

Relatório de Projeto

FASE 1

Grupo 9 | DAW | 21-10-23

47179 – Afonso Cabaço

48264 – João Pereira 48292 – Tiago Neves

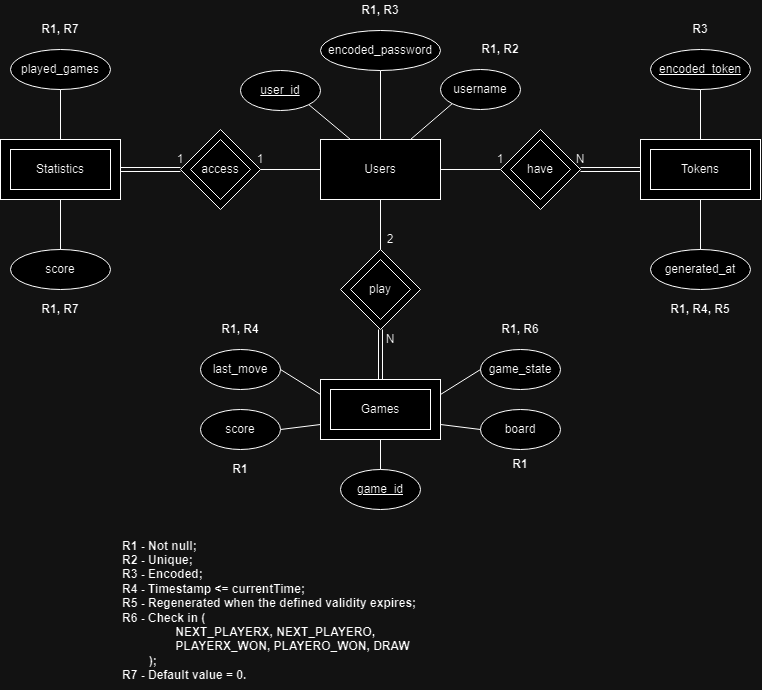
# INTRODUÇÃO

Neste documento iremos reportar os aspetos mais significativos sobre o design e implementação do projeto de DAW. Este projeto tem como objetivo realizar uma aplicação inspirada no jogo Gomoku.

# DATABASE MODELS

## Conceptual Model

O seguinte modelo EA foi concebido em grupo e usado como base para a estruturação do nosso projeto.



*Figure 1 - Modelo EA*

## Physical Model

Via o seguinte link poderão encontrar os scripts usados para a concretização do modelo, concebido anteriormente, na nossa base de dados.

https://github.com/isel-leic-daw/2023-daw-leic53d-2023-daw-leic53d-g09/tree/main/code/sql

# OPEN-API SPECIFICATION

As especificações do nosso projeto seguiram os conceitos da *OPEN-API*, para tal realizámos um ficheiro .*yaml* tal como pretendido para que todas especificações pudessem ser entendidas e testadas.

<https://github.com/isel-leic-ls/2223-2-LEIC42D-G09/blob/main/HTTPDoc.yaml>

# REQUEST DETAILS

Para efetuar o tratamento dos pedidos, foi preciso a criação de 5 classes diferentes, TaskServer, TaskWebAPI, TaskServices, TaskDataMem (caso o uso da app seja para dar uso a informação local, previamente criada pelos developers) e por último TaskDataPostgres (para usar informação de uma base de dados remota). Aquando o utilizador efetua um pedido ao servidor, este reencaminha o pedido para a webApi onde esta trata de efetuar o processamento dos dados vindos do pedido, ou seja os parâmetros que estão presentes na query, body, ou path do pedido e o uri que indica qual a funcionalidade a que o utilizador esta a querer aceder. Uma vez com os dados retirados e prontos para ser usados, estes são enviados para a próxima class TaskServices. É nesta class onde vai ocorrer o tratamento dos erros, que garante o envio correto da informação usada na class, TaskDataPostgres ou TaskDataMem, ambas com funcionalidade de inserção, remoção ou atualização da informação na database local ou em memória respetivamente. Temos de ter em conta que para certos pedidos é necessária a existência de autenticação por parte do utilizador. Para tal nesta fase, usamos uma aplicação dentro do modulo WebApi para fazer a verificação do header Authorization presente no pedido vindo do TaskServer. Caso este campo esteja vazio não será possível prosseguir e será mostrado um erro *403 – Forbidden*.

# CONNECTION MANAGEMENT

Com uso da função setUrl() da biblioteca *BaseDataSource* de java, definimos o caminho para a nossa base de dados e efetuar a ligação a base de dados. Uma vez a com a ligação estabelecida, para fazer uso da mesma usamos a função lambda *Connection.use{}* da mesma biblioteca, onde enviamos as querys pretendidas tendo em base a funcionalidade a qual o utilizador quer aceder no momento. Esta função é autoClosable, ou seja, quando o processo é finalizado a conexão é terminada, sendo necessária inicializar outro caso se queira efetuar mais coisas.

# DATA ACCESS

Devido a termos duas fontes de dados neste projeto, uma base de dados local e informação em memoria, o acesso a ambas é diferente.

Foi necessário a criação de data classes que representassem cada entidade da nossa aplicação, estas serviram para ajudar no armazenamento de informação e mais tarde no acesso a mesma tanto para a base de dados como em memória.

Devido a parte dos dados serem guardados numa base de dados local, foi necessário a criação de script sql para permitir a criação, inserção e remoção de dados da mesma.

# ERROR HANDLING/PROCESSING

O tratamento de erros e o seu processamento é realizado no módulo “TaskWebAPI” através da função auxiliar “requestHandler”. Sucintamente, esta função recebe como

parâmetros um request (pedido HTTP) e uma função, a função associada a um path, e retorna uma response. Realizamos um ciclo “try – catch” onde verificamos se no pedido é fornecido o meio de autenticação para acesso aos recursos do servidor, no

caso deste projeto o meio de autenticação é por via de “Bearer Token” e para a sua verificação utilizamos uma função auxiliar. Se a confirmação de autenticação for válida então o cliente/utilizador tem acesso ao recurso que pediu, caso contrário é apresentado uma mensagem de que não tem acesso ao recurso.

# CRITICAL EVALUATION

* Melhorar o gerenciamento dos erros e a forma de como os mostrar ao utilizador.
* Melhorar partes do código em geral.
* Rever as restrições para adicionar dados.

Relatório de Projeto

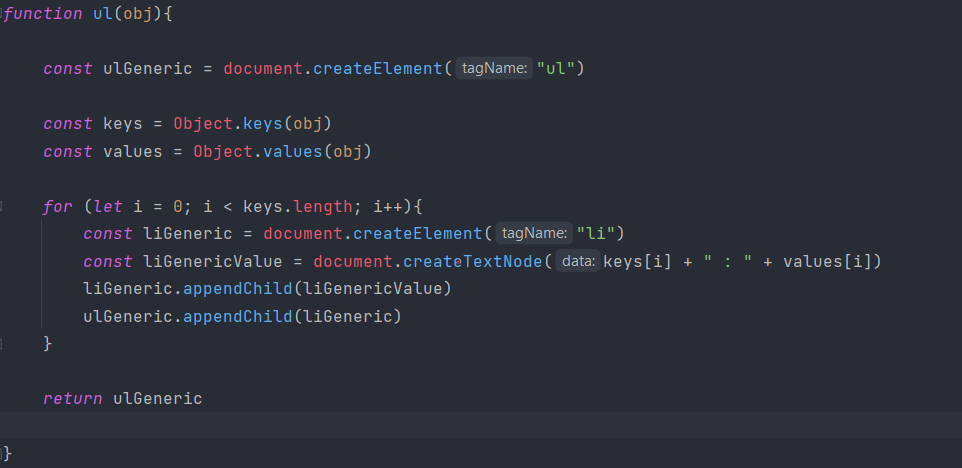
FASE 2

Grupo 9 | LS | 20-04-23

48291 – André Batista 48292 – Tiago Neves 48337 – Daniel Antunes

# STATIC CONTENT & SINGLE PAGE APPLICATION

Um dos objetivos da segunda fase era a realização de Single Page Application de forma a fornecer ao utilizador uma interface para as suas pesquisas, em específico das operações “GET”. Para facilitar a criação de páginas html, o grupo seguiu a sugestão do professor e criou uma função que organiza os objetos e os seus elementos obtidos da procura, ficando tudo ligado na ordem correta. Esta função encontra-se no ficheiro “htmlCreator.js” presente na pasta “static-content”.



# SKIP E LIMIT

Outro objetivo da segunda fase era a implementação de parâmetros de *query,* mais concretamente os parâmetros “skip” e “limit”.

No caso do parâmetro “skip”, este funciona como um offset e define a partir de qual índice o objeto deve retornar, ou seja, no caso de um pedido que pretende obter todas a “boards” de um utilizador, o parâmetro “skip=2” faz com que os objetos retornados sejam apenas as “boards” depois da “board” de índice 2.

Já no caso do parâmetro “limit”, este limita o número de objetos retornados, ou seja, se num pedido para obter todas a “boards” de um utilizador, o parâmetro “limit” for igualado a 3, apenas 3 “boards” serão retornadas.

Para a verificação e obtenção dos objetos corretos, dados os parâmetros antes referidos, fazemos a sua filtragem e processamento nas respetivas funções dos objetos pretendidos. Posteriormente, estes são passados nas “*SQL querys*” respetivas.

Relatório de Projeto

FASE 3

Grupo 9 | LS | 20-05-23

48291 – André Batista 48292 – Tiago Neves 48337 – Daniel Antunes

# SEARCH FUNCTION

Na presente fase do trabalho um dos objetivos era a criação de um mecanismo para a pesquisa de “boards”. O critério de procura ficou à escolha do grupo e por isso decidimos que a pesquisa por “boards” seria feito consoante o seu nome. Para que este mecanismo seja de uso mais explícito e facilitado para os clientes decidimos também que não seria preciso procurar a “board” pelo seu nome completo e sim por qualquer tipo de nome, frase ou apenas letra. Esta opção faz com que muitas vezes não apareça apenas a “board” pretendida mas também outras “boards” que tenham nome semelhante ao que foi procurado.

# STATIC CONTENT AND SINGLE PAGE APPLICATION (Continuation)

Nesta fase demos continuação ao *HTML e JavaScript, acrescentando* o resto das funcionalidades necessárias para um bom funcionamento da nossa SPA.

Estas novas funcionalidades podem ser encontradas, na pasta *static-content*,

e no ficheiro *handlers.js*.

A função que antes era usada para efetuar a criação das diferentes páginas *html*, deixou de ser usada, visto que houve e há a necessidade da criação de diferentes tipos de elementos da linguagem *html*, tais como o *form, input, label* entre outros.

Para tal e de maneira a reduzir o número de linhas de código, deixar o mesmo mais apresentável e por informação do professor, criamos funções auxiliares para os diferentes elementos de que precisávamos.

Criamos *handlers* para pedidos onde havia a necessidade da criação de novos dados. De momento tanto os *handlers* como as funções auxiliares estão no mesmo ficheiro, o que posteriormente será mudado para melhor organização e compreensão do projeto.

# CRITICAL EVALUATION

* Melhorar a organização geral do código.
* Reduzir a quantidade de código repetido, criando funções auxiliares que englobem toda a informação necessária.
* Eventual mudança do modelo EA na próxima fase.

Relatório de Projeto

FASE 4

Grupo 9 | LS | 10-06-23

48291 – André Batista 48292 – Tiago Neves 48337 – Daniel Antunes

# SINGLE PAGE APPLICATION (Continuation)

Um dos requisitos desta fase era a criação das *views* adequadas para a criação de um novo utilizador e para o acesso à aplicação (login). Para tal realizámos as funções *sign\_in* e *login* no ficheiro *handlers.js* para a criação destas *views*. Para a criação de um novo utilizador também adicionámos o campo *password,* de forma a criar as credenciais de criação e acesso de utilizadores mais corretas. Com os seus dados de acesso, *email* e *password*, um utilizador consegue realizar o login de forma simples e, como pretendido, o *token* do utilizador fica guardado numa variável presente no objeto *Response* da aplicação.

De forma a que o código ficasse mais organizado e percetível, além de ser também um aspeto referido nesta fase, realizámos funções auxiliares para a criação dos elementos *HTML,* estas funções podem ser encontradas no ficheiro *creates.js*.

# REFACTORING CODE

Como pretendido, realizámos uma análise critica sobre o nosso código de forma a identificarmos o maior número de repetições de código, erros ou até partes de código que pudessem afetar o desempenho da aplicação no geral.

# DEPLOYMENT/HOSTING ON RENDER

Nesta última fase foi-nos pedido para efetuar o deploy da nossa aplicação apartir de um serviço online, neste caso usamos o *Render,* um Plantform-as-a-Service (Paas) Provider.