Relatório Técnico da Fase 2 do Projeto SoccerNow

0. Como Correr o Programa Todo

O primeiro passo é correr o Docker na pasta onde se encontram os ficheiros *setup.sh* e *run.sh,* dentro da pasta SoccerNow e não na pasta SoccerNowJFX.

O segundo passo depende do tipo de UI que se quer abrir:

1) *WebUI:* tem de se aceder ao http://localhost:8080/

2) *JavaFX UI:* tem de se correr o ficheiro Main.java presente na pasta do SoccerNowJFX

1. Web UI

Para desenhar a interface gráfica no *browser*, decidimos fazer algo simples mas que demonstrasse todos os filtros que implementámos bem como as opções de busca por Jogadores, Equipas e Campeonatos.

Página Inicial

Quando se acede ao http://localhost:8080/ , a primeira página que aparece é a *homePage.html*. Nessa página, é possível escolher que tipo do login é que se quer fazer, ou Jogador ou Árbitro, através de 2 botões.

Login do Jogador

Quando se carrega no botão para fazer Login como Jogador, acede-se ao *loginJog.html*, onde é necessário introduzir o nome de um Jogador que já tem de existir previamente e uma password qualquer, já que o Caso de Uso A pede apenas um mockup login.

Login do Árbitro

Quando se carrega no botão para fazer Login como Árbitro, acede-se ao *loginArb.html*, onde é necessário introduzir o nome de um Árbitro que já tem de existir previamente e uma password qualquer, já que o Caso de Uso A pede apenas um mockup login.

Página do Dashboard/Navigation

Quando se carrega no botão *Entrar* nos respetivos logins, acede-se à página *navigation_ dashboard.html*, que contém:

1) *Dashboard de Filtros:* cinco botões, cada uma para aceder aos diferentes tipos de filtros: de Jogadores, de Arbitros, de Equipas, de Jogos e de Campeonatos.

2) *Dashboard para Buscas*: três botões, cada um para aceder aos diferentes tipos de buscas: de Jogadores, de Equipas e de Campeonatos.

Páginas Provenientes dos Botões do Dashboard de Filtros

Cada botão vai dar a um html diferente:

1) *filtrosJogadores.html* : dá para filtrar por nome (seja o nome escrito em letra pequena ou letra grande), pela posição em que o Jogador joga, pelo número de Golos que já marcou, pelo número de Jogos em que participou e pelo número de Cartões que já levou.

> Cada botão leva a um mesmo html chamado *resultadosFiltroJogador.html*. Aí são mostrados os diferentes Jogadores que se enquadram no filtro. Caso nenhum se enquadrem, vai para uma página de erro chamada *erro.html*.

2) *filtrosArbitros.html* : dá para filtrar por nome (seja o nome escrito em letra pequena ou letra grande), pelo número de Jogos que já oficializaram e pelo número de Cartões que já deram.

> Cada botão leva a um mesmo html chamado *resultadosFiltroArbitro.html*. Aí são mostrados os diferentes Árbitros que se enquadram no filtro. Caso nenhum se enquadrem, vai para uma página de erro chamada *erro.html*.

3) *filtrosEquipas.html :* dá para filtrar por nome (seja o nome escrito em letra pequena ou letra grande), pelo número de Jogadores, pelo número de vitórias ou empates ou derrotas, pelo número de conquistas e pelas posições que não estão presentes nos Jogadores da Equipa.

> Cada botão leva a um mesmo html chamado *resultadosFiltroEquipa.html*. Aí são mostradas as diferentes Equipas que se enquadram no filtro. Caso nenhuma se enquadre, vai para uma página de erro chamada *erro.html*.

4) *filtrosJogos.html* : dá para filtrar por jogos já realizados (ou seja, acabados), por jogos a serem realizados (ou seja, começados), pela quantidade de jogos, pela localização e pelo turno do jogo (se for de manhã, é realizado das 6h-11h; se for de tarde é realizado das 12h às 17h; e se for de noite, é realizado das 18h às 23h).

> Cada botão leva a um mesmo html chamado *resultadosFiltroJogo.html*. Aí são mostradas os diferentes Jogos que se enquadram no filtro. Caso nenhuma se enquadre, vai para uma página de erro chamada *erro.html*.

5) *filtrosCampeonatos.html* : dá para filtrar por nome, por equipa, por número de jogos realizados e por número de jogos a realizar.

> Cada botão leva a um mesmo html chamado *resultadosFiltroCampeonato.html*. Aí são mostradas os diferentes Campeonatos que se enquadram no filtro. Caso nenhuma se enquadre, vai para uma página de erro chamada *erro.html*.

Em cada uma destas cinco páginas há ainda também um botão *Voltar*, para poder visitar o Dashboard de Filtros e o Dashboard para Buscas de novo.

Páginas Provenientes dos Botões do Dashboard para Buscas

Cada botão vai dar a um html diferente:

1) *buscasJogadores.html* : neste momento, nesta página, é feita uma procura pelo NIF de um Jogador específico.

> Cada botão leva a um mesmo html chamado *resultadosBuscaJogador.html*. Aí é mostrado o Jogador que se enquadra na busca. Caso nenhuma se enquadre, vai para uma página de erro chamada *erro.html*.
>
> Nesta página, também é possível ter acesso a uma lista de todos os Jogadores.

2) *buscasEquipas.html* : neste momento, nesta página, é feita uma procura pelo nome de uma Equipa específica.

> Cada botão leva a um mesmo html chamado *resultadosBuscaEquipa.html*. Aí são mostradas as diferentes Equipas que se enquadram na busca. Caso nenhuma se enquadre, vai para uma página de erro chamada *erro.html*.
>
> Nesta página, também é possível ter acesso a uma lista de todas as Equipas.

3) *buscasCampeonatos.html* :

> Cada botão leva a um mesmo html chamado *resultadosBuscaCampeonato.html*. Aí são mostradas os diferentes Campeonatos que se enquadram na busca. Caso nenhuma se enquadre, vai para uma página de erro chamada *erro.html*.
>
> Nesta página, também é possível ter acesso a uma lista de todos os Campeonatos.

Em termos de Controllers, temos 6 WebControllers:

1) *HomeController*, que encaminha os Utilizadores para os diferentes logins, antes de entrar na aplicação propriamente dita.

2) *JogadorWebController*, que faz o login de um Jogador específico.

3) *ArbitroWebController*, que faz o login de um Árbitro específico.

4) *FiltrosWebController*, que vai fazer o controlo e distribuição de todos os *endpoints* para todos os filtros. Esta classe comunica com os diferentes handlers necessários para responder aos pedidos de filtros, que depois posteriormente comunicam com os respetivos repositórios. Alguns dos filtros são totalmente feitos a partir de *queries* nos repositórios, enquanto outros têm uma parte feita a partir do repositório e outra completada nos handlers.

5) *BuscasWebController*, que vai fazer o controlo e distribuição de todos os *endpoints* para todas as buscas.

6) *UpdateJogoWebController*, que vai realizar a o update do resultado final de um determinado Jogo.

Foi criado ainda um ficheiro *messages.properties* para ser usado para a internacionalização dos textos usados na WebUI, facilitando a tradução de todo o texto caso seja necessário.

Implementação

Quando filtramos o Jogador ou o Árbitro ou a Equipa ou o Campeonato por nome, não há distinção entre letras maiúsculas e minúsculas, ou seja, procurar por um Jogador chamado João dará os mesmo resultados caso se escreva 'João' ou 'joão'.

Na parte dos filtros do Jogo, há algumas considerações a ter em relação a dois deles:

-> *filtro por Jogos realizados:* considerámos jogos realizados aqueles que têm como status ACABADO

-> *filtro por Jogos a serem realizados:* considerámos jogos ainda por realizar aquele que têm como status COMECADO

Utilizámos este mesmo princípio para os seguintes filtros dos Campeonatos: *filtros por número de Jogos a realizar* e *filtros por número de Jogos realizados.*

Nos filtros dos Jogos, se procurarmos por um Jogo de manhã, o programa retorna jogos entre as 6 horas da manhã e as 11 horas; se procurarmos por um Jogo à tarde, o programa retorna jogos entre o meio-dia e as 17 horas da tarde; e, se procurarmos por um Jogo à noite, o programa retorna jogos entre as 18 horas e as 23 horas da noite.

Nos filtros dos Campeonatos, supomos que o segundo ("Filtro por por equipa"), queria dizer "Filtro por nome de equipa".

As buscas devolvem Jogadores, Equipas e Campeonatos mais pormenorizados do que os filtros correspondentes. Nessas páginas html há um botão para ter acesso à lista completa de Jogadores, Equipas e Campeonatos.

Testes após Implementação

Para verificar se todos os filtros e buscas implementadas funcionavam como deviam, foram testados os casos limites na própria interface, de forma a garantir que todas as possíveis falhas devolvem uma página de erro com o botão "Voltar".

Para testar os filtros dos Campeonatos, não podem ser usados os Jogos que já existem (feitos a partir de testes) quando se inicia o *SoccerNowApplication*, uma vez que esses Jogos são amigáveis, já que temos definida diferenças entre os Jogos que são amigáveis e os que são de campeonato:

1) Para ser um jogo de campeonato, o Árbitro tem de ter certificado –apesar de Jogos amigáveis também poderem ter um Árbitro com certificado

2) Os Jogos com o campo para o nome de Campeonato preenchido serão automaticamente reconhecidos como jogos de campeonato, enquanto os que tiverem esse campo a *null* serão reconhecidos como jogos amigáveis.

NOTA: O Caso de Uso I não foi implementado na parte dos filtros por falta de tempo e quando se carrega no botão para executar o updateFinalScore, o utilizador é levado para uma página html *naoImplementado.html*.

2. JavaFX UI

De forma a conseguir criar uma aplicação nativa, tivemos de criar um novo projeto fora do container, onde já estava o SoccerNow, que comunicasse com este.

Ao entrar na aplicação, temos três tipos de logins possíveis: Jogador, Árbitro e Organizador. No *HomeController*, é possível fazer estes tipos de logins.

Depois quando entramos em cada login, há uma série de opções (casos de uso) que podemos tomar, em três grandes grupos:

1) optionsJogador, em que no *optionsJogadorController*, o jogador com login feito pode ver a lista de Jogadores, a lista de Árbitros, a lista de Equipas, a lista de Jogos e a lista de Campeonatos. Depois também pode dar update às suas informações e pode adicionar ou remover a sua própria participação numa determinada equipa. Por fim, consegue dar delete a ele próprio (apagar o seu perfil).

2) optionsArbitro, em que no *optionsArbitroController,* o árbitro com login feito pode ver a lista de Jogadores, a lista de Árbitros, a lista de Equipas, a lista de Jogos e a lista de Campeonatos. Depois também pode dar update às suas informações e colocar o finalScore de cada Jogo. Por fim, consegue dar delete a ele próprio (apagar o seu perfil).

3) optionsOrganizador, em que no *optionsOrganizadorController,* o organizador pode ver a lista de Jogadores, a lista de Árbitros, a lista de Equipas, a lista de Jogos e a lista de Campeonatos. Pode também criar/registrar Jogadores, Árbitros, Equipas, Jogos e Campeonatos. Da mesma forma, pode atualizar Equipas e Campeonatos e pode atualizar Equipas aos Campeonatos. Por fim, também pode dar delete a um Jogador, a um Árbitro, uma Equipa, um Jogo ou um Campeonato (soft deleted).

Nesta aplicação, copiámos os enumerados do lado da aplicação principal com toda a lógica de negócio. Da mesma forma, fizemos o mesmo com os dtos, para facilitar a implementação. De resto, existe ainda uma pasta para todos os controllers, outra para os models e outra para os dtos.

Implementação

Para fazer login do Jogador ou do Arbitro, a classe MemoryLogin age como uma cache para guardar os dados do Utilizador que fez login. Esta classe é um singleton durante toda a execução do programa. Cada vez que alguém clica para fazer login enquanto a aplicação está a correr, a "cache" é limpa e faz reset.

Quando estamos a criar um Jogo, tem de se obrigatoriamente escolher as Equipas primeiro e depois dentro disso é que se escolhe os Jogadores de cada uma. Quando são escolhidos os Jogadores para a Equipa efetiva de Jogo 1, se estes estiverem presentes na segunda Equipa, já não podem ser escolhidos para a Equipa efetiva de Jogo 2 (sendo automaticamente retirados).

No *edge* case em que não há Jogadores suficientes para selecionar para a Equipa efetiva de Jogo 2, tem de se escolher uma segunda Equipa e subsequentemente voltar a escolher os Jogadores da Equipa efetiva de Jogo 1 – esses Jogadores não são desseleccionados, isso apenas é para voltar a haver o filtro para os Jogadores da segunda Equipa que podem ser escolhidos para a Equipa efetiva de Jogo 2.

Coisas a ter consideração quando se está a mexer na aplicação

Quando se querem selecionar elementos de uma lista, temos de segurar o CTRL e clicar em cada elemento que queremos adicionar (exemplo: para selecionar Jogadores para uma Equipa).

3. Casos de Uso dos Campeonatos

Controllers, Handlers, DTOs e Repositories

Desta vez, para complementar tudo o que já foi implementado, acabámos de implementar o *CampeonatoHandler* com todos os métodos que necessita para responder aos casos de uso da primeira fase.

Complementámos o *CampeonatoRepository* com as *queries* necessárias, tal como o *CampeonatoController* com os endpoints necessários para responder aos pedidos feitos. Da mesma forma, fizemos o *CampeonatoDto*.

Implementação

Quando criamos um Campeonato, temos obrigatoriamente de preencher dois atributos: ano e nome, sendo que ano não pode ser zero e o nome não pode ser null nem repetido.

Quando criamos um Campeonato, se tivermos equipas repetidas na lista, o código adiciona a equipa uma vez e ignora as repetições. Se receber o nome de uma equipa que não existe, salta essa à frente. O mesmo acontece com uma equipa que foi deleted (soft deleted).

Implementámos também o getAll dos Campeonatos e o getHistórico, que vai buscar todas as linhas EstatisticasEquipa de todos os Campeonatos em que uma dada Equipa participa ou já participou.

O update do Campeonato só permite dar update ao nome e ao ano. Para adicionar ou remover equipas, há funções próprias.

Foram implementadas também funções que vão buscar todas as equipas de um campeonato, vão buscar todos os campeonatos ou também um campeonato específico. Da mesma forma, criámos também duas funções que se relacionam com a tabela de pontos do campeonato: uma vai buscar uma List<EstatisticasEquipa> vem com toda a informação e por ordem de mais pontos para menos pontos, e a segunda vai representar essa lista em string para uma leitura mais fácil e para uso na UI.

O delete de um Campeonato é feito com base em não ter jogos marcados.

Quando um Campeonato cria os seus jogos, cria logo todos de uma vez. Baseámos a nossa implementação de criação de jogos do Campeonato no mundo real, onde antes do início da temporada, todos os jogos de um Campeonato específico são marcados. Desta forma, um Campeonato só pode ser deleted quando acabou todos os Jogos (ou seja, todos os Jogos têm como status ACABADO ou CANCELADO) ou antes de efetuar a marcação dos mesmos.

Agora, por causa de todas as alterações que fizemos com a introdução da lógica do Campeonato, ao criar um Jogo, caso não especifiquemos o nome do campeonato em que se vai inserir, o Jogo vai ser amigável. Caso contrário, se for Jogo de campeonato (ou seja, é especificado o nome do campeonato), um dos árbitros tem de ter certificado. É de mencionar ainda que para um Campeonato funcionar, têm de estar presentes no mínimo 8 equipas.

Da mesma forma, agora, ao fazer o updateFinalScore de um Jogo, a lógica de negócio verifica se o Jogo é de Campeonato. Se sim, ele adiciona todas as estatísticas do Jogo à EstatisticasEquipa da Equipa referente a esse Campeonato específico. Após isso, verifica o resultado, que tem de ser da forma X-Y, e muda o número de vitórias, empates e derrotas e o número de pontos de cada Equipa.

API Rest e Testes ao Código

Por ser algo bastante complexo, todos os testes foram maioritariamente feitos no Swagger e nas UI, de forma a tentar encontrar bugs ou erros maiores que impeçam o normal funcionamento da aplicação.

Foi extensivamente testada a criação de Campeonatos bem como a adição e remoção de equipas no mesmo. Também foi testada o delete do Campeonato, tendo sido testados todos os edge cases que achámos relevantes.

As classes que desenvolvemos na primeira fase do projeto das quais o Campeonato era sempre referenciado a *NULL* foram mudadas, de forma a puderem suportar as ligações com

as classes desenvolvidas na segunda parte do projeto. Tudo o que sofreu alterações foi *retested*, tanto pelo código de testes já escrito na fase 1 do projeto como pelo grupo, a partir do Swagger.

Os testes na classe Jogo foram os mais extensivos, com foco na criação e registo do resultado final, tendo resultado em longos períodos de *bug fixing*, devido a bugs encontrados em *edge* cases.

4. Gravação do Vídeo

Para gravarmos o vídeo obrigatório, primeiro determinámos todos os casos de uso e *edge* cases previamente. Desta forma, abaixo está tudo o que é testado no vídeo em termos de *edge* cases tanto no JavaFXUI como na WebUI:

Caso de Uso A) Testamos o login de um Jogador com um que não existe, testamos o login de Arbitro no login de um Jogador, testamos o login de um Jogador no login de um Arbitro, e testamos o login de um Arbitro que não existe. → Funcional tanto na parte da WebUI como na parte do JavaFXUI

Caso de Uso B) Testamos criar um Jogador/Arbitro com atributos necessários a NULL (por exemplo, o NIF), testamos criar um Jogador/Arbitro com NIF que já existe, testamos criar um Jogador com o NIF de um Arbitro que já existe, e testamos criar um Arbitro com o NIF de um Jogador que já existe.

Caso de Uso C) Testamos fazer Get de um Jogador/Arbitro que foi deleted, testamos fazer Get de um Jogador/Arbitro que não existe, testamos fazer Delete de um Jogador/Arbitro que não existe, e testamos fazer Update de um Jogador/Arbitro que não existe.

Caso de Uso D) Verificamos que um Get All de Jogador/Arbitro não retorna nenhum que tenha sido deleted, e verificamos que o Get e Get All de um Jogador/Arbitro que foi Updated com sucesso está igual ao esperado.

Caso de Uso E) Testamos criar uma Equipa com um nome igual a uma que já existe, e testamos criar uma Equipa com atributos necessários a NULL (neste caso o nome).

Caso de Uso F) Testamos fazer Get de uma Equipa que não existe, testamos tentar fazer Get de uma Equipa que foi Deleted, testamos tentar fazer Delete a uma Equipa com Jogos marcados, testamos remover e adicionar Jogadores (que não existam/que já estejam na equipa), verificamos que a Equipa fica como esperada após adicionar/remover um Jogador, e testamos fazer o Get dos seus Jogos/Histórico/Campeonatos.

Caso de Uso G) Verificamos que um Get All não retorna nenhuma equipa que tenha sido deleted.

Caso de Uso H) Testamos criar um Jogo com atributos necessários a NULL (por exemplo, a localização), testamos criar um Jogo com uma Equipa que não existe/com um Jogador que não existe/que não pertence à equipa, testamos criar um Jogo de Campeonato com um

Campeonato com menos de 8 equipas, testamos criar um Jogo de Campeonato sem um Arbitro certificado, e testamos criar um Jogo com um Jogador igual em ambas as Equipas.

Caso de Uso I) Testamos registar o resultado de um Jogo amigável, testamos registar o resultado de um Jogo que não existe, e testamos registar o resultado com atributos necessários a NULL (por exemplo, o placard). → Apenas implementado na parte do JavaFX.

Caso de Uso J) Testamos criar um Campeonato com um atributo necessário a NULL (por exemplo, o nome), e testamos criar um Campeonato com um nome que já existe.

Caso de Uso K) Testamos o Get de um Campeonato que não existe, e verificamos que Campeonatos Deleted não aparecem nos Gets.

Caso de Uso L) Testamos mudar o nome e ano do Campeonato, e testamos Adicionar/Remover Equipas que existem/não existem/que pertencem/não pertencem ao Campeonato.

Caso de Uso M) Verificamos que o Cancelamento de qualquer jogo COMECADO funciona.

O resto dos *Casos de Uso (N, O, P, Q, R)* pertence à parte dos filtros, ou seja, exclusivamente da parte da WebUI e todos os testes. Aqui, testamos, para além de quando os filtros correm bem, quando não há nada escrito ou quando é escrito algo mal.

Para além de todos estes *edge* cases, também testamos quando as criações, os updates, as verificações, os deletes e os filtros correm bem e de forma esperada.

Vídeos

Foram gravados dois vídeos: o *MainVideo_Grupo17* e o *SecondaryVideo_Grupo17* na pasta *Vídeos_Grupo17*, dentro da pasta *docs,* a demonstrar todos os *edge cases* e os casos funcionais descritos em cima.

O segundo vídeo foi gravado para lidar com alguns erros inesperados que aconteceram durante a gravação do vídeo principal: criação de um campeonato (para falhar e mostrar a mensagem de erro quando os campos necessários não estão preenchidos), criação de uma equipa (para falhar e mostrar a mensagem de erro quando o campo necessário não está preenchido) e um dos filtros em que aparece uma página em branco quando não devia. Está ainda demonstrado que um Jogador se pode remover ou adicionar a uma Equipa específica, algo que nos esquecemos de demonstrar no primeiro vídeo.

O *MainVideo_Grupo17* tem cerca de 31 minutos e o *SecondaryVideo_Grupo17* tem cerca de 3 minutos (não têm áudio, uma vez que tudo o que foi testado foi descrito em cima).

NOTA FINAL #1: Durante a gravação do vídeo, o boolean popular está a *true* e o boolean testar está a *false*.

NOTA FINAL #2: O trabalho entregue também tem os valores dos booleans desta forma.