SoccerNow

Gestão de Jogos de Futsal

Fase 1

Construção de Sistemas de Software - LEI 2024/25

Grupo 7
Gabriel Santos, fc59820
Marta Lourenço, fc58249
Miguel Lopes, fc54390

Descrição da arquitetura de camadas

A aplicação desenvolvida apresenta uma arquitetura em camadas, na qual cada camada possui as seguintes responsabilidades:

Camada de apresentação:
- Trata dos pedidos HTTP recebidos, sendo estes correspondentes aos casos de uso desenvolvidos
- Implementada nas classes anotadas com @RestController (*controllers*)

Camada de aplicação:
- Contém a lógica de negócio, coordenando a execução dos casos de uso
- Implementada nas classes anotadas com @Service (*handlers*)

Camada de negócio:
- Define as entidades e as suas relações, conceitos essenciais do negócio
- Implementada nas classes anotadas com @Entity (*entities*)

Camada de persistência:
- Orquestra as interações com a base de dados, incluindo persistir objetos novos/alterados e buscas baseadas no seu *id* ou outros atributos
- Implementada nas classes anotadas com @Repository (*repositories*)

A **camada de interface do utilizador** não foi implementada nesta fase do projeto, sendo a interface para a realização dos pedidos HTTP disponibilizada pelo Swagger.

Mapeamento JPA

Entidades

Com base na descrição do sistema a desenvolver, as entidades implementadas (anotadas com `@Entity`) estão descritas na tabela na próxima página com os seus atributos (exceto `id` e atributos que representam relações).

Geralmente, os atributos são anotados com `@Column`, podendo ou não especificar se esses valores devem ser `unique`, `nullable` ou se são um atributo temporal com `columnDefinition`.

- Atributos `id` são anotados da seguinte forma:
  ```
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  ```
- Atributos que representam **mapas** são anotados com `@ElementCollection`, `@CollectionTable` e `@MapKeyColumn` (`@MapKeyJoinColumn` no caso da chave ser uma entidade)
- Atributos que envolvam **enumerados** são anotados com a anotação `@Enumerated(EnumType.STRING)`

Entidade	Atributos
Arbitro	**`boolean certificado`**: O árbitro é certificado ou não?
Campeonato	**`Formato formato`**: Campeonato por pontos ou eliminatória **`Map<Clube, Integer> pontuacao`**: Pontos atribuídos a cada clube **`Map<Clube, Integer> classificacao`**: Classificação de cada clube
Clube	**`String nome`**: Nome do clube
Equipa	**`String nome`**: Nome da equipa **`int numVitorias`**: Total de vitórias da equipa
Estatisticas	**`Map<Equipa, Integer> resultado`**: Pontos atribuídos a cada equipa **`Equipa equipaVitoriosa`**: Equipa que ganhou o jogo **`Map<Jogador, Integer> golos`**: Número de golos marcados por cada jogador **`Map<Jogador, Cartao> cartoesAtribuidos`**: Cartões atribuídos a cada jogador
Jogador	**`Posicao posicao`**: Posição escolhida pelo jogador **`int numGolosMarcados`**: Total de golos recebidos pelo jogador **`Map<Cartao, Integer> cartoesRecebidos`**: Cartões recebidos pelo jogador
Jogo	**`LocalDate data`**: Data em que o jogo ocorre **`LocalTime horario`**: Horário em que o jogo ocorre **`Local local`**: Local onde o jogo ocorre **`Estado estadoAtual`**: Estado atual do jogo **`Arbitro arbitroPrincipal`**: Árbitro principal, no caso de haverem vários árbitros
Local	**`String nome`**: Nome do local **`String morada`**: Morada do local **`String codigoPostal`**: Código postal do local
Utilizador	**`String nome`**: Nome do utilizador **`String email`**: E-mail do utilizador **`String password`**: Palavra-passe do utilizador

Herança

Para diferenciar os dois tipos de Utilizador (Arbitro e Jogador), optou-se pelo uso da herança. A estratégia escolhida foi *Single Table* - ou seja, as entidades Arbitro e Jogador existem dentro da tabela da entidade Utilizador em vez de terem as suas próprias tabelas. A entidade Utilizador contém o atributo `id`, por isso este não aparece nas suas subclasses. O mapeamento da herança foi feito da seguinte forma:

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public class Utilizador {...}
```

Relações

As relações mapeadas na implementação estão descritas abaixo, com as entidades donas da relação assinaladas a sublinhado.

Arbitro - Jogo (M-N bidirecional): Vários jogos podem ser oficiados por vários árbitros

Arbitro: `List<Jogo> jogos`, anotado com `@ManyToMany` & `mappedBy`	**Jogo:** `List<Arbitro> arbitros`, anotado com `@ManyToMany` & `@JoinTable`

Clube - Campeonato (M-N bidirecional): Vários clubes participam em vários campeonatos

Clube: `List<Campeonato> campeonatos`, anotado com `@ManyToMany` & `mappedBy`	**Campeonato:** `List<Clube> clubes`, anotado com `@ManyToMany` & `@JoinTable`

Campeonato - Jogo (1-N unidirecional): Vários jogos associados a um campeonato

Campeonato: `List<Jogo> jogos`, anotado com `@OneToMany`	**Jogo:** Como a relação é unidirecional, não existe mapeamento no Jogo

Clube - Jogador (1-N bidirecional): Vários jogadores pertencem a um clube

Clube: `List<Jogador> jogadores`, anotado com `@OneToMany` & `mappedBy`	**Jogador:** `Clube clube`, anotado com `@ManyToOne` & `@JoinColumn`

Clube - Equipa (1-N bidirecional): Várias equipas pertencem a um clube

Equipa: `Clube clube`, anotado com `@ManyToOne` & `@JoinColumn`	**Clube:** `List<Equipa> equipas`, anotado com `@OneToMany` & `mappedBy`

Equipa - Jogador (M-N bidirecional): Vários jogadores pertencem a várias equipas

Equipa: `List<Jogador> jogadores`, anotado com `@ManyToMany` & `@JoinTable`	**Jogador:** `List<Equipa> equipas`, anotado com `@ManyToMany` & `mappedBy`

Equipa - <u>Jogo</u> (M-N bidirecional): Várias equipas participam em vários jogos

Equipa: `List<Jogo> jogos`, anotado com `@ManyToMany` & `mappedBy`	**<u>Jogo</u>:** `List<Equipa> equipas`, anotado com `@ManyToMany` & `@JoinTable`

<u>Estatisticas</u> - Jogo (1-1 bidirecional): Um conjunto de estatísticas para um jogo

<u>Estatisticas</u>: `Jogo jogo`, anotado com `@OneToOne` & `@JoinColumn`	**Jogo:** `Estatisticas estatisticas`, anotado com `@OneToOne` & `mappedBy`

Local - <u>Jogo</u> (1-N bidirecional): Vários jogos decorrem num local

Local: `List<Jogo> jogos`, anotado com `@OneToMany` & `mappedBy`	**<u>Jogo</u>:** `Local local`, anotado com `@ManyToOne` & `@JoinColumn`

Garantias de Negócio

De modo a garantir que a aplicação nunca existe num estado que viole as regras de negócio esperadas do contexto do *SoccerNow*, cada caso de uso conta com várias validações dos pedidos efetuados pelos utilizadores.

Conjunto C1

- **Jogador ou Árbitro:** Uma vez que jogador e árbitro usam o id da entidade estendida, e como utilizador usa GeneratedValue, os ids são únicos, não permitindo assim, um utilizador ser simultaneamente jogador e árbitro.
- **Identificação correta por tipo na obtenção:** Ao obter um utilizador, o sistema determina dinamicamente se se trata de um jogador ou árbitro. Isso garante que o cliente receba todos os dados relevantes ao tipo concreto.
- **Remoção de jogadores:** Antes de remover um jogador, é retirado esse membro do clube a que pertencia;
- **Remoção de árbitros:** A operação é bloqueada caso o árbitro ainda tenha jogos por cumprir, garantindo que o sistema se mantém consistente.
- **Atualização separada para jogadores e árbitros com validações específicas:** Métodos próprios de forma a aplicar as regras de cada papel. Assim, é possível atualizar uma maior amplitude de atributos, tanto aqueles herdados de utilizador como também atributos individuais das entidades.

Conjunto C2

Devido à divisão entre clubes e equipas, implementou-se os mesmos casos de uso das equipas para os clubes.

Criação de clubes:

- Antes de criar o clube, verifica-se se já existe um clube com o nome especificado. Se não existir, o clube é criado e persistido na base de dados através de `ClubeRepository`

Criação de equipas:

- Antes de criar a equipa, procede-se à validação dos campos introduzidos:
    - O clube existe?
    - Já existe alguma equipa com o nome especificado?
    - Os jogadores são exatamente cinco, únicos, do mesmo clube e pelo menos um deles é guarda-redes?
- Se as validações forem todas satisfeitas, a equipa é criada e a sua referência é acrescentada ao seu clube e aos seus jogadores.

Remoção de clubes:

- Para garantir a consistência do sistema, verifica-se se o clube não tem jogadores e se as suas equipas não têm jogos agendados/em curso.
- Se as validações forem satisfeitas, o clube e as suas equipas são apagados.

Remoção de equipas:

- Antes de remover a equipa especificada, verifica-se se esta tem jogos agendados/em curso, evitando causar inconsistências no sistema.
- Se a validação anterior for satisfeita, a referência para a equipa a apagar é removida do clube e de todos os seus jogadores. Por fim, a equipa é apagada

Atualização de clubes:

- Como o único campo que pode ser atualizado é o nome, efetua-se a mesma validação da criação de clubes. Se o nome escolhido não for o nome de outro clube, o clube é atualizado

Atualização de equipas:

- A operação de atualização de equipa permite alterar dois atributos da equipa: o seu nome e/ou os seus jogadores. Antes de efetuar as alterações, o sistema efetua as mesmas validações relativas ao nome e jogadores efetuadas no processo de criação de equipas
- Se as validações forem satisfeitas, o nome da equipa é alterado e/ou os jogadores são substituídos pelos especificados

Conjunto C3

Criar jogo:

- **Validade da data e horário:** Verifica-se se a data e horário especificados não correspondem a um momento no passado
- **Local existente e disponível:** Verifica-se se o lugar existe e não tem mais jogos marcados para o mesmo dia
- **Equipas:** Verifica-se que são exatamente duas e não pertencem ao mesmo clube
- **Disponibilidade dos jogadores:** Para cada jogador de cada equipa, verifica-se se não têm mais jogos marcados para o mesmo dia
- **Árbitros existentes e disponíveis:** Verifica-se se os árbitros existem e não têm mais jogos marcados para o mesmo dia
- **Árbitro principal:** Caso haja mais do que um árbitro, verifica-se se foi especificado um árbitro principal pertencente à lista de árbitros fornecida. Se houver apenas um árbitro e não foi especificado um árbitro principal, esse único árbitro será considerado o principal

Registar resultado:

- **Jogo existe:** Verifica-se se o jogo especificado existe e ainda não está marcado como concluído
- **Validade das equipas e jogadores:** Verifica-se se as equipas e jogadores especificados correspondem aos do jogo
- **Validade dos valores:** Verifica-se se os valores das estatísticas são válidos

Adicionalmente, foi implementado um caso de uso para **criar um local**, pois este é necessário para a criação de um jogo.