Relatório SoccerNow

Pedro Marques nº48674

José Matusse nº56996

Objetivo

SoccerNow é uma aplicação de gestão e organização de jogos e torneios de futsal, permite a criação e remoção de utilizadores que serão ou árbitros ou jogadores, de jogos e torneios

Arquitetura em camadas

O programa desenvolvido SoccerNow segue uma arquitetura em camadas típica de aplicações SpringBoot, havendo vários níveis de abstração de maneira a promover a separação de responsabilidades, facilitar a manutenção e evolução independente de cada camada. A arquitetura implementada segue o padrão de desenho Model-View-Controller (MVC)

1. **Camada de apresentação (Controllers)**

   Responsabilidades:
   - Receber e processar requisições HTTP dos clientes
   - Validar dados de entrada
   - Delegar processamento de negócios para a camada de serviço
   - Formatar e retornar respostas apropriadas

   Implementados:
   - EquipaController
   - JogadorController
   - UserController

2. **Camada de serviço (Services)**

   Responsabilidades:

   - Implementar a lógica de negócios
   - Coordenar operações entre diferentes repositórios
   - Aplicar regras de negócio e validações
   - Gerir transações

   Implementados:

   - ArbitroService
   - EquipaService
   - JogadorService

3. **Camada de persistência (Repositories)**

   Responsabilidades:

   - Fornecer abstração para operações de acesso a dados
   - Implementar consultas específicas ao banco de dados
   - Traduzir entre objetos de domínio e representação de banco de dados

   Implementados:

   - ArbitroRepository
   - EquipaRepository
   - JogadorRepository

4. **Camada de Modelo (Models)**

   Responsabilidades:

   - Representar as entidades do domínio e seus relacionamentos
   - Encapsular o estado e comportamento das entidades
   - Definir mapeamento ORM para persistência

   Implementados:

   - Arbitro
   - Equipa
   - Estatisticas
   - Jogador
   - User

Diagrama de classes

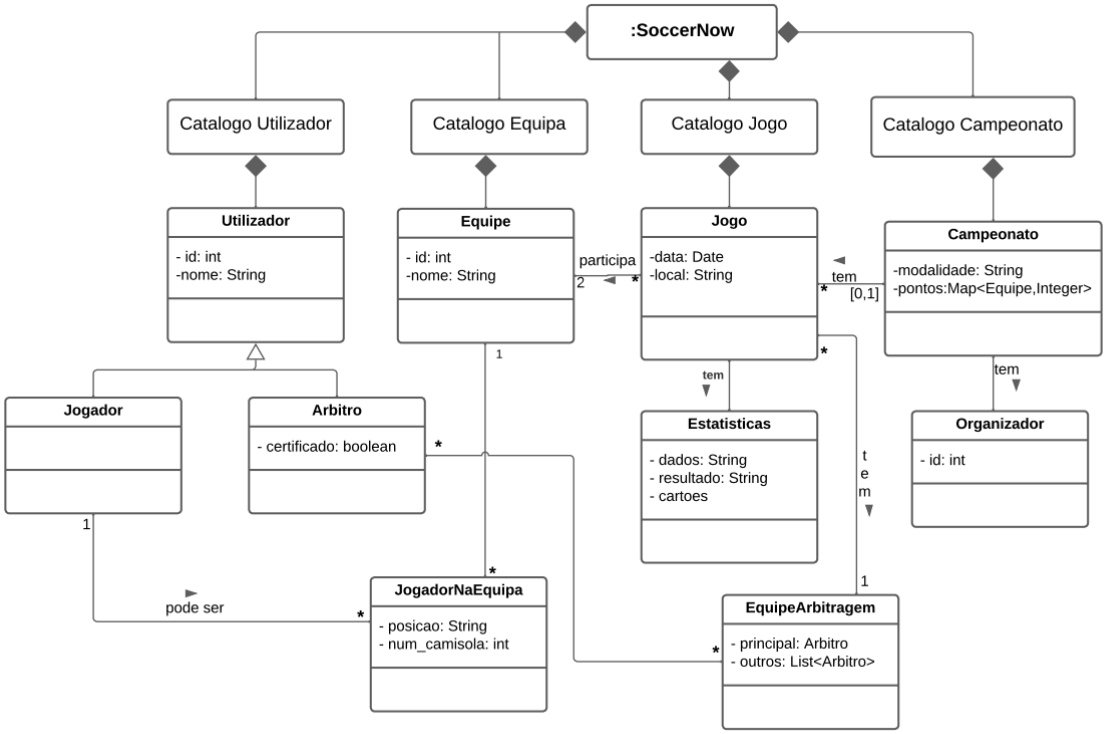
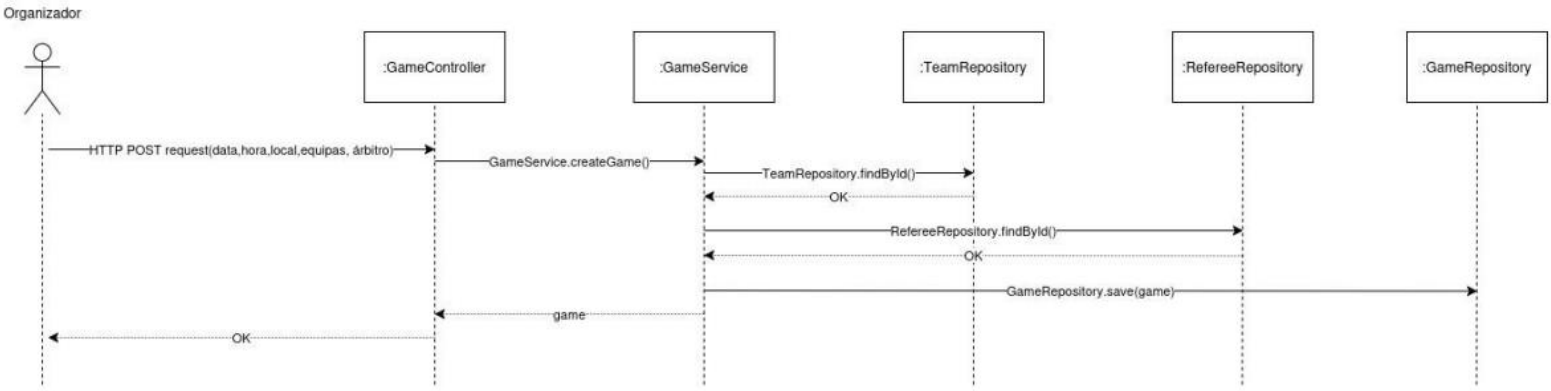
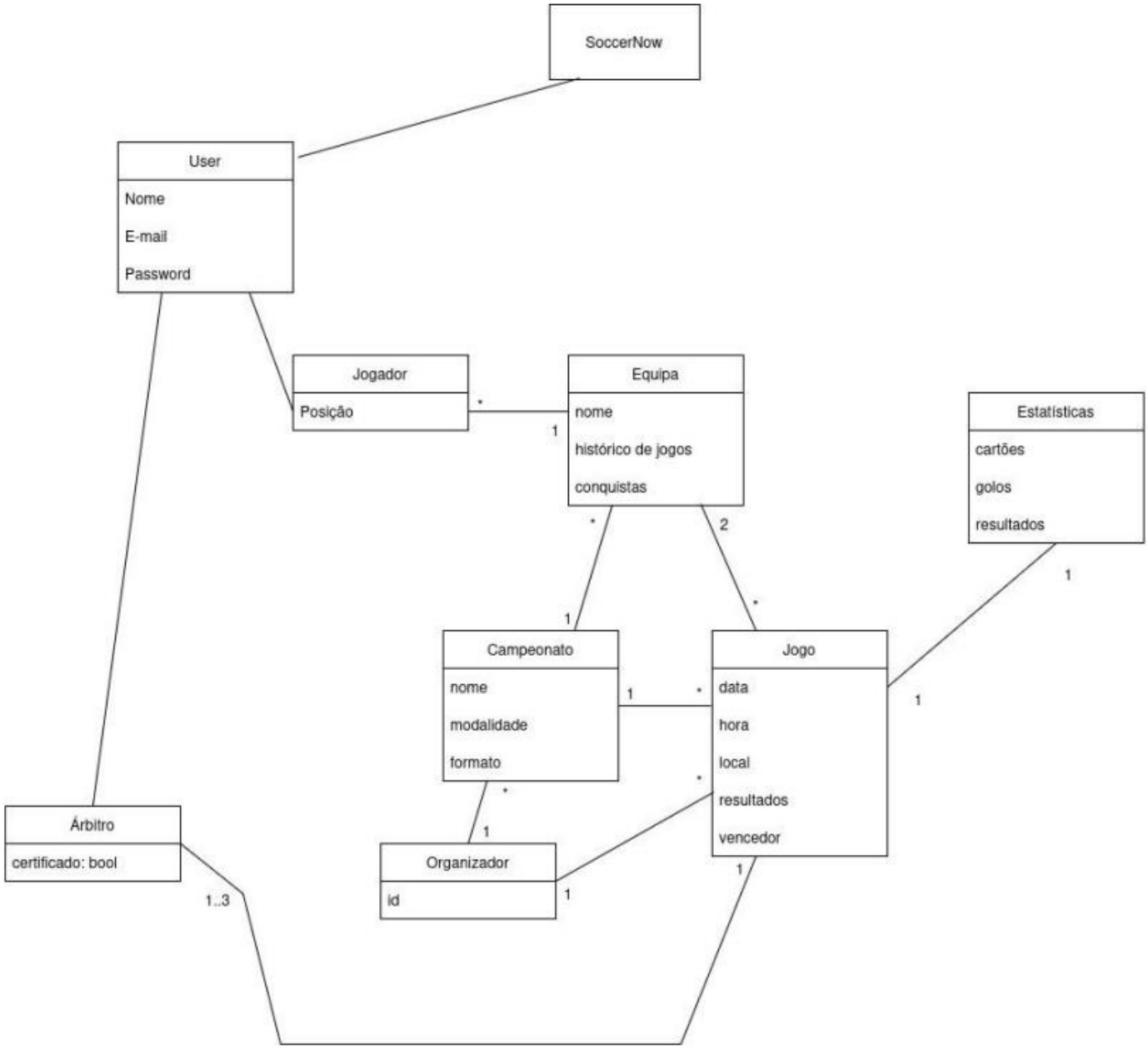


Diagrama de sequência (SSD)



Modelo de Domínio



Mapeamento JPA

Mapeamento Jogador-Equipa (ManyToMany)

Na classe Jogador.java, utilizamos mapeamento “@ManyToMany” uma vez que cada jogador pode integrar várias equipas, utilizamos também a anotação “@JoinTable” para criar uma tabela intermediária “jogador_equipa” para armazenar as associações entre jogadores e equipas, permitindo múltiplas associações em ambas as direções. O proprietário desta relação é o Jogador (sem “mappedBy”), enquanto Equipa referencia o mapeamento com “mappedBy=”equipas”” de maneira a centralizar a configuração do relacionamento e evitar duplicação. Utilizamos também a anotação “@JsonIgnoreProperties” em ambos os lados para evitar recursão infinita durante a serialização JSON.

Mapeamento Jogador-Estatísticas (OneToMany)

Um jogador possui múltiplas estatísticas, mas cada uma pertence a apenas um jogador, usamos também o “CascadeType.ALL” para garantir que operações realizadas num jogador sejam propagadas para as suas estatísticas. Em Estatisticas fazemos “@JoinColumn(name=”jogador_id”)” de maneira a ter a coluna jogador_id, que referencia a tabela users onde estão todos os usuários (Jogadores e Árbitros) armazenados, na entidade Jogador definimos esta entidade como proprietário da relação, de maneira a ser mais eficiente, cada jogador tem registo das suas estatísticas.

O modelo Estatísticas ainda está por ser implementado na sua totalidade, ainda tem algumas características em falta

Mapeamento da herança em User-Jogador

Usamos aqui uma estratégia de tabela única com a anotação “@DiscriminatorValue(“Jogador”)”, isto permite consultas mais simples e simplifica o esquema da persistência.

Decidimos desenhar esta relação desta maneira para ser mais simples e haver reutilização de código, através do polimorfismo.