



DEPARTAMENTO DE ELETRÓNICA, TELECOMUNICAÇÕES E INFORMÁTICA
UNIVERSIDADE DE AVEIRO

RELATÓRIO DE BASE DE DADOS

APRESENTAÇÃO FINAL

SISTEMA PARA GESTÃO DE EVENTOS E-SPORTS

2021-2022

DANIEL JORGE BERNARDO FERREIRA
102885

AFONSO MARIA DA COSTA AZEVEDO
104272

ÍNDICE

1. INTRODUÇÃO	1
2. CONTRIBUIÇÃO.....	1
3. IMPORTANTE	2
4. ENTREGÁVEIS	3
5. ANÁLISE DE REQUISITOS.....	4
5.1 ENTIDADES.....	4
5.2 CARACTERÍSTICAS DO SISTEMA	5
6. DESENHO CONCEPTUAL	6
6.1 DIAGRAMA ER	6
6.2 MODELO RELACIONAL	7
7. DIAGRAM AUTO-GERADO FINAL	8
8. ALTERAÇÕES INTRODUZIDAS DURANTE O DESENVOLVIMENTO	9
9. INTERFACE.....	10
9.1 FORMS	11
9.2 USE-CASES	12
10. PROCESSO DE NORMALIZAÇÃO.....	13
11. STORED PROCEDURES (SP)	14
12. FUNCTIONS (UDF).....	15
13. TRIGGERS.....	16
14. GERADORES.....	17
15. NOTAS FINAIS	18

1. INTRODUÇÃO

Para o nosso projeto da unidade curricular de Base de Dados decidimos desenvolver um sistema de gestão de torneios de Esports. Este seria utilizado num ambiente competitivo, em torneios de vídeo jogos tanto por utilizadores regulares que apenas pretendem acompanhar as suas equipas e jogadores favoritos, como por membros das próprias equipas e organizadores de eventos.

O seu desenvolvimento tem como objetivo a aplicação prática dos conhecimentos adquiridos nas aulas teóricas e práticas desta unidade curricular em toda a sua plenitude, desde o pensamento e desenho do modelo da base de dados à sua gestão e manipulação por sistemas de software.

Este relatório surge como seguimento do trabalho apresentado anteriormente (Proposta), onde foi apresentado todo o procedimento relativo à análise de requisitos e desenho conceptual do sistema.

Este relatório irá focar-se mais na implementação e funcionamento do sistema.

2. CONTRIBUIÇÃO

Participação individual global

Daniel Ferreira	70%
Afonso Azevedo	30%

3. IMPORTANTE

Visto que o nosso trabalho é muito pesado, chegando a ocupar mais de 100 MB, não conseguimos enviar todo o trabalho aqui.

O limite no e-learning é de 50 MB.

Deixámos de fora a grande parte do trabalho, no que conta:

- A pasta Proposta, que contém todo o material apresentado na proposta do projeto.
- A pasta Apresentação, que contém os ficheiros com a apresentação final, incluindo a demo.
- A pasta Python, que contém todos os scripts python (geradores).
- O script, switchServer.py, criado para alterar automaticamente as connection_strings de todo o projeto tal como o seu ficheiro de configuração serverconfig.ini.
- Outros ficheiros não tão importantes.

Todo o trabalho, incluindo estes diretórios e ficheiros que fomos obrigados a deixar de parte, está disponível no nosso repositório no github:

<https://github.com/DanielFerreira011102/EsportsBD>

4. ENTREGÁVEIS

Em conjunto com este relatório entregamos vários ficheiros, descritos abaixo. No nosso projeto temos 5 pastas principais:

1. Esports – onde se encontram as classes, components, forms e resouces.
2. Python – contém todos os geradores (em python).
3. SQL – que contem todos os ficheiros *.sql* usados para o desenvolvimento da base de dados do projeto, divididos por tópico.
4. Proposta – contém o material apresentado durante a proposta.
5. Apresentação – contém o material usado na segunda apresentação, inclui os slides e demo. A demo apresentada não corresponde ao resultado final, visto que fizemos alterações entretanto.

Todos os ficheiros desenvolvidos durante o projeto estão também no nosso repositório online:

<https://github.com/DanielFerreira011102/EsportsBD>

Pode assistir à nossa demo apresentada na aula:

<https://www.youtube.com/watch?v=aPymw0M-1Fs>

Por fim, também pode assistir à demo onde mostramos o que atualizámos:

https://www.youtube.com/watch?v=OQUFi_k-h-Y

Notas:

1. Para compilar a aplicação, pode ser necessário mudar a connection string do servidor que se pretende utilizar. Para esse processo ser mais simples, decidimos criar um script em python com o nome *switchServer.py* que por default (sem argumentos) alterna entre o meu servidor local e o servidor remoto fornecido pela universidade. Basicamente percorre pelos ficheiros e dá replace na String que pretendemos mudar. O script também aceita como argumentos *old_conn_string*, *new_conn_string* e *path*.
2. Para gerar a base de dados completa, existe dentro da paste SQL um ficheiro *fullScript.sql* gerado automaticamente que cria todo o schema e insere todos os dados. Como o script é extenso, com mais de 110 mil linhas então pode demorar alguns minutos (~1-2 mins) a correr.

5. ANÁLISE DE REQUISITOS

Permitirá cobrir estatísticas e análise de torneios profissionais ou amadores.
Permitindo procurar informação referente aos atributos de cada entidade promovida destacando, dessa forma, informação de certos utilizadores, organizações, equipas, etc.

Nota:

1. Os requisitos apresentados neste capítulo são os propostos inicialmente.
Não correspondem inteiramente ao estado atual do trabalho.

5.1 ENTIDADES

User: Usuário do serviço proposto.

Team Staff: Staff da equipa, i.e. team manager, head coach, analyst...

Player: Usuário designado como jogador em uma equipa, pode ser capitão da equipa.

Equipa: Equipa com um conjunto de jogadores de um certo jogo.

Event staff: Usuário associado a uma certa organização, cada um com o seu cargo e responsabilidades.

Organization: Organização responsável por criar e regular eventos/torneios e-sports.

Tournament/Event: Competição entre várias equipas a um título.

Match/Series: Partida entre duas equipas.

Mapa: Definido em cada ronda/fase de uma match, existe dentro do jogo.

Player Stats: Conjunto de estatísticas de um jogador em um mapa, avalia a performance de um jogador.

Stat: Variável de performance de um jogador, o seu nome e valor dependem do jogo.

Game: Vídeo-jogo multiplayer, i.e. CS:GO, Valorant, Rocket League, League of legends...

Developer company: Empresa responsável por desenvolver e publicar um jogo.

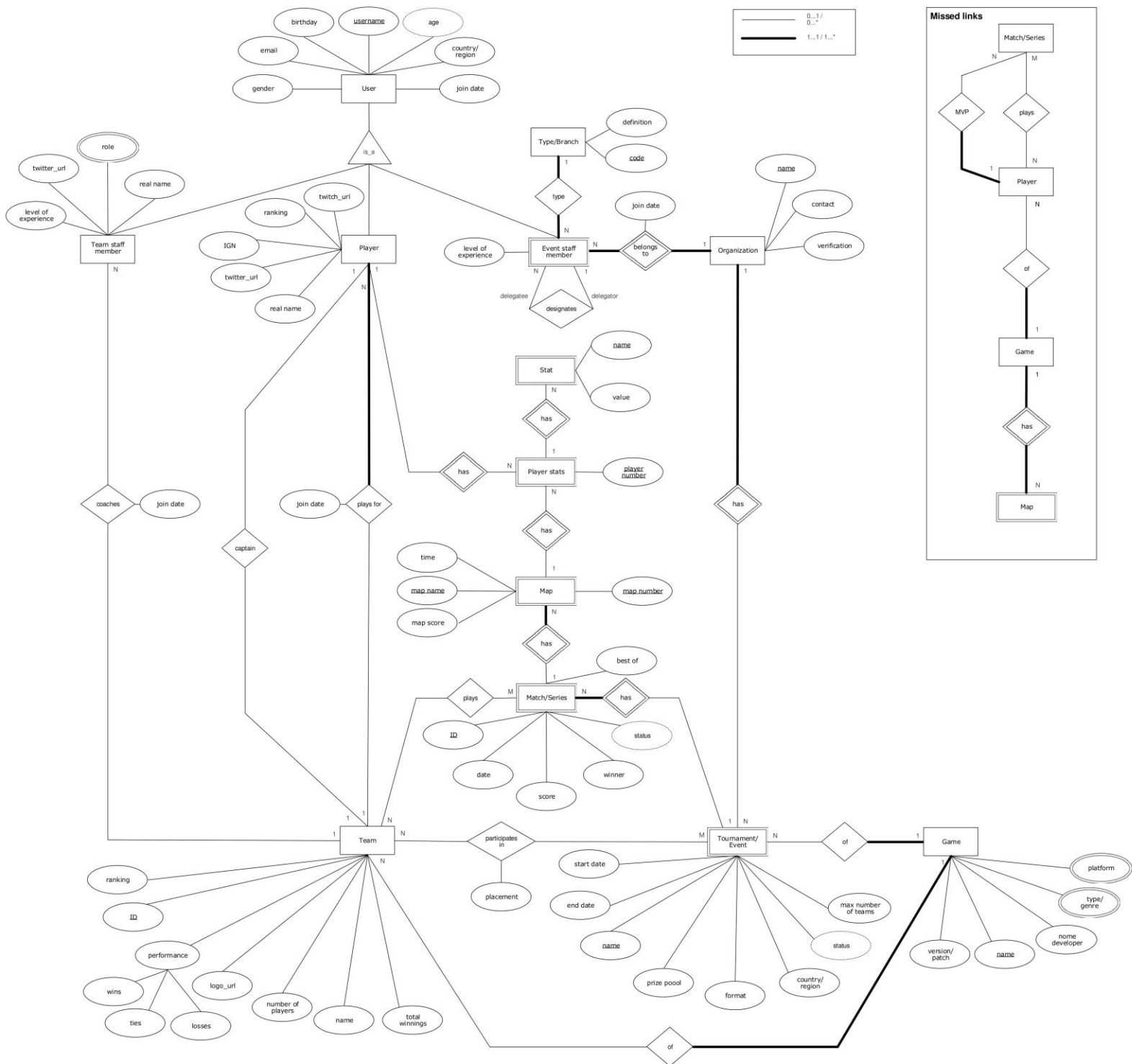
5.2 CARACTERÍSTICAS DO SISTEMA

- O nosso sistema de gestão de torneios de esports vai ser utilizado por vários users cada um deles caracterizado pelo seu email, data de nascimento, nome de utilizador, país/região e género. Pretende-se guardar também a sua data de inscrição no serviço.
- Um utilizador pode pertencer à staff de uma equipa, ser um jogador ou ser até mesmo um membro de uma organização responsável por estruturar eventos/torneios.
- Um membro da staff da equipa pode ter vários cargos na equipa.
- Cada player representa uma equipa. Pretendemos recordar as redes sociais dos jogadores para promover a marca de cada um.
- A staff e vários jogadores formam uma equipa que vai participar nos torneios e defrontar outras equipas em partidas (Match/Series).
- As partidas podem ter um ou mais mapas e as stats de cada jogador serão guardadas para cada mapa.
- O videojogo (Game) é caracterizado pelo seu tipo/género, nome e version/patch. Além disso, cada jogo deverá conter as plataformas associadas i.e. PS5, PC... e também a respetiva empresa desenvolvedora.
- O membro da staff de eventos é caracterizado pelo seu nível/anos de experiência, e pela sua função na organização. Um membro pode designar novos utilizadores para trabalhar na organização sobre a sua responsabilidade. A organização irá promover os torneios/eventos.
- Os torneios/eventos são definidos pela sua data de abertura, data de fecho, código, nome, formato, país/região, status e número máximo de equipas, um torneio será composto por várias equipas e naturalmente partidas (Match/Series). Será atribuída a cada equipa registada uma colocação no torneio.

6. DESENHO CONCEPTUAL

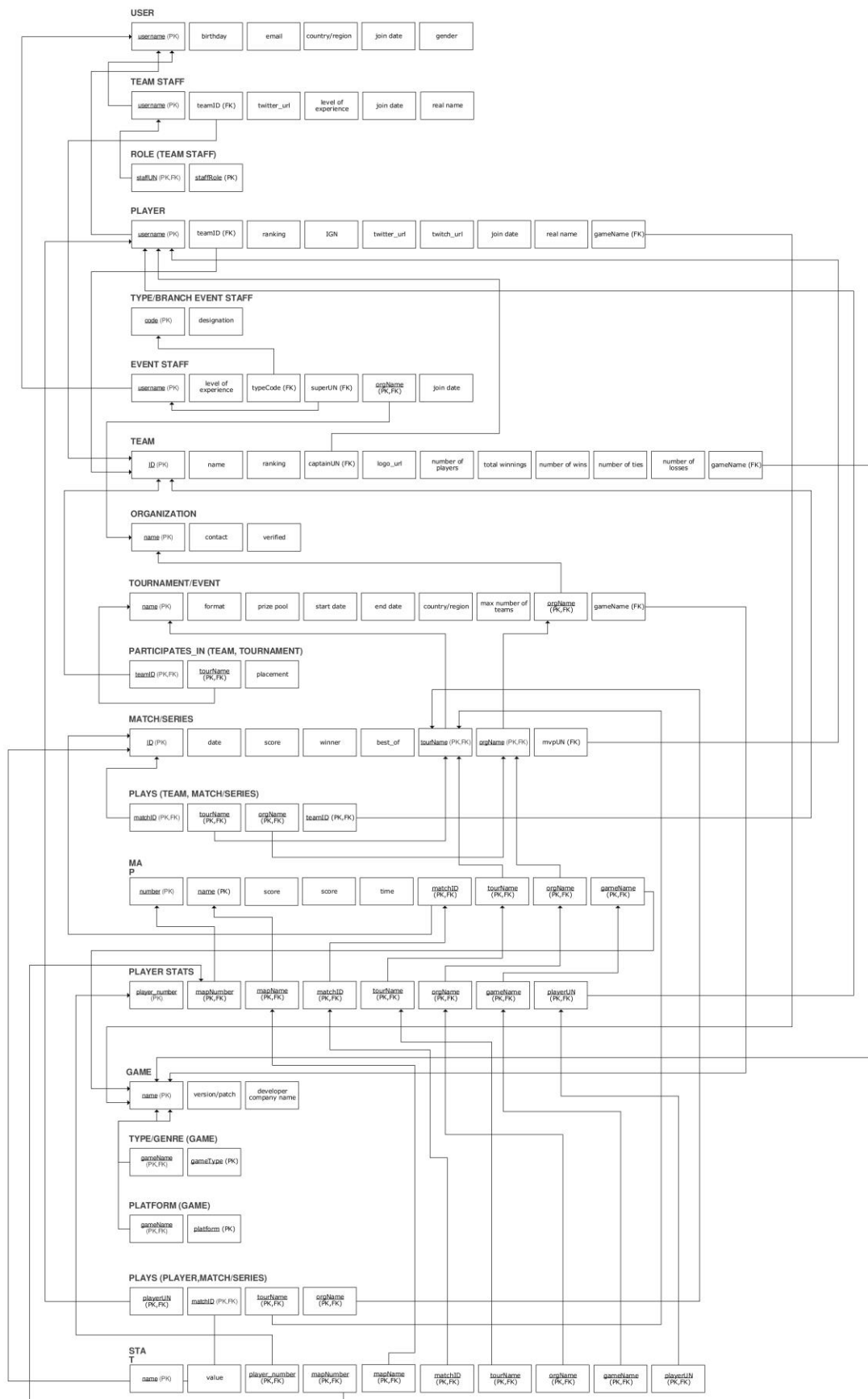
6.1 DIAGRAMA ER

[ERD.pdf](#)



6.2 MODELO RELACIONAL

[MR.pdf](#)



8. ALTERAÇÕES INTRODUZIDAS DURANTE O DESENVOLVIMENTO

- Na entidade GAME foi redefinida a coluna ‘platform’ para ser um atributo simples, para reduzir o número tabelas e porque não era uma característica crucial no nosso sistema.
- Foi eliminada a tabela DEVELOPER_COMPANY que colocámos como atributo dentro de GAME. No início pensávamos em detalhar também informação sobre cada produtora dos jogos, mas achámos que já tínhamos muitas tabelas e esta em específico não é muito importante.
- Por sugestão do professor durante a apresentação da proposta, alterámos a estrutura de algumas relações de forma a não existir uma cadeia enorme de entidades fracas e existir entidades com uma Primary key composta por muitas colunas diferentes.
- No MAP decidimos remover a coluna ‘nome’ da Primary Key, isto segue o ponto mencionado antes. Acabámos por decidir que um único ID bastava para classificar um mapa.
- Nas entidades ORGANIZATION e PLAYER foi adicionada uma imagem, para ajudar na identificação. Ainda na ORGANIZATION foi removida a coluna ‘verificação’ que achámos redundante.
- Desde a primeira apresentação do projeto foram criadas as tabelas TOURNAMENT_WINNER, SERIES_MVP, SERIES_RESULT, TEAM_PLAYS, TEAM_JOIN_MESSAGE e ORG_JOIN_MESSAGE. Parte destas tabelas foram criados durante o processo de normalização, outras foi porque achámos do nosso interesse e faziam sentido ser adicionadas.

9. INTERFACE

Para desenvolver o nosso projeto recorreremos ao IDE Visual Studio 2022 com integração do framework Windows Forms da plataforma .NET, utilizado para construir o UI da aplicação. Todo o código foi feito na linguagem C#.

Decidimos desenvolver a interface para melhor organizar os dados e permitir um possível utilizador entender facilmente o propósito e estrutura da aplicação. Apesar de estarmos cientes que o design da interface não iria ser avaliado, achámos o UI da aplicação como parte integral para se movimentar e perceber o sistema.

De forma a melhorar a experiência de visualização e manipulação de grandes quantidades de dados, procurámos desde o início permitir ao utilizador fazer pesquisas, filtragens e ordenação de acordo com os vários atributos nas tabelas. Devido à sua integração, é possível em todas as tabelas ordenar por um atributo ao carregar no cabeçalho respetivo. No entanto, este processo não foi feito em SQL (ainda que possível), pois achámos mais fácil e escalável integrar a solução fornecida pela própria Microsoft, em C#.

Na ocorrência de erros ou falha na correlação e validade dos dados utilizámos a componente MessageBox para mostrar um diagnóstico do que aconteceu, indicando ao utilizador que a sua ação não foi autenticada. Para além disso, também utilizámos esta componente muitas vezes para dar avisos, de maneira a tornar o sistema numa simulação mais realista.

9.1 FORMS

Ao todo, temos 11 forms diferentes:

LoginOrRegistForm: responsável pelo login e registo de um user.

MainForm: wrapper para os restantes forms, apresenta-se como um “menu” havendo tabs de navegação para cada outro form.

PlayersForm: contém informação sobre os jogadores, divididos por jogo.

TeamsForm: contém informação sobre as equipas, divididas por jogo.

EventsForm: contém informação sobre os torneios, divididos por jogo.

MatchesForm: contém informação sobre as partidas, divididas por jogo.

OrgsForm: contém informação sobre as organizações.

MyAccountForm: form onde é possível verificar e editar dados da sua conta.

MyTeamForm: form com 5 stages dependendo do estado do utilizador. fornece o caminho para criar ou registar-se a uma equipa. Caso o utilizador seja um Manager ou o líder da equipa permite alterações no contexto da equipa, adicionando e removendo membros, ou eliminando a própria equipa.

SearchForm: responsável pela funcionalidade de search, pode ser acedido sempre que clicamos no enter dentro da textbox ou clicando no icon à esquerda.

MyHomeForm: form inicial apresentado quando o utilizador se regista ou faz login, apresenta uma breve introdução sobre o projeto.

9.2 USE-CASES

Nós implementámos diversas funcionalidades/tarefas diferentes na nossa aplicação, pelo que descrever cada uma delas seria um pouco excessivo. Dessa forma, iremos falar das duas mais importantes.

1. Procurar por um PLAYER, EQUIPA, ORGANIZAÇÃO, etc:

- Esta foi uma das tarefas que achámos mais interessantes. Dada uma string conseguimos pesquisar por todas as colunas, não só pelas PKs, das tabelas indicadas, mostrando todos os resultados encontrados numa ListView.
- A sua implementação foi de certa forma complexa. Inicialmente, pensava que teria de usar Full-text Indexes e Full-text Search, ou utilizar alguma ferramenta como SQL search. No entanto depois de vasculhar na web encontrei uma solução que consegui adaptar para o cenário que queria facilmente, e implementei então o SP 'search' que se encontra no ficheiro *procedures.sql*. Apesar de ter utilizado uma solução que encontrei na web, fui obrigado a alterar toda a estrutura da SP para se adaptar ao meu cenário e fornecer a informação que precisava, e por isso considero como trabalho meu.
- Depois foram ainda desenvolvidos outros procedures que retornam a informação detalhada do record relativo à row selecionada na listView. Nomeadamente, os procedures 'getSearchPlayer', 'getSearchTeam', etc.

2. Enviar join requests para a uma TEAM:

- Esta foi uma nova funcionalidade que implementámos durante o desenvolvimento. Achámos desde início, que fornecer alguma maneira de um utilizador se juntar a uma equipa era uma das tarefas mais importantes do projeto. Mas não sabíamos como deveria ser feito. Como mencionei anteriormente, um dos objetivos deste sistema era simular uma aplicação, dito isso, apenas ter um botão que quando clicado automaticamente entravas na equipa, seria irrealista e sem sentido.
- Decidimos, então, fazer um sistema estilo de pedido/resposta, onde um utilizador envia um request que ficaria stored como record na tabela TEAM_JOIN_REQUESTS. Managers e líderes das equipas têm acesso a esses requests e escolhem entre aceitar ou rejeitar os pedidos. Naturalmente, os pedintes conseguem também cancelar o request sempre que quiserem.

10. PROCESSO DE NORMALIZAÇÃO

A normalização é o processo de organização de dados numa base de dados.

Ao implementar os geradores de records em python deparámo-nos com alguns problemas ao inserir os dados.

Algumas relações nas tabelas não estavam exatamente bem estruturadas, por exemplo, a tabela TEAM antes tinha uma coluna com referência ao team_captain (PLAYER), no entanto, como a tabela PLAYER também tinha referência à sua equipa, para criar um record teríamos de primeiro inserir o valor da coluna como NULL e depois dar update para acrescentar o captain ou a equipa, o que não é certamente uma boa solução.

Para resolver esse problema, recorremos ao processo de Normalização, criámos a nova tabela TEAM_CAPTAIN de forma que TEAM não tivesse dependências sobre o PLAYER, tornando a inserção dos dados mais simples.

O processo de normalização foi apenas feito quando achámos necessário ou quando fazia sentido normalizar as relações entre tabelas para tornar a base de dados mais flexível ou eliminar redundância e dependências inconsistentes.

Algumas das mudanças que fizemos e mencionámos em cima foi devido a este mesmo processo.

11. STORED PROCEDURES (SP)

Stored Procedures podem aceder ou modificar dados numa base de dados, mas não estão ligados a uma base de dados ou objeto específico, o que oferece uma série de vantagens.

Devido à dispersão e partilha dos dados de todas as entidades em várias tabelas, decidimos criar Stored Procedures para a sua inserção e edição, que recebem todos os atributos, validam e distribuem pelas relações, abstraindo toda a complexidade associada a esta tarefa do código da interface.

Decidimos utilizar Procedures sobretudo para executar operações de DML e simplificar a manipulação dos dados.

```

1 DROP PROCEDURE IF EXISTS resetID
2 DROP PROCEDURE IF EXISTS whatNeedsIndexes
3 DROP PROCEDURE IF EXISTS createIndexes
4 DROP PROCEDURE IF EXISTS deletePT
5 DROP PROCEDURE IF EXISTS helpIndex
6 DROP PROCEDURE IF EXISTS createTeamPlayerExists
7 DROP PROCEDURE IF EXISTS createTeamPlayerDoesNotExist
8 DROP PROCEDURE IF EXISTS createTeamPlayerJoinRequest
9 DROP PROCEDURE IF EXISTS removeStaffFromTeam
10 DROP PROCEDURE IF EXISTS removePlayerFromTeam
11 DROP PROCEDURE IF EXISTS acceptTeamPlayerExists
12 DROP PROCEDURE IF EXISTS acceptTeamStaffExists
13 DROP PROCEDURE IF EXISTS acceptPlayerDoesNotExist
14 DROP PROCEDURE IF EXISTS acceptTeamStaffDoesNotExist
15 DROP PROCEDURE IF EXISTS deleteTeam
16 DROP PROCEDURE IF EXISTS deleteUser
17 DROP PROCEDURE IF EXISTS editUser
18 DROP PROCEDURE IF EXISTS search
19 DROP PROCEDURE IF EXISTS SearchAllTables
20 DROP PROCEDURE IF EXISTS searchAll
21 DROP PROCEDURE IF EXISTS getSearchPlayer
22 DROP PROCEDURE IF EXISTS getSearchTeam
23 DROP PROCEDURE IF EXISTS getSearchOrg
24 DROP PROCEDURE IF EXISTS getSearchSeries
25 DROP PROCEDURE IF EXISTS getSearchTour
26 DROP PROCEDURE IF EXISTS rejectRequest

```


12. FUNCTIONS (UDF)

As funções criadas ajudaram-nos a controlar o progresso do nosso projeto. Utilizámos UDFs para realizar operações de ‘check’ de forma a verificar a integridade dos dados.

As funções também se revelaram importantes na consulta da informação da base de dados e como ferramenta de processamento.

Decidimos optar mais por utilizar UDFs do que Procedures para trabalhar com DQL. A simples razão é que apesar de Stored Procedures terem uma série de vantagens sobre UDFs, SPs não podem ser utilizados em instruções SELECT e dessa forma, não podem ser aproveitados na criação de outras queries semelhantes. Dito isso, este era o nosso plano quando começámos o trabalho, mas não conseguimos retirar tanto partido desse benefício quanto desejávamos.

```

1 DROP FUNCTION IF EXISTS CheckUserExists;
2 DROP FUNCTION IF EXISTS CheckUserMatchPassword;
3 DROP FUNCTION IF EXISTS getUsersNotPlayers;
4 DROP FUNCTION IF EXISTS getUsersNotEventStaff;
5 DROP FUNCTION IF EXISTS getUsersNotTeamStaff;
6 DROP FUNCTION IF EXISTS getTeams;
7 DROP FUNCTION IF EXISTS checkPlayerIGL;
8 DROP FUNCTION IF EXISTS getGamePlayerData;
9 DROP FUNCTION IF EXISTS getRandomTeamCaptain;
10 DROP FUNCTION IF EXISTS getPlayerInfo;
11 DROP FUNCTION IF EXISTS getGameTeamData;
12 DROP FUNCTION IF EXISTS getTeamInfo;
13 DROP FUNCTION IF EXISTS getOrgLogo;
14 DROP FUNCTION IF EXISTS getTeamPlayersRegion;
15 DROP FUNCTION IF EXISTS getTeamGameRegion;
16 DROP FUNCTION IF EXISTS getGameEventData;
17 DROP FUNCTION IF EXISTS getEventInfo;
18 DROP FUNCTION IF EXISTS getWinner;
19 DROP FUNCTION IF EXISTS getMatchData;
20 DROP FUNCTION IF EXISTS getMatchInfo;
21 DROP FUNCTION IF EXISTS getMatchWinner;
22 DROP FUNCTION IF EXISTS CheckPlayerExists;
23 DROP FUNCTION IF EXISTS CheckIGNRegistered;
24 DROP FUNCTION IF EXISTS CheckTeamExists;
25 DROP FUNCTION IF EXISTS CheckTeamExists;
26 DROP FUNCTION IF EXISTS CheckUserHasTeam;
27 DROP FUNCTION IF EXISTS CheckUserHasTeamJoinRequest;
28 DROP FUNCTION IF EXISTS getTeamMembers;
29 DROP FUNCTION IF EXISTS getTeamInfoFromUsername;
30 DROP FUNCTION IF EXISTS getTeamCaptain;
31 DROP FUNCTION IF EXISTS getExtendedPlayerInfo;
32 DROP FUNCTION IF EXISTS getExtendedTeamStaffInfo;
33 DROP FUNCTION IF EXISTS getNotificationsCount;
34 DROP FUNCTION IF EXISTS getTeamRequests;
35 DROP FUNCTION IF EXISTS getUserData;
36 DROP VIEW IF EXISTS getNewID

```

13. TRIGGERS

Não considerámos os triggers como algo que devêssemos focar muito.

Apesar de triggers serem úteis para encontrar erros no nível da base de dados (manter integridade dos dados) e terem a vantagem de poder ser utilizados por aplicações distintas sem integrar o código cada vez, para este contexto mais académico, foi mais simples fazer todo o processamento em C#. Não só porque estamos mais habituados a trabalhar com este tipo de linguagens, mas também porque nos permitia fazer debugging e refactoring do código mais facilmente.

Mesmo assim, decidimos criar um trigger, para mostrar que não estamos completamente ignorantes em relação a este objeto.

Para evitar que fossem inscritas mais equipas nos torneios do que o número máximo de equipas permitido, criamos um trigger que previne este cenário. Este trigger foi criado no ficheiro *triggers.sql* na pasta SQL.

```
GO
CREATE TRIGGER dontInsertIfTeamsFilled
ON PARTICIPATES_IN INSTEAD OF INSERT
AS
BEGIN
    DECLARE @count INT
    DECLARE @tournament VARCHAR(50)
    DECLARE @numberOfTeams INT

    SELECT @tournament = tournament FROM INSERTED

    SELECT @count = COUNT(*) FROM PARTICIPATES_IN WHERE @tournament = tournament
    SELECT @numberOfTeams = number_teams FROM TOURNAMENT WHERE @tournament = [name]
























    IF @count >= @numberOfTeams
        RAISERROR('Tournament slots are already filled.',16,1)
    ELSE
        INSERT INTO PARTICIPATES_IN SELECT * FROM INSERTED
END
GO
```

14. GERADORES

Todos os nossos geradores foram desenvolvidos em python, com o módulo open-source pyodbc. Implementámos um script para cada gerador que se limita a implementar as funções necessárias para criar os records relativos a certas tabelas.

Temos um script especial responsável por popular as tabelas, que executa essas funções criando e inserindo os records na base de dados.

Esta foi, sem dúvida, a parte mais trabalhosa do projeto. Procurámos gerar records que não só se adequassem ao tipo de dados das tabelas, mas que realmente fizessem sentido. Para além disso, tivemos de criar os nossos próprios datasets, que utilizámos para gerar os nomes das equipas, nomes de utilizador, que também foi uma tarefa bastante fatigante.

 countries	16/06/2022 20:24	Documento de tex...	5 KB
 CSVParserTemplate	21/06/2022 03:07	Python File	1 KB
 firstnames	16/06/2022 18:28	Documento de tex...	143 KB
 GenerateEvent	17/06/2022 16:58	Python File	6 KB
 GenerateEventStaff	13/06/2022 20:11	Python File	2 KB
 GenerateGame	13/06/2022 20:07	Python File	1 KB
 GenerateMatch	17/06/2022 18:13	Python File	6 KB
 GenerateOrganization	13/06/2022 12:16	Python File	7 KB
 GeneratePlayer	16/06/2022 19:11	Python File	10 KB
 GenerateTeam	16/06/2022 17:57	Python File	9 KB
 GenerateTeamCaptain	13/06/2022 16:36	Python File	1 KB
 GenerateTeamStaff	13/06/2022 02:45	Python File	3 KB
 GenerateUser	16/06/2022 17:39	Python File	4 KB
 GetSQLData	17/06/2022 16:35	Python File	2 KB
 lastnames	13/06/2022 01:25	Documento de tex...	162 KB
 orgs	16/06/2022 23:46	Documento de tex...	14 KB
 players	16/06/2022 18:56	Documento de tex...	428 KB
 PopulateTables	17/06/2022 18:14	Python File	5 KB
 remember	21/06/2022 03:07	Documento de tex...	1 513 KB
 teams	16/06/2022 18:18	Documento de tex...	65 KB
 TODO	13/06/2022 20:15	Ficheiro	1 KB
 tournaments	16/06/2022 23:26	Documento de tex...	26 KB
 users	13/06/2022 02:05	Documento de tex...	17 411 KB

15. NOTAS FINAIS

Neste projeto posemos à aprova as técnicas aprendidas nas aulas praticas durante o ano letivo, bem como os conceitos teóricos.

Devido à falta de tempo não conseguimos fazer tudo o que tínhamos proposto inicialmente e que pretendíamos fazer, como por exemplo, criar o form ‘MyOrganization’, e uma “screen” específica para cada TEAM, PLAYER, ORGANIZATION, etc.

O que iria abrir a possibilidade de retirar informação mais detalhada e dar uso a tabelas como STATs para verificar a performance dos jogadores em cada partida. Se isso tivesse sido feito, seria também interessante ter implementado o conceito de linkagem entre “screens” onde clicando na label da equipa de um PLAYER específico, o utilizador seria redirecionado para a “screen” dessa equipa.

São ideias para o futuro, que tornariam o trabalho mais interessante. Por agora, acho que o nosso trabalho foi bastante bem conseguido e implementámos funcionalidades e contextos suficientes para o tempo que tínhamos.