´

### Departamento de Engenharia de Eletrónica e Telecomunicações e de Computadores

**Trabalho Prático (1ª Fase)**

Grupo 6:

Pedro Diz, nº47718

Vasco Branco, nº48259

João Pereira, nº48264

Professores: Paulo Pereira e Pedro Pereira

Semestre de verão de 2023/2024

**Índice**

[Introdução 3](#_Toc137468591)

[Modelo Conceptual 4](#_Toc137468592)

Modelo Físico……………………………………………………………………………………………………………………………5

Especificação Open-API…………………………………………………………………………………………………………….5

Detalhes num Pedido a API……………………………………………………………………………………………………….5

Gerenciamento de Conexões…………………………………………………………………………………………………….6

Acesso a Dados…………………………………………………………………………………………………………………………6

Tratamento de Erros…………………………………………………………………………………………………………………6

Avaliação Critica……………………………………………………………………………………………………………………….6

# **Introdução**

O objetivo deste trabalho é implementar um sistema de informação que gere sessões de vídeo jogos.

O domínio da aplicação consiste nestas entidades:

* Player: Um Player é caracterizado por ter um número único, um nome e um email.
* Game: Um Game é caracterizado por ter um número único, um nome único, um developer e um conjunto de géneros associados.
* Session: Uma Session é caracterizada por ter um número único, a quantidade de jogadores envolvidos na sessão, a data de início da sessão, o jogo e os jogadores associados

**Modelo Conceptual**

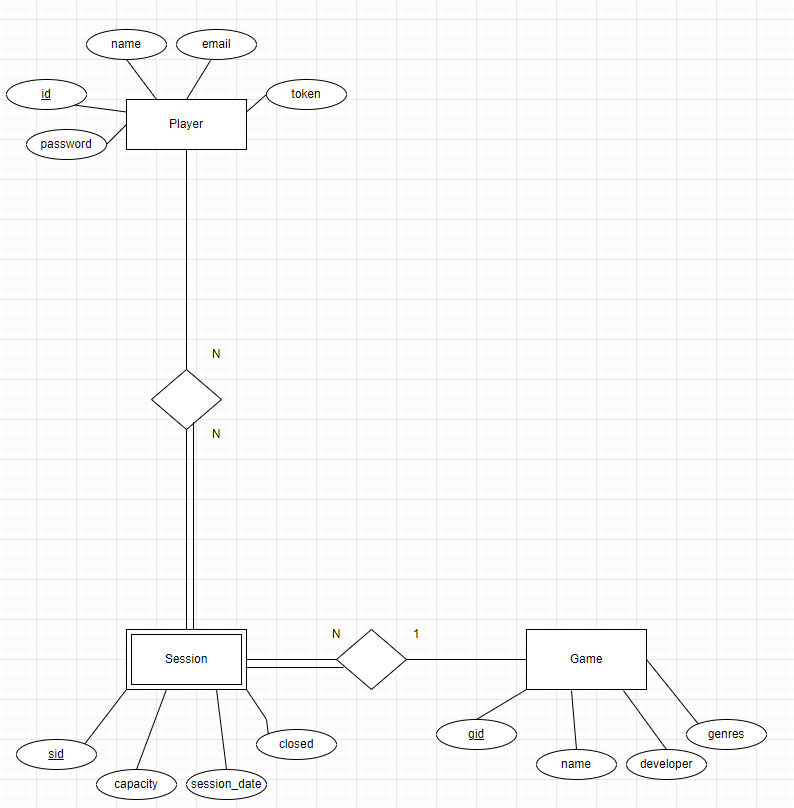


Figura 1- Modelo Conceptual

O modelo contem as 3 entidades referidas. De notar que Session é fraco de Player e de Game porque, se não houver nem players nem game, não existe session.

Restrições de Integridade:

* Email tem o seguinte formato: [a@a.a](mailto:a@a.a)
* Email e Name são chaves candidatas de Player
* O token em Player e único
* Session\_date tem o seguinte formato: YYYY-MM-DD HH:MM:SS
* Name é chave candidata de Game

**Modelo Físico**

O modelo físico pode ser encontrado [aqui](https://github.com/isel-leic-ls/2324-2-LEIC42D-G06/blob/main/Backend/sql/createTables.sql).

Todas as tabelas baseiam-se no modelo conceptual.

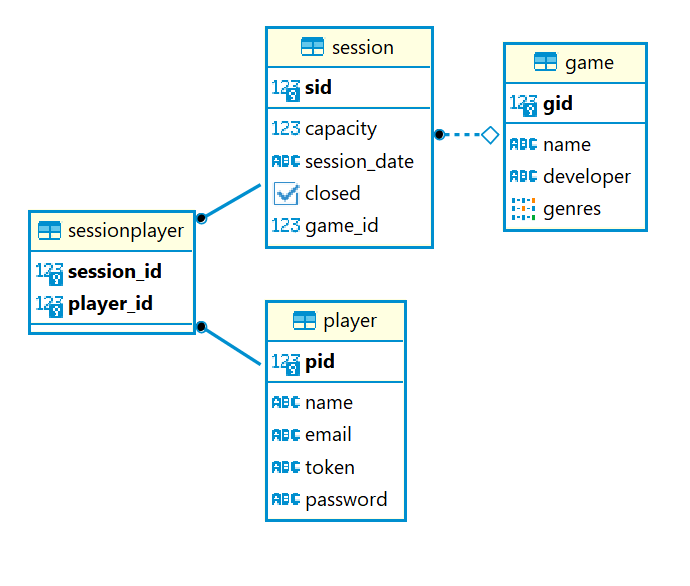


Figura - Modelo físico

**Especificação Open-API**

TODO

**Detalhes num Pedido a API**

O pedido chega ao servidor, o servidor depois faz roteamento do pedido para o handler apropriado. Depois disto acontecer, os seguintes passos são efetuados:

* O handler corre uma função chamada errorAwareScope que corre o código necessário a satisfazer o pedido. Caso haja algum erro, esta função faz o tratamento do erro via o exceptionHandler que possui.
* Dentro da função referida acima, a primeira coisa a realizar é extrair os parâmetros presentes na URI. É também extraído o token caso um pedido precisa de autenticação.
* A seguir é realizado a deserialização do JSON se o pedido tiver body.
* É chamado o serviço associado que faz validações sobre os parâmetros passados. Dentro deste serviço é chamado o repositório para persistir, ler, alterar dados.

**Gerenciamento de Conexões**

Sempre que se interage com o DBMS, vamos buscar uma conexão via a função getConnection() e usamos a função use {} que automaticamente fecha a conexão após correr o código no seu scope. Como nunca colocamos o autoCommit a falso nas funções do repositório, todo o código corre numa única transação.

**Acesso a Dados**

O nosso acesso de dados foi implementado com 3 interfaces. Existe 1 interface para gerir players, outra para gerir games e outra para gerir sessions. Existem 2 implementações destas interfaces. Uma delas para fazer as operações CRUD em memória que, cujos nomes normalmente seguem a seguinte estrutura: Mem{Entidade}Repo e a outra para fazer as operações CRUD numa base de dados postgres em que os nomes dos repositórios seguem a estrutura : Jdbc{Entidade}Repo. Os repositórios JDBC recebem como parâmetro o datasource postgres com o qual iriam interagir.

**Tratamento de Erros**

As exceções do backend são gerados em todos os módulos principais. Por exemplo:

* Repositório não consegue encontrar session com id = x.
* Serviço falha numa validação.
* Api não consegue extrair o parâmetro da rota.

Todos estas exceções são depois tratadas pelo exceptionHandler usado na função exceptionAwareScope { }. Dentro deste exceptionHandler é convertido a exceção para um status code utilizando um mapa associativo. É depois criado uma resposta com uma descrição do problema.

**Avaliação Critica**

Todo