Relatório SoccerNow

Trabalho realizado por:

Erickson Cacondo fc53653

André Santos fc57538

Miguel Sanchez fc64675

1. Introdução

Este relatório apresenta uma análise do projeto SoccerNow, juntamente com diagramas (modelo de classes combinado, modelo de domínio e diagrama de sequência) que refletem a implementação do código. O objetivo é documentar a estrutura, implementação e arquitetura do projeto

2. Estrutura e Arquitetura do Projeto

O projeto segue uma arquitetura em camadas bem definida, separando as responsabilidades em:

- **Camada de Apresentação (Controller):** Responsável por expor a funcionalidade através de uma API REST. Utiliza controladores Spring (@RestController) para mapear pedidos HTTP para métodos de serviço. Os controladores (UserController, PlayerController, TeamController, JogoController) recebem e retornam DTOs (Data Transfer Objects) ou entidades do domínio, interagindo com a camada de serviço.
- **Camada de Negócio (Service):** Contém a lógica de negócio principal da aplicação. Os serviços (UserService, PlayerService, TeamService, JogoService) orquestram as operações, coordenam interações entre repositórios e implementam regras de negócio e validações. Utiliza a anotação @Service do Spring e @Transactional para gestão de transações.

  Modelo de Domínio (Domain): Representa as entidades centrais do negócio (User, Player, Referee, Team, Jogo, Campeonato, Cartao, Estatisticas, Resultado). Estas classes são anotadas com JPA (@Entity, @Table, @Id, @GeneratedValue, @ManyToOne,

@OneToMany, @ManyToMany, @OneToOne, etc.) para mapeamento objetorelacional.
@Table, @Id, @GeneratedValue, @ManyToOne, @OneToMany, @ManyToMany, @OneToOne, etc.) para mapeamento objetorelacional.

- **DTOs (Data Transfer Objects):** Utilizados para transferir dados entre as camadas, especialmente entre a camada de apresentação e a camada de serviço (UserDTO, TeamDTO).
- **Configuração:** Inclui configurações da aplicação, como a configuração do OpenAPI/Swagger (OpenApiConfig.java) para documentação da API.
- **Exceções:** Define exceções personalizadas (ApplicationException, NotFoundException) para tratamento de erros específicos da aplicação.

Esta estrutura promove a separação de conceitos, manutenibilidade e testabilidade do código.

3. Análise do Modelo de Domínio

O modelo de domínio, conforme implementado nas classes e visualizado no diagrama **modelo_dominio_v2.png** (ver Figura 3 na secção 8.3), representa as entidades principais. As classes correspondem bem às entidades, com as seguintes observações e nomenclaturas corretas:

- **User (Abstrata):** Classe base para Player e Referee, utilizando herança InheritanceType.JOINED. Contém atributos comuns como id, name, email, password.
- **Player:** Estende User, adicionando preferredPosition e uma relação @ManyToMany com Team (players).
  Referee: Estende User, adicionando o atributo booleano certified.
- **Team:** Contém id, name e uma relação @ManyToMany com Player (players).
- **Jogo:** Entidade central. Contém id, dateTime, location, amigavel, homeScore, awayScore. Possui relações:
  - @ManyToOne com Team (obrigatórias, homeTeam e awayTeam).
  - @ManyToOne com Campeonato (opcional).
  - @ManyToMany com Referee (referees).
  - @ManyToOne com Referee (primaryReferee, opcional).
  - @OneToMany com Cartao (mapeada por "jogo").

    - @OneToMany com Estatisticas (mapeada por "jogo").
    - @OneToOne com Resultado (mapeada por "jogo").
- **Campeonato:** Contém id, nome, modalidade, formato e uma relação @OneToMany com Jogo (não explicitamente mapeada no lado Campeonato no código).
- **Cartao:** Contém id, tipo e uma relação @ManyToOne com Jogo.
- **Estatisticas:** Contém id, gols e uma relação @ManyToOne com Jogo.
- **Resultado:** Contém id, placar, uma relação @ManyToOne com Team (equipaVitoriosa, opcional) e uma relação @OneToOne com Jogo.

4. Justificativas do Mapeamento JPA

A persistência de dados no projeto é gerida através do Java Persistence API (JPA) com Hibernate como provedor. As escolhas de mapeamento objeto-relacional foram feitas com base nos requisitos do domínio e nas boas práticas de JPA:

- **Identificação de Entidades (@Entity):** Todas as classes persistentes (User, Player, Referee, Team, Jogo, Campeonato, Cartao, Estatisticas, Resultado) foram anotadas com @Entity.
- **Chaves Primárias (@Id, @GeneratedValue):** A estratégia GenerationType.IDENTITY faz para a geração automática de chaves primárias (id).
  Mapeamento de Herança (@Inheritance(strategy = InheritanceType.JOINED)): Para a hierarquia User (Player, Referee),foi escolhida a estratégia JOINED, criando tabelas separadas ligadas pela chave primária.
- **Mapeamento de Relações:**
  - **@ManyToOne:** Utilizada para Jogo -> Team, Cartao -> Jogo, etc.
  - `optional = false` garante obrigatoriedade em Jogo -> Team.
  - **@OneToMany:** Utilizada em Jogo -> Cartao, Jogo -> Estatisticas. O `mappedBy` é usado para indicar a entidade proprietária.
  - **@ManyToMany:** Aplicada em Team <-> Player e Jogo <-> Referee. A anotação @JoinTable configura as tabelas de junção.
  - **@OneToOne:** Utilizada para Jogo <-> Resultado. @JoinColumn define a chave estrangeira no lado proprietário (Resultado).

- **Operações em Cascata (cascade):** CascadeType.ALL foi aplicado em relações a partir de Jogo (para Cartao, Estatisticas, Resultado), propagando operações quando o ciclo de vida é dependente.
- **Campos Transitórios (@Transient):** Utilizado para métodos auxiliares não persistidos.

Estas escolhas visam criar um esquema de base de dados relacional normalizado e eficiente.

5. Garantias do Sistema (Lógica de Negócio)

A camada de serviço implementa diversas validações e regras de negócio:

- **Integridade Transacional (@Transactional):** Garante atomicidade das operações de escrita na base de dados.
- **Validação na Criação de Jogos (JogoService.criarJogo):**
  - Equipas Distintas: Verifica se homeTeamId != awayTeamId.
  - Existência de Entidades: Confirma a existência de homeTeam, awayTeam e árbitros via repositórios. *Nota: A validação/ associação de Campeonato está ausente no código para a criação de jogo.*
  - Certificação de Árbitros: Se !jogo.isAmigavel(), verifica se todos os árbitros associados estão certificados.
  - Árbitro Principal: Verifica se o primaryRefereeId, se fornecido, existe e pertence à lista de árbitros.
- **Validação no Registo de Resultados (JogoService.registarResultado):**
  - Existência do Jogo.
  - Prevenção de Duplicação: Verifica se já existe um resultado associado ao jogo.
  - Existência da Equipa Vitoriosa (Opcional).
- **Gestão de Utilizadores (UserService):** Trata a criação/atualização de User, Player e Referee.
- **Gestão de Equipas (TeamService):** Gere a associação de jogadores a equipas.
- **Tratamento de Exceções:** Utilização de ApplicationException e NotFoundException.

Estas garantias asseguram que as operações respeitam as regras de negócio e mantêm a consistência dos dados.

6. Análise da Camada de Serviço

Os serviços implementam a lógica de negócio:

- **UserService:** Gere User, Player e Referee.
- **PlayerService:** Operações básicas para Player.
- **TeamService:** Gere Team e associação de Players.
- **JogoService:** Lógica complexa para criar jogos e registar resultados, com validações. *Nota: Interage diretamente com repositórios (TeamRepository, RefereeRepository) em vez de serviços intermédios para algumas validações.*

Observações: Uso apropriado de @Transactional e injeção de dependências. JogoService implementa validações importantes, mas com acesso direto a repositórios e sem a lógica de campeonato na criação.

7. Análise da Camada de Apresentação (Controladores)

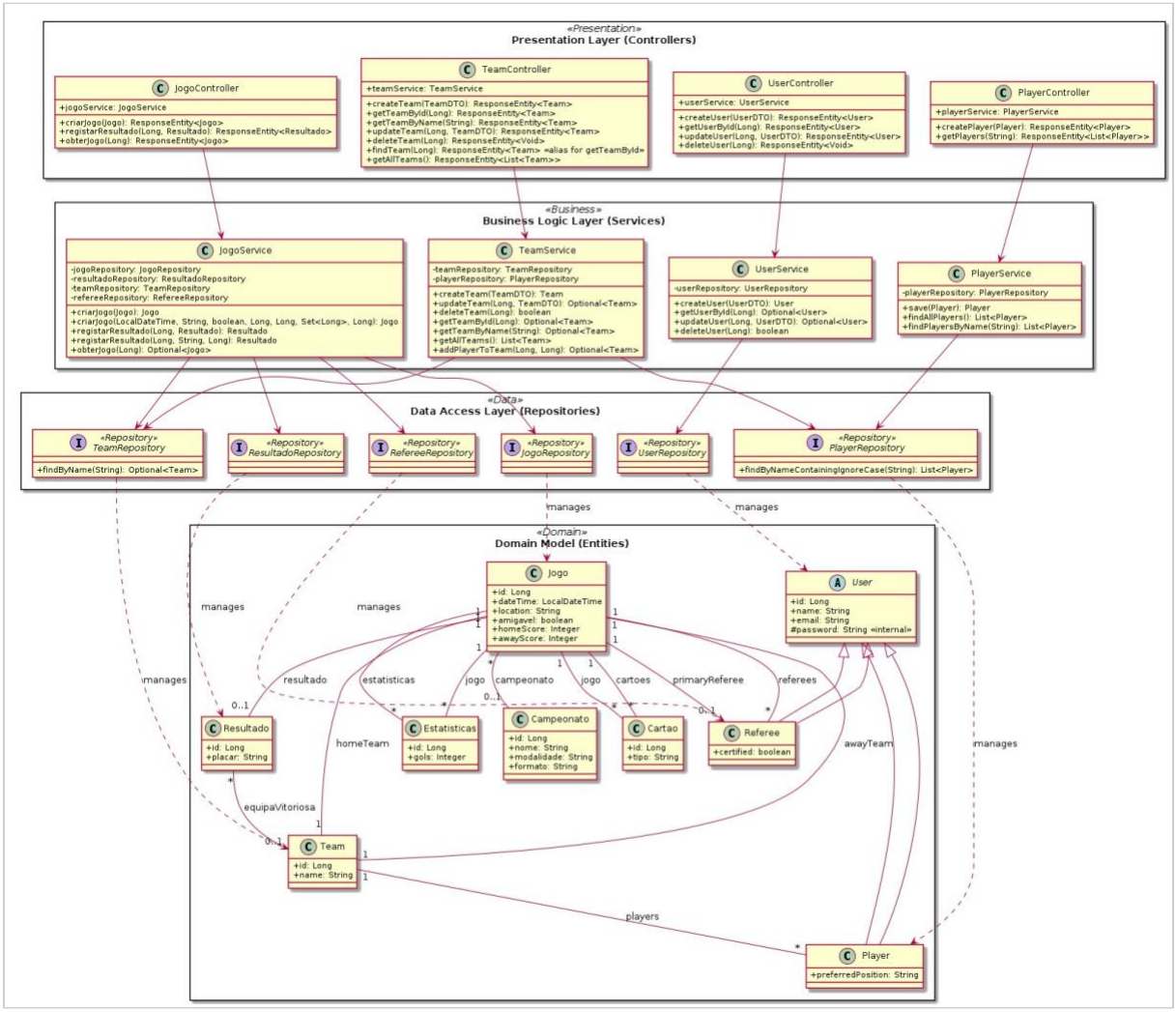
Os controladores expõem as funcionalidades via API REST:

- Mapeiam endpoints HTTP para métodos de serviço.
- Utilizam anotações Spring MVC/WebFlux (@RestController, @PostMapping, etc.).
- Recebem DTOs ou entidades (@RequestBody).
- Retornam ResponseEntity com entidades ou listas. *Nota: Na criação de jogo, o controlador recebe e retorna a entidade `Jogo` diretamente.*

8. Análise dos Diagramas

8.1 Diagrama de Classes

Figura 1: Diagrama de Classes.



Este diagrama (**diagrama_classes**) apresenta uma visão integrada da arquitetura, mostrando as classes/ interfaces nas camadas de Apresentação, Negócio e Dados, e incluindo o Modelo de Domínio na parte inferior.

Diagrama de Sequência (SSD) - Criação de Jogo

Figura 2: Diagrama de Sequência (SSD) - Criação de Jogo.

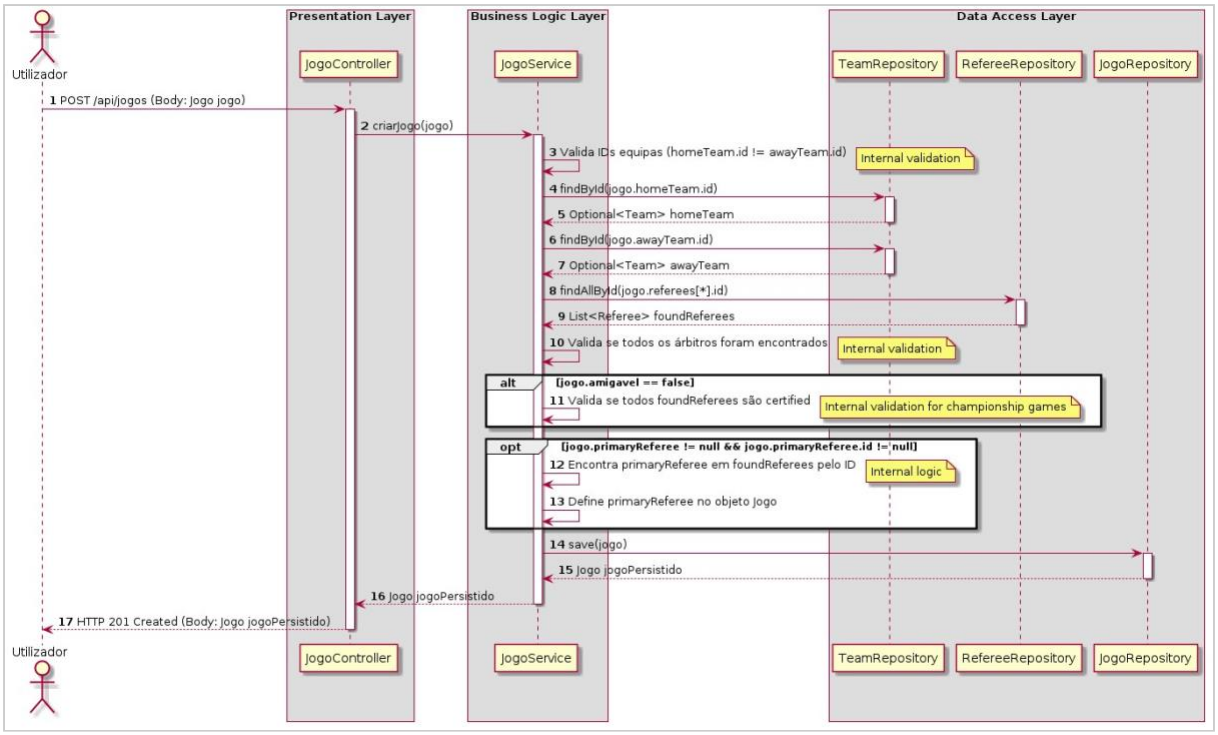


Figura 2: Diagrama de Sequência (SSD) para Criação de Jogo (POST /api/jogos).

Este diagrama (**ssd_criar_jogo**) ilustra o fluxo real de criação de um jogo:

- O `JogoController` recebe a entidade `Jogo` diretamente no corpo do pedido.
- O `JogoService` realiza validações internas e interage diretamente com `TeamRepository` e `RefereeRepository` para buscar equipas e árbitros.
- A lógica de validação/associação de Campeonato durante a criação está ausente.
- A validação de certificação de árbitros (para jogos não amigáveis) e a lógica para definir o árbitro principal estão representadas.
- A persistência ocorre via `JogoRepository`.
- O `JogoController` retorna a entidade `Jogo` persistida diretamente na resposta HTTP 201.

Este SSD reflete as interações e dependências reais observadas no código para esta operação.

8.2 Diagrama do Modelo de Domínio

Figura 3: Diagrama do Modelo de Domínio.

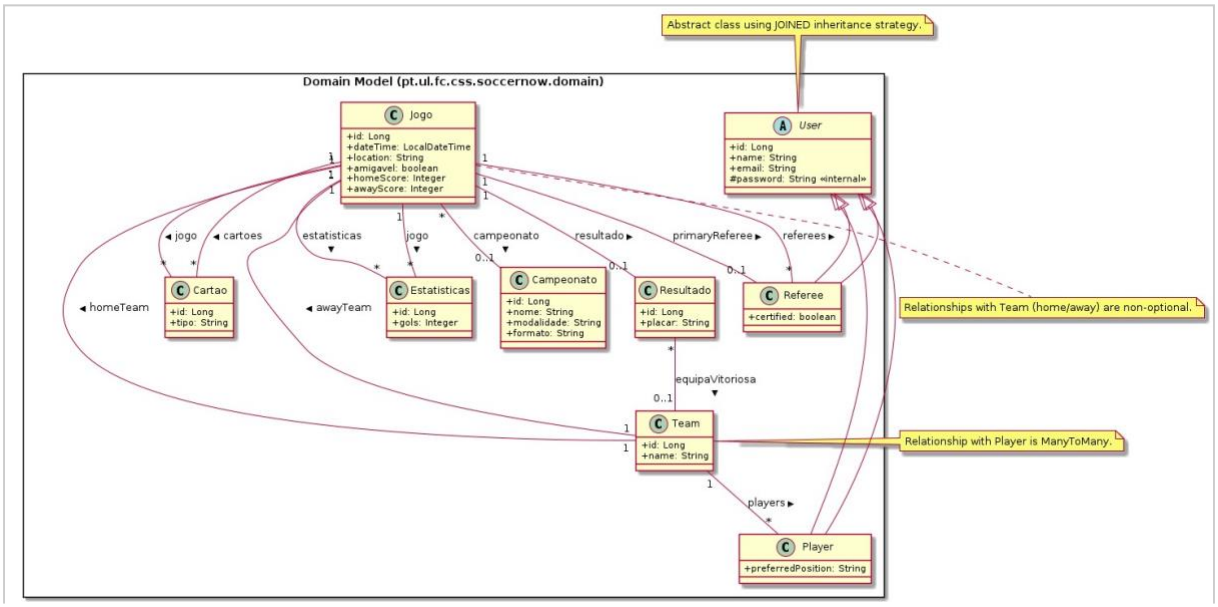


Figura 3: Diagrama do Modelo de Domínio.

(**modelo_dominio**) representa as entidades do domínio e suas relações conforme implementado no código.