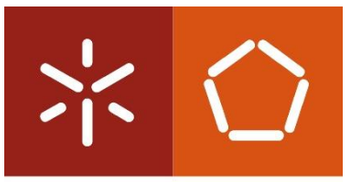

Universidade do Minho

Departamento de Informática

Sistemas Distribuídos

Alarme Covid

Grupo Nº 27

Gustavo Lourenço (A89561)
Leonardo Marreiros (A89537) Martim Almeida (A89501)

25 de janeiro de 2021

Conteúdo

1 Introdução **2**

2 Arquitetura e Solução do Projeto **2**

2.1 Classes . . . 2

2.1.1 Mapa . . . 2

2.1.2 Utilizador e Utilizadores . . . 2

2.1.3 NotificaLocalLivre, VerificaContacto e Receiver . . . 3

2.1.4 Sistema . . . 3

2.1.5 Servidor . . . 3

2.1.6 Cliente . . . 3

2.2 Comunicação Cliente-Servidor . . . 3

2.2.1 TaggedConnection . . . 3

2.2.2 Demultiplexer . . . 3

2.2.3 ByteArrayInputStream . . . 3

2.2.4 ByteArrayOutputStream . . . 4

2.3 Funcionalidades do Sistema . . . 4

2.3.1 Register . . . 4

2.3.2 Login . . . 4

2.3.3 Lotacao . . . 4

2.3.4 Move . . . 4

2.3.5 Disponibilidade . . . 4

2.3.6 Infecao . . . 5

3 Conclusão **5**

Capítulo 1

Introdução

O Projecto trata-se da implementação de uma plataforma *Alarme Covid* através do uso de uma comunicação Cliente-Servidor, utilizando Sockets e Threads.
A Plataforma terá as seguintes funcionalidades:

- Autenticação e Registo de um Utilizador. Sempre que o Utilizador deseja utilizar uma das funcionalidades da Plataforma, terá que estar autenticado;
- Informar a localização do Utilizador ao Servidor;
- Informar o Utilizador sobre o número de pessoas presentes numa localização;
- Informar o Utilizador quando uma determinada Localização está vazia, de forma a poder mover-se para lá;
- Comunicar ao Servidor que se encontra Infetado. Neste ponto, o Utilizador estará incapacitado de interagir com o servidor, uma vez que supostamente estará em isolamento.
- O Servidor informar todos os Utilizadores que tenha cruzado com um Utilizador que se encontra doente, uma vez que podem estar em risco de contágio;

Capítulo 2

Arquitetura e Solução do Projeto

2.1 Classes

2.1.1 Mapa

Como apresentado no enunciado, a plataforma consta com a implementação de um mapa por onde os Utilizadores se movimentam. Nesta classe implementamos um *ReentrantLock*, de forma a dar Lock do Mapa sempre que necessário, e uma Matriz N x N da classe privada Local, apresentada posteriomente.

Local

A classe Local que representa um Local apresentado no Mapa consta com a implementação de um *ReentrantReadWriteLock*, de uma *Condition*, da lotação do local e de uma Lista que contém todos os *Usernames* dos vários Utilizadores que se encontram atualmente no Local.

2.1.2 Utilizador e Utilizadores

Como, novamente, apresentado no enunciado, a plataforma necessita de utilizadores, uma vez que a comunicação entre Cliente e Servidor só é estabelecida caso se esteja autenticado

pelo Servidor. Assim, a Classe Utilizador contém toda a informação sobre o Utilizador (*Username*, *Password*, se encontra Infetado, se está *Logged In* à plataforma e localização atual.) Para além disso, o Utilizador consta com a implementação de um *ReentrantReadWriteLock*, de uma *TreeSet* que contém os *Usernames* do vários Utilizadores do Sistema que o Utilizador se cruzou e com os métodos de serialização dos dados do Utilizador.
A Classe Utilizadores é responsável pelo armazenamento e manutenção do Utilizadores presentes na Plataforma. Os utilizadores são armazenados num *HashMap*, cuja *Key* é o *Username* do Utilizador. Esta classe consta também com a implementação de um *ReentrantLock*.

2.1.3 NotificaLocalLivre, VerificaContacto e Receiver

Classes Auxiliares à implementação da funcionalidade de verificar se um local se encontra vazio e da funcionalidade de informar os Utilizadores que tenham cruzado com um Utilizador infectado.

2.1.4 Sistema

Classe responsável pela execução de todas os serviços enviados de um Cliente ao Servidor da Plataforma e contém a implementação do Mapa, de tamanho 10 x 10, e do armazenamento dos vários Utilizadores presentes na Plataforma.

2.1.5 Servidor

O Servidor da Plataforma contém a implementação de um *Worker*, responsável pelo estabelecimento da comunicação com o Cliente da Plataforma. Para além disso, o Servidor contém a informação de quantos *Workers* existem por conexão e o Tamanho do Mapa (valores que são 3 e 10 por defeito respectivamente). Consta também com a implementação de uma Lista que contém as várias *TaggedConnections* realizadas, sendo que esta lista é necessária para a realização da Funcionalidade de avisar um Utilizador sobre ter estado em contacto com outro Utilizador infetado.

2.1.6 Cliente

O Cliente é a Classe pela qual um Utilizador consegue interagir com o Servidor e com o Sistema da Plataforma. Para cada pedido efetuado, é criada uma nova *thread*.

2.2 Comunicação Cliente-Servidor

2.2.1 TaggedConnection

É pela TaggedConnection que o Cliente e o Servidor fazem os seus *Requests* e *Replies* através do envio de *Frames*.

A Frame é um Classe que contém a *Tag* do Pedido do Cliente, a opção da *Request* do Cliente, ou seja, a Funcionalidade pedida e um *Array* de **Byte** que contém a informação transmitida entre o Cliente e o Servidor.

2.2.2 Demultiplexer

Esta classe encapsula uma *TaggedConnection* e tem como objetivo garantir que as *Replies* enviadas a uma TaggedConnection cheguem à *Thread* apropriada. Para isso, o *Demultiplexer* delega o envio de mensagens por parte do Cliente para a *TaggedConnection* e disponibiliza a operação *receive* que bloqueia a *Thread* invocadora até chegar uma mensagem com a *Tag* especificada por esta, devolvendo então o contéudo.

2.2.3 ByteArrayInputStream

Utilizamos *ByteArrayInputStream* para conseguir separar o *Array* de *Bytes* obtido através da receção de uma mensagem pela *TaggedConnection*/*Demultiplexer* nas suas devidas Variáveis ou Classes.

2.2.4 ByteArrayOutputStream

Utilizamos *ByteArrayOutputStream* para conseguir enviar Variáveis ou Classes como um *Array* de *Bytes* através da *TaggedConnection/Demultiplexer*.

2.3 Funcionalidades do Sistema

2.3.1 Register

Neste comando, o cliente escreve uma linha com o nome de utilizador, palavra-passe e coordenadas onde se encontra.

Primeiramente, a linha é dividida em *Tokens* e é criado um novo objeto Utilizador, objeto esse que é enviado para o Servidor fazendo uso da *TaggedConnection* com a Opção "1". No Servidor, é recebido o Utilizador enviado pelo Cliente e é verificado se esse utilizador existe na coleção de Utilizadores existentes nesse momento. Se existir, vai enviar uma resposta na forma de um *Boolean* de volta ao Cliente, onde vai ser gerada uma *UserExistsException*. Caso não exista, se as coordenadas inseridas estiverem dentro do tamanho do mapa escolhido, o novo utilizador é adicionado à coleção e é também adicionado ao Mapa na posição pretendida. No caso das coordenadas inseridas serem uma posição inválida, é enviada uma resposta ao Cliente e gerada uma *TamanhoInvalidoException*.

2.3.2 Login

Neste comando, o cliente escreve uma linha com o nome de utilizador e a respetiva palavra-passe. Novamente, esta linha é *parsed* e enviado o nome de utilizador e palavra passe ao Servidor fazendo uso da *TaggedConnection* com a Opção "2". No Servidor, caso a palavra passe inserida seja igual à palavra-passe presente na coleção de utilizadores referente ao utilizador em questão, se o utilizador não tiver informado infeção anteriormente e se não estiver com sessão iniciada noutro dispositivo, então é feito o *Login* com sucesso.

Caso a palavra-passe esteja errada, é enviada uma resposta ao Cliente e lançada uma *PasswordMissmatchException*.

Caso o utilizador tenha informado contágio anteriormente (e tenha inserido a palavra-passe correta), não consegue entrar na conta, é enviada uma resposta ao Cliente e lançada uma *InfectedUserException*.

Na eventualidade do utilizador inserir a palavra-passe correta e não estar infetado mas estiver com sessão iniciada noutro dispositivo, é gerada uma *UserAlreadyLoggedInException*.

2.3.3 Lotacao

Neste comando, o cliente escreve uma linha com as coordenadas da posição sobre a qual pretende saber a lotação. Esta linha é *parsed* e são enviados os diferentes componentes para o Servidor fazendo uso da *TaggedConnection* com a Opção "3".

No Servidor, caso as coordenadas sejam uma posição válida, é enviada ao cliente a resposta com a lotação da posição pretendida.

Caso as coordenadas sejam inválidas, é enviada uma resposta ao Cliente e gerada uma *TamanhoInvalidoException*.

2.3.4 Move

Neste comando, o cliente escreve uma linha com as coordenadas da posição para a qual se pretende deslocar. Esta linha é *parsed* e são enviados os diferentes componentes para o Servidor fazendo uso da *TaggedConnection* com a Opção "4".

No Servidor, caso as coordenadas sejam uma posição válida, o utilizador em questão é trocado de local no Mapa e é enviada uma mensagem de confirmação para o cliente.

Caso as coordenadas sejam inválidas, é enviada uma resposta ao Cliente e gerada uma *TamanhoInvalidoException*.

2.3.5 Disponibilidade

Neste comando, o cliente escreve uma linha com as coordenadas da posição para a qual se pretende saber se está livre. Esta linha é *parsed* e são enviados os diferentes componentes para o Servidor fazendo uso da *TaggedConnection* com a Opção "5".

Caso as coordenadas sejam uma posição válida, o Servidor irá notificar o Cliente assim que a localização estiver livre. Isto é conseguido através da criação de uma *Thread* que utiliza a classe *NotificaLocalLivre* que irá enviar a resposta para o Cliente quando o Local estiver livre. Isso ocorre através da *Condition* do Local (obtida através do seu *WriteLock*) e dos seus métodos *await()* e *signalAll()*.

Caso as coordenadas sejam inválidas, é enviada uma resposta ao Cliente e gerada uma *TamanhoInvalidoException*.

2.3.6 Infecao

Neste comando, o cliente informa que contraiu a infeção e é enviado para o servidor o utilizador em questão fazendo uso da *TaggedConnection* com a Opção "6".

No servidor, esse utilizador é dado como infetado, é removido do Mapa, é enviada uma resposta de confirmação ao Cliente onde é feito o *Logout* automático da conta.

Além disso, são notificados todos os utilizadores que estiveram em contacto com o utilizador infetado, através do envio da lista dos clientes que entraram em contacto com o cliente infectado para uma *Thread* que utiliza a classe *VerificaContacto* para verificar se o utilizador com sessão iniciada por um Cliente se encontra nesta lista.

Capítulo 3

Conclusão

No desenvolvimento deste trabalho, achamos que todo o material fornecido pela UC, seja das Aulas Práticas como das Aulas Teóricas, bastante útil para o desenvolvimento deste Projeto. Com a sua realização, conseguimos ter um conhecimento mais abrangente e uma maior destreza na utilização de *threads*, paralelamente aprofundando a capacidade de entender como gerir concorrência e acesso a dados de uma forma segura entre um servidor e vários clientes.

Encontramos certas dificuldades na comunicação Cliente-Servidor, principalmente antes da utilização do *ByteArrayInputStream* e do *ByteArrayOutputStream*.

Achamos um aspeto a melhorar neste trabalho a separação dos métodos com o Sistema Distribuído implementado por estes.

Apesar de não implementarmos a 2ª Funcionalidade Adicional, achamos que foi um trabalho bastante conseguido.