

Projeto de Sistemas Distribuídos

Plataforma scoreDEI

Resultados desportivos em direto

Inês Martins Marçal 2019215917

Noémia Quintano Mora Gonçalves 2019219433

Índice

Descrição da Arquitetura	2
Autenticação	2
Detalhes sobre o controlo de acesso a utilizadores:	3
Endpoints	3
Organização do código e funcionalidades	4
Detalhes de implementações extra efetuadas:	4
Classes que representam as entidades presentes na base de dados:	5
Classes que auxiliam a passagem da informação dos métodos presentes no ProjetoController.java para os respetivos HTMLs	6
Classes Repository e Service implementadas	7
Explicação detalhada dos métodos associados aos <i>endpoints</i> descritos acima	10
Funcionalidades que revelaram ser mais complexas e como foram ultrapassadas	17
Testes efetuados à plataforma	18

Descrição da Arquitetura

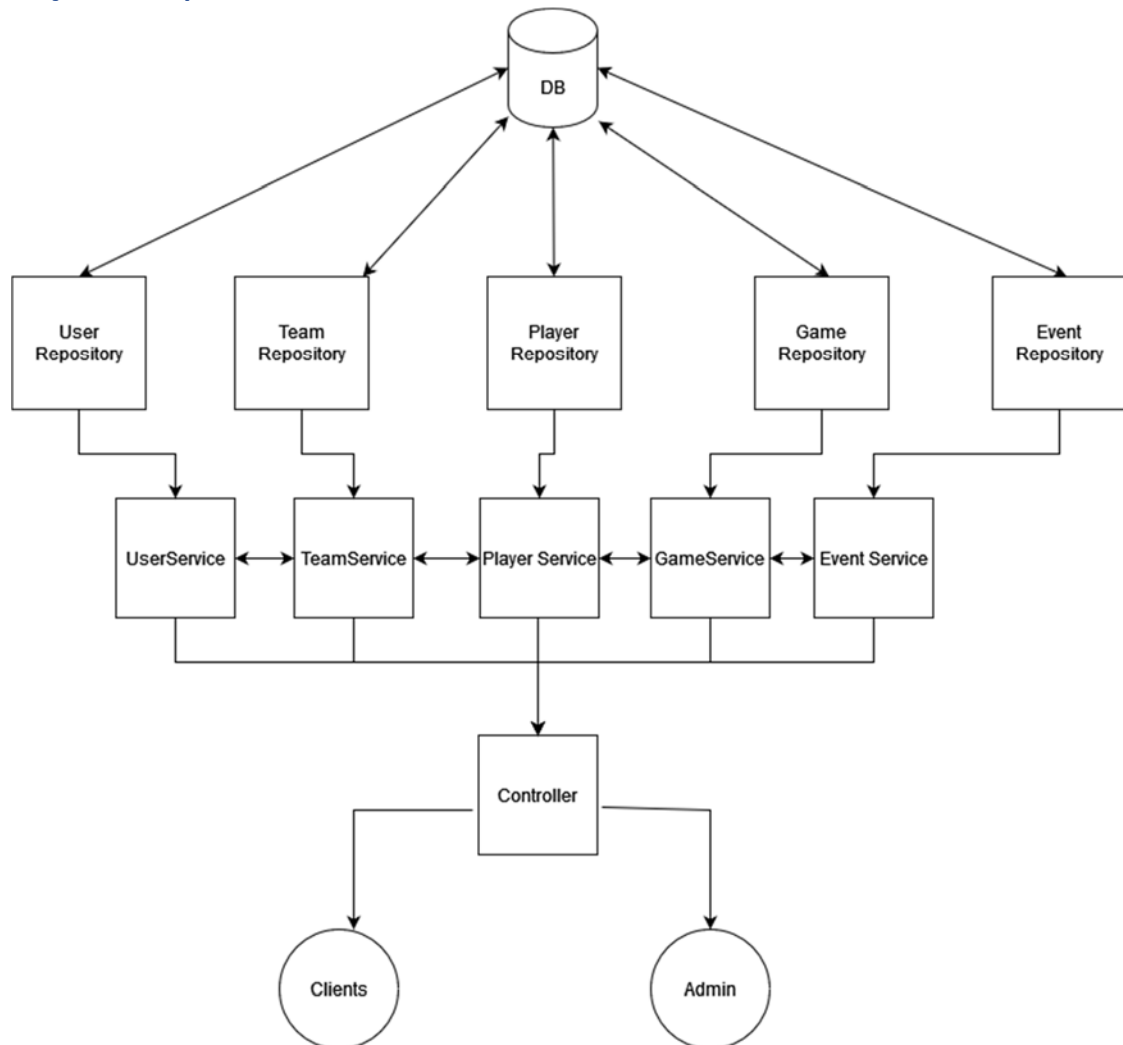


Figura 1: Arquitetura detalhada da plataforma scoreDei

No esquema, as setas permitem visualizar o sentido do fornecimento de serviços, ou seja, o *UserRepository* fornece funções ao *UserService* e este por sua vez fornece-as ao *Controller* para que este responda aos pedidos dos *Clients* e *Administradores*.

Tal como podemos visualizar no esquema, a plataforma apresenta as funcionalidades divididas em diferentes classes, os repositórios das diferentes classes interagem com a base de dados, os serviços, por sua vez, utilizam as funções dos repositórios para implementar a lógica da aplicação e verificar se as restrições implementadas são cumpridas e por fim o *controller* faz uso dos serviços disponíveis para dar resposta aos pedidos dos utilizadores, apresentado as páginas e as informações pedidas.

Autenticação

A autenticação dos utilizadores foi implementada através de uma classe com *session scope*, esta apresenta dois atributos do tipo boolean, o *isAuthenticated* que indica se o utilizador se encontra autenticado e o *isAdmin* que indica se o utilizador tem permissões de administrador ou não. Antes de ser carregada uma página, cujo acesso é apenas permitido a utilizadores autenticados ou a administradores, estas variáveis são consultadas e caso o utilizador não cumpra os requisitos este é redirecionado para a vista */home*. Os valores dos atributos destas

variáveis são apenas mudados quando ocorre um *login* bem-sucedido, um *logout* ou uma sessão terminada.

Detalhes sobre o controlo de acesso a utilizadores:

Os diferentes tipos de utilizador têm na página */home* um menu de acesso adequado às funcionalidades a que lhes foi concedida permissão, sendo estas as seguintes:

Utilizador anónimo -> tem permissões para visualizar as estatísticas e os detalhes dos jogos a decorrer.

Utilizador autenticado -> todas as permissões do utilizador anónimo e também a possibilidade de adicionar eventos a um jogo.

Administrador -> todas as permissões do utilizador autenticado e também a possibilidade de criar equipas, jogadores, jogos e outros utilizadores

Endpoints

De seguida encontram-se os *endpoints(links)* mapeados pelo controlador da aplicação:

GET *"/* -> redireciona para a vista *home* e caso não exista *admins* ou *users* na base de dados este link permite criar um *admin* automaticamente. Posteriormente é apenas necessário fazer login com as seguintes credenciais:

Email: admin@gmail.com

Pass: admin

GET *"/home* -> permite a visualização dos jogos a decorrer e a navegação entre vistas permitidas ao utilizador em questão

GET *"/detais* -> permite a visualização dos detalhes de um determinado jogo

GET *"/login* -> permite ao utilizador realizar a autenticação

GET *"/logout* -> permite ao utilizador sair da sessão

POST *"/authentication* -> utilizado pelo */login* para realizar a autenticação e no final redireciona o utilizador para a vista */home*

GET *"/create-user* -> permite a um administrador criar um novo utilizador

POST *"/save-user* -> utilizado pelo */create-user* para colocar os novos dados na base de dados

GET *"/create-team* -> permite a um administrador criar uma nova equipa

POST *"/save-team* -> utilizado pelo */create-team* para colocar os novos dados na base de dados

GET *"/create-player* -> permite a um administrador inserir um novo jogador numa equipa já existente

POST *"/save-player* -> utilizado pelo */create-player* para colocar os novos dados na base de dados

GET *"/create-game* -> permite a um administrador inserir um novo jogo entre as equipas já inseridas no sistema

POST “/save-game” -> utilizado pelo */create-game* para colocar os novos dados na base de dados

GET “/edit-player” -> permite ao administrador editar um jogador

POST “/save-edit” -> utilizado pelo */edit-player* para colocar os novos dados na base de dados

GET “/edit-team” -> permite ao administrador editar uma equipa

POST “/save-team” -> utilizado pelo */create-team* para colocar os novos dados na base de dados

GET “/edit-game” -> permite ao administrador editar um jogo

POST “/save-game” -> utilizado pelo */create-team* para colocar os novos dados na base de dados

Organização do código e funcionalidades

- Na pasta beans encontra-se o ficheiro *AuthenticationInformation.java*, uma vez que este se trata de ser um bean.

- Na pasta data encontram-se as classes: *Event.java*, *Game.java*, *Player.java*, *Team.java* e *User.java*. Estas classes contêm as entidades/objetos presentes na base de dados.

- Na pasta formData foram colocadas as classes *FormResult.java*, *LoginFormData.java*, *MenuItem.java*, *Stats.java* e *TeamForm.java*. Estas classes permitem auxiliar o processo de passagem da informação dos métodos presentes no ficheiro *ProjetoController.java*, associados a um *endpoint*, para o *HTML*. Estas entidades são passadas vazias para o *HTML*, onde posteriormente os seus atributos parciais dos objetos irão ser preenchidos com a informação fornecida pelo utilizador, ou passam informação para o *HTML* que não é derivada diretamente de uma classe.

- Na pasta projeto encontram-se os ficheiros *Repository.java* e *Service.java* de cada uma das classes presentes na pasta *data*. O repositório contém as *queries* que podem vir a ser utilizadas nos serviços para gerir uma tabela, cada serviço apresenta, assim, um repositório associado. Nesta pasta foram ainda colocados os ficheiros *ProjetoApplication.java* e o *ProjetoController.java* onde se encontram todos os *endpoints* definidos e os métodos associados aos mesmos/funcionalidades.

- Na pasta templates contida na pasta resources encontram-se os *HTMLs* associados a todos os *endpoints*.

Detalhes de implementações extra efetuadas:

- Na página */create-event* ao ser selecionada a equipa na qual o evento ocorreu, apenas será disponibilizado os jogadores dessa mesma equipa, não sendo possível ao utilizador selecionar um que não corresponde à mesma;

- Ao submeter uma entidade criada, por exemplo um evento, o utilizador é redirecionado para uma página *save-event.html* onde não só aparece uma mensagem que indica o sucesso ou insucesso da operação como também é disponibilizado dois *links*, um que permite regressar à página */home* e outro que permite criar outra instância da entidade que estava a ser criada pelo utilizador. Em caso de insucesso da operação é apresentada a razão;

- Não poderá existir duas equipas com o mesmo nome;
- Não poderá existir dois jogadores com o mesmo nome dentro da mesma equipa;
- O utilizador só poderá criar eventos a partir do momento em que o jogo é começado;
- O utilizador não poderá começar, interromper ou pausar o jogo após o mesmo já ter sido terminado;
- O utilizador não poderá terminar novamente o jogo;
- Caso um jogador já tenha 2 cartões amarelos, não poderá criar mais eventos (funciona como se lhe tivesse sido atribuído um vermelho);
- Caso um jogador tenha um cartão vermelho é expulso do jogo, não sendo possível assim selecioná-lo em eventos futuros;
- Caso um utilizador tenha efetuado *login* é disponibilizado uma opção *logout* para sair da sua conta;
- Os resultados apresentados nas estatísticas são dinâmicos pelo que são apresentados no decorrer dos jogos e apresentam a informação do momento em que o utilizador fez o pedido à aplicação;
- Caso exista mais do que um melhor jogador, ou seja, com o mesmo número de golos marcados, é escolhido o primeiro que se encontra nesse mesmo conjunto;
- O utilizador ao visualizar as estatísticas tem a possibilidade de ordenar as estatísticas de acordo com o nome da equipa, com o número de golos, vitórias, derrotas e empates clicando nos títulos correspondentes a estes parâmetros da tabela.

Classes que representam as entidades presentes na base de dados:

Classe Event → Cada evento apresenta um id, um tipo, uma data e hora da ocorrência do mesmo, um jogo e ainda uma equipa associada.

Classe Game → Cada jogo é definido por um id, uma data e hora da ocorrência do mesmo, uma localização e uma *equipaA* e uma *equipaB*. Ao haver eventos associados a este, foi também definida uma lista para os mesmos.

Classe Player → Cada jogador é definido por um id, por um nome, por uma data de aniversário, pela equipa que o representa e ainda pela posição em campo em que joga.

Classe Team → Cada equipa é definida por um id, por um nome, por um logótipo (*url* do mesmo) e ainda por um conjunto de jogadores constituintes da mesma.

Classe User → Um utilizador apresenta um id, um nome, uma password, um e-mail, uma morada, um contacto e por fim uma *flag* a indicar se o mesmo se trata de um *admin* ou não.

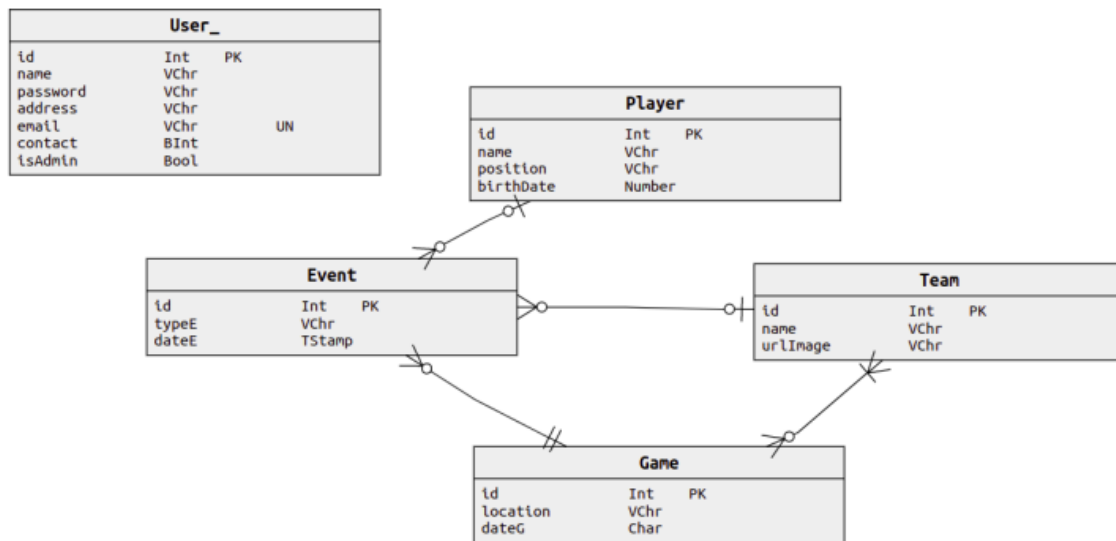


Figura 2: Diagrama de Entidade Relacionamento

Classes que auxiliam a passagem da informação dos métodos presentes no `ProjetoController.java` para os respetivos HTMLs

Classe FormResult → Esta classe permite que após a criação de um objeto (evento, utilizador etc..) seja apresentada uma mensagem a indicar o sucesso ou não da operação. Para isso esta recebe o tipo do objeto que está a ser criado, *entityType* e no final, após ter sido verificado o sucesso ou insucesso da operação, é definido o parâmetro *result* que indica este mesmo estado.

Classe LoginFormData → Esta classe auxilia o processo de *login*, recebendo assim um e-mail e uma *password* fornecidos pelo utilizador. Esta informação é depois passada ao *endpoint* autenticação.

Classe FormEvent → Pretendia-se que após a seleção da equipa que originou o evento apenas fosse disponibilizado ao utilizador os jogadores pertencentes à mesma no *dropdown* seguinte. Pelo que para contornar os conflitos apresentados pelo parâmetro *required* na implementação desta funcionalidade foi criada esta classe que auxilia a passagem de informação entre o método *create-event* e o respetivo *html*. Deste modo, a mesma apresenta como atributos um *playerA*, um *playerB* e ainda um evento. Como o *dropdown* correspondente a este atributo é *required* e havendo dois tipos de jogadores, só que os que não correspondiam à equipa encontravam-se *hidden*, caso fosse utilizado apenas o atributo *player* da classe *Event*, o parâmetro *required* não conseguia perceber de que jogador se tratava.

Classe MenuItem → Permite auxiliar a disponibilização de opções aos utilizadores. Para isto recebe uma *string* que indica a operação em causa e o *link* associado à mesma que irá permitir realizar a operação desejada.

Classe StatsData → Esta classe permite facilitar a passagem das características de cada equipa para o *HTML*, recebendo assim o nome da equipa, o número de jogos realizado pela mesma, o número de vitórias, o número de derrotas e por fim o número de empates.

Classe TeamForm → Permite auxiliar o processo de criação de equipas, apresentando um parâmetro que indica o nome da equipa que se pretende adicionar à base de dados e a flag que indica se se pretende importar os jogadores pertencente à mesma da API.

Classes Repository e Service implementadas

Nos repositórios encontram-se as *queries* que permitem efetuar a pesquisa na base de dados e nos serviços. Posteriormente, recorrendo a estas *queries* presentes nos repositórios e tendo em conta as restrições dos serviços em questão, foram implementados os métodos necessários para obter as funcionalidades pretendidas.

UserRepository.java:

User_ findByEmail(): Query que retorna o *User_* associado ao email

UserService.java:

boolean addUser(User_ user_): Método que recebe como parâmetro o *User_* a ser adicionado ao sistema. A *password* que é adicionada consiste no *hashCode* da *password* inserida originalmente pelo utilizador. Antes da inserção verifica-se se existe algum *User_* já associado ao email colocado, caso exista a função retorna *false*. Por outro lado, se não existir, o utilizador é colocado na base de dados e a função retorna *true*.

int authenticate(LoginFormData loginData): Método que recebe como parâmetro uma instância da classe auxiliar *loginFormData*, onde se encontram o email e *password* do correspondente pedido de autenticação. Este método começa por verificar se existe um utilizador cujo email associado é o que foi inserido e, caso exista, verifica ainda se o *hashCode* da *password* inserida corresponde ao que se encontra na base de dados. Caso a autenticação esteja correta é verificado se o utilizador possui permissões de administrador, caso possua, a função retorna 1, caso contrário retorna 2. Se os dados inseridos não corresponderem aos presentes na base de dados o valor retornado é 0.

TeamRepository.java:

Team findByName(String name): Query que retorna a *Team* cujo atributo *name* seja igual ao parâmetro *name* recebido.

TeamService.java:

boolean addTeam(TeamForm teamForm): Método que recebe como parâmetro uma instância da classe auxiliar *teamForm*. Esta, por sua vez, recolhe os dados da API externa e adiciona-os à base de dados. Deste modo, começa-se por verificar se já existe uma equipa no repositório com o mesmo nome da que se pretende adicionar e caso já exista a função retorna *false*. Após essa verificação, é procurada a equipa na API externa, caso não exista, optou-se por não a adicionar e a função retorna *false*, caso contrário é adicionada à base de dados. Se se tiver ainda selecionado a opção *importPlayers* irão ser adicionados os dados dos jogadores extraídos da API externa à base de dados e a função retorna *true*.

List<Team> getAllTeams(): Método que retorna uma lista com todas as equipas existentes na base de dados.

StatsData getStats(String team): Método que recebe como parâmetro uma *String team* e retorna uma instância da classe auxiliar *getStats* que apresenta as estatísticas da equipa cujo

atributo *name* é igual ao parâmetros *team*. Começa-se por recorrer à função do *gameService* *getAllGamesTeam* que retorna todos os jogos em que uma equipa participou. Para cada jogo utiliza-se à função *getScoreTeam* do *eventService* que retorna o resultado do jogo, verificando-se, posteriormente, se o mesmo corresponde a uma vitória, derrota ou empate, guardando toda essa informação na instância *stats* da classe *getStats* que depois é retornada.

PlayerRepository.java:

Player findbyNameAndTeam(String name, String team): Query que procura um jogador cujos atributos *name* e *team* correspondam aos respetivos parâmetros da função *name* e *team*.

List<String> getBestScore(): Query que permite obter a lista jogadores ordenados por ordem decrescente do número de golos marcados. A mesma é utilizada para calcular o melhor marcador de golos.

int findByName(String name): Query que permite encontrar quantos jogadores tem este nome passado como parâmetro.

void changePlayer(String newName, String name, String position, Date birthDate): Query que permite alterar um jogador com um determinado nome.

List<Player> getAllPlayers(): Query que permite obter todos os jogadores.

PlayerService.java:

boolean addPlayer(Player player): Método que recebe como parâmetro um jogador *player*. Este método começa por verificar se o mesmo já encontra na base de dados, em particular, na mesma equipa um jogador com o mesmo nome, se tal se verificar, é retornado o valor *false* uma vez que foi definido que não poderia existir jogadores com o mesmo nome dentro da mesma equipa. Caso contrário o valor retornado é *true*.

public String BestScorer(): Verifica-se inicialmente se a lista que contém os jogadores ordenados por ordem decrescente do número de golos marcados contém algum elemento. E caso isto se verifique, é retornado o primeiro elemento dessa lista, ou seja, o melhor marcador de golos de todos os jogos efetuados.

List<Player> getAllPlayers(): Permite obter todos os jogadores.

int findByName(String name): Permite encontrar um jogador com o nome *name*.

void changePlayer(String newName, String name, String position, Date birthDate): Permite alterar um jogador existente recebendo um novo nome, posição e data de nascimento.

GameRepository.java:

Game findbyId(String id): Query que procura o jogo cujo atributo *id* corresponde ao parâmetros *id*.

List<Game> findbyTeam(String team): Query que procura todos os jogos em que uma equipa jogou, quer como *teamA* ou como *teamB*.

List<Event> OrderEventOfGame(Game game): Query que permite obter todos os eventos de um game e ordena-los, posteriormente, por ordem cronológica.

GameService.java

boolean addGame(Game game): Método que recebe como parâmetros o *game* que se pretende adicionar e retorna um *boolean*. Começa-se por verificar se a equipa não vai jogar com ela própria, se tal situação se verificar a função retorna *false*, caso contrário o mesmo é adicionado à base de dados e é retornado *true*.

List<Game> getAllGames(): Método que retorna a lista de todos os jogos.

List<Game> getAllGamesTeam(String team): Método que recebe como parâmetros a *String team* e retorna a lista de jogos onde a equipa cujo atributo *name* é o parâmetro *team*.

List<Game> findById(int id): Método que recebe como parâmetro o id de um jogo, retornando este jogo, posteriormente.

List<Event> OrderEventOfGame(Game game): Método que recebe um jogo e que retorna, posteriormente, a lista de todos os eventos do mesmo ordenada de forma cronológica.

EventRepository.java:

List<Event> findbyTypeE(String type, int id): Query que retorna a lista de eventos do tipo *type* do jogo cujo atributo id corresponde ao parâmetro id .

List<Event> allEventsOrderByDate(int id): Query que retorna uma lista ordenada dos eventos do jogo cujo atributo id é o valor do parâmetro id.

int countYellowCards(int id, String playerName): Query que retorna o número de cartões amarelos que o jogador, cujo atributo nome é o parâmetro *playerName*, tem até ao momento num jogo, cujo o atributo id é o parâmetro id.

int countRedCards(int id, String playerName): Query que retorna o número de cartões vermelhos que o jogador, cujo atributo nome é o parâmetro *playerName*, tem até ao momento num jogo, cujo o atributo id é o parâmetro id.

int getScoredbyTeamAndGame(String team, int gameId): Query que retorna o número de golos que a equipa, cujo atributo nome é o parâmetro *team* tem até ao momento num jogo, cujo atributo id é o parâmetro *gameId*.

EventService.java:

boolean addEvent(Event event): Método que recebe como parâmetro o evento que se pretende adicionar à base de dados. Começa-se por verificar se já existem eventos para o jogo em questão, caso o evento a ser adicionado seja o primeiro é verificado se o mesmo é do tipo "Start of the Game", caso não o seja, é retornado 1. É também verificado se já foi inserido um evento "End of Game" e se tal se verificar, a função retorna 2. Seguidamente, fazem-se verificações de acordo com o tipo de evento que se está a receber, se for um evento do tipo "Start of the Game" verifica-se se já existe um evento de início de jogo para o jogo em questão e se tal acontecer a função retorna 3. Para o evento do tipo "Goal" verifica-se apenas se o jogador em questão possui cartões vermelhos ou dois ou mais cartões amarelos anteriores ao evento que se pretende adicionar, caso isso aconteça é retornando o valor 4. Em relação, ao evento do tipo "Red Card", antes de ser adicionado o evento pretendido, é verificado se o jogador apresenta já um cartão vermelho ou se já apresenta dois ou mais cartões amarelos, sendo retornado o valor 4 caso isso aconteça. Dois ou mais eventos "Yellow Card" para um mesmo jogador resulta na expulsão do mesmo, não podendo realizar mais eventos, sendo retornado também o valor 4. Caso o evento a adicionar não esteja contra nenhuma das

restrições, este é adicionado à base de dados e a função retorna o valor 0. Decidiu-se retornar um valor diferente para cada condição no sentido de ser possível, no método *save-event()*, ao se adicionar um evento, mostrar ao utilizador se o mesmo foi adicionado com sucesso ou insucesso. Neste último caso, é indicado de acordo com o valor retornado por este método a razão do insucesso.

Failure creating event

The player has been sent off

[Home](#) [Create another instance](#)

Figura 3: Exemplo de mensagem da criação de um evento em que o jogador já tinha um cartão vermelho

List<Event> getAllEvents(): Método que retorna uma lista com todos os eventos.

List<String> getTypeEvents(): Método que retorna a lista de tipos possíveis para um evento.

int getScoreTeam(String team, int gameld): Método que retorna a quantidade de golos que a equipa, cujo atributo *name* é o parâmetro *team*, tem no jogo em causa, cujo atributo *id* é o parâmetro *gameld*.

Explicação detalhada dos métodos associados aos *endpoits* descritos acima

Ao *endpoit /home* encontra-se associado o método *home()* que recebe como parâmetro o *model* que, por sua vez, permite a passagem de informação deste método para o respetivo *HTML*. No sentido de ser possível quer aos utilizadores autenticados quer aos não autenticados acompanhar o jogo é passado o atributo *allGames* para o *HTML*, podendo-se assim visualizar todos os jogos existentes. Para que o utilizador possa escolher outras ações é apresentado um menu com um conjunto de opções, sendo que cada uma tem um *link* associado. Deste modo definiu-se uma lista de *MenuItems*, classe explicada no tópico anterior, através da qual é possível alterar a vista consoante seja um administrador, um utilizador autenticado ou não autenticado.

Para um utilizador não autenticado, o menu consiste apenas na opção de login, já o menu de um utilizador autenticado apresenta todas as opções do não autenticado e ainda um link que permite ao utilizador acrescentar eventos a um jogo que esteja a decorrer. Por outro lado, um administrador tem acesso a tudo o que um utilizador autenticado tem e no seu menu aparecem ainda as opções de criar um utilizador, criar uma equipa, criar um jogador ou criar um jogo. A partir do momento em que um utilizador ou administrador efetua o log in corretamente fica também disponível um botão de log out.

Além destas opções, para ambos é também possível visualizar as estatísticas de cada equipa. Tal como referido acima, este menu é passado, posteriormente, para o *HTML*. Para o caso em utilizador tenha colocada as credências erradas ou não esteja registado na base de dado foi definido que este método recebe um atributo *String error*, proveniente do método *authentication()*. Este, por sua vez, permite indicar ao utilizador que foram introduzidas as credências erradas ao este ser redirecionado para a página */home*.

No *HTML*, que permitirá dar origem à *vista home.html* são percorridas as opções que se encontram no atributo do *HTML*, *MenuItems*, proveniente do método *home()*, sendo apresentado o tipo de opção através de uma *string* e o *link* associado à mesma. Com vista a que seja apresentado uma listagem dos jogos existentes é mostrado para cada um o nome da *equipaA*, o nome da *equipaB*, a localização do jogo e os detalhes do mesmo. Este método recebe também, proveniente do método *home()*, outro atributo *HTML*, o *logged*, que indica se o utilizador efetuou o *login* ou não, e caso o tenha feito é apresentado a mensagem *welcome*.

ScoreDEI				ScoreDEI			
Menu				Welcome			
Login Show Stats				Logout Create User Create Team Create Player Create Game Show Stats			
Games				Games			
Team A	Team B	Location		Team A	Team B	Location	
Ajax	Manchester United	Portela	City Show details	Ajax	Manchester United	Portela	City Show details Register Events
Flamengo	Ajax	Lisboa	Show details	Flamengo	Ajax	Lisboa	Show details Register Events

Figura 4: Exemplo de diferentes menus: à esquerda o de um utilizador não autenticado e à direita o de um administrador

Ao *endpoint /show-stats* encontra-se associado o método *showStats()* que recebe um *int* como parâmetro que identifica o parâmetro pelo qual as estatísticas aparecem ordenadas. Este método permite enviar para o *HTML* as estatísticas correspondentes à equipa em causa. Para isso é obtido através do método *getAllTeams()* da classe *teamService* as equipas existentes, a estas é aplicado o método *getStats()* que através da avaliação dos eventos corresponde a cada jogo permite saber o resultado e contabilizar as vitórias, derrotas, empates e número de jogos. Estas estatísticas são ordenadas segundo o parâmetro escolhido através de um *switch* que tem uma função *lambda* para a comparação de cada parâmetro. Após a ordenação e posterior inversão, necessário para os parâmetros numéricos, as estatísticas são enviadas para o *HTML* juntamente com os atributos *BestScorerName* e o *BestScorerGoals* que indicam, respetivamente, o nome e o número de golos do melhor marcador. No final é retornado a página *showStats.html*, onde são apresentadas as estatísticas relativas a cada equipa.

No *HTML* que dá origem à *vista showStats.html* é percorrido o atributo *allStats*, vindo do método referido anteriormente, que apresenta as estatísticas de cada uma das equipas. Ao mesmo tempo que é efetuada esta operação é construída a tabela que irá apresentar toda esta informação. Para mudar a ordem de apresentação clica-se no cabeçalho correspondente ao parâmetro pelo qual se quer realizar a ordenação.

[Home](#)

Stats

Team	Victories	Defeats	Ties	Number of Games
Flamengo	1	0	0	1
Manchester United	0	0	1	1
Ajax	0	1	1	2
Chelsea	0	0	0	0
Real Madrid	0	0	0	0
Barcelona	0	0	0	0
Liverpool	0	0	0	0
Benfica	0	0	0	0

Best Scorer	Number of Goals
Hugo Moura	1

Figura 5: Parte da vista das estatísticas em que as equipas se encontram ordenadas por número de vitórias

Ao endpoint `/details` encontra-se associado o método `showdetails()`, que, por sua vez, recebe como parâmetro o id do jogo para que seja possível procura-lo na base de dados. Caso este exista são passados para o respetivo *HTML* os seguintes atributos: um atributo *OrderEvent* que corresponde aos eventos de um jogo ordenados de forma cronológica, que por sua vez, foram obtidos pelo método *OrderEventOfGame()* da classe *gameService*; um atributo *game* que apresenta os detalhes de um jogo (url da imagem, nome das duas equipas e ainda a localização do mesmo); e um atributo *resultA* e *result B* que contém o resultado de cada uma das equipas respetivamente naquele jogo.

Deste modo, *no details.html* é apresentado o nome de cada equipa com o *url* do logotipo associado, o número de golos da equipaA e da equipaB e ainda a localização do jogo em causa. Para que seja possível visualizar os eventos associados a este jogo é criada uma tabela que indica o tipo de evento, o jogador que o efetuou, caso o tipo de evento o exija, e ainda a data e hora em que o mesmo ocorreu.

[Home](#)

Game Details

Team A: Flamengo



Goals: 1

Team B: Ajax



Goals: 0

Location: Lisboa

Events

Event	Player	Time
Start of the Game		29-05-2022 02:16
Goal	Luan Sales do Nascimento	29-05-2022 02:16
Paused Game		29-05-2022 02:19

Figura 6: Parte da vista onde são apresentados os detalhes de um jogo existente

O método `loginForm()` encontra-se associado ao `endpoint /login`. Através deste é enviado um objeto `LoginFormData` vazio, explicado no tópico classes anterior, para o HTML.

Por sua vez, `no login.html` é definida duas caixas de texto, uma onde utilizador irá colocar o email definido para a sua conta e outra para a respetiva `password`. A informação introduzida pelo utilizador nestas caixas é colocada nos campos email e `password` do `LoginFormData`. No final, o utilizador submete estes dados e o `LoginFormdata` é enviado para o `endpoint /authentication`.

[Home](#)

ScoreDei Log in



Email

Password

Login

Figura 6: Parte da vista onde é efetuado o login

O método `authentication()`, associado ao `endpoint /authentication`, recebe o objeto `LoginFormData` proveniente do `login.html`, tal como referido anteriormente. Por sua vez, este é passado como parâmetro para o método `authenticate()` do `userService`. Posteriormente, caso este método, onde é verificado se o utilizador existe e se a `password` corresponde ao mesmo, retorne um valor diferente de zero passa-se a indicar que o mesmo se encontra autenticado nesta sessão. Ao mesmo tempo se o valor retornado for igual a 1 é também indicado que nesta sessão o utilizador se trata de um administrador. No final, é redirecionado para a página `/home`.

O método `createUserForm()`, associado ao `endpoint /create-user`, permite passar para o HTML um objeto `user_` vazio. No final deste é retornado a página `create-user.html`.

Por sua vez, `o create-user.html` recebe como parâmetro um `user_` vazio, proveniente do método referido anteriormente. Aqui é definido uma caixa de texto para os seguintes parâmetros do `user_`: nome, e-mail, password, contacto e morada. Simultaneamente, os dados obtidos nestas caixas de texto irão preencher os respetivos campos do objeto `user_`. De seguida, é definido também duas opções (`true` ou `false`) e que o utilizador seleccionar é a que irá preencher o campo `isAdmin` do `user_`. No fim, o objeto `user_` é enviado para o `endpoint /save-user`.

[Home](#)

User Registration



Name

Email

Password

Address

Contact

Admin Permissions
☐ True
☒ False

Save User

Figura 7: Parte da vista onde são apresentados os detalhes de um jogo existente

Tal como referido acima, `o método save-user` recebe como parâmetro o `user_` proveniente do `create-user.html`. Neste método é criado um objeto `FormResult`, explicado anteriormente, para a entidade `user`. Consoante o retorno do método `addUser` da `class UserService` é definido o resultado da operação `create-user` que irá ser passada para o `FormResult` criado anteriormente. Deste modo, caso tenha sido possível criar o `user` é definido o resultado da operação como sendo `Sucess`, caso contrário como sendo `Failure`.

Posteriormente, o objeto *FormResult* é passado para o *result.html*. Por sua vez, neste é definido uma mensagem a indicar o resultado da criação o *user*, utilizando os parâmetros recebidos através do *FormResult*. É criado ainda um *endpoint* para a página */home* e outro que permite criar outra instância do mesmo objeto.

Sucess creating user

[Home](#) [Create another instance](#)

Figura 8: Parte da vista que indica o sucesso ou insucesso da operação efetuada

No método *logout()*, associado ao *endpoint /logout*, é alterado não só o valor do atributo *isAdmin* (caso o utilizador naquela sessão for um administrador) como também o atributo *isAuthenticated* (que indica que o utilizador se encontra autenticado naquela sessão) para *false*. Redireciona-se, assim, no final o utilizador para a página */home*.

No método *create-player()*, associado ao *endpoint /create-player*, é passado para o *create-player.html* não só um atributo *player* relativo ao objeto *player* vazio criado como também um atributo *allteams* referente às equipas existentes, obtidas através do método *getAllTeams()* da classe *teamService*. No final deste método é retornado a página *create-player.html*.

Por sua vez, o *create-player.html* recebe como parâmetro o atributo *player* proveniente do método descrito anteriormente. Aqui é definido uma caixa de texto para o nome do *player*, cuja a informação inserida na mesma irá preencher o campo *name* deste objeto. De seguida é definido um *dropdown* não só para a posição dos jogadores no campo como também para a escolha da equipa a que o jogador pertence. Por outro lado, para que se possa escolher a equipa a que o jogador pertence é percorrido o atributo *allTeams*, proveniente do método *create-player()*, sendo assim possível apresentar o nome de cada equipa no *dropdown*. É também pedido ao utilizador que indique a data de nascimento do jogador que está a ser criado. No final, é enviado o *player* para o *endpoint /save-player*.

Deste modo, o método *savePlayer Submisson()*, associado ao *endpoint /save-player*, recebe como parâmetro o *player* proveniente do *save-player.html*. Aqui é criado um novo objeto *FormResult* para a entidade *player*. Consoante o resultado do método *addPlayer* da classe *servicePlayer*, responsável por verificar se o jogador existe na base de dados, é definido o resultado da operação *create-player* que irá ser passada para o *FormResult* criado. Deste modo, caso tenha sido possível a criação do jogador, o parâmetro *result* da classe *FormResult* é definido como *Success*, caso contrário como *Failure*. De seguida, o objeto *FormResult* é enviado para o respetivo *HTML*, explicado no *save-user*. No final é retornado a página *result.html*.

Home Player Registration

Name

Player's position

Date

Liverpool
Barcelona
Real Madrid
Chelsea
Ajax
Manchester United
Flamengo

Figura 9: Parte da vista onde é efetuado o registo do jogador

No método `createTeamForm()`, associado ao `endpoint create-team`, é definido um atributo `team` como sendo um novo `TeamForm()`, que é, consequentemente, enviado para o `create-team.html`.

Por sua vez, `este create-team.html` recebe como parâmetro o atributo `team` proveniente do método descrito anteriormente. É ainda criado uma caixa de texto que irá permitir preencher o campo nome do objeto `TeamForm()`. Além desta caixa de texto é ainda apresentado ao utilizador duas opções, cuja escolha irá definir o valor do atributo `importPlayers` do objeto `TeamForm()`. No final, o utilizador ao submeter a criação da equipa, o atributo `team` é enviado para o `endpoint /save-team`.

Consequentemente, o método `saveTeam Submission()` recebe como parâmetro o `TeamForm()` enviado pelo `create-team.html`. Após a criação de um novo objeto `FormResult` para a entidade `team` é efetuado o mesmo raciocínio que o descrito para o método `savePlayer_Submisson()`.

No método `createGame()`, associado ao `endpoint /create-game`, é enviado para o `HTML` não só um jogo vazio, definido como atributo `game`, como também todas as equipas existentes através do atributo `allTeams`. Este último é obtido a partir do método `getAllTeams()` da classe `teamService`.

Por sua vez, o `create-team.html` recebe como parâmetro o atributo `game`, proveniente do método referido anteriormente. Aqui é criada uma caixa de texto, cuja informação irá preencher o campo correspondente à localização do jogo que está a ser criado. Além desta caixa de texto, foram também definidos dois `dropdowns`, em que em cada um é possível selecionar a equipa que se pretende para a realização do jogo em causa. As equipas apresentadas no `dropdown` foram obtidas de modo análogo ao explicado nos métodos anteriores, ou seja, através do atributo `allTeams` proveniente do método `createGame()`. O utilizador terá ainda de preencher a data em que se irá realizar o jogo que está a criar. O objeto `game` é enviado, no final, para o `endpoint /save-game`.

Deste modo, o método `saveGame()` recebe como parâmetro o objeto `game` proveniente do `create-game.html`. Após a criação de um novo objeto `FormResult` para a entidade `game` procede-se ao mesmo raciocínio que o efetuado para o método `savePlayer_Submisson()`.

[Home](#)

Team Registration

Name

Import Players

☐ True

☒ False

Figura 10: Parte da vista onde é efetuado o registo da equipa

[Home](#)

Game Registration

Location

Date

Team A

Benfica
Liverpool
Barcelona
Real Madrid
Chelsea
Ajax
Manchester United
Flamengo

Team B

Benfica
Liverpool
Barcelona
Real Madrid
Chelsea
Ajax
Manchester United
Flamengo

Figura 11: Parte da vista onde é efetuado o registo de um jogo

Por último, o método `createEvent()`, associado ao endpoint `/create-event`, recebe como parâmetro o id do jogo que originou o evento, procurando se o mesmo existe na base de dados, através do método `findByid` da classe `GameService`. Caso o jogo exista, é passado para o `create-event.html` um atributo `game` que representa os detalhes do jogo em causa, um atributo `fevent` que corresponde a um objeto `FormEvent` vazio e ainda o atributo `typesevents` que foi obtido pelo método `getTypeEvents()` da classe `eventService`. No final é retornado a página `create-event.html`. Porém, caso o jogo não exista na base de dados, o utilizador é redirecionado para a página `/home`. Qualquer utilizador não autenticado é também redirecionado para a página `/home`, uma vez que a operação de criar eventos é apenas permitida a utilizadores autenticados.

Por sua vez, o `create-event.html` recebe como parâmetro o objeto `FormEvent` proveniente do método referido acima. Neste são criados 3 *dropdowns*, um para o tipo de evento que se pretende criar, outro para a equipa no qual o mesmo ocorreu e outro para o jogador que o efetuou. É ainda pedido ao utilizador que indique a data do evento. Após selecionadas as opções, as mesmas irão preencher os campos dos atributos do objeto `FormEvent`. Como é necessário atribuir o valor do id do `game` ao parâmetro `game` da classe evento, este é obtido diretamente pelo atributo `game` proveniente do método `createEvent()`, não sendo assim necessário ser mostrado ao utilizador, pelo foi definido como *hidden*.

Além dos ids definidos para cada um dos dropdowns, foi ainda definido outro que englobasse o *dropdown* das equipas e dos jogadores de forma a que fosse possível esconder ou mostrar os mesmos em conjunto. Relativamente ao dropdown events, sempre que fosse selecionado uma opção no mesmo era efetuado uma chamada à função `check()`, implementada em *JavaScript*. Esta por sua vez, permite obter o elemento do *HTML* que apresenta o id events, neste caso o *dropdown*, e, posteriormente, a opção que foi selecionada no mesmo. Caso o evento escolhido tenha sido um golo, um cartão amarelo ou um cartão vermelho são pedidos aos utilizadores obrigatoriamente os 3 *dropdowns* mais a data em que ocorreu o evento. Caso contrário o *dropdown* das equipas e dos jogadores é escondido.

Por outro lado, em relação ao *dropdown* das equipas, sempre que é selecionada uma opção no mesmo é chamada a função `hideplayer()`, implementada em *JavaScript*. Tal como também já foi mencionado antes, através do id teams é possível saber a opção selecionada. Perante isto, se a equipa A for a selecionada apenas irão ser mostrados os jogadores pertencentes a esta equipa. No final é enviado o objeto `fevent` para o endpoint `/save-event`.

[Home](#)

Register Events

Type of event

Team

Player

Date

Figura 12: Parte da vista onde é efetuado o registo de um evento

[Home](#)

Register Events

Type of event

Goal

Team

Choose Option

Player

Choose Option

Date

05/29/2022, 06:59 PM

Save Event

[Home](#)

Register Events

Type of event

Start of the Game

Date

05/29/2022, 06:59 PM

Save Event

Figura 13: Alteração da vista onde é efetuado o registo de um evento consoante o tipo do mesmo

Explicando mais detalhadamente a função `check()` e a função `hideplayer()` implementadas em JavaScript no `create-event.html`:

Na função `check()`, após se ter obtido a opção que foi selecionada no *dropdown* das equipas, é verificado se esta corresponde a um evento do tipo golo, cartão amarelo ou cartão vermelho, e caso isto se verifique o *dropdown* das equipas e dos jogadores é mostrado, definindo-os através dos ids como sendo *required*. Caso contrário são escondidos.

Por outro lado, na função `hideplayer()` é verificado qual das equipas foram selecionadas, no caso de ter sido a equipa A, apenas é mostrado no *dropdown* seguinte os jogadores correspondentes a esta equipa, sendo colocado escondidos os jogadores da equipa B. Deste modo, os jogadores correspondentes à equipa A foram definidos como um parâmetro *required* e os restantes não *required*. No caso de ter sido escolhida a equipa B é efetuado o procedimento contrário.

A classe `FormEvent`, que contém como atributos além de um evento, um `playerA` e um `playerB`, foi criada no sentido de contornar os conflitos apresentados pelo parâmetro *required* do `html`, que, por sua vez, foi atribuído a todos os *dropdowns*. Assim, é necessário que seja, posteriormente, definido qual destes dois jogadores irá ocupar o campo `player` de um objeto evento. Para isso, o método `savePlayer_Submisson()` verifica, inicialmente, qual dos jogadores se encontra a `null` e o que não apresentar este estado irá definir o campo `player` do respetivo evento. Posteriormente, procede-se ao mesmo raciocínio que o efetuado para o método `savePlayer_Submisson()`, só que neste caso para a entidade `event`.

Funcionalidades que revelaram ser mais complexas e como foram ultrapassadas

- Guardar uma imagem diretamente na base de dados era complexo por isso optou-se por guardar o `url` da imagem do logo que se obteve pela `API` como um atributo da classe `Team`. Deste modo, quando é necessário apresentá-la numa vista é o `html` que carrega o `url` da imagem associado à equipa.

- A funcionalidade de ordenar a tabela das estatísticas mostrou-se complexa, sendo que acabou por utilizar-se o método `getStats()` em conjunto com um parâmetro passado pelo `url` que permite identificar a ordenação pretendida. Optou-se por esta abordagem pois apenas uma `query` apresentava uma elevada complexidade e desta forma podem ser adicionadas novas



ordenações sem dificuldade, adicionando apenas o case no *switch* e fazendo a função lambda de comparação.

- A funcionalidade extra de apenas ser disponibilizado ao utilizador os jogadores pertencentes à equipa seleccionada na criação de um evento gerou algumas dificuldades ao nível do *HTML*. Para a realizar acabou por se implementar duas funções de *JavaScript* que envolveram uma pesquisa adicional sobre estes conceitos.



Testes efetuados à plataforma


- De seguida encontram-se os testes efetuados à plataforma para cada uma das funcionalidades pretendidas. Os que passaram com encontram-se assinalados com um *check* e os que reprovaram com uma cross.

Logout



Descrição	Passed	Failed
Ser possível a um utilizador sem privilégios carregar no botão <i>logout</i>	O utilizador sem privilégios consegue carregar no botão 	O utilizador sem privilégios não consegue carregar no botão
Ser possível a um utilizador com privilégios de administrador carregar no botão <i>logout</i>	O utilizador com privilégios de administrador consegue carregar no botão 	O utilizador com privilégios de administrador não consegue carregar no botão

Menu Home



Descrição	Passed	Failed
O menu <i>home</i> disponibilizado a um utilizador não autenticado deverá apresentar as opções <i>de login</i> , <i>show stats</i> e <i>ainda show details</i> para cada jogo existente	O menu <i>home</i> apresenta as caraterísticas descritas 	O menu <i>home</i> não apresenta as caraterísticas descritas
O menu <i>home</i> disponibilizado a um utilizador autenticado sem privilégios deverá apresentar as opções de <i>logout</i> , <i>show stats</i> e <i>ainda as opções show details e register events</i> para cada jogo existente	O menu <i>home</i> apresenta as caraterísticas descritas 	O menu <i>home</i> não apresenta as caraterísticas descritas




O menu <i>home</i> disponibilizado a um utilizador autenticado com privilégios de administrador deverá apresentar as opções de <i>logout</i> , <i>show stats</i> , <i>create user</i> , <i>create team</i> , <i>create player</i> , <i>create game</i> e ainda as opções <i>show details</i> e <i>register events</i> para cada jogo existente	O menu <i>home</i> apresenta as características descritas 	O menu <i>home</i> não apresenta as características descritas
--	--	---

Login




Descrição	Passed	Failed
O utilizador efetuou um login válido	O utilizador consegue entrar na plataforma 	O utilizador não consegue entrar na plataforma
O utilizador efetuou um login inválido	O utilizador não consegue entrar na plataforma, sendo redirecionado para a página <i>home</i> onde irá ser apresentado a mensagem <i>invalid credentials</i> 	O utilizador consegue entrar na plataforma




Create-user

Descrição	Passed	Failed
Colocar letras no campo contacto	Quando o utilizador submete os dados é apresentada uma mensagem a indicar para o mesmo preencher o campo contacto com o formato correto 	O utilizador consegue submeter os dados e consequentemente criar o <i>user</i>
Colocar um email no formato errado no respetivo campo	Quando o utilizador submete os dados é apresentada uma mensagem a indicar para o mesmo preencher o campo email com o formato correto utilizando um @ 	O utilizador consegue submeter os dados e consequentemente criar o <i>user</i>





O utilizador anónimo tenta aceder à página de criação de um utilizador pelo endereço	O utilizador anónimo é redirecionado para a página inicial 	O utilizador anónimo consegue aceder à página de criação de um utilizador
O utilizador autenticado tenta aceder à página de criação de um utilizador pelo endereço	O utilizador autenticado é redirecionado para a página inicial 	O utilizador autenticado consegue aceder à página de criação de um utilizador
Um administrador submete um novo utilizador sem ter preenchido todos os campos	É pedido para preencher os campos em falta 	O novo utilizador é guardado na base de dados


Create-Team

Descrição	Passed	Failed
O administrador não consegue criar uma equipa se a mesma não existir na API	O administrador não consegue criar a equipa e ao submeter os dados irá ser redirecionado para o <i>endpoint /save-team</i> , onde é apresentado uma mensagem a indicar que a operação foi falhou 	O administrador consegue criar a equipa
O administrador consegue criar uma equipa utilizando as disponíveis na API e posteriormente importar os jogadores da mesma	O administrador consegue criar a equipa e ao submeter os dados irá ser redirecionado para o <i>endpoint /save-team</i> , onde é apresentado uma mensagem a indicar que a operação foi bem-sucedida 	O administrador consegue criar a equipa
O administrador consegue criar uma equipa utilizando as disponíveis na API sem importar os jogadores da mesma	O administrador consegue criar a equipa e ao submeter os dados irá ser redirecionado para o <i>endpoint /save-team</i> , onde é apresentado uma mensagem a indicar que a operação foi bem-sucedida 	O administrador consegue criar a equipa




O utilizador anónimo tenta aceder à página de criação de uma equipa pelo endereço	O utilizador anónimo é redirecionado para a página inicial 	O utilizador anónimo consegue aceder à página de criação de uma equipa
O utilizador autenticado tenta aceder à página de criação de uma equipa pelo endereço	O utilizador autenticado é redirecionado para a página inicial 	O utilizador autenticado consegue aceder à página de criação de uma equipa
Um administrador submete uma nova equipa sem ter preenchido todos os campos	É pedido para preencher os campos em falta 	A nova equipa é guardada na base de dados





Create-Player





Descrição	Passed	Failed
O administrador consegue criar um jogador e adicioná-lo às equipas existentes na base de dados	O administrador consegue criar o jogador e ao submeter os dados irá ser redirecionado para o <i>endpoint /save-player</i> , onde é apresentado uma mensagem a indicar que a operação foi bem-sucedida 	O administrador consegue criar o jogador
O administrador não consegue criar um jogador com o mesmo nome que um já existente na base de dados na mesma equipa	O administrador não consegue criar o jogador e ao submeter os dados irá ser redirecionado para o <i>endpoint /save-player</i> , onde é apresentado uma mensagem a indicar que a operação falhou 	O administrador consegue criar o jogador
O utilizador anónimo tenta aceder à página de criação de um jogador pelo endereço	O utilizador anónimo é redirecionado para a página inicial 	O utilizador anónimo consegue aceder à página de criação de um jogador
O utilizador autenticado tenta aceder à página de criação de um jogador pelo endereço	O utilizador autenticado é redirecionado para a página inicial 	O utilizador autenticado consegue aceder à página de criação de um jogador





Um administrador submete um novo jogador sem ter preenchido todos os campos	É pedido para preencher os campos em falta 	O novo jogador é guardado na base de dados
---	---	--



Create-Event

Descrição	Passed	Failed
O utilizador autenticado consegue criar um evento do tipo golo	O utilizador autenticado consegue criar um evento do tipo golo e ao submeter os dados irá ser redirecionado para o <i>endpoint /save-event</i> , onde é apresentado uma mensagem a indicar que a operação foi bem-sucedida 	O utilizador autenticado não consegue criar um evento do tipo golo
O utilizador autenticado consegue atribuir um cartão vermelho a um jogador	O utilizador autenticado consegue atribuir um cartão vermelho a um jogador e ao submeter os dados irá ser redirecionado para o <i>endpoint /save-event</i> , onde é apresentado uma mensagem a indicar que a operação foi bem-sucedida 	O utilizador autenticado não consegue atribuir um cartão vermelho a um jogador
O utilizador autenticado consegue atribuir um cartão amarelo a um jogador	O utilizador autenticado consegue atribuir um cartão amarelo a um jogador e ao submeter os dados irá ser redirecionado para o <i>endpoint /save-event</i> , onde é apresentado uma mensagem a indicar que a operação foi bem-sucedida 	O utilizador autenticado não consegue atribuir um cartão amarelo a um jogador





O utilizador autenticado consegue indicar quando começou um jogo	<p>O utilizador autenticado consegue indicar quando começou um jogo e ao submeter os dados irá ser redirecionado para o <i>endpoint /save-event</i>, onde é apresentado uma mensagem a indicar que a operação foi bem-sucedida</p> 	O utilizador autenticado não consegue assinalar quando é que o jogo começou
O utilizador autenticado consegue indicar quando terminou o jogo	<p>O utilizador autenticado consegue indicar quando terminou o jogo e ao submeter os dados irá ser redirecionado para o <i>endpoint /save-event</i>, onde é apresentado uma mensagem a indicar que a operação foi bem-sucedida</p> 	O utilizador autenticado não consegue assinalar o término do jogo
O utilizador autenticado consegue indicar o momento de interrupção de um jogo	<p>O utilizador autenticado consegue indicar o momento de interrupção de um jogo e ao submeter os dados irá ser redirecionado para o <i>endpoint /save-event</i>, onde é apresentado uma mensagem a indicar que a operação foi bem-sucedida</p> 	O utilizador autenticado não consegue indicar o momento da interrupção do jogo
O utilizador autenticado consegue indicar o momento de pausa de um jogo	<p>O utilizador autenticado consegue indicar a pausa do jogo e ao submeter o momento da mesma irá ser redirecionado para o <i>endpoint /save-event</i>, onde é apresentado uma mensagem a indicar que a operação foi bem-sucedida</p> 	O utilizador autenticado não consegue indicar o momento de pausa num jogo

O utilizador autenticado não consegue indicar o término do mesmo jogo duas vezes	O utilizador autenticado ao tentar terminar o mesmo jogo duas vezes irá ser redirecionado para o <i>endpoint /save-event</i> , onde é apresentado a mensagem: <i>"The game already end"</i> . O evento não é criado. 	O utilizador autenticado consegue assinalar o término de um jogo mais do que uma vez
O utilizador autenticado não consegue pausar um jogo depois de terminado	O utilizador autenticado ao tentar pausar um jogo depois de terminado irá ser redirecionado para o <i>endpoint /save-event</i> , onde é apresentado a mensagem: <i>"The game already end"</i> . O evento não é criado. 	O utilizador autenticado consegue assinalar a pausa um jogo depois de terminado
O utilizador autenticado não consegue interromper um jogo depois de terminado	O utilizador autenticado ao tentar interromper um jogo depois de terminado irá ser redirecionado para o <i>endpoint /save-event</i> , onde é apresentado a mensagem: <i>"The game already end"</i> . O evento não é criado. 	O utilizador autenticado consegue assinalar a interrupção de um jogo depois de terminado
O utilizador autenticado não consegue indicar o começo do mesmo jogo duas vezes	O utilizador autenticado ao tentar começar o mesmo jogo duas vezes irá ser redirecionado para o <i>endpoint /save-event</i> , onde é apresentado a mensagem: <i>"The game has not started already"</i> . O evento não é criado. 	O utilizador autenticado consegue assinalar o começo do jogo mais do que uma vez


Após a atribuição de um cartão vermelho, o jogador não poderá originar mais eventos, pois é expulso	Um jogador que recebeu cartão vermelho não poderá realizar mais eventos, pelo que o utilizador autenticado ao submeter o mesmo é redirecionado para o <i>endpoint /save-event</i> , onde é apresentado a mensagem: <i>"The player has been sent off"</i> . O evento não é criado.	O jogador após um cartão vermelho continua a poder realizar mais eventos
		
O utilizador autenticado não consegue criar nenhum evento sem ter sido criado em primeiro lugar um evento a indicar o começo do jogo	O utilizador autenticado que criou um evento antes de o jogo ter começado, ao submeter o mesmo é redirecionado para o <i>endpoint /save-event</i> , onde é apresentado a mensagem: <i>"The game has not started already"</i> . O evento não é criado.	O utilizador autenticado consegue criar eventos antes do jogo ter sido assinalado o começo do jogo
		
Após ser atribuído dois cartões amarelos a um jogador, ele é expulso não podendo realizar mais eventos (contando como se fosse atribuído um vermelho)	Ao terceiro cartão amarelo o jogador não poderá realizar mais eventos e o utilizador autenticado é redirecionado para o <i>endpoint /save-event</i> , onde é apresentada uma mensagem a indicar que a operação foi bem-sucedida	O jogador após o segundo cartão amarelo consegue continuar a realizar eventos
		
O utilizador autenticado não consegue criar eventos após o jogo ter terminado	O utilizador autenticado ao tentar criar um evento após o jogo ter começado é redirecionado para o <i>endpoint /save-event</i> , onde é apresentado a mensagem: <i>"The game already end"</i> . O evento não é criado.	O utilizador autenticado continua a poder criar eventos após o jogo ter terminado
		

O utilizador anónimo tenta aceder à página de criação de um evento pelo endereço	O utilizador anónimo é redirecionado para a página inicial 	O utilizador anónimo consegue aceder à página de criação de um evento
Um administrador submete um novo evento sem ter preenchido todos os campos	É pedido para preencher os campos em falta 	O novo evento é guardado na base de dados

Create-Game


Descrição	Passed	Failed
O utilizador anónimo tenta aceder à página de criação de um jogo pelo endereço	O utilizador anónimo é redirecionado para a página inicial 	O utilizador anónimo consegue aceder à página de criação de um jogo
O utilizador autenticado tenta aceder à página de criação de um jogo pelo endereço	O utilizador autenticado é redirecionado para a página inicial 	O utilizador autenticado consegue aceder à página de criação de um jogo
O administrador cria um jogo em que a equipa A é a mesma que a equipa B	Falha ao criar o jogo 	O jogo é criado com sucesso
Um administrador submete um novo jogo sem ter preenchido todos os campos	É pedido para preencher os campos em falta 	O novo jogo é guardado na base de dados

Show-Stats

Descrição	Passed	Failed
Ordenar as estatísticas por ordem alfabética do nome de equipa	As estatísticas são apresentadas por ordem alfabética do nome de equipa 	As estatísticas são apresentadas por ordem outra ordem que não a alfabética do nome de equipa

Ordenar as estatísticas por ordem decrescente do número de vitórias	As estatísticas são apresentadas por ordem decrescente do número de vitórias 	As estatísticas são apresentadas por ordem outra ordem que não a ordem decrescente do número de vitórias
Ordenar as estatísticas por ordem decrescente do número de derrotas	As estatísticas são apresentadas por ordem decrescente do número de derrotas 	As estatísticas são apresentadas por ordem outra ordem que não a ordem decrescente do número de derrotas
Ordenar as estatísticas por ordem decrescente do número de empates	As estatísticas são apresentadas por ordem decrescente do número de empates 	As estatísticas são apresentadas por ordem outra ordem que não a ordem decrescente do número de empates
Ordenar as estatísticas por ordem decrescente do número total de jogos	As estatísticas são apresentadas por ordem decrescente do número total de jogos 	As estatísticas são apresentadas por ordem outra ordem que não a ordem decrescente do número total de jogos
Tanto um utilizador não autenticado, como um autenticado sem privilégios ou um autenticado com privilégios (<i>admin</i>) consegue visualizar as estatísticas das equipas existentes	Qualquer utilizador consegue visualizar as estatísticas das equipas existentes 	Alguns ou nenhum dos utilizadores conseguem visualizar as estatísticas das equipas existentes

Show-details

Descrição	Passed	Failed
Tanto um utilizador não autenticado, como um autenticado sem privilégios ou um autenticado com privilégios (<i>admin</i>) consegue visualizar os detalhes dos jogos existentes	Qualquer utilizador consegue visualizar os detalhes dos jogos existentes 	Alguns ou nenhum dos utilizadores conseguem visualizar os detalhes