1559          GROUP BY Game_Masters.Utilizador.Nome, Is_Banned, Game_Masters.Utilizador.Email);
1560 GO
1561
1562 GO
1563 CREATE FUNCTION udf_getBannedUser_ID(@ID int)
1564 RETURNS TABLE
1565 AS
1566 RETURN (SELECT Nome as 'User Name', Email as 'User Email' FROM Game_Masters.Utilizador WHERE Game_Masters.Utilizador.Email = @ID AND Game_Masters.Utilizador.Is_Banned = '1');
1567 GO
1568
1569 GO
1570 CREATE FUNCTION udf_getBannedUser_Name(@Nome VARCHAR(50))
1571 RETURNS TABLE
1572 AS
1573 RETURN (SELECT Nome as 'User Name', Email as 'User Email' FROM Game_Masters.Utilizador WHERE Game_Masters.Utilizador.Nome LIKE '%'+@Nome+'%' AND Game_Masters.Utilizador.Is_Banned = '1');
1574 GO
1575

```

Fig.5 : User-Defined Functions

## Views

No nosso projecto decidimos não utilizar **Views** visto que com os conhecimentos que adquirimos nas aulas teóricas, concluímos que o uso de **UDFs** seria o mais indicado, por serem facilmente compiladas e optimizadas, por serem mais seguras(**UDFs** permitem adicionar a opção de Schemabinding o que previne a alteração ou remoção de tabelas utilizadas por essa função), para além do facto de poderem suportar consultas de nível de complexidade mais elevado.

# Índices

No nosso projecto foram usados índices devido ao volume de dados que a nosso modelo de dados vai ter de suportar, de tal modo que as consultas efectuadas iriam começar a ter tempos de resposta superiores aqueles que o utilizador pretende. De modo a evitar isso, precisamos de manter as tabelas organizadas de maneira a que a consulta pode ser executada num intervalo de tempo muito curto.

Para isso acontecer criamos índices **NonClustered**, nos atributos que achamos serem alvo de consulta frequente e também devido ao facto de índices do tipo **Clustered** serem criados automaticamente.

Utilizamos estes índices com um **FILLFACTOR = 80**, que determina a percentagem de espaço em cada página, e com **pad\_index** activo, que aplica as páginas secundárias o valor do **FILLFACTOR**.

Basicamente utilizamos os índices de modo a beneficiar a performance da pesquisa dos dados.

```

167 GO
168 GO
169 CREATE NONCLUSTERED INDEX index_account_name ON Game_Masters.Conta(NickName)
170 WITH (FILLFACTOR=80, pad_index=ON);
171
172 GO
173 CREATE NONCLUSTERED INDEX index_provider_address ON Game_Masters.Fornecedor(Endereco)
174 WITH (FILLFACTOR=80, pad_index=ON);
175
176 GO
177 CREATE NONCLUSTERED INDEX index_game_provider ON Game_Masters.Jogo(Fornecedor)
178 WITH (FILLFACTOR=80, pad_index=ON);
179
180 GO
181 CREATE NONCLUSTERED INDEX index_game_name ON Game_Masters.Jogo(NomeJogo)
182 WITH (FILLFACTOR=80, pad_index=ON);
183
184 GO
185 CREATE NONCLUSTERED INDEX index_game_register ON Game_Masters.Jogo(numRegisto)
186 WITH (FILLFACTOR=80, pad_index=ON);
187
188 GO
189 CREATE NONCLUSTERED INDEX index_game_category ON Game_Masters.Jogo(Categoria)
190 WITH (FILLFACTOR=80, pad_index=ON);

```

Fig.6: Indices NonClustered

# Triggers

No nosso modelo foi usado um Trigger After Insert,Update para prevenir que os utilizadores seja gestores de duas contas ao mesmo tempo.

```

GO
CREATE TRIGGER trigger_user ON Game_Masters.Conta
AFTER INSERT, UPDATE
AS
SET NOCOUNT ON;
DECLARE @count AS INT;
SELECT @count=COUNT(Game_Masters.Conta.ID) FROM Game_Masters.Conta JOIN inserted ON Game_Masters.Conta.ID = inserted.ID;

IF @count > 1
BEGIN
RAISERROR ('O utilizador nao pode ser gestor que mais do que 1 conta.', 16, 1);
ROLLBACK TRAN;
END;
GO

```

Fig.7: Trigger After Insert, Update

# Descrição da Aplicação

O utilizador começa por fazer login à sua conta ou regista os seus dados:

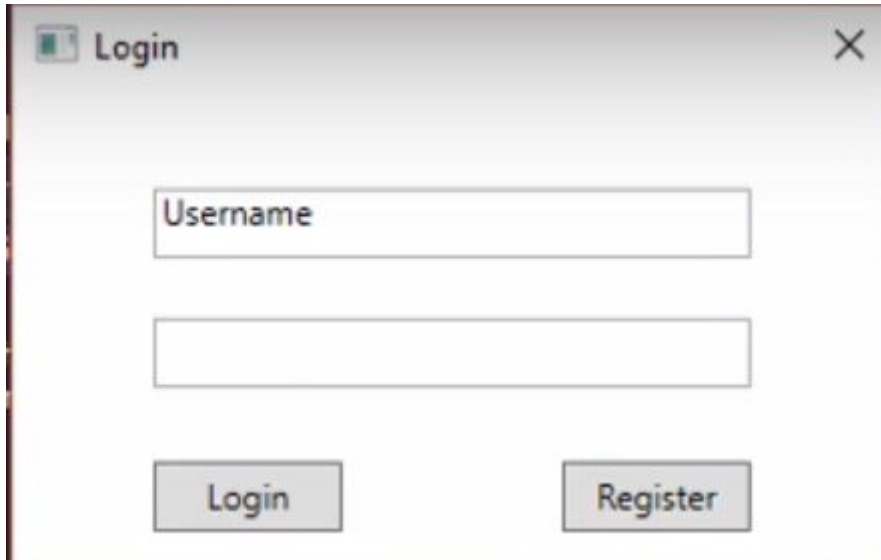
A screenshot of a web application window titled "Login". It features a title bar with a close button (X). The main content area has a "Username" label above a text input field. Below this is another empty text input field. At the bottom, there are two buttons: "Login" and "Register".

Fig.8: Login Interface

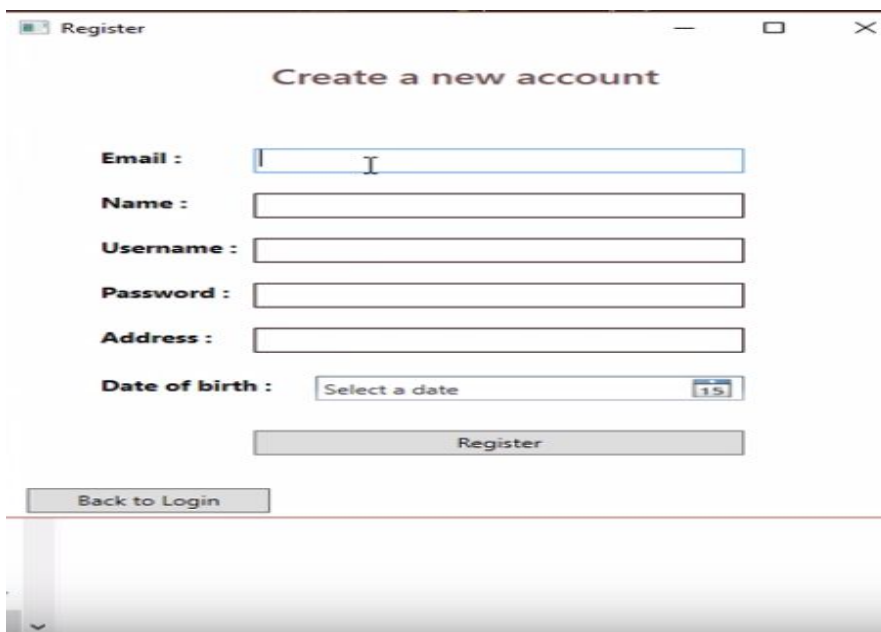
A screenshot of a web application window titled "Register". It features a title bar with standard window controls. The main content area has the heading "Create a new account". Below this, there are several form fields: "Email :", "Name :", "Username :", "Password :", "Address :", and "Date of birth :". The "Date of birth" field includes a date picker showing "15". At the bottom, there is a "Register" button and a "Back to Login" button.

Fig.9: Register Interface

Tendo os seus dados de utilizador registados e o seu login feito a seguinte figura demonstra o aspecto da nossa interface:

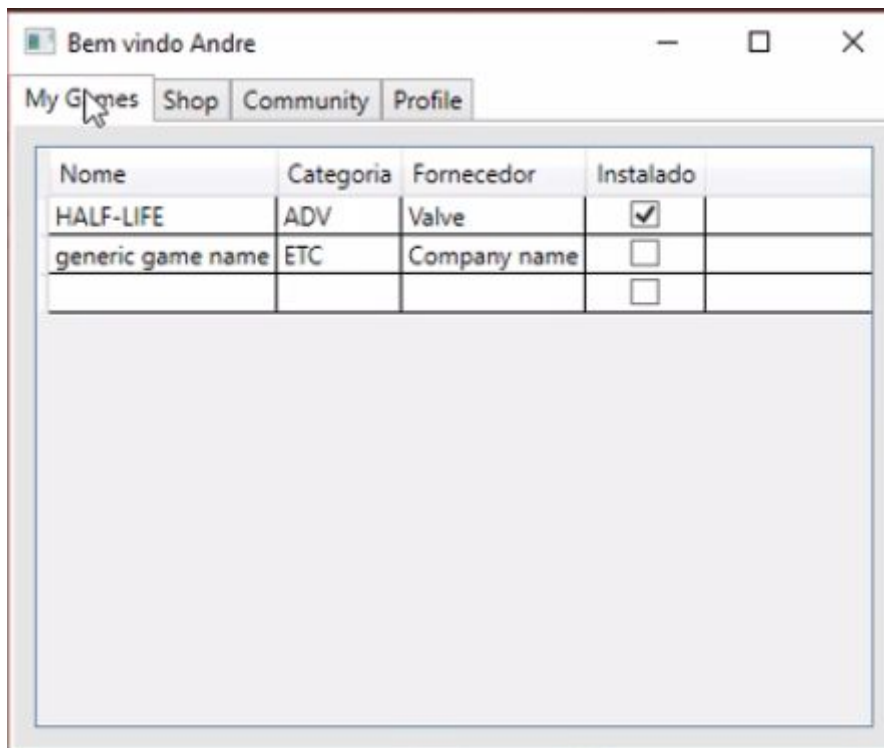


Fig.10: User Interface

Dentro desta interface temos vários campos tais como o My Games que demonstra todos os jogos que o utilizador têm na sua biblioteca assim como uma informação se o jogo está instalado ou não. De seguida temos o campo Shop em que o utilizador introduz o jogo ou nome do fornecedor que quer comprar e clicando duas vezes no jogo ele fica comprado e é adicionado ao My Games



Fig.11: Shop Interface



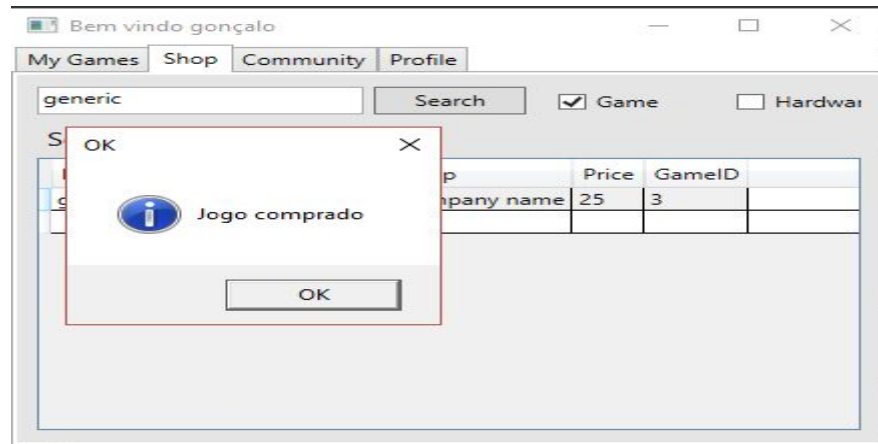


Fig.12: Jogo Comprado

De seguida temos o campo Community que mostra a lista de amigos do utilizador assim como a informação base desses utilizadores. Basicamente a Comunidade só mostra os utilizadores amigos do utilizador.

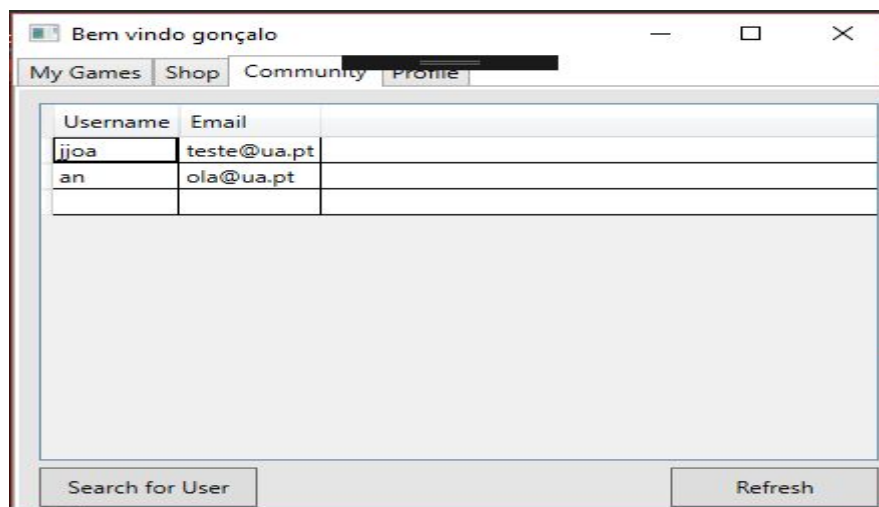


Fig.13: Community Interface

Dentro desta parte podemos fazer pesquisa dos utilizadores da comunidade:



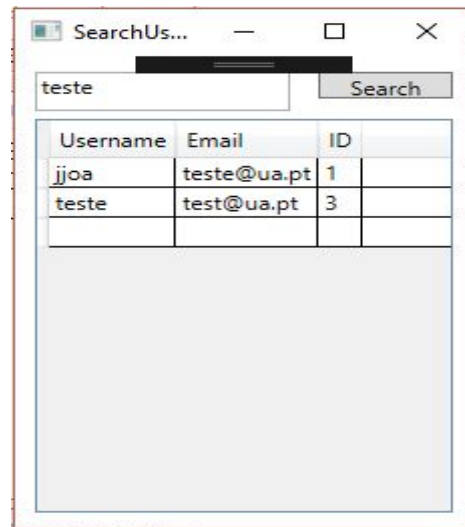


Fig.14: Search For User

Finalmente temos o campo Profile em que temos a informação do utilizador e é neste campo em que o utilizador pode fazer update aos seus dados

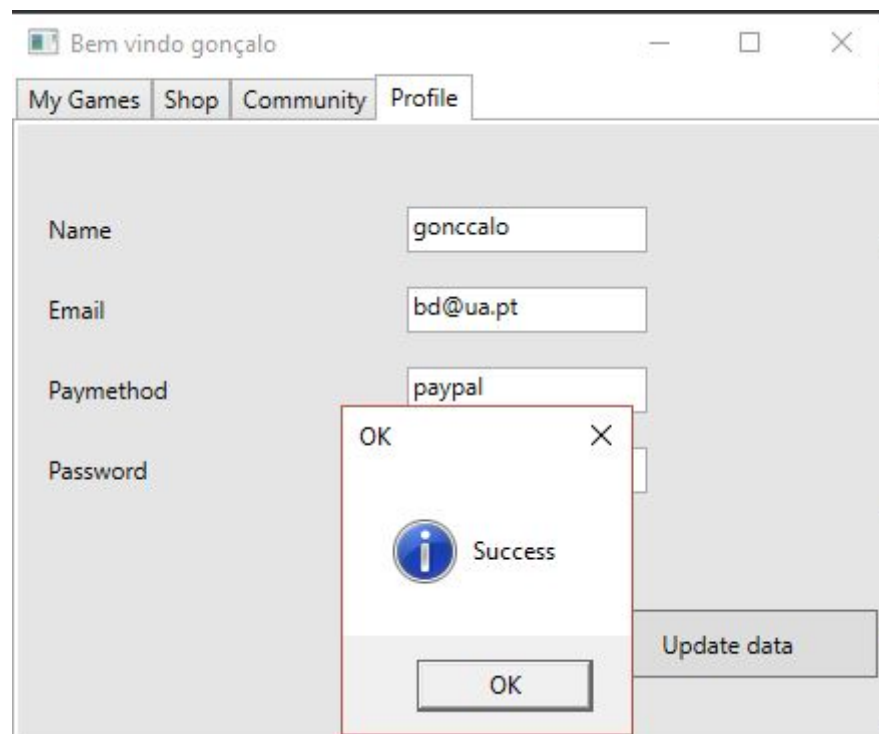


Fig.15: Update User Information

# Notas

Tal como foi referido anteriormente, nós resolvemos remodelar o nosso modelo de dados recentemente e por essa razão temos diagramas e esquemas novos e também resolvemos retirar algumas coisas que estavam mal feitas na nossa apresentação que foi feita dia 06/06/2017 nomeadamente o uso de **Check Constraints** para impedir Inserts e ou Updates com valores impróprios visto que da maneira como estava não podíamos inserir dados nas tabelas por causa do **Check Constraint**.

## Conclusão

O objectivos inicialmente propostos não foram bem atingidos mas no final achamos que conseguimos desenvolver, com base nos conteúdos leccionados na unidade curricular, um modelo de dados robusto e funcional.

No final, queremos agradecer ao professor Joaquim Sousa Pinto pela sua disponibilidade e por nos ter ajudado no desenvolvimento do nosso projecto.

## Referências

- SQL Server:  
<https://www.microsoft.com/pt-pt/sql-server/sql-server-2016>
- Stack Overflow:  
<https://stackoverflow.com/>