Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores 2023/2024 Unidade Curricular de Desenvolvimento de Aplicações Web
Docente Felipe Freitas

Área Departamental de Engenharia de Eletrónica e Telecomunicações e de Computadores

Fase 1

Autores: 48269 José Borges

48259 Vasco Branco

46080 Sérgio Capela

Relatório para a Unidade Curricular de Desenvolvimento de Aplicações Web Licenciatura em Engenharia Informática e de Computadores

19 – 01 – 2024

Índice

Introdução 1
Modelo Conceptual 2
Modelo Físico 3
Especificação Open-API 4
Navegação pela Aplicação 4
Detalhes do Pedido 5
Gerenciamento da Conexão 5
Acesso a Dados 5
Processamento/Tratamento de Erros 6
Avaliação Crítica 6

Introdução

O objetivo deste projeto é o desenvolvimento de um sistema baseado na web que permitirá que vários jogadores joguem o jogo de Gomoku. O domínio da aplicação é baseado em 4 entidades diferentes:

1. User: Um *user* é caracterizado por um número único, um username único e uma password única.

2. Game: Um *game* representa um jogo, identificado por um número único, contendo informação sobre os jogadores que nele participam, o estado e o tamanho do tabuleiro com as posições ocupadas pelas peças, as regras e a variante do mesmo.

3. Ranking: *Ranking* representa a qualificação de cada jogador, sendo identificado pelo número único de user, contendo também o número total de jogos, o número de vitórias e também o número de derrotas do jogador.

4. Lobby: Um *Lobby* representa uma sala de espera para a qual os jogadores são inseridos quando manifestam vontade de jogar, sendo assim contém o identificador do jogador que procura um jogo (e posteriormente também o do seu adversário), as regras, a variante, o tamanho do tabuleiro selecionados pelos jogadores, e também o estado do jogo.

Modelo Conceptual

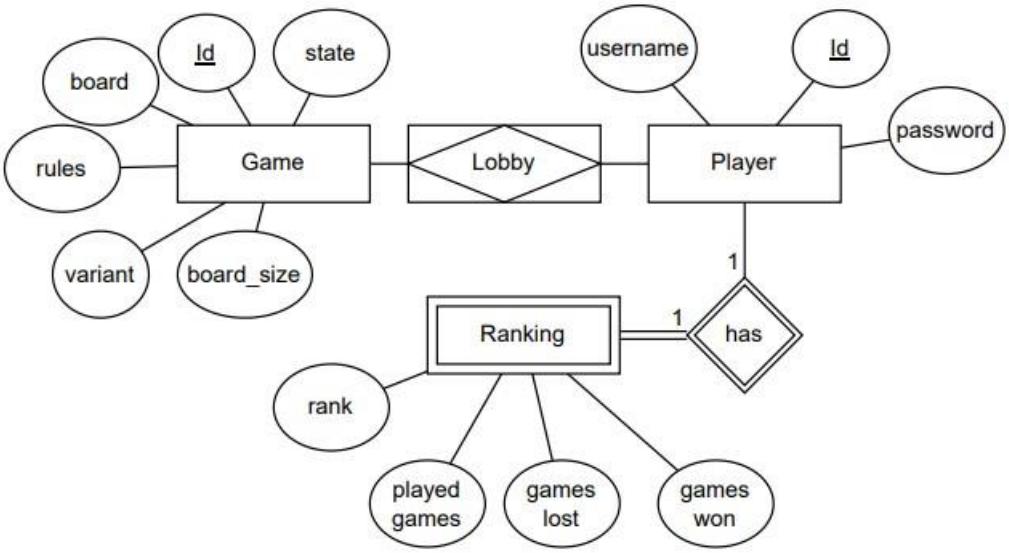

Figura 1- Modelo EA

Como referido anteriormente o nosso modelo é composto por 4 entidades, sendo 1 delas fraca. Como demonstrado na figura 1, a entidade Ranking é fraca da entidade Player uma vez que quando não existe uma Player não é possível que exista um Ranking.

Modelo Físico

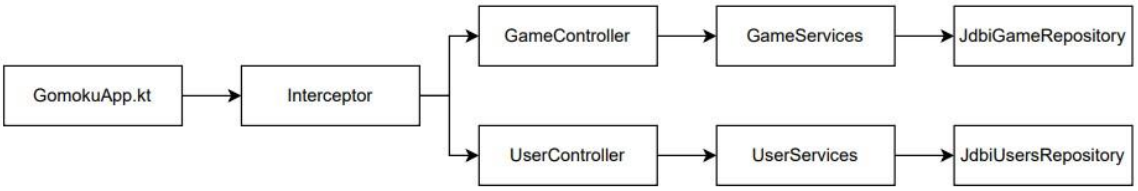

Figura 2 - Modelo Físico

Para a implementação do modelo físico recorremos a REST API que consiste na criação de serviços web com o objetivo de facilitar a comunicação e a integração entre sistemas distribuídos e heterogêneos.

- GomokuApp.kt - Constitui o ponto de entrada para a aplicação.
- Interceptor - Interceta o pedido para verificar a existência de um token.
- GameController e UserController - Implementação das rotas HTTP que compõem a REST API da aplicação.
- GameServices e UserServices - Implementação da lógica de cada funcionalidade da aplicação.
- JdbiGamesRepository e JdbiUsersRepository – Acesso à informação da aplicação.

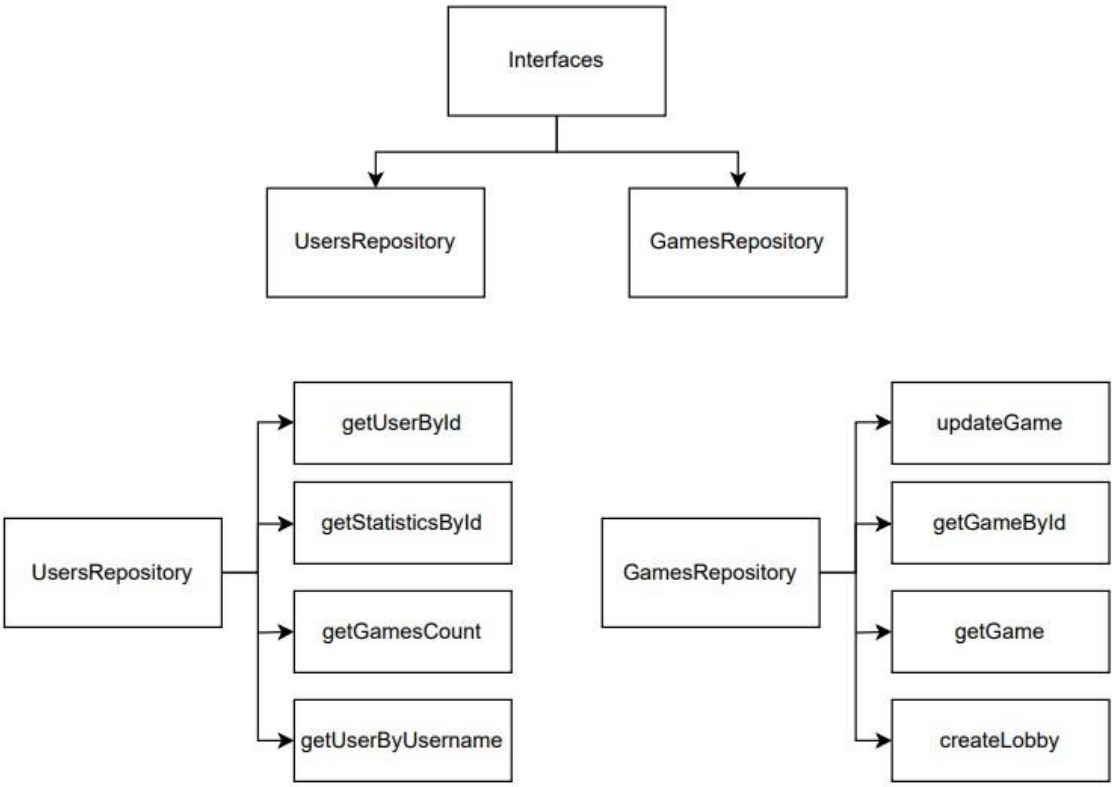

Figura 3 - Interfaces

Para garantir que as funcionalidades da aplicação são executadas corretamente foram implementadas 2 interfaces: UsersRepository e GamesRepository.

Especificação Open-API

Toda a informação sobre a aplicação pode ser encontrada *aqui*.

Navegação pela Aplicação

Foi implementada uma SPA (Single Page Application) que carrega o conteúdo à medida que o utilizador interage com a aplicação fazendo com que apenas sejam atualizadas as partes da página que assim o necessitam. A estrutura de navegação da aplicação baseia-se em todos os utilizadores terem à sua disponibilidade uma barra de navegação que os permite navegar para as páginas Home, S*tatistics, About us* e também realizar a sua iniciar ou finalizar a sua sessão na nossa aplicação, mas apenas os utilizadores autenticados podem criar uma sala de espera e posteriormente jogar.

Detalhes do Pedido

Quando um utilizador efetua um pedido HTTP na aplicação, o GomokuApp envia a informação com a rota e método utilizados para que no respetivo Controller possa ser executada a funcionalidade correta com a informação passada no path ou no body enviando essa informação para os respetivos Services que efetuam as verificações necessárias para garantir que a informação seja enviada para a base de dados local efetuando a funcionalidade da aplicação no respetivo Repository. Certas funcionalidades apenas podem ser efetuadas se o User estiver autenticado na aplicação, para isso desenvolvemos funcionalidades de signin para posteriormente efetuarmos essa verificação no Interceptor.

Gerenciamento da Conexão

Utilizando a função setUrl() definimos o caminho para a base de dados, seguidamente com o handle recebido como parâmetro estabelecemos a conexão com a base de dados utilizando as funções de extensão *createQuery* e *createUpdate*, para obtenção de dados (apartir da instrução select) e realização de atualizações (apartir da instrução update) respetivamente, e enviamos as queries necessárias para realizarmos a respetiva funcionalidade que desejamos. Essa conexão é descartada assim que é finalizada, uma vez que o método de conexão que utilizamos a termina automaticamente.

Acesso a Dados

Para aceder à informação da base de dados foram criadas as classes JdbiUsersRepository e JdbiGamesRepository que acedem às interfaces anteriormente mencionadas na figura 3. Cada classe dá override às funções da respetiva interface estabelecendo uma ligação com a base de dados e enviando queries anteriormente criadas para que possam ser efetuadas as funcionalidades da aplicação.

Processamento/Tratamento de Erros

O tratamento de erros é processado da seguinte forma: Em ambos os *controllers* (*Game* e *User*), todos os caminhos mapeados podem ter dois resultados diferentes *Success* ou *Failure*. Ambos os valores são *typealias* no caso de *Success* é *Either.Right<S>* e no caso de *Failure Either.Left<F>*. *Either* é uma *sealed class* que possui duas *data classes, Left* e *Right*. Para cada caminho foi criado uma *sealed class* que representa todos os erros possíveis. Estas *sealed classes* são utilizadas no *typealias* que representam o retorno de todas as funções no *services* do formato *Either<GameIdFetchError,GameModel>* , considerando *GameModel* o retorno duma função do *services* e GameIdFetchError a *sealed class* com os possíveis erros. Assim no *controller* é possível em caso de erro recolher qual o motivo do erro em concreto e disponibilizar um *status code* e uma mensagem personalizados para cada caso. Também é importante realçar que foi utilizado o JdbiTransactionManager. Este mecanismo fornece a possibilidade de iniciar, finalizar e desfazer transações, garantindo assim que as operações na base de dados sejam atômicas. Sendo mais concreto, tal é conseguido através do mêtodo run que utiliza, dentro da sua implementação, um mêtodo extensão do tipo Jdbi (inTransaction) que ao ser chamado recebe um bloco de código que representa as ações a ser efetuadas pela transação dentro do seu scope.

Avaliação Crítica

Na realização desta fase do projeto melhorámos algum código feito nas fases anteriores, evitando assim a repetição de código, e implementámos algumas novas funcionalidades.