

**GameOn   
Trabalho Prático – 2ª Fase**

|  | Daniel Caseiro |
| --- | --- |
|  | Henrique Fontes |

| Orientadores | Afonso Remédios |
| --- | --- |
|  | Nuno Leite  Walter Vieira |

Relatório GameOn (2ª Fase) realizado no âmbito de Sistemas de Informação,  
do curso de licenciatura em Engenharia Informática e de Computadores  
Semestre de Verão 2022/2023

Junho de 2023

**Instituto Superior de Engenharia de Lisboa**  
Licenciatura em Engenharia Informática e de Computadores

**GameOn   
Trabalho Prático – 2ª Fase**

| 46052 | Daniel André Caseiro |
| --- | --- |
| 48295 | Henrique Fontes |
|  |  |



|  | Orientadores:  Afonso Remédios |
| --- | --- |
|  | Nuno Leite  Walter Vieira |





Relatório GameOn (2ª fase) realizado no âmbito de Sistemas de Informação,  
do curso de licenciatura em Engenharia Informática e de Computadores  
Semestre de Verão 2022/2023

Maio de 2023

# **Resumo**

O projeto consiste na criação de um sistema de gestão de jogos, onde é possível criar jogadores, jogos, partidas, atribuir crachás, iniciar conversas e outras funcionalidades relacionadas.

A primeira fase do projeto resume-se à implementação do modelo de dados, incluindo todas as restrições de integridade, e a criação do código PL/pgSQL que permite criar o modelo físico, remover o modelo físico e preencher a base de dados. Também foram criadas funções, procedimentos armazenados, uma vista e gatilhos.

Na segunda e presente fase do projeto o intuito foi a manipulação de dados através da especificação JPA. Foram implementadas diversos componentes que permitem aceder a processos já estabelecidos na primeira fase do projeto, tal como a adição de novas funcionalidades. Para isso, foram criados diferentes mecanismos com o objetivo de simplificar processos, tal como a criação de funções genéricas.

Conclui-se que a criação de um sistema de gestão de jogos é uma tarefa complexa, mas com a implementação adequada do modelo de dados e do código PL/pgSQL e JPA, é possível criar um sistema eficiente e escalável. Por fim, é possível constatar que a implementação de mecanismos automáticos ajuda a garantir a integridade dos dados da base de dados.

**Palavras-chave:** funções; base de dados; chat; crachá; integridade; jogador; PL/pgSQL; pontos; procedimento armazenado; restrições; SQL; vista; JPA.

**Índice**

[Resumo 5](https://docs.google.com/document/d/1EU3Lgs4XdtL8TAuF0BkiR3VL3cA85uEK5K82EegVwfI/edit#heading=h.gjdgxs)

[Abstract 7](https://docs.google.com/document/d/1EU3Lgs4XdtL8TAuF0BkiR3VL3cA85uEK5K82EegVwfI/edit#heading=h.30j0zll)

[Lista de Figuras 11](https://docs.google.com/document/d/1EU3Lgs4XdtL8TAuF0BkiR3VL3cA85uEK5K82EegVwfI/edit#heading=h.3znysh7)

# 

# **Lista de Figuras**

[Figura 1 - Diagrama do Modelo Entidade Associação.](https://docs.google.com/document/d/1EU3Lgs4XdtL8TAuF0BkiR3VL3cA85uEK5K82EegVwfI/edit#heading=h.2r0uhxc) 2

# **Introdução**

Este relatório descreve o trabalho realizado com o objetivo de atingir determinados objetivos de aprendizagem. A tarefa envolve a criação de uma aplicação Java que ofereça diversas funcionalidades, conforme definido na fase 1 deste projeto.

A primeira parte do trabalho consiste em desenvolver uma aplicação capaz de acessar as funcionalidades 2d a 2l, conforme descrito na fase inicial. Essas funcionalidades envolvem a manipulação de dados relacionados a jogos, jogadores, partidas, atribuição de crachás, conversas e outras funcionalidades correlacionadas. É necessário implementar a lógica que permite interagir com essas funcionalidades de forma adequada.

Além disso, é necessário implementar a funcionalidade 2h sem recorrer a qualquer procedimento armazenado ou função PL/pgSQL. Essa funcionalidade, descrita na fase 1 do trabalho, requer uma abordagem diferente para sua execução, utilizando outros recursos disponíveis na aplicação Java.

Posteriormente, na segunda parte do trabalho, o objetivo é utilizar o mecanismo de optimistic locking para aumentar em 20% o número de pontos associados a um crachá específico. Isso é realizado fornecendo o nome do crachá e o identificador do jogo ao qual ele pertence. Se necessário, é permitido realizar alterações no esquema do banco de dados para suportar essa modificação. Além disso, é importante testar essa funcionalidade, identificando e tratando adequadamente situações de alteração concorrente conflituante que possam inviabilizar a operação.

No relatório, serão descritas a forma como as situações de erro foram criadas para testar a alínea anterior, bem como as estratégias utilizadas para apresentar mensagens de erro adequadas aos usuários da aplicação.

Por fim, a terceira parte do trabalho consiste em repetir a tarefa anterior, ou seja, realizar o aumento de pontos associados a um crachá, mas utilizando o controle de concorrência pessimista. Será necessário explorar as funcionalidades disponíveis nesse mecanismo para garantir a integridade e consistência dos dados durante a operação.

Este relatório fornecerá uma visão geral de cada uma dessas etapas, descrevendo os passos tomados, as decisões de implementação, os resultados obtidos e as conclusões alcançadas ao final do projeto.

# **2.Formulação do Problema**

O objetivo principal deste projeto é a criação de uma aplicação Java para um sistema de gestão de jogos, visando enfrentar diferentes desafios e alcançar diversas metas. Os resultados esperados abrangem a criação da aplicação Java, que inclui funcionalidades como a criação de jogadores, jogos, partidas, atribuição de crachás, início de conversas e outras tarefas relacionadas.

De forma a ser construída uma aplicação que execute todos os processos necessários de forma eficaz e simples, é necessário construir uma interface intuitiva, que disponibilize as operações requisitadas para esta fase do projeto de forma clara. Para isso procura-se a implementação de mecanismos de entrada de dados, tal como de exposição de dados requisitados pelo utilizador.

A solução que procuramos passa por uma solução genérica que visa simplificar processos e diminuir substancialmente a quantidade de código escrito.

Para esta implementação será necessário aceder a funcionalidades implementadas na primeira fase do projeto através de acessos diretos à base de dados , bem como através de componentes ORM estabelecidos.

Para além da implementação do acesso às funcionalidades previamente construídas, é necessário também implementar a funcionalidade do exercício 2h) da passada fase do projeto. Esta funcionalidade visa a existência de duas implementações distintas: a utilização de procedimentos armazenados ou funções PL/pgSQL, buscando uma solução alternativa com recursos disponíveis na aplicação Java, tal como a procura de uma solução reutilizando os procedimentos armazenados que a funcionalidade original contém.

Outro dos objetivos a esclarecer é a utilização do mecanismo de *optimistic* e *pessimistic locking*. Desta forma garante-se que as transações estarão cobertas de uma camada que saiba lidar com possíveis conflitos que possam ocorrer. A utilização destes mecanismos ocorre na implementação de uma nova funcionalidade, que visa o aumento de 20% do número de pontos associados a um determinado crachá específico. Pretende-se também a realização de uma função que teste essa solução e que apresente uma mensagem de erro adequada em caso de alteração concorrente conflitante que inviabilize a operação

Resumidamente, o objetivo é desenvolver uma aplicação Java robusta, eficiente e escalável, capaz de gerir jogos de forma adequada, mantendo a integridade dos dados e proporcionando uma boa experiência para os utilizadores.

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# **3.Solução Proposta - Grandes Ideias**

A solução proposta para abordar os desafios e metas estabelecidos no enunciado do projeto de criação de uma aplicação Java para um sistema de gestão de jogos pode ser dividida em duas partes: a construção dos componentes e estrutura a ser utilizados e a resolução dos problemas propostos. A primeira parte visa a facilitação e simplificação da segunda, que depende desta para funcionar conforme o requisitado.

## **3.1.1 Componentes**

Esta secção dedica-se à apresentação das soluções encontradas para a criação de componentes que procuram funcionar de forma consistente e regular nas implementações a ser estabelecidas.

## **3.1.1.1 Mapeamento Objeto-relacional**

O processo de criação das entidades JPA para o relatório envolve identificar as entidades principais do sistema e criar classes Java correspondentes a elas. Essas classes são mapeadas como entidades persistentes usando as anotações apropriadas. Os atributos são definidos e mapeados para as colunas do banco de dados, e as relações entre as entidades são estabelecidas. Em seguida, o provedor JPA é configurado no projeto através de um arquivo de configuração. Com as entidades prontas, é possível utilizar as operações de persistência fornecidas pela API JPA para interagir com o banco de dados, realizando consultas e manipulando os dados. As entidades JPA facilitam a persistência e manipulação dos dados no contexto do projeto.

O uso do JPA Buddy auxiliou significativamente no processo de criação das entidades JPA para o relatório. O JPA Buddy é uma ferramenta que oferece recursos avançados para a geração automática de código JPA. Foi assim possível acelerar o desenvolvimento das entidades, pois são fornecidas funcionalidades como a geração automática de classes de entidade a partir do banco de dados e vice-versa. Isso economizou tempo e reduziu a quantidade de código manual necessário. Além disso, o JPA Buddy também facilitou a configuração do provedor JPA, permitindo uma integração suave com o projeto. Em resumo, o uso do JPA Buddy agilizou o processo de criação das entidades JPA, proporcionando maior produtividade e simplificando as tarefas de mapeamento e configuração.

**3.1.1.2 Componentes ORM**

O código apresentado inclui um Mapper genérico chamado GenericMapper, que implementa a interface IMapper. Esse Mapper genérico possui métodos para criar, ler, atualizar e excluir entidades. Ele foi projetado para ser flexível e pode ser usado com diferentes tipos de entidades e chaves. O GenericMapper utiliza o objeto DataScope e o EntityManager para gerir transações e operações de acesso ao banco de dados. Essa implementação do Mapper genérico facilita o desenvolvimento de operações CRUD, permitindo a reutilização de código e simplificando o acesso aos dados das entidades.

Além disso, o código também apresenta um Repositório genérico chamado GenericRepository, que implementa a interface IRepository. Esse Repositório genérico permite realizar operações de acesso aos dados, como adicionar, excluir, atualizar e buscar elementos. O GenericRepository utiliza o Mapper genérico GenericMapper para executar essas operações, aproveitando a flexibilidade e a funcionalidade fornecidas pelo Mapper genérico. Essa implementação do Repositório genérico simplifica o acesso aos dados das entidades, promovendo a reutilização de código e facilitando as operações básicas de acesso ao banco de dados.

## **3.1.1.3 Datascope**

O código apresentado inclui a classe *DataScope*, que é uma subclasse da classe abstrata *AbstractDataScope*, que implementa também a interface *AutoCloseable*. A classe estende a funcionalidade do *AbstractDataScope* para criar um âmbito de dados específico.

A classe DataScope herda todos os métodos e comportamentos definidos na classe *AbstractDataScope*, como o controlo de transações, o acesso ao EntityManager e a validação do trabalho realizado. Esta também adiciona o seu próprio construtor, que chama o construtor da classe pai usando a palavra-chave *super*.

A utilização da classe *DataScope* permite criar um âmbito de dados delimitado, em que as transações são geridas automaticamente e as operações de acesso à base de dados são executadas de forma segura. Ao finalizar o âmbito, o encerramento automático é garantido pelo mecanismo *AutoCloseable*.

## **3.1.2 Estrutura**

Esta secção procura apresentar a estruturação do nosso projeto. Desta forma pretende-se também clarificar como é que os processos são estabelecidos e como todos os componentes se conjugam para executar as diversas operações a realizar.

## 

## **3.1.2.1 App**

A classe *App r*epresenta a aplicaçãodesenvolvida ao longo do projeto*,* oferecendo uma interface de linha de comando para possibilitar a interação com as funcionalidades do sistema de gestão de jogos, tratando exceções e apresentando resultados. Esta classe permite que o usuário execute, para além das novas funcionalidades implementadas, diversas operações realizadas na primeira fase do projeto.

## **3.1.2.2 Executor Operation**

Esta classe é responsável pela execução das diversas operações do sistema de gestão de jogos. Possui um construtor que recebe um *Entity Manager* como parâmetro de forma a manter as transações consistentes e anotações que distinguem se os métodos das operações em questão são funções, vistas ou procedimentos armazenados.

As classes *ExecutorDB* e *RegisterDB são utilizadas aqui de forma a executar as operações e registar funções e vistas relacionadas a estas operações, respectivamente.*

## **3.1.2.2 Register DB**

O intuito da criação da classe *Register DB* é o registo das funções e vistas relacionadas às operações. Para isso, esta classe recebe também um *Entity Manager* no construtor e é utilizada na classe previamente mencionada *Executor Operation.* De forma a realizar estes registos, utiliza-se a classe *Service* para registar estas operações, isto após serem estabelecidos os nomes e argumentos dos métodos a serem invocados.

## **3.1.2.3 Executor DB**

A classe *Executor DB* encontra-se no mesmo nível da classe *Register DB* e por isso também recebe o mesmo *Entity Manager* da classe *Executor Operation,* que a invoca.

A primeira operação realizada por esta classe é o método *executeMethod()* que obtém um método através de técnicas de reflexão e através deste obtém as anotações referidas anteriormente, que distinguem se o método é uma função, vista ou procedimento armazenado. Após esta distinção o método correspondente é invocado, executa-se a função correspondente vinda da classe *Service* e apresenta-se os resultados caso estes existam.

## **3.1.2.4 Service**

Esta classe é utilizada pelas classes *Register DB* e *Executor DB*. Regista-se determinada função ou vista através da classe *Register DB* e associa-se à variável *currentFunction*, que depois será executada posteriormente via *Executor DB.*

## **3.2 Resolução dos problemas propostos**

A realização das funcionalidades propostas pode ser dividida em três partes principais: o acesso às funcionalidades *2d)* a *2l)*, aimplementação da alínea 2h em *JPA* com e sem recurso às funções e procedimentos armazenados realizados previamente em *SQL* e por último a nova operação que acrescenta 20% dos pontos da pontuação de um determinado *Cracha*.

## **3.2.1 Acesso ás funcionalidades da Fase 1**

## **3.2.2 Implementação da alínea 2h em JPA**

## **3.2.3 ex2**

# 

# **5.Conclusões**

Neste relatório foi abordada a segunda fase do trabalho, na qual o foco foi o desenvolvimento de uma camada de acesso a dados utilizando uma implementação de JPA e um subconjunto dos padrões de desenho. Além disso, foi desenvolvido também uma aplicação em Java que fizesse o uso adequado dessa camada de acesso a dados.

Durante esta fase, para além da atenção dada à correta utilização do processamento transacional, por meio dos mecanismos disponíveis no JPA, também foi tido em conta a integridade dos dados e a consistência das operações. Foi enfatizada a importância de libertar adequadamente as conexões e recursos quando estes não estiverem em uso, visando a eficiência e o bom desempenho da aplicação.

Foram desenvolvidos testes adequados para identificar situações de alteração concorrente conflitante que pudessem inviabilizar a operação, que expressam mensagens de erro adequadas nesses casos.

No desenvolvimento do trabalho, foram valorizados o tratamento de erros em todas as etapas, bem como a gestão transacional, utilizando o nível de isolamento adequado de forma explícita.

Em termos de organização do código, o foco foi simplificar o código ao máximo utilizando componentes comuns de forma estruturada.

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# **Referências**

| [1] | Fundamentals of Database Systems (7th Edition) Ramez Elmasri, Shamkant B. Navathe Pearson Education, 2015. |
| --- | --- |
| [2] | Transaction processing : concepts and techniques (5th Edition) Jim Gray, Andreas Reuter Morgan Kaifmann, 1993. |