scoreDEI

Relatório do trabalho prático 2

Sistemas Distribuídos

Edição 2021/ 2022

Licenciatura em Engenharia Informática

Trabalho realizado por:
João Filipe Guiomar Artur, nº 2019217853
Rui Eduardo Carvalho Marques, nº 2019216539

Introdução

No contexto da unidade curricular de Sistemas Distribuídos, edição 2021/2022, da Licenciatura em Engenharia Informática, foi proposta a criação de uma plataforma para consulta de resultados desportivos em tempo real. Esta plataforma, denominada de *scoreDEI,* destina-se a utilizadores desta aplicação *web* poderem consultar e receber atualizações de resultados desportivos de jogos registados na base de dados.

A construção da plataforma *scoreDEI* permite a exploração e aplicação de conhecimentos de programação relacionados com Sistemas Distribuídos, adotando a linguagem de programação Java. O desenvolvimento do trabalho pressupõe a noção de uma arquitetura MVC (*Model-View-Controller*) e de como é constituído e programado um sistema de *Web* com camada de dados e de *frontend,* com recurso às *frameworks Spring Boot* e *Thymeleaf*. Espera-se a integração de uma base de dados na aplicação *web* através de *Java Persistence Application Programming Interface* (JPA).

Finalmente, para obter mais facilmente dados para a base de dados, deve haver uma integração com uma API externa (https://v3.football.api-sports.io/) utilizando tecnologia REST.

Arquitetura de Software & Funcionamento da API scoreDEI

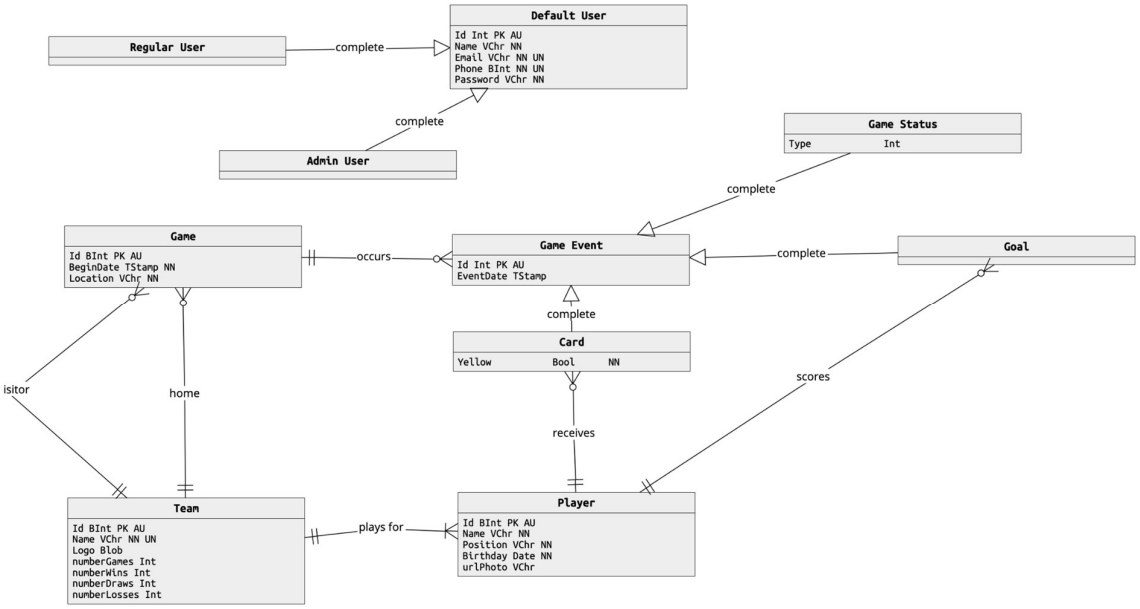

Fig.1 : Modelo de Entidade Relacionamento (ER)

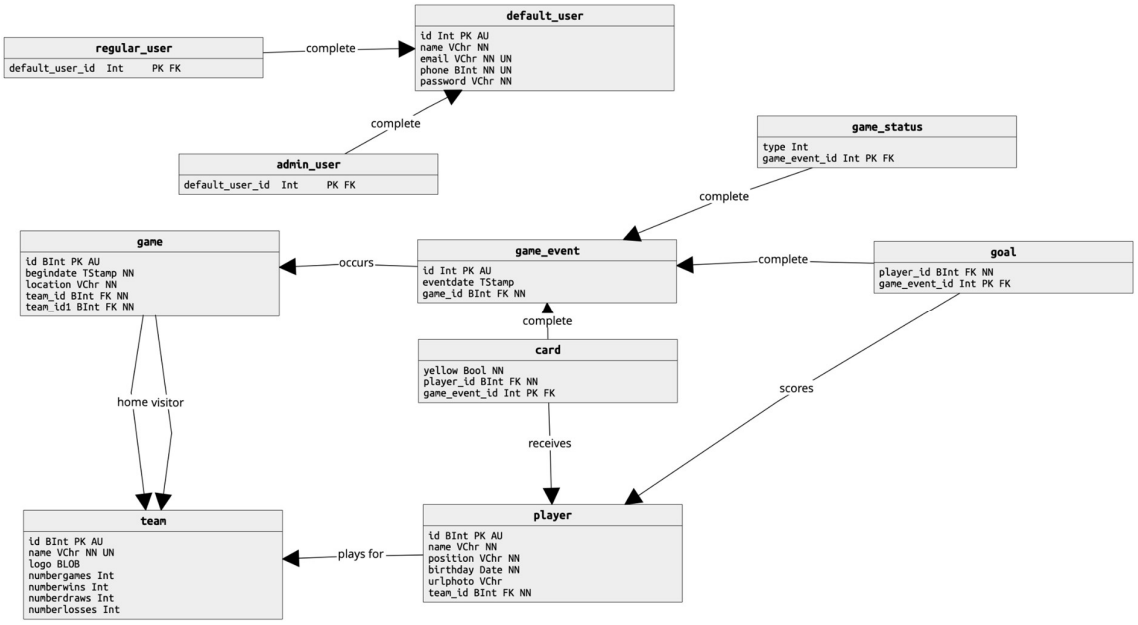

Fig.2 : Modelo físico

A base de dados estruturada para este projeto possui 10 tabelas. Cada tabela representa as seguintes entidades do modelo Entidade-Relação (E-R):

- *User* (ou *default_user*): Entidade responsável pelos utilizadores com atributos: (i) *id*; (ii) nome; (iii) *email*; (iv) *phone* (número de telemóvel); (v) *password*, cuja codificação com SHA-256 é guardada na base de dados.
- *AdminUser* (ou *admin_user*): Entidade filha do *User* para representar os utilizadores administradores.
- *RegularUser* (ou *regular_user*): Entidade filha do *User* para representar os utilizadores normais.
- *Team* (ou *team*): Entidade que representa uma equipa de futebol. Possui os seguintes atributos: (i) *id* da equipa; (ii) nome da equipa; (iii) *logo* da equipa, guardado através de um *byte array*; (iv) número de jogos terminados disputados; (v) número de vitórias; (vi) número de empates; (vii) número de derrotas.
- *Player* (ou *player*): Entidade para representar um jogador de futebol. Possui os seguintes atributos: (i) *id* do jogador; (ii) nome do jogador; (iii) posição; (iv) data de nascimento; (v) *url* com a foto do jogador (este atributo pode ser nulo pois só é adicionado se vier da API externa);
- *Game* (ou *game*): Entidade que representa um jogo de futebol entre duas equipas. Possui os seguintes atributos: (i) *id* do jogo; (ii) data e hora do jogo; (iii) localização do jogo.
- *GameEvent* (ou *game_event*): Entidade responsável por representar os eventos de um jogo. Possui os seguintes atributos: (i) *id* do evento; (ii) hora de ocorrência do evento (nota: embora esteja definido na imagem como *Timestamp*, esta variável é do tipo *Time*)
- *GameStatus* (ou *game_status*): Entidade filha de *GameEvent* e representa, através do seu atributo *type*, os vários estados de um jogo: (i) iniciado; (ii) terminado; (iii) interrompido; (iv) continuado.

- *Card* (ou *card*): Entidade filha de *GameEvent* e representa um cartão atribuído a um jogador. A entidade distingue se é vermelho ou amarelo através do seu atributo *booleano isYellow*.
- *Goal* (ou *goal*): Entidade filha de *GameEvent* e representa um golo marcado por um jogador.

A aplicação *web* scoreDEI é constituída por 4 camadas: (i) camada de controladores (controllers); (ii) camada de serviços (services); (iii) camada de repositórios (repositories); (iv) camada de base de dados.

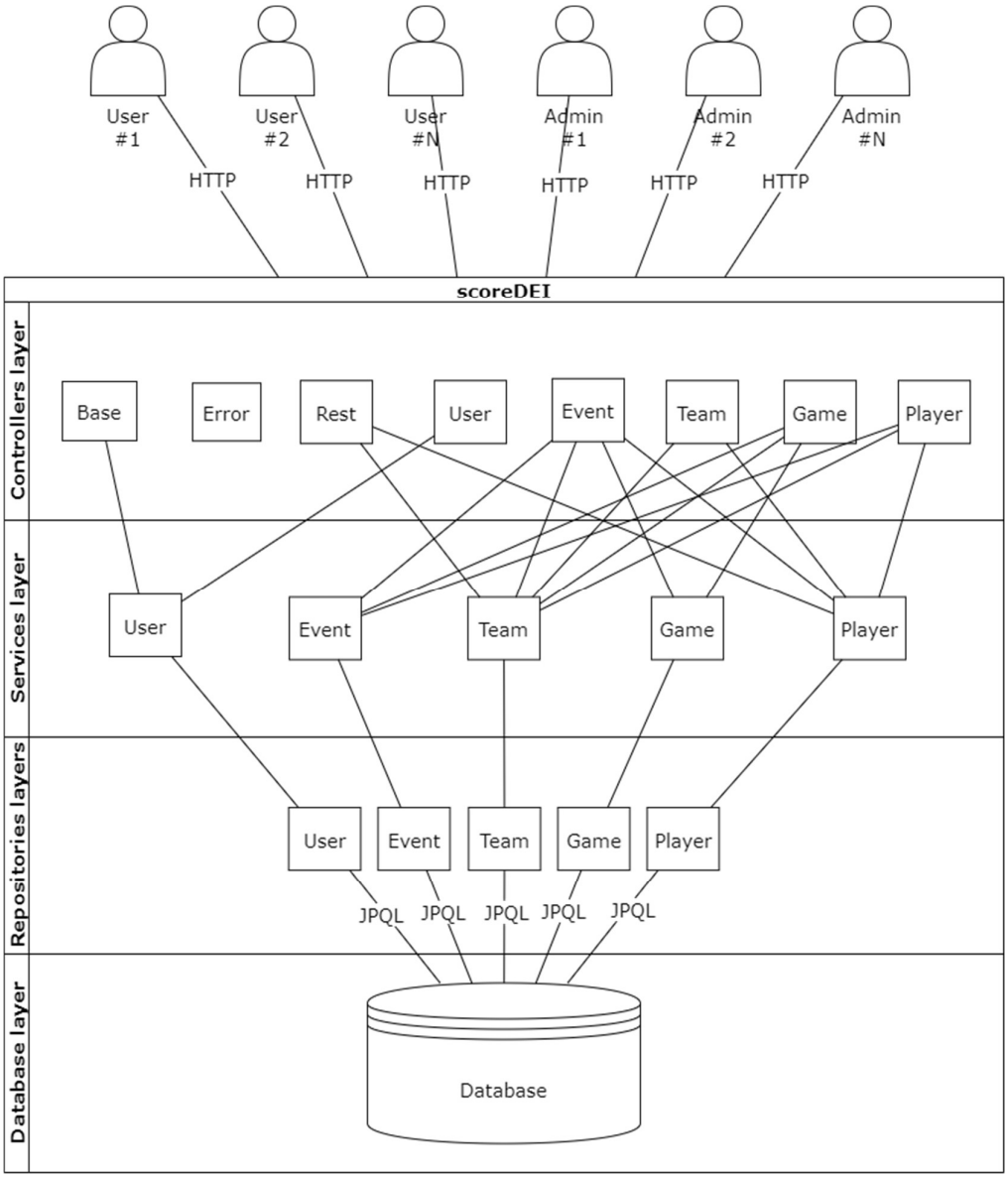

Fig.3 : Arquitetura

O utilizador interage através das várias vistas com os controladores, via o protocolo HTTP, para explorar as várias funcionalidades e visualizar os dados. Por sua vez, cada controlador comunicar com um ou mais serviços, localizados na camada imediatamente abaixo, para obter as respostas aos pedidos provenientes dos utilizadores. Após a receção de um pedido, um serviço reencaminha esse pedido para o respetivo repositório. Este comunica com a base de dados, via query JPQL, para obter os dados requisitados. Por último, a resposta com os dados pedidos sobe as várias camadas até ser apresentada numa vista ao utilizador.

Implementação Backend

Ao nível backend da API, a plataforma é constituída por vários controladores (*controlllers)*, cada um destes destinado à gestão das funcionalidades da API: (i) utilizadores; (ii) equipas; (iii) jogadores; (iv) jogos; (v) eventos; (vi) erro; (vii) autenticação e inserção inicial de dados e (viii) comunicação com a API externa.

Os vários controladores possuem vários *endpoints* para a manipulação e apresentação dos dados da base de dados. Para se visualizar a informação de uma forma acessível e fácil, os *controllers* estão associados a várias *views*, servindo de interface para com o utilizador. A transferência de dados entre os controllers e as views é feita através dos Models. Desta forma, é possível apresentar ao utilizador a informação da base de dados e permite, ao mesmo, inserir também dados nela.

Comunicação com a base de dados

Para a gestão da base de dados, realizando as operações CRUD (*Create*, *Update* e *Delete*), foram criadas classes *Services* (*UserService*, *TeamService*, *PlayerService*, *GameService* e *EventService*) para auxiliarem os *DataControllers*, nesse processo. Estas classes possuem, como atributo, a respetiva *interface Repository* para poder buscar a informação à base de dados com as funções com *queries* JPQL da *interface*.

Esta fase de implementação demonstrou-se ser a fase onde foi gasto a maioria do tempo da implementação. Os vários requisitos de funcionalidades da plataforma requerem vários *endpoints* e, consequentemente, várias funções e *queries* JPQL.

Tratamento de Erros

Quando ocorre um erro nos *endpoints* criados ou o utilizador tenta aceder algo que não possui permissão ou que não existe, o utilizador é redirecionado a um *endpoint* de erro. Este redireciona-o para uma vista *default* de erro, evitando que o utilizador receba uma *Whitelabel Error Page*.

Controlo de Utilizadores

O controlo de acesso a informação e/ou funcionalidades é efetuado por filtros de verificação. Existem dois filtros: (i) filtro de autenticação; (ii) filtro de administradores.

Para aceder a determinadas funcionalidades, nomeadamente a criação de eventos, requer-se que o utilizador esteja autenticado. Quando o utilizador tenta aceder a este conjunto de funcionalidades, existem duas hipóteses: (i) o utilizador encontra-se autenticado e procede

normalmente; (ii) o utilizador não está autenticado e é reencaminhado para o processo de autenticação.

No caso do segundo filtro, para além da autenticação, requer-se que o utilizador tenha privilégios de administrador para aceder a algumas funcionalidades como, por exemplo, a criação e gestão de equipas, jogadores e/ou jogos. Assim, uma tentativa de acesso a estas funcionalidades tem duas respostas possíveis: (i) o utilizador dispõe de privilégios de administrador e prossegue normalmente; (ii) o utilizador não está autenticado ou, caso esteja, não dispõe de tais privilégios e, portanto, é-lhe negado o acesso, sendo redirecionado para uma vista que o informa de tal.

Comunicação com a API externa

Para obter os dados da API de futebol, https://v3.football.api-sports.io/, foi criado um Rest Controller. Neste controller encontram-se apenas 3 *GET endpoints*, dado que a comunicação com esta API serve para popular mais facilmente a base de dados com equipas e jogadores. Para comunicar com a API, efetuou-se o registo na *dashboard* para obter uma API *Key* necessária para enviar nos *headers* dos *endpoints* da API.

O primeiro *endpoint* desenvolvido destina-se apenas a verificar o estado e a conexão com a API de futebol. Em caso de sucesso, é retornada uma mensagem com status 200 e informação sobre o utilizador da API *Key*.

Os restantes *endpoints* permitem obter equipas e jogadores. Para obter mais equipas e utilizadores, os parâmetros escolhidos para comunicar com a API foram o id da liga, o ano da temporada e, para o caso dos utilizadores, o número da página (cada resposta apenas fornece informação relativa a 20 jogadores. Após a receção da resposta, em formato JSON, os dados são processados para se extrair apenas as informações necessárias ao projeto e se criarem entidades para inserir na base de dados.

A comunicação com a API externa demonstrou-se como uma das fases mais complexas do projeto. Inicialmente, houve dificuldade na comunicação com a base de dados. Contudo, este problema foi, rapidamente, resolvido. O processo de *parsing* da mensagem JSON da API também foi mais complexo do que esperado, devido ao formato da mensagem e elevada quantidade de atributos e dados da resposta.

Distribuição de Tarefas

Todos os membros do grupo implementaram partes de *frontend* e *backend* do projeto. No entanto, as funcionalidades do projeto foram, maioritariamente, implementadas por:

- *Interfaces* de *frontend* e ligação aos *controllers* - João Artur;
- Ligação *backend* entre a base de dados e os *controllers* - Rui Marques;
- Tratamento dos erros - João Artur;
- Autenticação e autorização - João Artur;
- Ligação com a API externa - Rui Marques;

Testes

Na fase de testagem, foram realizados os seguintes testes para verificar as funcionalidades pretendidas:

Teste	Resultado
Fazer login - administrador	Funciona
Fazer login - não administrador	Funciona
Fazer login - login inválido	Rejeita
Fazer logout - administrador	Funciona
Fazer logout - não administrador	Funciona
Registar utilizador - administrador	Funciona
Registar utilizadores - não administrador/ sem autenticação	Rejeita
Editar utilizadores - administrador	Funciona
Editar utilizadores - não administrador/ sem autenticação	Rejeita
Visualizar lista de utilizadores - administrador	Funciona
Visualizar lista de utilizadores - não administrador/ sem autenticação	Rejeita
Eliminar utilizador - administrador	Funciona
Eliminar utilizador - não administrador/ sem autenticação	Rejeita
Criar equipa - administrador	Funciona
Criar equipa - não administrador/ sem autenticação	Rejeita
Editar equipa - administrador	Funciona
Editar equipa - não administrador/ sem autenticação	Rejeita
Visualizar lista de equipas	Funciona
Eliminar equipa - administrador	Funciona
Eliminar equipa - não administrador/ sem autenticação	Rejeita
Criar jogador - administrador	Funciona

Criar jogador - não administrador/ sem autenticação	Rejeita
Editar jogador - administrador	Funciona
Editar jogador - não administrador/ sem autenticação	Rejeita
Visualizar lista de jogadores	Funciona
Eliminar jogador - administrador	Funciona
Eliminar jogador - não administrador/ sem autenticação	Rejeita
Criar jogo - administrador	Funciona
Criar jogo - não administrador/ sem autenticação	Rejeita
Editar jogo como administrador	Funciona
Editar jogo - não administrador/ sem autenticação	Rejeita
Visualizar lista de jogos	Funciona
Criar evento - com autenticação	Funciona
Criar evento - sem autenticação	Rejeita
Editar jogo depois deste ter começado	Rejeita
Adicionar ao jogo qualquer tipo de evento antes do evento de início de jogo	Rejeita
Adicionar ao jogo qualquer tipo de evento após evento de interrupção de jogo	Rejeita
Adicionar dois cartões amarelos a um jogador	Funciona
Adicionar um evento (golo ou cartão) a um jogador com um cartão vermelho	Rejeita
Acompanhar jogo	Funciona
Listagem de eventos ordenados cronologicamente	Funciona
Listagem de equipas ordenáveis	Funciona
Melhor marcador	Funciona
Atualização dos jogos realizados da equipa	Funciona
Atualização de golos dos jogadores	Funciona

Atualização de cartões dos jogadores	Funciona
Atualização dos resultados dos jogos	Funciona
Popular base de dados com utilizadores	Funciona
Chamada à API externa para obter equipas	Funciona
Chamada à API externa para obter jogadores	Funciona

Conclusão

Após a conclusão do desenvolvimento, consideram-se alcançados os seguintes objetivos: (i) criação de sistema de *Web* com camada de dados e *Frontend* através do *Spring Boot* e *Thymeleaf*; (ii) programação de uma base de dados *Postgres* através da *Java Persistence Application Programming Interface*; (iii) integração de um serviço externo com recurso à tecnologia REST para obtenção de dados; (iv) implementação de mecanismos de autenticação e autorização.

Em suma, verifica-se que os conhecimentos, tanto sobre a unidade curricular de Sistemas Distribuídos como sobre aplicações *Web*, foram aprofundados e ampliados. Mesmo que, num cenário real, um sistema de acompanhamento de jogos de futebol em tempo real semelhante não seja tão simples como o que foi proposto e desenvolvido, ganhou-se uma melhor noção sobre como este tipo de sistemas distribuídos com eventos em tempo real funcionam num contexto real.