Área Departamental de Engenharia de Eletrónica e Telecomunicações e de Computadores

No âmbito da unidade curricular

Laboratório de Software

Autores: 49476 Gonçalo Pinto
49459 Manuel Santos
49436 Ricardo Oliveira

Docentes: Engenheiro Paulo Pereira

Engenheiro Pedro Pereira

Maio de 2024

ÍNDICE

INTRODUÇÃO

Este relatório vista apresentar uma visão abrangente do projeto desenvolvido ao longo do semestre, na Unidade Curricular de Laboratório de Software, que compreende a análise, desenho e implementação de um sistema de informação para gestão de sessões de videojogos *multiplayer*.

O projeto implica o desenvolvimento de um sistema completo, incluindo *back-end* e *front-end.* No *back-end*, foi desenvolvida uma API HTTP que expõe os recursos necessários para a interação com o sistema. Essa API é responsável por fornecer diversas funcionalidades, como a gestão de jogadores, jogos e respetivas sessões.

No *front-end*, será criada, numa futura fase, uma *Single Page Application* (SPA) que oferecerá uma interface gráfica para os utilizadores interagirem com o sistema. Nessa interface, serão adicionadas e atualizadas funcionalidades para consumir os recursos disponibilizados pela API.

Nesta API é possível realizar as seguintes funcionalidades:

Gestão de jogadores:

- Criação de jogadores;
- Listar os jogadores existentes;
- Aceder aos detalhes de um jogador através do seu identificador.

Gestão de jogos:

- Adição de um jogo;
- Listar os jogos existentes;
- Aceder aos detalhes de um jogo através do seu identificador;

Gestão de sessões:

- Criação de sessões de jogos;
- Adicionar um jogador a uma sessão;
- Listar as sessões existentes;
- Aceder aos detalhes de uma sessão através do seu identificador.

BASE DE DADOS

Nesta secção estão descritos o funcionamento e a implementação da base de dados utilizada no projeto, com os seus modelos físico e conceptual.

Uma base de dados constitui um sistema estruturado e seguro para a gestão de dados, abrangendo as operações de armazenamento, consulta e atualização de forma eficiente. É composta por entidades, onde cada uma representa uma informação específica. Denomina-se base de dados relacional quando possibilita estabelecer relações entre diferentes conjuntos de dados.

A base de dados do nosso projeto contém 4 tabelas de informação diferentes:

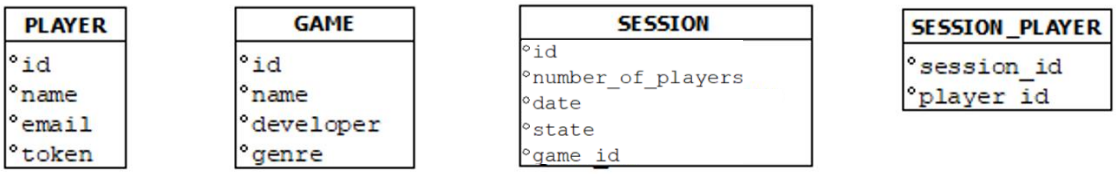

Figura 1 - Tabelas de informação

Todas as entidades desenvolvidas possuem uma chave primária, com o intuito de identificar o elemento.

Modelo conceptual

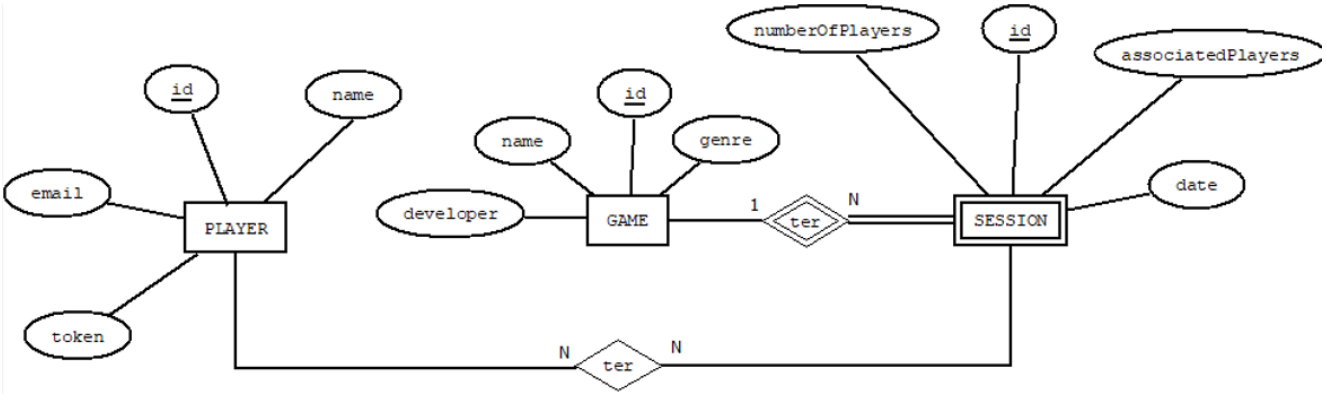

Figura 2 - Modelo Conceptual

Na figura 2, é apresentado o modelo conceptual que foi elaborado com base nas caraterísticas e funcionalidades do projeto. Este modelo ajuda a compreender e assegurar que todos os dados envolvidos sejam claros e desempenhem funções específicas. O modelo conceptual adotado é do tipo entidade-relacionamento (ER), o qual permite uma representação mais precisa e detalhada dos dados.

No contexto de um modelo ER, existem várias tipologias de restrições nos relacionamentos entre as entidades:

- 1:1 – Relação um para um, onde cada entidade de um conjunto participa apenas uma vez na relação.
- 1:N – Relação um para muitos, em que as entidades de um conjunto participam apenas uma vez na relação, enquanto as do outro conjunto podem participar múltiplas vezes, pelo menos duas.
- N:1 – Relação muitos para um, em que as entidades de um conjunto podem participar múltiplas vezes na relação, enquanto as do outro conjunto participam apenas uma vez.
- N:N – Relação muitos para muitos, onde as entidades de ambos os conjuntos podem participar múltiplas vezes na relação.

Modelo Físico

O modelo físico é o ponto em que se transforma o modelo conceptual em algo funcional, isto é, é o momento em que se começa a escrever código para o sistema de base de dados e a testar a sua operacionalidade na prática. Geralmente, estes modelos são criados utilizando ferramentas específicas para a construção de bases de dados, com recurso à linguagem de programação *PostgreSQL*.

```
create table game (
    gid serial primary key,
    gname varchar(80),
    gdeveloper varchar(80),
    ggenre int[] not null
);
```

```
create table player (
    pid serial primary key,
    pname varchar(80),
    pemail varchar(80),
    ptoken uuid
);
```

```
create table session (
                    sid serial primary key,
                    snofplayers int,
                    sessionDate timestamp,
                    state varchar(6),
                    sgame_id int references game(gid)
);
```

```
create table session_player (
    session_id INT REFERENCES session(sid),
    player_id INT REFERENCES player(pid),
     PRIMARY KEY (session_id, player_id)
);
```

Figura 3 - Modelo Físico

ORGANIZAÇÃO DO SOFTWARE

Especificação da Open-API

O ficheiro YAML que contém a especificação da Open-API do projeto pode ser encontrado na diretoria “docs” do repositório GITHUB do grupo.

Link: https://github.com/isel-leic-ls/2324-2-LEIC42D-G05/tree/main/docs/documentation.yaml

Detalhes do Request

Tal como sugerido no enunciado do projeto, a nossa solução divide-se em 4 camadas principais:

- **SessionsServer.kt:** Serve como ponto de entrada para a aplicação do servidor, responsável por configurar e iniciar o servidor web, além de definir as rotas e controladores para os endpoints da API. Estabelece rotas para operações relacionadas com jogadores, jogos e sessões de jogo, e instancia serviços como a classe WebApi para interagir com a base de dados ou memória. Por exemplo, são definidas rotas para listar jogadores, criar jogos e gerir sessões. Após configurar o servidor, é iniciado um logger para monitorizar seu funcionamento, e o servidor é iniciado na porta 9000. Utiliza-se uma injeção de dependências para definir o armazenamento pretendido de modo a generalizar o código, não dependendo o *back-end* de nenhum tipo de armazenamento específico.

- **SessionsWebApi.kt:** É a implementação das rotas HTTP que constituem a API REST da aplicação web. Este módulo, trata das operações relacionadas com jogadores, jogos e sessões de jogo. Fornece métodos para listar jogadores, obter detalhes de um jogador específico, criar um jogador, listar jogos, obter detalhes de um jogo específico, criar um jogo, listar sessões, criar uma sessão, obter detalhes de uma sessão específica e adicionar um jogador a uma sessão existente.

  Além disso, esta implementação configura os serviços necessários para interagir com a base de dados ou memória e trata dos pedidos HTTP correspondentes às operações definidas. Por exemplo, o método listPlayers obtém a lista de jogadores, createPlayer cria um jogador e lista as sessões existentes. Cada método lida com a conversão dos dados recebidos e enviados para o formato JSON, utilizando a biblioteca de serialização.

- **SessionsServices.kt:** É responsável pela implementação da lógica de cada uma das funcionalidades da aplicação. Define métodos para criar jogadores, obter detalhes de jogadores, listar jogadores, criar jogos, obter detalhes de jogos, listar jogos, criar sessões, adicionar jogadores a sessões, obter detalhes de sessões e listar sessões. Além disso, este módulo trata da paginação das sessões e da obtenção do token do jogador a partir do cabeçalho de autorização. Utiliza a classe Data para interagir com a base de dados ou memória, realizando as operações necessárias para cada funcionalidade da aplicação.

- **SessionsData.kt/SessionsDataMem.kt:** É a camada de memória onde é feito o acesso a dados, seja os que estão armazenados em memória ou os que estão na base de dados.

Gestão da Ligação

Para gerir a base de dados, tanto para consultar informação como para inserir, é necessário estabelecer uma conexão com ela. Para isso, utilizamos o seguinte código em Kotlin:

```kotlin
val dataSource: PGSimpleDataSource = PGSimpleDataSource()
new *
init {
    dataSource.setURL("jdbc:postgresql://localhost/postgres?user=postgres&password=postgres")
}
```

Figura 4 - Conexão à base de dados

PGSimplesDataSource() é uma classe disponibilizada pelo driver JDBC do *PostgreSQL*, que possibilita a conexão a uma base de dados *PostgreSQL*.

setUrl() é uma função do objeto *dataSource* que define o URL da nossa base de dados, facilitando assim a ligação e interação com a mesma.

Acesso a Dados

De modo a facilitar a alteração do tipo de acesso aos dados (armazenado em memória ou na base de dados) foi implementada a interface ***Data,*** que define a assinatura das funções responsáveis pelo acesso a dados.

Esta interface define um conjunto de operações padronizadas para aceder e manipular dados, enquanto as classes de entrada e saída fornecem estruturas de dados para representar informações recebidas do cliente e enviar respostas formatadas de volta ao cliente, respetivamente.

Ao implementar a interface Data, as operações definidas podem utilizar as estruturas de entrada para receber dados do cliente e transformá-los em operações específicas de acesso a dados.

Por sua vez, as estruturas de saída são usadas para formatar os dados recuperados da fonte de dados antes de serem enviados de volta ao cliente como resposta, garantindo assim uma comunicação eficiente entre as diferentes camadas da aplicação, e facilitando a manipulação e a formatação dos dados de acordo com as necessidades do sistema e do cliente.

```kotlin
interface Data{

    fun createGame(name:String, developer:String, genres: Set<Genre>?): Int
    fun getGameDetails(id: UInt): Game?
    fun getGameList(genres: Set<Genre>, developer: String?, limit:Int, skip:Int ): List<Game>

    fun createPlayer(name: String, email: String, token: UUID): Int
    fun getPlayerById(id: UInt): PlayerDC?
    fun getListOfPlayers(limit: Int, skip: Int): List<PlayerDC>

    fun createSession(playerID:UInt, capacity: UInt, gid: UInt, date: LocalDateTime): Int
    fun getDetailsOfSession(id: UInt): Session?
    fun getListSession(gid: UInt, date: LocalDateTime?, state: Services.TYPESTATE?, pid: UInt?): List<Session>
    fun addPLayerToSession(id: UInt, sid:UInt): Session
    fun getPlayerToken(token: String): String?
    fun getPlayerByEmail(email: String): PlayerDC?
```

Figura 5 – Interface Data

```kotlin
@Serializable
Codiumate: Test this class
data class SessionRequestClient(val playerID :UInt, val capacity: Int, val gid: Int, val date: String)
@Serializable
Codiumate: Test this class
data class AddPlayerID(val id: Int, val sessionId: Int)
@Serializable
Codiumate: Test this class
data class SessionsFilter(val gid: Int, val date: String? = null, val state: String? = null, val pid: Int? = null)

@Serializable
Codiumate: Test this class
data class PlayerID(
    val name: String,
    val email: String
)


@Serializable
Codiumate: Test this class
data class GameDataHandler(
    val gameName: String,
    val gameDev: String,
    val gameGenre: String
```

Figura 6 – Estruturas de Entrada

```kotlin
@Serializable
Codiumate: Test this class
data class SessionOutput(
    val id: UInt,
    val nofplayers:Int,
    val sessionDate: LocalDateTime, val game: GameOutput,
    val state: Services.TYPESTATE = Services.TYPESTATE.OPEN,
    val associatedPlayers: MutableList<UInt>?
)
@Serializable
Codiumate: Test this class
data class GameOutput(
    val gameId: UInt,
    val gameName: String,
    val gameDev:String,
    val gameGenre: Set<Genre>
)

@Serializable
Codiumate: Test this class
data class PlayerOutput(
    val id: UInt,
    val name: String,
    val email: String,
)
```

Figura 7 – Estruturas de Saída

Processamento e Tratamento de Dados

No contexto da webAPI e dos serviços, é crucial estabelecer um processo eficiente de processamento e tratamento de erros para garantir a fiabilidade e estabilidade das operações. A função *tryRun*() desempenha um papel central nesse contexto.

Ao lidar com o processamento de dados, especialmente em operações que envolvem interações com a base de dados ou serviços externos, é inevitável que ocorram potenciais situações de erro. A função *tryRun*() é projetada para tentar executar um bloco de código específico e, caso ocorra uma exceção durante essa execução, ela é capaz de lidar com a situação de forma apropriada.

Por exemplo, quando ocorrem exceções conhecidas, como *IllegalStateException* (erro 400 – BAD REQUEST), *IllegalArgumentException* (erro 400 – BAD REQUEST) ou *NoSuchElementException* (erro 404 – NOT FOUND), a função *tryRun*() responde de maneira precisa, ao retornar um status code correspondente, que indica o tipo de erro ocorrido, juntamente com uma mensagem descritiva para facilitar a compreensão do problema pelo utilizador ou cliente da API.

Além disso, existem outras verificações de integridade dos dados, como por exemplo a validação do formato de email introduzido pelo utilizador. Se o email fornecido não corresponder ao formato esperado, é lançada uma exceção com a mensagem "Invalid email".

Quando ocorrem exceções não previstas, a função *tryRun*() fornece uma resposta padrão de erro interno do servidor, utilizando um status code 500 (Internal Server Error) para indicar que ocorreu um problema inesperado.

Single Page Application

Uma *Single Page Application* (SPA) é um tipo de aplicação web que opera numa única página, gerada por um código em Javascript, CSS e HTML, e carrega dinamicamente o conteúdo conforme o utilizador interage com a aplicação. Numa SPA, o código todo necessário à aplicação é carregado de uma só vez no navegador quando o utilizador acede ao site pela primeira vez, o que faz com que não exista a necessidade do utilizador recarregar a página web a cada interação realizada.

Para a realização do projeto foram criados os seguintes ficheiros:

- **Index.html -** Ficheiro HTML que define a estrutura da página inicial da aplicação "Game Scheduling App", através de elementos e divisões.
- **Index.js** - Ficheiro JavaScript que configura as rotas da aplicação, e define as funções executadas com base no respetivo URL.

Foram ainda criados elementos CSS para estilizar a SPA, bem como a criação de diferentes *handlers* (e respetivas vistas) responsáveis pela criação do HTML de uma página Web.

Funcionamento prático na Web

Na realização deste projeto, foi criada uma página Web para suportar no *front-end* as funcionalidades existentes no *back-end*, através dos ficheiros especificados na secção acima.
A aplicação foi dividida nas seguintes páginas:

- **Login** – Nesta secção existe um formulário para a introdução de um nome, um *username* e um email para a criação de um jogador. Caso estas credenciais já existam, invés de se criar um jogador, é efetuado o login na conta que o mesmo já tinha.
  Para isto acontecer, foi utilizado um método POST que envia os dados inseridos para o servidor. O id retornado é guardado numa variável global definida no projeto, para ser possível aceder aos dados do próprio utilizador e, assim, realizar as diferentes operações da aplicação.

- **Player Details** – Logo após a criação ou o login do jogador, o utilizador é redirecionado para a página com os seus detalhes, que explicitam o nome, o *username*, o email e todas as sessões onde o jogador está, sendo possível visualizar os detalhes dessa sessão e até sair dela. Para tal, utilizou-se o atributo *href* do HTML.

- **List Games** – Nesta secção é possível procurar jogos existentes através do seu nome, do desenvolvedor, ou de géneros (podendo haver pesquisas combinadas e.g. jogos de um determinado desenvolvedor com determinado(s) género(s)). Para tal, foi utilizado o método GET para obter os jogos que cumprem estes requisitos e apresentá-los ao utilizador (redireciona o utilizador para a lista de jogos apresentadas sendo possível ver os detalhes de cada jogo). Nesta secção é possível também criar um jogo através de um formulário existente onde o utilizador introduz o nome, desenvolvedor e género(s) do jogo. Desta forma, foi utilizado o método POST para criar o jogo (redireciona o utilizador para a página de detalhes do jogo criado).

- **Games** – Após a pesquisa de jogos, o utilizador é redirecionado para esta secção, que apresenta o(s) jogo(s) que satisfaz(em) os filtros da pesquisa realizada. Deste modo, é possível ver os detalhes de um jogo, clicando no botão apresentado.

- **Game Details** – Nesta secção são apresentados os detalhes de um jogo, como o nome, o desenvolvedor, o(s) género(s), e uma foto (se disponível). Deste modo, é possível ver as sessões desse jogo, clicando no botão apresentado.

- **List Sessions** - Nesta secção é possível procurar sessões existentes através do seu jogo, da sua data e hora, do seu estado (aberto ou fechado) ou do username do criador da mesma. Para tal, foi utilizado o método GET para obter as sessões que cumprem estes requisitos e apresentá-los ao utilizador (redireciona o utilizador para a lista de sessões apresentadas sendo possível ver os detalhes de cada sessão, juntar-se à sessão, apagá-la ou editá-la). Nesta secção é possível também criar uma sessão através de um formulário existente onde o utilizador introduz o número da capacidade de jogadores, o jogo e a data e hora. Desta forma, foi utilizado o método POST para criar a sessão (redireciona o utilizador para a página de detalhes da sessão criada).

- **Sessions** - Após a pesquisa de sessões, o utilizador é redirecionado para esta secção, que apresenta a(s) sessão(s) que satisfaz(em) os filtros da pesquisa realizada. Deste modo, é possível ver os detalhes de uma sessão, clicando no botão apresentado.

- **Session Details** - Nesta secção são apresentados os detalhes de uma sessão, como a capacidade, a data, o estado, os detalhes do jogo e os jogadores na sessão. Deste modo, é possível ver os detalhes dos jogadores na sessão, clicando sobre eles.

AVALIAÇÃO CRÍTICA

Em suma, a realização deste projeto está a fazer com que consigamos aplicar os conhecimentos obtidos ao longo da licenciatura, bem como superar algumas dificuldades que temos. Com esta API conseguimos realizar diversas funcionalidades, como criar um jogador, um jogo e uma sessão e adicionar um jogador a uma sessão existente.

Foi implementada uma parte de *front-end* para realizar operações que obtém dados da base de dados a apresentam-nos ao utilizador.

Nesta fase do projeto foram adicionadas mais funcionalidades ao *front-end*, tendo a capacidade de além da obtenção de dados, a possibilidade de postá-los também.

Nas fases futuras iremos aperfeiçoar o nosso código, incluindo mais funcionalidades e o respetivo ficheiro de Docker.