



UNIVERSIDADE DE ÉVORA

Relatório Sistemas Operativos II

Rúben Peixoto e Marcelo Feliz
37514 e 38073

24 de Abril 2020

Índice

1	Introdução	1
2	Configurações Iniciais	3
3	Servidor	4
3.1	Base de Dados	4
4	Cliente	6
4.1	Base de Dados	7
4.2	Thread	7
5	Makefile	8
6	Javadoc	9
7	Problemas	10
7.1	Problema 1	10
7.2	Problema 2	10
7.3	Problema 3	10
8	Conclusão	11
9	Referências	12

Lista de Figuras

1	Directoria do projecto	1
2	Ficheiro xml com as configurações para a base de dados PostgreSQL	3
3	Tabelas usadas em PostgreSQL	4
4	Registo de um utilizador	6
5	Menu da interface	6
6	Estrutura da tabela User na base de dados embebida	7
7	Javadoc em index.html	9
8	Problema com a thread	10

1 Introdução



Figura 1: Directoria do projecto

Para este projecto a linguagem adoptada foi o Java com a versão 8. O propósito deste trabalho é criar uma estrutura de comunicação entre cliente e servidor. Nesta estrutura o cliente requer serviços ao servidor. Há certas operações em que o servidor fornece dados, de tipos primitivos ou de objectos, ao cliente. Para nos abstrairmos não só dos protocolos de comunicação TCP/IP como também da serialização dos objectos, o grupo usou Java RMI.

Posto isto, e como é imperativo que o servidor guarde informação em memória persistente, o grupo decidiu usar o Postgresql, no lado do servidor, para esse efeito. A directoria "esquema_db/", que se encontrar ao mesmo nível da pasta do projecto "SPN/", contém os scripts necessários para criar as tabelas e inserir conteúdo nas mesmas.

Do lado do cliente, como existe a possibilidade de o servidor falhar, foi necessário implementar uma base de dados embebida com o propósito de arquivar pedidos que, por desventura, não chegaram ao servidor. Por isso, o grupo escolheu usar a ferramenta "HyperSQL".

Uma vez que o servidor podia falhar, o grupo decidiu usar uma Thread, no lado do cliente, que de 10 em 10 segundos, tenta reenviar os pedidos desse cliente. O grupo percebeu a vantagem de ter um processo em paralelo no lado do cliente e, por isso, decidimos colocar mais uma funcionalidade. Essa funcionalidade consiste em verificar, de todas as necessidades do utilizador, que está a usar o programa, tem até à altura, se existe algum produto que

possa satisfazer a(s) sua(s) necessidade(s). Caso exista pelo menos um produto que respeite esse requisito, é enviado uma mensagem para a interface do utilizador a informar essa situação.

Para as acções de compilar e executar: o cliente, o servidor, e o serviço de nome, é utilizado um Makefile.

2 Configurações Iniciais

Quando o programa é executado, mais especificamente, do lado do servidor, este vai buscar as informações de configuração necessárias para conectar-se à base de dados Postgres. Este ficheiro está na mesma directoria que as pasta "esquema_db/" e "SPN/" e é denominado "DBProperties.xml". Na figura a baixo pode ver a estrutura do ficheiro XML.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
  <entry key="host">localhost</entry>
  <entry key="database"> ... </entry>
  <entry key="user"> ... </entry>
  <entry key="password"> ... </entry>
</properties>
```

Figura 2: Ficheiro xml com as configurações para a base de dados PostgreSQL

O grupo não viu a necessidade de fazer o mesmo para a base de dados HyperSQL uma vez que a tabela está dentro da pasta do projecto, na directoria "SPN/nbproject/UserDB" com o nome "requests".

Quando o cliente ou o servidor são inicializados, é necessário colocar argumentos para tal. No entanto, esses argumentos são utilizados no Makefile, não sendo necessário colocar informações tais como o IP e a porta à mão.

3 Servidor

Foram criados serviços que, de momento, não são usados pelo cliente, no entanto, o grupo considera que, estes possam vir a ser necessários no futuro.

Todos os métodos do servidor utilizam operações da base de dados PostgreSQL. As operações para a base de dados foi implementada no ficheiro "DBOperations.java" e para o servidor "ServiçoImpl". Este último implementa a interface definida no ficheiro "Servico.java".

3.1 Base de Dados

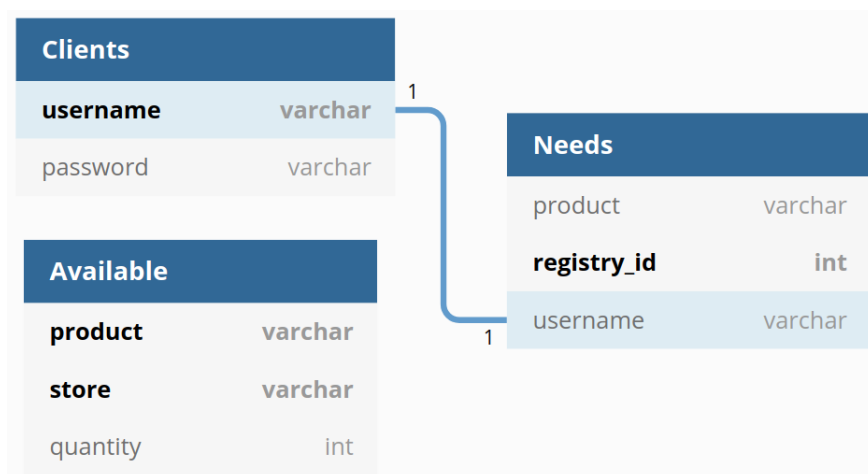


Figura 3: Tabelas usadas em PostgreSQL

Os parâmetros para a conexão à base de dados PostgreSQL são fornecidos através de um ficheiro chamado "DBProperties.xml". Este ficheiro fica ao mesmo nível que a pasta "SPN/".

As tabelas dedicadas para este projecto, são três: Clients, Needs e Available.

Na tabela Clients pretende-se que se guarde apenas a informação de um utilizador (Username e Password). Embora a password não seja utilizada nesta fase, o grupo considerou inserir este campo.

A tabela Needs contém todas as necessidades até ao momento. Sempre que uma necessidade é criada, é gerado um código de registo único.

A tabela Available tem todos os produtos que foram encontrados até ao momento. A quantidade neste projecto não foi usada, no entanto, numa segunda fase, iremos averiguar se este campo é necessário ou não. A nossa indecisão deve-se ao facto de que há artigos que são difíceis de serem quantificados, por exemplo: artigos que só se pode quantificar através do peso como é o caso da carne picada. Por outro lado, também se pode acontecer pedir duas embalagens de 500 gramas cada de camarão.

Na próxima fase já teremos decidido qual a melhor opção e retiramos ou não, o campo "quantity".

4 Cliente

O grupo decidiu que o nome do utilizador ia ser pedido durante o funcionamento da aplicação, porque o grupo pensa que no futuro será mais fácil quando se for implementar a parte gráfica. A parte de autenticação, que aparece na figura abaixo, está muito simples, isto é, pede apenas um username. A password será implementada numa segunda fase.

```
===== BEM VINDO =====  
Insira o seu username: █
```

Figura 4: Registo de um utilizador

Após se entrar com um username é mostrado um menu, tal como mostra a Figura 4.

```
===== Menu =====  
1 -> Consultar Necessidades  
2 -> Registar Necessidade  
3 -> Consultar Produtos  
4 -> Reportar Produto  
5 -> Consumir Produto  
6 -> Sair da Aplicação
```

Figura 5: Menu da interface

Nesta figura as opções do menu têm como objectivo:

1. Consultar Necessidades
 - Esta opção mostra todas as necessidades do utilizador com um certo "username".
2. Registar Necessidades
 - Envia ao servidor a informação do produto que o utilizador precisa.
3. Consultar Produtos
 - Mostra todos os produtos divulgados por todos os utilizadores.

4. Reportar Produto

- O cliente envia uma informação ao servidor a informar que foi visto um certo produto numa determinada loja.

5. Consumir Produto

- Um cliente informa a base de dados que vai consumir o produto.

6. Sair da Aplicação

- Faz com que a aplicação termine.

4.1 Base de Dados

User	
username	varchar
product	varchar
operation	int
store	varchar

Figura 6: Estrutura da tabela User na base de dados embebida

O programa "client" usa uma base de dados embebida que tem como objectivo guardar todos os pedidos feitos ao servidor que, por algum motivo, deu erro. Normalmente estes pedidos são registos: registo de necessidades e registo de produtos encontrados. O grupo não viu a necessidade de implementar mais tabelas para além da "User".

4.2 Thread

A thread , tal como falado na Introdução, que tem dois funcionalidades:

1. Mostrar um aviso caso já tenham sido encontrados artigos que o utilizador precisa.
2. Verifica se existe pedidos que não chegaram ao servidor. Caso exista algum, esta Thread vai tentar reenviar os pedidos para o servidor.

5 Makefile

Tal como foi dito na Introdução, este projeto usa um Makefile. As operações que estão neste ficheiro são:

- *\$make compile*
 - Cria a directoria "build/classes/", compila todos os ficheiro com a extensão .java e coloca os ficheiros compilados nessa directoria.
- *\$make start_rmi*
 - Inicia os serviço de nomes.
- *\$make server*
 - Inicializa o servidor. A cada execução é colocado o jar para o PostgreSQL. **O servidor só funciona quando o serviço de nomes estiver ativo.**
- *\$make client*
 - Inicializa o cliente. A cada execução é usado o jar para o HyperSQL.
- *\$make clean*
 - Apaga todos os ficheiros compilados na directoria build/classes/.

Tanto o servidor como o cliente, recebem argumentos antes de serem executados, no entanto, essa informação é passada no próprio Makefile.

6 Javadoc

The screenshot displays the Javadoc index.html page. On the left, there is a sidebar with two sections: 'Packages' and 'All Classes'. The 'Packages' section lists: spn.client, spn.db, spn.db.tabelas, spn.embedded_db, and spn.server. The 'All Classes' section lists: AccessDBProperties, Available, Client, Clients, DBConnection, DBConnector, DBOperations, EmbeddedDBConnection, FailedRequestOrganizer, Needs, Servico, ServicoImpl, Servidor, and TerminalInterface. The main content area has a top navigation bar with 'PREV', 'NEXT', 'FRAMES', and 'NO FRAMES'. Below this is a 'Packages' section with a table listing the packages and their descriptions. The table has two columns: 'Package' and 'Description'. The packages listed are spn.client, spn.db, spn.db.tabelas, spn.embedded_db, and spn.server. Below the table is a navigation bar with 'OVERVIEW', 'PACKAGE', 'CLASS', 'USE', 'TREE', 'DEPRECATED', 'INDEX', and 'HELP'. The 'OVERVIEW' tab is selected. At the bottom, there is another navigation bar with 'PREV', 'NEXT', 'FRAMES', and 'NO FRAMES'.

Package	Description
spn.client	
spn.db	
spn.db.tabelas	
spn.embedded_db	
spn.server	

Figura 7: Javadoc em index.html

A pasta "Javadoc" tem as informações necessária sobre todas as classes implementadas neste projecto e situa-se na directoria "so2-t01-37514-38073". Para abrir a documentação é preciso aceder ao ficheiro "index.html".

7 Problemas

7.1 Problema 1

```
4
Insira o produto: X
                !!! Atenção !!!
Foi encontrado um ou mais produtos que podem ser do seu interesse!
=====
||      Produto      |      Loja      ||
=====
||      carne       |      jumbo      ||
=====
peixe
```

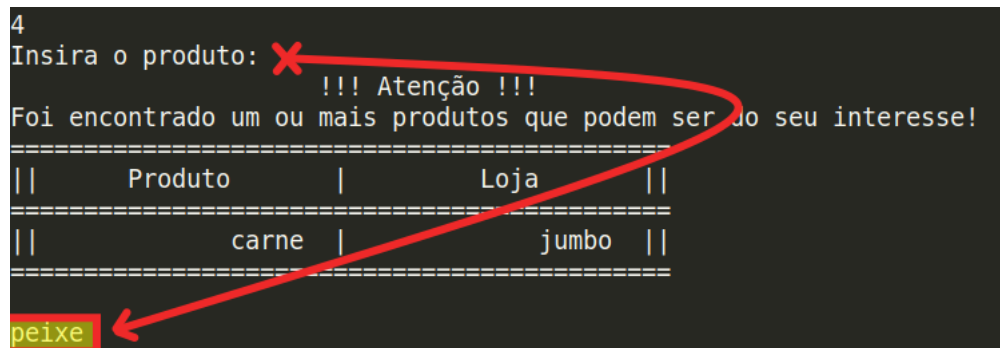


Figura 8: Problema com a thread

Uma vez que usamos uma Thread para anunciar a existência de artigos que estão disponíveis que o utilizador precisa, existe a possibilidade, sempre que se vai inserir um comando na interface ou se vai adicionar um novo produto ou uma nova necessidade, e há algum artigo a reportar, de aparecer este aviso no meio dessas operações. Embora esta situação não seja elegante, o programa continua a funcionar sem qualquer problema.

7.2 Problema 2

Este problema, também derivado ao uso da Thread, acontece quando se escolhe a opção 6 (Sair da Aplicação). Quando se insere esta opção, normalmente o programa demora, no pior dos casos, 10 segundos para encerrar. Tal acontece porque a Thread tem um método que suspende a sua execução durante 10 segundos.

7.3 Problema 3

Sempre que há uma nova necessidade é criada um código único e sempre que as necessidades são consultadas aparece o produto juntamente com esse código. O grupo não conseguiu encontrar mais nenhuma situação onde se poderia usar melhor este código.

8 Conclusão

Tirando os problemas enunciados, o grupo considera que o trabalho cumpre com o que foi pedido no trabalho.

Numa próxima fase tentaremos corrigir não só os problemas enunciados como também verificar se o campo "quantity", da tabela "Available", permanece.

9 Referências

- [1] José Saias. Aulas práticas
- [2] PostgreSQL 11.7. <https://www.postgresql.org/docs/11/index.html>
- [3] HyperSQL. <http://hsqldb.org/doc/2.0/guide/index.html>
- [4] Class Properties. <https://www.baeldung.com/java-properties>