ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

OnTrack - Sistema de Gestão e Registo da Localização de Automóveis e Camiões

47147 André Gonçalves Jardim

Orientadores Nuno Datia

Relatório de segunda fase realizado no âmbito de Sistemas de Informação,
do curso de licenciatura em Engenharia Informática e de Computadores
Semestre de Verão 2021/2022

Junho de 2022

Resumo

Este projeto tem como objetivo criar uma aplicação Java que permita aceder a funcionalidades do sistema desenvolvido na primeira fase do trabalho utilizando JPA.

Palavras-chave: acesso a dados; base de dados; JPA; transações.

Abstract

The goal of this project is to create a java application to access functionalities from the system designed in the first phase of this project using JPA.

Keywords: data access; databases; JPA; transactions.

Índice

Lista de Figuras

1. Introdução

Este projeto visa criar uma aplicação para aceder a funcionalidades do sistema de informação desenvolvido na primeira fase para a empresa *OnTrack*.

1.1 Organização do documento

No capítulo 2 são apresentadas as alterações efetuadas sobre a primeira fase do projeto.

No capítulo 3 é apresentada a estrutura do projeto.

No capítulo 4 são descritos os problemas encontrados e as soluções para os mesmos.

No capítulo 5 são apresentados alguns detalhes de implementação.

No capítulo 6 é apresentada a solução implementada para o processamento de registos utilizando *optimistic locking*.

2. Alterações ao Modelo Físico

O modelo físico criado na primeira fase tinha um erro em que cada cliente poderia ter mais do que 1 frota de veículos, esse erro foi corrigido nesta fase adicionando a restrição *unique* na coluna *nif_cliente* da tabela *frotas_veiculos*.

```
create table frotas_veiculos
(
    id          serial primary key,
    nif_cliente int not null unique,
    foreign key (nif_cliente) references clientes (nif) on update cascade
);
```

Figura 1 – Alterações na tabela *frotas_veiculos*

3. Organização do Projeto

O projeto foi organizado em 3 *packages*:
domain – contém as interfaces e classes de domínio anotadas com anotações JPA;
data – trata do acesso à base de dados;
ui – trata da apresentação das funcionalidades ao utilizador e da recolha dos dados necessários para as executar.

3.1 *Domain*

Este *package* contém as classes que representam as entidades da base de dados. Cada classe tem as anotações necessárias para que o JPA funcione corretamente.

3.2 *Data*

Este *package* contém os repositórios necessários para o funcionamento da aplicação e a classe *JPAContext* que é a base de toda a funcionalidade do projeto e trata de fornecer os repositórios para as camadas que necessitem de aceder à base de dados e de chamar os procedimentos armazenados com a devida gestão transacional.

Este *package* depende de ***domain***.

3.3 *UI*

Este *package* contém os métodos necessárias para a apresentação de dados ao utilizador assim como a recolha dos dados para a execução dos métodos que acedem à base de dados.

Este *package* depende de ***domain*** e de ***data***.

4. Problemas

Neste capítulo são descritos os problemas que surgiram e as soluções criadas para os resolver.

4.1 Incompatibilidade entre Tipos

Na base de dados, os clientes têm um atributo do tipo *BIT* que indica se estes foram removidos ou não. Em java, este atributo foi implementado utilizando o tipo *boolean*, no entanto, ao fazer a persistência de um destes objetos, ocorria um erro porque o tipo *boolean* é incompatível com o tipo *BIT*, foi então criada uma classe que converte *boolean* em *BIT* e vice-versa.

```java
@Converter
public class BooleanToBit implements AttributeConverter<Boolean, Object> {
    @Override
    public Object convertToDatabaseColumn(Boolean value) {
        try {
            PGobject po = new PGobject();
            po.setType("BIT");
            if (value != null && value) {
                po.setValue("1");
            } else {
                po.setValue("0");
            }
            return po;
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }

    @Override
    public Boolean convertToEntityAttribute(Object value) {
        return value == Boolean.TRUE;
    }
}
```

Figura 2 – Conversor de *boolean* em BIT

4.2 Atualização de Atributos Chave

Na primeira fase decidiu-se utilizar o NIF de um cliente como a sua chave o que impossibilita a atualização do NIF através do JPA, assim apenas é possível atualizar o nome, a morada, o número do cartão de cidadão e a referência.

4.3 Entidades sem Atributo Chave

Na primeira fase não se sentiu a necessidade de colocar atributos chave em algumas entidades, o que se revelou um problema nesta fase, dado que o JPA necessita que todas as entidades tenham uma chave.

Para tentar resolver o problema sem efetuar alterações na base de dados foi utilizada a marca temporal como chave na entidade **registos inválidos**, no entanto, dado que a marca temporal não é única, poderão continuar a ocorrer problemas que só podem ser resolvidos adicionando uma chave na base de dados.

5. Detalhes de Implementação

Neste capítulo são descritos alguns detalhes de implementação.

5.1 Classe *JPAContext*

Quando esta classe é instanciada é criada uma *EntityManagerFactory* que é utilizada para criar um *EntityManager*. São também instanciados os repositórios necessários.

Esta classe contém os métodos ***beginTransaction***, que inicia uma transação se nenhuma estiver a decorrer, e ***commit*** que faz *commit/rollback* quando necessário.

Contém também os métodos necessárias para aceder a algumas funcionalidades do sistema criado tais como a contagem de alarmes, processamento de registos, inserção de clientes e veículos e a remoção de registos inválidos expirados.

5.2 Repositórios

Todos os repositórios criados têm um construtor que recebe um *EntityManager* e fazem *override* aos métodos ***findByKey***, que procura e devolve uma entidade através da sua chave utilizando um *named query*, ***add***, que adiciona um objeto ao *EntityManager*, ***remove***, que remove um objeto do *EntityManager* e ***find*** que executa uma *query jpql* e retorna uma lista de resultados.

5.3 Validação de Dados

Foi criada a classe *Validation* no *package domain* que valida alguns dados tais como a matrícula de um veículo e o NIF de um cliente, para que estes possam ser validados antes de serem atribuídos a objetos, diminuindo assim o número de transações desnecessárias.

6. *Optimistic Locking*

Neste capítulo é descrita a solução utilizada para o processamento de registos utilizando *optimistic locking* e a testagem da mesma.

6.1 Implementação

Para efetuar o processamento de registos utilizando *optimistic locking* foi criada uma cópia da classe *RegistoNaoProcessado* (*RegistoNaoProcessadoOpt)* que contém a anotação *"*@OptimisticLocking(type=OptimisticLockingType.CHANGED_COLUMNS)" para que quando o registo não processado seja removido (após o seu processamento) seja lançado um erro caso tenham ocorrido alterações concorrentes conflituantes desde a leitura inicial.

6.2 Teste

Para testar a solução implementada, após a leitura dos registos não processados presentes na base de dados o processamento dos registos é suspenso até o utilizador inserir uma linha na consola, podendo assim causar alterações na base de dados.