SoccerNow

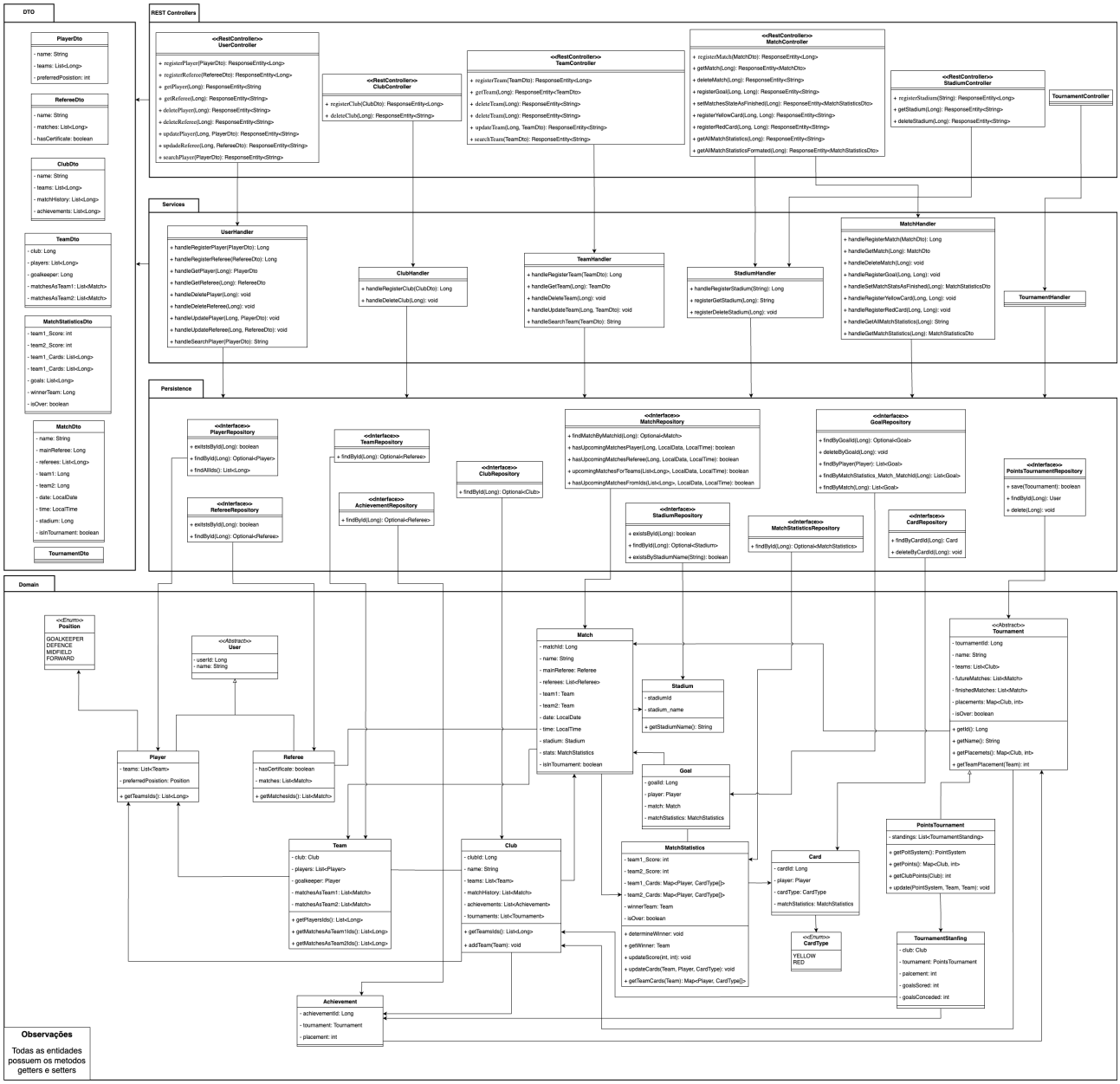
Luis Santos fc58437
Liliana Valente fc59846
Denis Bahnari fc59878

no âmbito da disciplina
Construção de Sistemas de Software

Ciências ULisboa

Contents

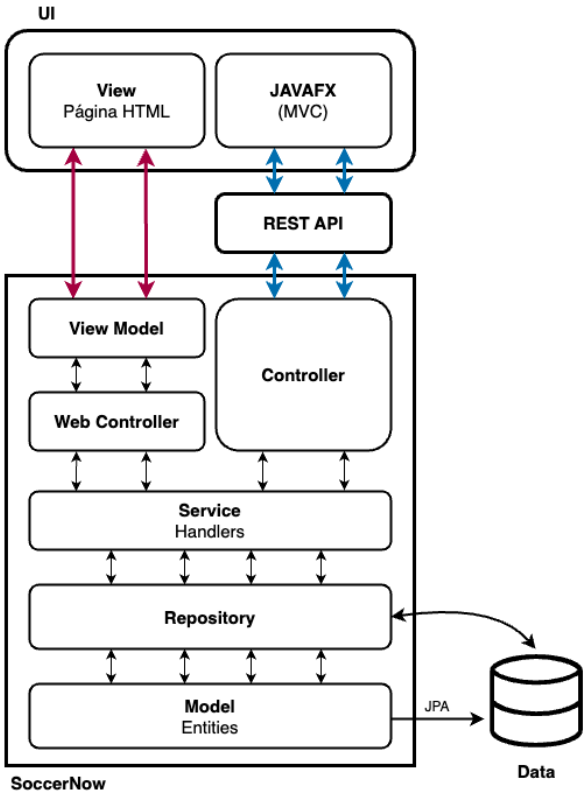
Diagrama de Classes



Estrutura Full Stack e Divisão por Camadas

Estruturamos o código seguindo uma arquitetura em camadas que separa claramente as responsabilidades da aplicação SoccerNow.

Na 2ª fase do projeto, foram implementados filtros funcionais para jogadores, árbitros, equipas, jogos e campeonatos, permitindo uma pesquisa mais precisa e eficiente. Estes filtros foram posteriormente integrados na interface web com Thymeleaf, seguindo a estrutura MVC, estudada durante o semestre. Todos os casos de uso propostos no enunciado foram implementados com sucesso. Para além da interface web, foi também desenvolvida a interface gráfica em JavaFX, orientada ao registo de utilizadores, equipas, jogos e campeonatos, permitindo a gestão visual deles (update, remove e get). Foram ainda acrescentados nesta fase testes para o Torneio que garantem o correto funcionamento das principais funcionalidades da aplicação.



Exploração de decisões técnicas no Thymeleaf

No desenvolvimento da interface web com Thymeleaf, aplicámos o padrão Page Controller, seguindo sempre o padrão de arquitetura de software MVC. Aplicando este padrão separamos os Web Controllers de forma a que cada um esteja somente responsável pela gestão de uma página. Também foi necessário criar View Models listados abaixo para intermediar a comunicação entre o controller e a view, cumprindo assim o modelo MVC.

```java
public class ViewModelReferee {

    private String name;
    private boolean hasCertificate;
    private List<String> matches;

    // getters e setters

}
```

```java
public class ViewModelPlayer {

    private String name;
    private List<String> teams;
    private String preferedPosition;

    // getters e setters

}
```

```java
public class ViewModelPointsTournament {

    private String tournamentName;
    private List<String> clubs;
    private List<String> matches;
    private boolean isOver;


    // getters e setters

}
```

```java
public class ViewModelClub {

        private Long clubId;
        private String name;
        private int nTeams;
        private List<String> tournaments;
        private List<String> achievements;

    // getters e setters

}
```

```java
public class ViewModelMatch {

    private Long id;
    private String principalReferee;
    private List<String> referees;
    private String team1;
    private String team2;
    private LocalDate date;
    private LocalTime time;
    private String stadium;
    private String score;
    private List<String> goals;
    private List<String> team1_yellowCards;
    private List<String> team2_yellowCards;
    private List<String> team1_redCards;
    private List<String> team2_redCards;
    private boolean isOver;
    private String tournament;
    private Map<Long, String> players;

    // getters e setters

}
```

```java
public class ViewModelTeam {

    private String club;
    private List<String> players;
    private String goalkeeper;
    private List<String> matches;

    // getters e setters

}
```

Foi criada uma página de login, permitindo a autenticação por mock de administradores do sistema SoccerNow. Esta implementação garante que, sempre que um utilizador é autenticado, é criada também uma sessão (HttpSession) para esse mesmo utilizador, onde o seu nome de utilizador é armazenado. Todos os Web Controllers da aplicação estão protegidos por uma condição que verifica a existência dessa sessão antes de permitir o acesso a qualquer endpoint. Caso a sessão não exista ou o utilizador não esteja autenticado, o sistema redireciona automaticamente para a página de login, assegurando assim o controlo de acesso às funcionalidades da aplicação e não permitindo o acesso a não administradores.

Após a autenticação somos redirecionados para o menu inicial:

Players: Na área dos jogadores, inicialmente é exibido todos os jogadores que existem registados, a tabela tem como colunas o nome do jogador, a sua posição preferida e o nome dos clubes a que pertençe. Para filtrar os jogadores, basta selecionar o tipo de filtro que se pretende aplicar, os seus valores e filtrar.

Referees: Na área dos árbitos, é exibido todos os árbitos regitados, com o seu nome, certificado, e todas as partidas que está associado. Sendo possivel filtrar os árbitos, escolhendo o tipo de filtro e o seu repsetivo atributo.

Clubs: Na áreda dos clubes, são mostrados todos os clubes registados, com o seu nome, numero de equipas, os torneios em que participa e as suas achievements, sendo ainda possivel ver com mais detalhes todas as equipas de cada clube. É possivel filtrar os clubes com base nos atributos necessários, preenchendo o questionário.

Matches: Na área das partidas, é mostrado todas as partidas registadas, os clubes que se confrontam em cada jogo, a pontuação atual, o local, a data, hora, o torneio em que participa, e se a partida já terminou ou não. Para filtar os jogos basta preencher os atributos com os valores pretendidos e filtrar.É possivel observar cada partida com mais detalhes, sendo possivel ver as suas estatisticas, como os golos e os cartões. Assim como é possivel registar eventos da propria partida, selecionando o jogador e o tipo de envento a ser registado.

Torneios: Na área dos torneios, inicialmente é exibido todos os torneios existentes, o se nome, as suas partidas, o estado do torneio e a tabela pontos de cada clube do torneio. É possivel filtrar os torneios com base nos atributos pretendidos.

Exploração de decisões técnicas no JavaFX

A implementação da interface JavaFX baseou-se nos seguintes conceitos e padrões:

Model-View-Controller (MVC) com FXML:

- View (FXML): Definição declarativa das interfaces utilizando FXML, permitindo uma clara separação entre o layout e a lógica.
- Controller: Classes que gerem a interação do utilizador e comunicam com o backend aquando do processamento de eventos
- Model (no âmbito deste projeto receberam o nome de DTO, estando em sintonia com o backend): Objetos de transferência de dados que representam as entidades do sistema.

Todas as áreas seguem a mesma lógica:

- O FXML é carregado
- Um evento acontece (clicar num botão, por exemplo)

- Essa ação é gerida pelo controller, que despoleta a série de acontecimentos necessária para processar os dados, atualizar o modelo, e refletir as mudanças na interface gráfica

Exploração de decisões técnicas no backend

Filtros com recurso a vários Endpoints

Para os filtros abaixo foi utilizada uma estratégia de filtragem um a um. Assim para cada tipo de filtro foi criado um handler específico. Numa primeira instância é realizada uma pesquisa ao repositório para retornar todos os objetos nele registados e, após isso, é realizada a filtragem.

Filtros do Utilizador (Árbitro e Jogador)

```
// Handler que filtra dos jogadores pelo numero maximo ou minimo de golos marcados
public List<ViewModelPlayer> handleFilterPlayersByGoalsScored(Integer maxGoals,
    Integer minGoals) {
        // logica
}

// Handler que filtra dos jogadores pelo numero maximo ou minimo de cartoes mostrados
 public List<ViewModelReferee> handleFilterRefereesByCardsShown(Integer maxCards,
     Integer minCards) {
        // logica
}
```

Queremos que a interface gráfica do Thymeleaf forneça *feedback* positivo na maior parte dos casos para evitar frustração do utilizador e por isso optamos por implementar, por exemplo, o filtro por nome através do uso da query abaixo. Assim, ao invés de retornar um resultado somente estes coincidem na perfeição, o utilizador obtém um resultado quando os caracteres da String do parâmetro ocorrem na String do nome do jogador.

```
public List<ViewModelPlayer> handleFilterPlayersByName(String name) {
    if (name == null || name.trim().isEmpty()) {
        throw new InvalidUserDataException("Name cannot be empty!");
    }
    List<Player> players = pr.findByNameContainingIgnoreCase(name);
    List<ViewModelPlayer> playersDto = new ArrayList<>();
    for (Player player : players) {
        playersDto.add(PlayerMapper.toViewModel(player));
    }
    return playersDto;
}
```

```
// No PlayerRepository
List<Player> findByNameContainingIgnoreCase(String name);
```

Filtros com recurso a Streams do Java

Para os Filtros abaixo foi utilizada uma estratégia de filtragem que utiliza somente 1 endpoint e consequentemente toda a filtragem foi realizada com recurso a somente 1 handler. Numa primeira instância é realizada uma pesquisa ao repositório para retornar todos os objetos nele registados e, após isso, se o parâmetro da assinatura do método não for nulo então é filtrado de acordo com o seu valor com recurso a *streams* do Java.

Filtros da Equipa

```
public List<FormatedClubDto> handleSearchFilterClub(String name, Integer minPlayers,
    Integer maxPlayers, Integer nWins, Integer nDraws, Integer nLosses, Integer
    nAchievements, Integer achievementPosition, String missingPlayerPosition) {
        // logica
}
```

Filtros do Jogo

```
public List<FormatedMatchDto> handleSearchFilterMatch(Boolean isOver, Integer
    minGoals, Integer maxGoals, String stadiumName, String matchPeriod) {
        // logica
}
```

Filtros do Campeonato

```
public List<FormatedPointsTournamentDto> filterByTournament(String name, String
    clubName, Integer minRealizedMatches, Integer maxRealizedMatches, Integer
    minToDoMatches, Integer maxToDoMatches) {
        // logica
}
```