Relatório Técnico da Fase 1 do Projeto SoccerNow

1. Definição de Entidades

Nesta fase do projeto foi-nos pedido que desenvolvêssemos um Modelo de Domínio e um Diagrama de Classes que representasse uma solução para a aplicação do SoccerNow. Uma vez que esta é uma aplicação com lógica de negócio complexa, acordámos que o *Domain Model* seria o Padrão da Camada de Negócio que mais se adequaria ao nosso projeto.

Desta forma, definimos que teríamos como Entidades as seguintes classes:

▷ Classe *Utilizador*, uma classe abstrata que vai ser composta por apenas quatro atributos, o nome, a password, o número de jogos em que cada utilizador participou e o NIF.

▷ Classe *Jogador*, uma das classes concretas que estendem a Classe *Utilizador*, que representa um único jogador.

▷ Classe Árbitro, uma das classes concretas que estendem a Classe *Utilizador*, que representa um único árbitro.

▷ Classe *Campeonato*, uma classe abstrata que vai ser composta por alguns atributos como o nome, o ano, entre outros, que representa um único campeonato.

▷ Classe *CampeonatoPontos*, uma das classes concretas que estende a Classe *Campeonato*, que representa um único campeonato feito por pontos e não por eliminatórias.

▷ Classe *CampeonatoEliminatorias*, uma das classes concretas que estende a Classe *Campeonato*, que representa um único campeonato feito a partir de eliminatórias e não por pontos.

▷ Classe *Jogo*, uma classe concreta, que representa um único *Jogo* e tudo o que isso envolve, desde árbitros, jogadores, equipas a jogar, entre outros.

▷ Classe *Equipa*, uma classe concreta, que representa uma única *Equipa* com tudo o que isso envolve, desde nome, n jogadores participantes, entre outros.

▷ Classe *EstatísticasEquipa*, uma classe concreta, que representa as estatísticas de cada *Equipa* num *Campeonato* específico.

▷ Classe *EquipaJogo*, uma classe concreta, que representa uma equipa de 5 jogadores (e tudo o que ela envolve) que vai jogar num determinado *Jogo*.

▷ Classe *Estatísticas*, uma classe abstrata, que poderá ser estendida pelos vários tipos de *Estatísticas*: *Golo*, *CartaoVermelho* e *CartaoAmarelo*.

▷ Classe *Golo*, uma classe concreta que estende a Classe *Estatísticas*, que representa um golo e tudo o que este envolve, desde o Jogador que a deu, a equipa para que está a jogar, o jogo em que ocorreu, etc.

▷ Classe *CartãoVermelho*, uma classe concreta que estende a Classe *Estatísticas*, que representa um cartão vermelho e tudo o que este envolve.

▷ Classe *CartãoAmarelo*, uma classe concreta que estende a *Estatísticas*, que representa um cartão vermelho e tudo o que este envolve.

Também acordámos que teríamos dois novos tipos de dados:

▷ Enumerado *Posição*, que representará todas as posições de jogadores possíveis: GK – Guarda-Redes (GoalKeeper), DEF – Defesa, MED – Médio e ATQ – Atacante.

▷ Enumerado *Status*, que representará todos os estados possíveis de um jogo: Acabado, Começado e Cancelado.

▷ Enumerado *EstatisticaTipo*, que representará todos os tipos de estatísticas possíveis: Golo, CartaoAmarelo e CartaoVermelho.

▷ Enumerado *EstadoEntidade*, que representará se o Jogador/Árbitro/Equipa já foi soft deleted ou não: ATIVO, se não tiver sido soft deleted, e INATIVO se tiver sido soft deleted.

2. Suposições que tomámos segundo o enunciado

Após comunicarmos as nossas dúvidas sobre o enunciado ao professor, iremos tomar as seguintes opções acerca do projeto:

▷ Supomos que o Organizador dos jogos e campeonatos seremos nós enquanto grupo por agora.

▷ Estabelecemos que é o Árbitro quem vai colocar o resultado e as estatísticas de cada Jogo.

▷ Prosseguiremos com o sistema de pontos como se de futebol se tratasse: 0 pontos para uma equipa que perca, 1 ponto para ambas as equipas quando empatam e 3 pontos para uma equipa que ganhe.

▷ Um Jogador que prefira ser guarda-redes pode não chegar a ser guarda-redes num determinado jogo porque o Organizador é que define as posições dos jogadores de uma determinada equipa num determinado jogo.

▷ Um Jogador não pode ser Arbitro e vice-versa.

3. Relações entre as diferentes entidades e atributos

▷ Nas classes que mapeámos para entidades também é possível verificar a anotação *@Id*, que faz com que todos os objetos inseridos nas bases de dados tenham a sua própria chave primária.

▷ Nas classes que mapeámos para entidades também é possível verificar a anotação *@Version* num atributo com o mesmo nome, uma vez que usamos Optimistic Offline Locking.

▷ Na classe *Jogador* é usada a anotação *@Enumerated*, que faz com que o enumerado *Posição* tome a forma de uma string quando introduzido na tabela dos jogadores. Já na classe *Jogo*, a anotação *@Enumerated* vai fazer com que o enumerado *Status* tome a forma de um número.

▷ Na classe abstrata *Utilizador*, vamos ter os atributos id, nome, password, numJogos e NIF.

▷ Na classe concreta *Jogador*, que estende *Utilizador*, vamos ter os atributos posição e equipas (lista de equipas a que o jogador pertence), estando esta última mapeada com *@ManyToMany*, já que um jogador pode estar em várias equipas e uma equipa pode ter vários jogadores.

▷ Na classe concreta *Arbitro*, vamos ter o boolean certificado.

▷ Na classe concreta *Equipa*, vamos ter os atributos id, nome, historicoJogos (lista mapeada com uma relação *@ManyToMany* com *Jogo*, já que cada jogo tem mais que uma equipa), historicoResultados (lista mapeada com uma relação *@OneToMany*(mappedBy = "equipa") com *EstatísticasEquipa*, já que cada equipa pode ter várias estatísticas de campeonato), jogadores (lista mapeada com uma relação *@ManyToMany*(mappedBy = "equipas" com *Jogador*, já explicada anteriormente), e campeonatos (lista mapeada com uma relação *@ManyToMany* com *Campeonato*, já que uma equipa pode participar em vários campeonatos e vice-versa).

▷ Na classe abstrata *Campeonato*, vamos ter os atributos id, nome, ano e equipas (lista de equipas presente num campeonato mapeada com uma relação *@ManyToMany*(mappedBy = "campeonatos") com *Equipa*, uma relação explicada anteriormente).

▷ Na classe concreta *CampeonatoPontos*, estendida de *Campeonato*, vamos ter o atributo tabela (lista de *EstatísticasEquipa* mapeada com a relação *@OneToMany*(mappedBy = "campeonato") com EstatísticasEquipa, já que um campeonato tem várias estatísticas de cada equipa).

▷ Na classe abstrata *Estatísticas*, vamos ter os atributos id, nome, jogador (mapeado com uma relação *@OneToOne* com *Jogador*, já que cada estatística está associada a um jogador específico), equipa (mapeado com uma relação *@OneToOne* com *Equipa*, já que cada estatística tem associada o jogador de uma equipa particular), campeonato (mapeado com uma relação *@OneToOne* com *Campetonato*, já que cada estatística específica foi feita num campeonato específico), e jogo (mapeado com uma relação *@OneToOne* com *Jogo*, já que cada estatística específica acontece num jogo específico).

▷ Nas classes concretas *CartaoAmarelo* e *CartaoVermelho,* que estendem a classe *Estatísticas*, vamos ter um atributo arbitro, mapeado com uma relação *@OneToOne*, já que cada uma destas estatísticas é dada por um árbitro.

▷ A classe concreta *Golo,* que estende a classe *Estatísticas*, não tem atributos

▷ Na classe concreta *EstatísticasEquipa*, vamos ter os atributos id, pontos, numVitorias, numEmpates, numDerrotas, equipa (a que se refere as estatísticas, mapeada com uma relação *@ManyToOne* com *Equipa*, já que uma estatística está ligada a uma equipa),

campeonato (a que se referem as várias estatísticas das várias equipas, mapeada com uma relação *@ManyToOne* com Campeonato), e estatísticas (lista de estatísticas detalhada de cada equipa, mapeada com uma relação *@OneToMany* com Estatisticas).

▷ Na classe concreta Jogo, vamos ter os atributos id, data, horário, localização, placarFinal, status (do jogo), campeonato (mapeado através uma relação *@OneToOne* com *Campeonato*, já que um determinado jogo pertence a um determinado campeonato), equipas (lista de equipas presentes no jogo, mapeado por uma relação *@OneToMany*(mappedBy = "jogo") com *EquipaJogo*, já que um jogo tem duas equipas), arbitroPrincipal (mapeado por uma relação *@OneToOne* com *Arbitro*, já que cada jogo precisa de um árbitro principal), árbitros (lista de todos os árbitros presentes em cada jogo, mapeada por uma relação *@ManyToMany* com *Arbitro*), vencedor (a equipa vencedora da partida, mapeada com uma relação *@OneToOne* com *Equipa*, já que cada jogo tem apenas um vencedor), e estatísticas (lista de estatísticas detalhada de cada jogo, mapeada com uma relação *@OneToMany* (mappedBy = "jogo") com *Estatísticas*).

▷ Na classe concreta *EquipaJogo*, vamos ter os atributos id, equipa (que representam as equipas de 5 jogadores efetivamente em campo, através de uma relação *@OneToOne* com *Equipa*), jogadores (lista de jogadores presentes em cada equipa efetiva de cada jogo, através de uma relação de *@ManyToMany* com *Jogador*, uma vez que várias *EquipasJogo* têm vários jogadores e vice versa), gk (guarda-redes, um jogador específico, presente em cada *EquipaJogo*, através de uma relação *@OneToOne* com *Jogador*, já que cada equipa efetiva em campo tem apenas um guarda-redes), e jogo (já que cada uma destas equipas efetivas pode ser titular num jogo específico, mapeado através da relação *@ManyToOne* com *Jogo*).

4. Herança entre diversas classes/entidades

Neste projeto, também implementámos vários exemplos de herança, através da anotação *@Inheritance*, mais especificamente:

▷ SINGLE_TABLE

1) Entre as classes que são estendidas da classe *Utilizador* e esta mesma classe, uma vez que achámos mais benéfico ter tanto *Jogador* como *Arbitro* numa única tabela, com mais uma coluna a indicar o seu tipo, porque conseguimos com mais facilidade aceder aos seus atributos.
2) Entre as classes que são estendidas da classe *Estatísticas* e esta mesma classe uma vez que, como quase todos os atributos são partilhados entre todas as classes estendidas –*Golo*, *CartaoVermelho* e *CartaoAmarelo*-, então a tabela não ficaria com muitos NULLs, poupando espaço e sendo mais fácil aceder aos atributos de cada linha.

▷ TABLE_PER_CLASS

1) Entre as classes que são estendidas pela classe *Campeonato* e esta mesma classe, uma vez que achámos daria um acesso mais rápido aos atributos mas também um uso mais eficiente de espaço. Como nos foi pedido para preparar o sistema para um segundo possível tipo de campeonato, não sabemos ao certo o quanto os atributos de *CampeonatoPontos* e *CampeonatoEliminatorias* vão diferir. Por isso, em vez de termos uma única tabela com possivelmente muitos NULLs, decidimos ter uma tabela por cada tipo de campeonato.

5. Controllers, Handlers, DTOs e Repositories

Para responder às questões sobre Utilizadores, temos um único *UtilizadorHandler* e um único *UtilizadorRepository*. Contudo, temos dois controllers –*JogadorController* e *ArbitroController*-, de modo a fazer a separação entre as operações de *Jogador* e de *Arbitro* e ser mais fácil visualizar os casos de uso pedidos no enunciado. Da mesma forma, temos um DTO separado para Jogador e outro para Arbitro.

Para já, na Fase 1 do Projeto, ainda só temos implementados o *EquipaController*, o *JogoController* e o *EquipaJogoController*, além dos controllers para Utilizadores. Uma vez que os Casos de Uso referentes aos Campeonatos estão na Fase 2 do Projeto, decidimos ainda não implementar o *CampeonatoController*.

Contudo, já começámos a implementar o *CampeonatoHandler*, para conseguir comunicar com outros handlers que também já implementámos para além do *UtilizadorHandler*: *EquipaHandler*, *EquipaJogoHandler*, *EstatísticasHandler* e *JogoHandler*.

Também já implementámos algumas queries em alguns dos repositórios que já criámos, nomeadamente em: *UtilizadorRepository* e *EquipaRepository*. (Além destes, foram criados o *EquipaJogoRepository*, o *CampeonatoRepository, EstatisticasRepository* e *JogoRepository*.)

Em termos de DTOs, além do *JogadorDto* e do *ArbitroDto*, também criámos o *EquipaDto*, *JogoDto*, *EquipaJogoDto*, *EstatisticasDto* de modo a melhorar o transporte de informações através da API REST.

No *JogadorDto* e no *ArbitroDto*, todos os campos têm de estar preenchidos no endpoint *create* (à exceção do id, como seria de esperar, já que é preenchido pela base de dados). No *update*, temos de preencher todos os campos novamente. O que fazer quanto ao *EquipaDto* e ao *JogoDto* será explicado durante o ponto 6.

6. REST

Para darmos o *update* em tanto os Árbitros como os Jogadores, em vez de utilizarmos o verbo PATCH, utilizámos o verbo PUT para trazer mais escalabilidade ao processo e também porque estamos a planear já uma futuro implementação de UI onde os campos do *update* são automaticamente preenchidos pelos campos atuais.

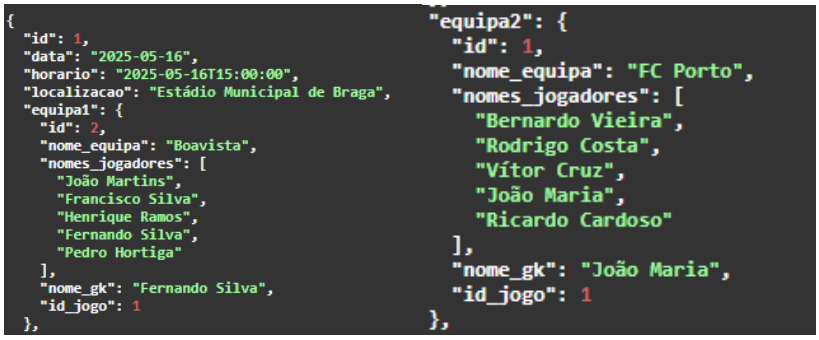
Para além dos casos de uso presentes no enunciado, criámos mais alguns *endpoints* adicionais:

▷ Para auxiliar a realização de alguns testes e complementar os casos de uso referentes aos utilizadores e às equipas, criámos os "getAll" para os Jogadores, Arbitros e Equipas. Para além disto, criámos um endpoint que adicione uma Equipa a um Jogador específico e outro que remova uma Equipa de um Jogador específico.

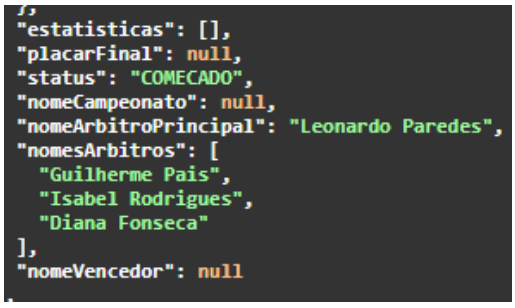
▷ Já no JogoController, com o mesmo propósito, foi feito um endpoint "getAll".

▷ Criámos endpoints também para mostrar que é possível responder às perguntas extras colocadas no enunciado, a que chamámos casos de uso extra aquando dos commits para o GitLab.

▷ Quando fazemos o createJogo, temos de preencher tudo o que está nas duas primeiras fotos abaixo (menos tudo o que envolva ids, visto que a base de dados faz esse trabalho:



As Estatísticas têm de permanecer a NULL. Depois também tem de ser preenchido pelo menos o Árbitro Principal, mesmo que o campo da lista de árbitros fique a NULL.



▷ Aquando da utilização do PATCH no Swagger com o endpoint /updatefinalscore:

-o campo com o Campeonato tem de ficar a NULL

-o campo do vencedor tem de ser preenchido com o nome da Equipa vencedora (ou "EMPATE", caso haja um)

-o placarFinal tem de ser preenchido com, por exemplo, "2-1"

-caso não queiram que nenhum GOLO/CARTAOAMARELO/CARTAOVERMELHO seja registado, devem apagar as {} com tudo no seu interior do campo as estatísticas

▷ Durante o PATCH do updateJogo, podemos mudar a data, hora, localização e árbitros. É de mencionar que as duas equipas escolhidas não podem ser mudados mas é possível mudar os jogadores escolhidos para o jogo através do *update* EquipaJogo.

▷ Durante a criação de uma equipa no *endpoint* /createEquipa, a lista de Jogadores pode ser inicializada a NULL.

7. Implementação e Testes

O terminal do bash / linha de comandos demora bastante tempo a demonstrar que a base de dados está pronta para usar, por isso, é melhor ver pelo terminal do Docker, que indica mais rapidamente o momento em que a base de dados acaba de ser populada e em que a aplicação fica pronta a usar.

Em relação a testes: a alguns deles foram feitos através do Swagger mas outros também foram feitos na classe *SoccerNowApplication*, o que nos auxiliou a verificar vários bugs ao longo da nossa implementação.

Para facilitar, na parte dos testes na classe *SoccerNowApplication*, decidimos popular a base de dados com tudo o que achámos importante, desde Equipas, a Jogadores, a Jogos marcados e a Árbitros. Quando inicializamos a execução do programa, as passwords de cada *Utilizador* são geradas automaticamente, bem como as posições em que os Jogadores jogam. É de mencionar também que enquanto a base de dados é populada, são feitos testes para verificar se o create retorna a resposta esperada.

Nota Importante: Na classe SoccerNowApplication temos neste momento dois booleans:

→ popular: se se quiser popular a base de dados ao correr a aplicação, deve-se colocar este boolean a *true*, mas se se quiser inicializar uma base de dados completamente vazia, deve ficar a *false*.

→ testar: se se quiser correr os testes ao correr a aplicação, deve-se colocar este boolean a *true*, caso contrário, deve-se colocar a *false*.

Nota importante: Estes dois booleans **não podem ser verdadeiros ao mesmo tempo**, uma vez que o *testar* assume uma base de dados vazia no início dos testes que são hardcoded.

```java
@Bean
@Transactional
public CommandLineRunner demo() {
    return (args) -> {
        System.out.println("\n" + "do some sanity tests here" + "\n");

        //Set to TRUE if you want to Populate the DB upon running the app
        Boolean popular = false;

        //Set to TRUE if you want to run tests upon running the app
        Boolean testar = true;
```

(Para ser fiel ao que é pedido no enunciado, começamos com o boolean testar = true, uma vez que um dos pontos avaliados são os testes. ➔ Por causa do atributo testar, mesmo com o popular a falso, a base de dados vai estar populada com 2 Jogadores, 2 Árbitros e 2 Equipas que foram soft deleted.)

```
-----STARTING TESTS-----


Starting Tests Related to Arbitro


Let's Start by Creating a Valid Arbitro

Hibernate: select u1_0.id,u1_0.dtype,u1_0.nif,u1_0.nome,u1_0.num_jogos,u1_0.passwo
u1_0 where u1_0.nif=?
Hibernate: select nextval('utilizador_seq')
Hibernate: insert into utilizador (nif, nome, num_jogos, password, version, certif

Check if the Arbitro was Successfuly Created: true


Now Let's Try to Create a New Arbitro With the Same NIF

Hibernate: select u1_0.id,u1_0.dtype,u1_0.nif,u1_0.nome,u1_0.num_jogos,u1_0.passwo
u1_0 where u1_0.nif=?

Creating Not Done, Since Two Utilizadores Can't Have the Same NIF: true
```

Figura 1 - Exemplo dos testes que foram implementados

Nestes testes, estão incluídos testes sobre os dois tipos de Utilizadores e sobre a Equipa. Contudo, não temos testes implementados no código em si para o Jogo (mas testámos no Swagger manualmente se tudo no Jogo funcionava). **Mais aprofundadamente**, testámos a criação e o delete de Jogadores, de Árbitros e de Equipas; testámos a adição de um jogador a uma equipa e o seu contrário; testámos a criação de Jogos; testámos que dois Utilizadores não podem ter os mesmos NIF; testámos que não dá para criar Utilizadores com atributos em falta; testámos os logins dos Utilizadores; testámos que não dá para criar Equipas com o mesmo nome; testámos a remoção de Jogadores das Equipas...

Nota Importante: Após testar os pedidos REST dos *endpoints* /removerEquipaDoJogador e /removerJogadorDaEquipa, conseguimos perceber que realizam o que é pedido no Swagger, contudo não passam nos testes na classe *SoccerNowApplication.* É um pequeno bug que temos noção que existe e que pretendemos resolver futuramente.

Nota Importante: Todos os testes não presentes na classe SoccerNowApplication foram feitos manualmente no Swagger.

Nas queries extras (dos casos de uso extras) e em mais algumas presentes durante toda a implementação do código, todos os Árbitros, Equipas e Jogadores que foram soft deleted continuam a aparecer, visto que os seus dados ainda são usados para casos de uso específicos, especialmente aqueles que usam dados antigos/históricos. Para o extra "Equipas com mais vitórias" também estão presentes as equipas deleted.

Durante a nossa implementação, uma vez que ainda não nos focámos muito nos Casos de Uso dos Campeonatos, por isso, por agora, todos os atributos relacionados com os Campeonatos e as EstatísticasEquipas ficarão a NULL nesta fase do projeto.

Neste momento, é possível ter jogadores ou árbitros com o mesmo nome, uma vez que estamos a utilizar um atributo NIF em cada Utilizador. Isto resolve o problema de ter utilizadores com o mesmo nome tal como resolve o problema de não poder ter um Jogador a ser Arbitro (e vice versa). É de mencionar também que no futuro (fase 2 do projeto) as funções de pesquisa baseadas por nome dos dois tipos de utilizador vão passar a usar NIF.

É importante mencionar ainda que em todos os Jogos, a lista de Árbitros não contém o Árbitro Principal – todos os Jogos têm pelo menos Árbitro Principal mesmo que a lista de árbitros auxiliares esteja vazia (NULL).

Estipulámos também que no update da Equipa apenas podemos dar update ao nome à equipa. Para tirar ou adicionar jogadores novos, temos dois *endpoints* adicionais no *JogadorController* para esse propósito.

Em termos de verificações, há algumas que foram feitas durante a implementação do código, tais como:

▷ Verificar que um jogador não pertence às duas equipasJogo em simultâneo para um jogo específico.

▷ Verificar que um jogador que está numa equipaJogo pertence realmente à Equipa por que está a jogar

▷ Verificar que os elementos de uma EquipaJogo existem antes de as ir buscar para a sua criação

Para colmatar possíveis erros, fomos verificar também, entre outros:

▷ Caso jogadores ou árbitros não serem encontrados, em vez de dar erro, há uma mensagem no Swagger a dizer que não foi encontrado. A mesma coisa verifica-se com as funções para apagar algum utilizador: se não for encontrado, no Swagger, mostra uma mensagem explicando que não foi encontrado nenhum utilizador com aquele id.

▷ Se todos os DTOs têm todos os campos importantes preenchidos (ou seja, não têm nenhum campo/atributo importante a NULL).

▷ Se todos os DTOs têm as equipasJogo preenchidas com 5 Jogadores cada.

8. Garantias Para Lógica de Negócio

Alguns pontos lógicos que devem ocorrer de determinadas situações estão implementados:

▷ Jogador cria um Jogo -> Número de Jogos do Jogador vai aumentar

▷ Jogador marca um Golo -> Jogador vai ter essa Estatística associada a ele

▷ Jogador começa a pertencer a uma Equipa -> Equipa tem registo dele

▷ Jogador deixa de pertencer a uma Equipa -> Equipa deixa de ter registo dele

▷ Jogador é eliminado -> Equipa deixa de ter registo deste Jogador, Jogo passa a ter NULL em qualquer Estatística ou qualquer Equipa a que seja associado, ou seja, todas as entidades que têm a *foreign key* desta Equipa a eliminar, pomos no seu lugar NULL

▷ Árbitro participa num Jogo -> Número de Jogos do Árbitro vai aumentar

▷ Árbitro é eliminado -> Jogo passa a ter NULL em qualquer posição em que o Árbitro esteja.

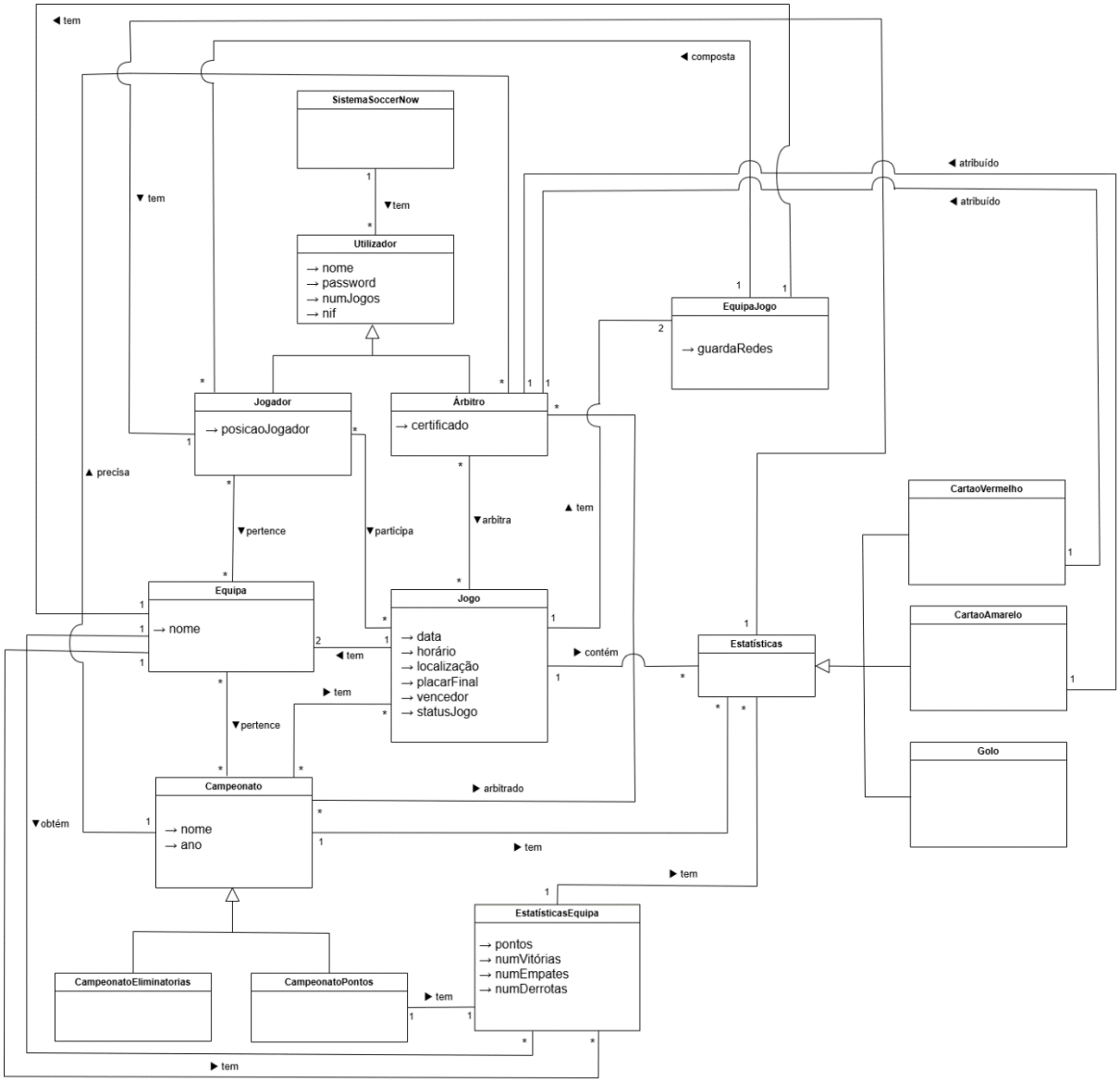
▷ Árbitro é eliminado -> Todas as entidades que têm a *foreign key* deste Árbitro a eliminar, pomos no seu lugar a NULL

▷ Jogo termina -> Status passa de COMEÇADO para ACABADO

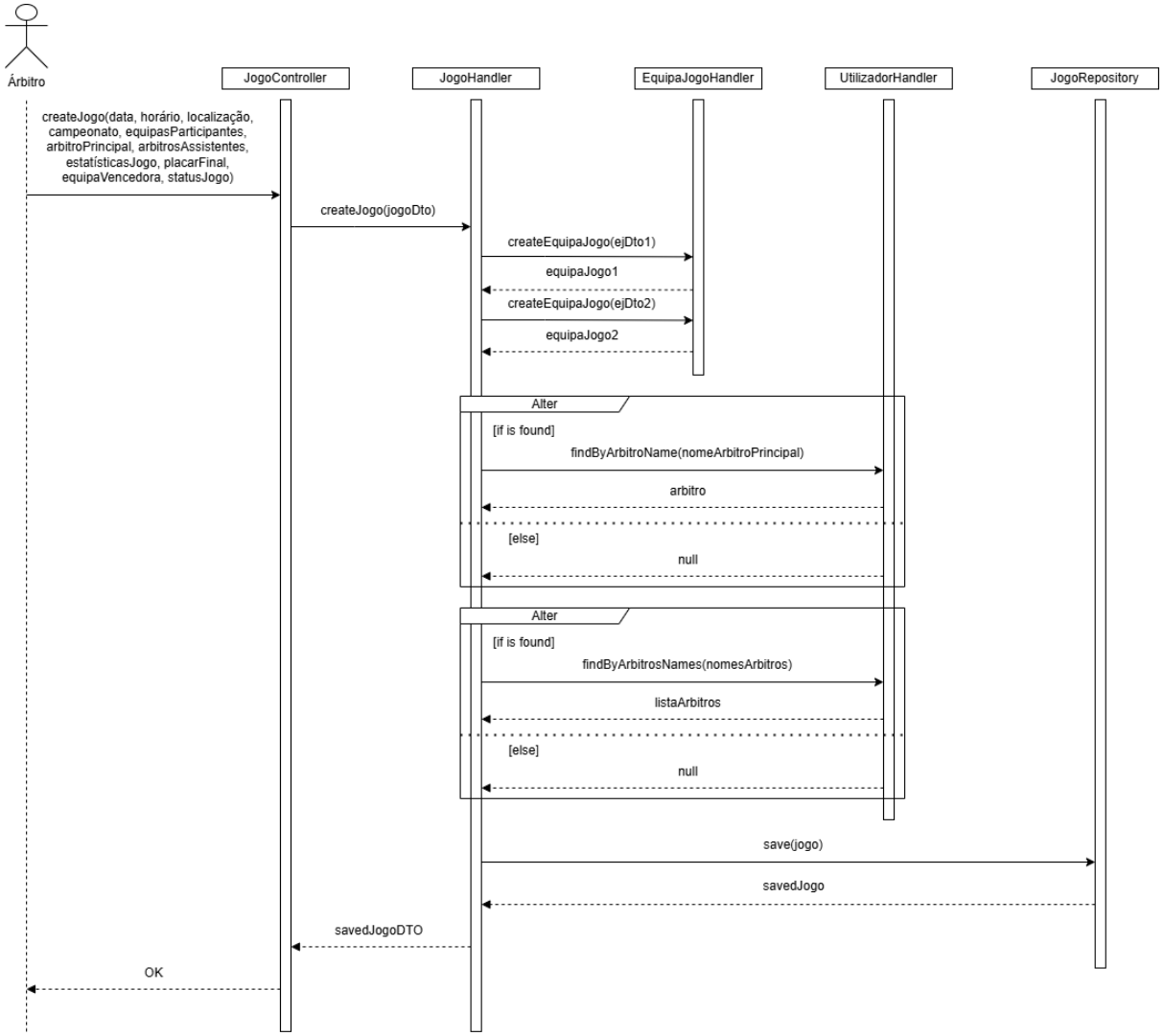
▷ Equipa é eliminada/removida -> Todas as entidades que têm a *foreign key* desta Equipa a eliminar, pomos no seu lugar NULL

▷ Não é possível criar duas Equipas com o mesmo nome -> Se se tentar, retorna um erro a recusar criar a Equipa

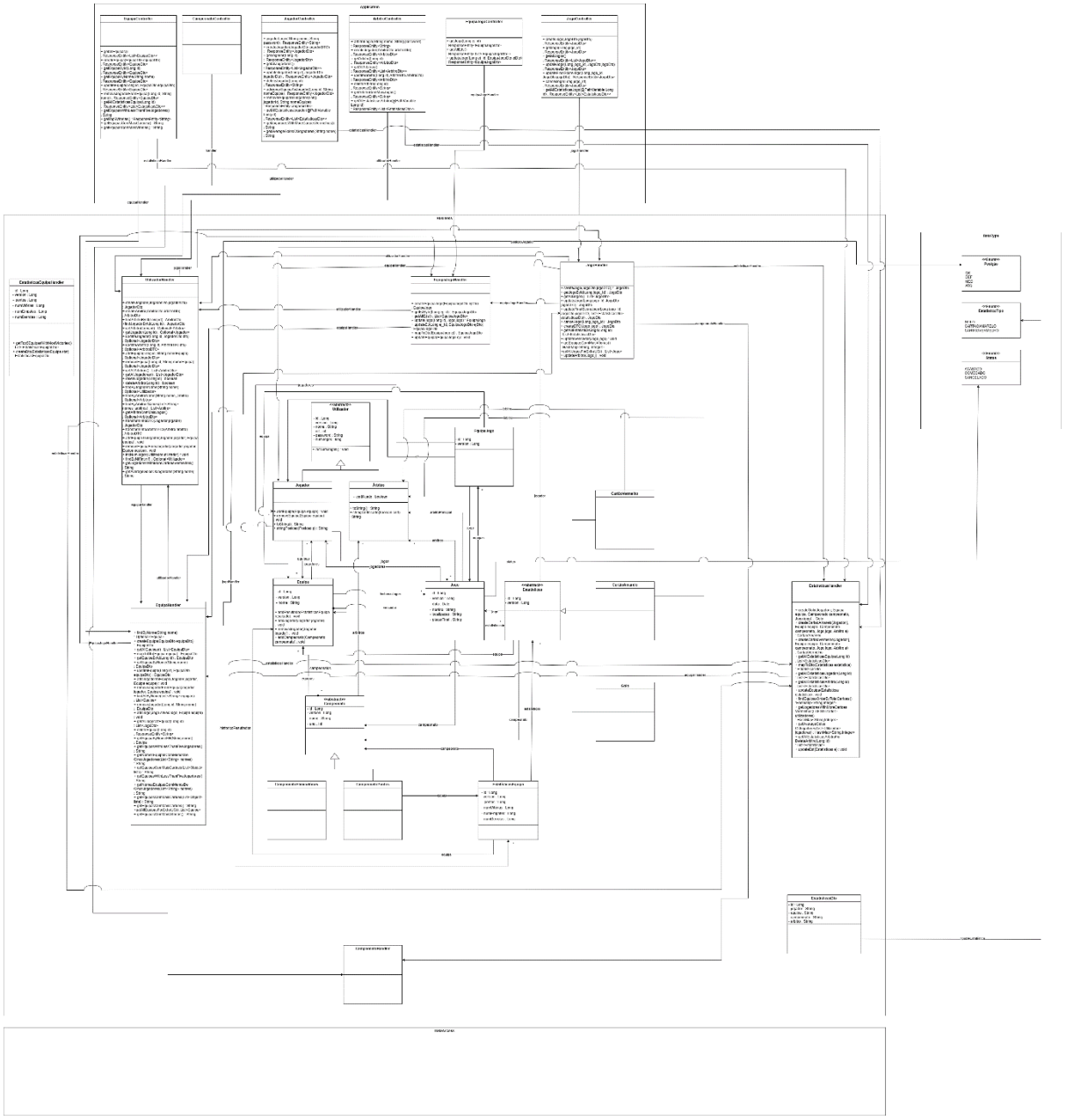
9. **Modelo de Domínio**



10. **SSD do Caso H**



11. **Diagrama de Classes**



(Os pngs das imagens dos últimos 3 pontos estão disponíveis na mesma pasta fora deste documento.)