Sistemas Distribuídos

MIEI - 3º ano - 1º semestre
Universidade do Minho

Trabalho Prático - Alarme Covid

Grupo 16

Luís Manuel Pereira - A77667
Ricardo Miguel Santos Gomes - A93785
Sara Alexandra Gomes Marques - A89477

Braga, 25 de janeiro de 2021

Conteúdo

1 Introdução **2**

2 Código Desenvolvido **3**
2.1 Cliente 3
2.1.1 Client 3
2.1.2 AdminMenu 3
2.2 Packet 4
2.3 CovidAlarm 4
2.3.1 User 5
2.3.2 Position 5
2.3.3 UserMap 5
2.4 Servidor 5
2.4.1 ObjectStream 5

3 Conclusão **6**

1. Introdução

Este relatório surge no âmbito da unidade curricular de Sistema Distribuídos integrada no 3º ano do Curso de Engenharia Informática.

Neste trabalho foi desenvolvido um programa com uma funcionalidade básica semelhante à aplicação STAYAWAY COVID. Deste modo, o código desenvolvido deverá permitir a um utilizador comunicar com o servidor para conhecer o número e localização de outros utilizadores potencialmente infetados com Covid-19, para além de comunicar a sua própria localização e casos de contágio.

Para este fim foi desenvolvido o programa descrito de seguida, utilizando a linguagem de programação Java e recorrendo aos métodos estudados nesta UC (sockets, threads e locks/unlocks).

2. Código Desenvolvido

2.1 Cliente

Os utilizadores poderão interagir com o programa através das interfaces (menus) implementadas na classe *Client*. Para além disto, também existirá uma classe *AdminMenu* com um funcionamento semelhante à *Client*, mas cuja interface está prevista para o uso exclusivo do administrador do sistema.

2.1.1 Client

Esta classe implementa a interface com a qual um utilizador interage após o seu login ou registo no sistema. É de notar que a opção "Consultar Mapa"será apenas visível para um utilizador classificado como especial (cujo registo poderá apenas ser feito através do menu de administrador).

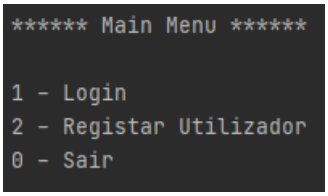

Figura 2.1: Menu inicial do Client

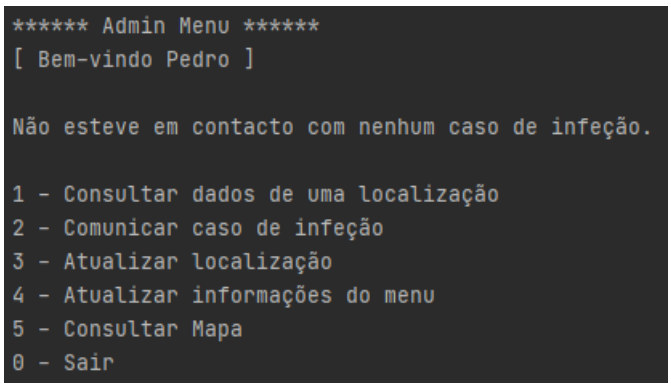

Figura 2.2: Menu de um utilizador especial

2.1.2 AdminMenu

Esta classe implementa a interface que será exclusivamente utilizada pelo administrador do sistema, logo existe separadamente da aplicação usada pelos restantes utilizadores, e apresenta opções diferentes no seu menu.

```
****** Admin Menu ******

1 - Registar Utilizador Especial
2 - Consultar Contas
3 - Consultar Mapa
4 - Gravar Servidor
0 - Sair
```

Figura 2.3: Menu do Administrador

2.2 Packet

Todas as instruções recebidas por um utilizador a partir da classe *Client* ou *AdminMenu* serão traduzidas num *Packet* que inclui toda a informação relevante para o sistema depois efetuar alterações e/ou apresentar os dados registados no mesmo.

```
public class Packet implements Serializable {
    private final int option;
    private final String username;
    private final String password;
    private final Boolean state;
    private final Boolean special;
    private final int m;
    private final int n;
```

Figura 2.4: Informação incluída num Packet

Cada *Packet* gerado terá a sua informação transmitida por *DataOutputStreams* e *DataInputStreams* durante a comunicação Cliente/Servidor.

2.3 CovidAlarm

A classe *CovidAlarm* realiza a ligação entre Cliente e Servidor, tratando de cada pedido do utilizador mediante a informação incluída no *Packet* recebido e chamando os métodos necessários à sua realização.

Para além disto, a classe guarda toda a informação relevante relativamente aos utilizadores incluídos no sistema e aos seus deslocamentos.

```
public class CovidAlarm implements Serializable {
    @Serial
    private static final long serialVersionUID = 3726281774063155278L;
    private Map<String, User> users;
    UserMap usermap;
    private ReentrantLock lock;
    private String info;
```

Figura 2.5: Classe CovidAlarm

É também importante notar que a realização de qualquer pedido do utilizador nesta classe, por depender da consulta e/ou alteração de dados do servidor, está limitada por *locks* para evitar conflitos aquando a utilização da aplicação por vários utilizadores (implicando o uso de vários *threads*).

2.3.1 User

A classe *User* inclui toda a informação relativa a um utilizador do sistema: o seu nome de utilizador, password, estado de infeção, tipo (se é especial ou não), a lista de utilizadores próximos de si (na mesma posição), e a sua posição no mapa.

```
public class User implements Serializable {
    @Serial
    private static final long serialVersionUID = -7149009667160496245L;
    private String username;
    private String password;
    private Boolean state; // Infected: True or False
    private Boolean special;
    private List<User> nearbyUsers;
    private Position current;
```

Figura 2.6: Dados de um *User*

2.3.2 Position

Esta classe inclui dois integers que definem uma dada posição no mapa. É relevante notar que também foi definida a função hashCode() nesta classe, de modo a permitir o seu uso como *key* no HashMap usado pelo *UserMap*.

2.3.3 UserMap

A classe *UserMap* inclui um Map que agrupa, para cada posição, a lista de *Users* nela presentes.

2.4 Servidor

Podemos considerar que o servidor do sistema está dividido entre as classes *Server* e *ServerWorker*. O *ServerWorker* recebe um *socket* associado a cada cliente que se conecta ao servidor, para além do *covidAlarm* correspondente aos dados guardados em memória, e inicializa o servidor. Por sua vez, o *Server* irá iniciar uma nova *thread* com cada *ServerWorker* criado.

2.4.1 ObjectStream

Consideramos útil a criação de uma classe *ObjectStream*, com o objetivo de permitir guardar e carregar o estado do servidor. Para isto, recorremos ao uso da classe *CovidAlarm* para organizar os dados, e desenvolvemos as funções saveServer() e loadServer() com vista a permitir o armazenamento dos mesmos num ficheiro ServerData.dat.

3. Conclusão

Através do desenvolvimento deste projeto, consideramos que o grupo adquiriu uma compreensão mais aprofundada do modo de funcionamento de um sistema baseado em comunicações cliente/servidor através de *sockets*, para além da importância do uso de *threads* em aplicações que requerem o acesso a dados por parte de múltiplos utilizadores simultaneamente.

A implementação de notificações de aviso revelou-se um ponto particularmente desafiante devido à sua dependência do movimento dos restantes utilizadores da aplicação, remetendo para a importância do controlo de concurrência e da organização de dados necessária neste tipo de sistemas. Consideramos também que o código desenvolvido poderia ter sido mais extensivamente comentado.

Em geral, acreditamos que foi elaborado um bom trabalho, tendo este cumprido todos os requisitos propostos.