Sistemas Distribuídos

Grupo 50
Guilherme Martins a89532
João Pereira a89607
José Costa a89519
Tiago Freitas a89570

A89532 a89607 a89519 A89570

Conteúdo

1 Introdução

Neste trabalho foi-nos proposto o desenvolvimento de uma plataforma inspirada na aplicação *STAYAWAY COVID*, baseada no rastreamento de possíveis focos de contágio, com o objetivo de prevenir um utilizador de se infetar e, caso já esteja infetado, alertar com quem ele já esteve.

Este sistema funcionará sob a forma de cliente/servidor em Java, com a utilização de *sockets* e *threads*, em que o cliente, pré-registado ou não, se autentica e pode comunicar a sua localização e que está infetado, se for o caso. Tem também acesso à localização de outros utilizadores no mapa, de forma a poder movimentar-se de uma maneira mais segura e poderá, por fim, receber notificações, quando alguém com quem esteve indica que está infectado e quando uma localização que ele pediu para verificar, se encontra vazia.

2 Classes

2.1 AlarmeCovid

De seguida vão ser descritas as classes correspondentes à lógica de negócio **AlarmeCovid**.

2.1.1 AlarmeCovidLN

Classe responsável por armazenar e gerir toda a informação do programa (servidor), isto é, tudo é que é relacionado com os utilizadores e o mapa do sistema, assim como possui os métodos responsáveis pelas funcionalidades que o servidor acede para obter informação do estado do sistema, de modo a responder aos pedidos do cliente. Optamos por utilizar um mapa de tamanho N = 10, com o intuito de termos um bom espaço para testes e, ao mesmo tempo meramente demonstrativo de como o servidor se comportaria numa situação real.

2.1.2 Utilizador

Classe que representa um utilizador do programa, caracterizado por *username*, *password* e sua localização, onde também podemos saber se este está ou não infetado, *loggado*, em risco e se tem permissão para aceder ao mapa. Cada utilizador contém a lista de pessoas pelas quais já passou.

2.1.3 Célula

Classe que representa cada célula da matriz relativa ao mapa utlizado, e que guarda a informação de quem esteve ou está em cada localização.

Possui também a informação de quantas pessoas se encontram lá, apenas no momento atual.

2.1.4 Localização

Classe que consiste num par de coordenadas (x,y) de um determinado utilizador, isto é, a linha e a coluna da matriz (mapa).

2.2 Cliente

Vamos, agora, explicar como funciona o cliente e como se interliga com o resto do sistema.

2.2.1 ClienteSimples

Classe responsável por apresentar ao cliente a interface, a partir da qual ele pode interagir, e por receber as mensagens enviadas pelo cliente, que serão passadas, através do *stub* ao servidor. As mensagens recebidas podem ser:

- **Login**: para efetuar o *login* do utilizador;
- **Register**: para registar um utilizador novo;
- **Comunicar localização atual**: comunica a localização do utilizador;
- **Verificar ocupação de uma localização**: comando que serve para saber o número atual de utilizadores numa dada posição;
- **Verificar se uma localização está vazia**: comando que nos permite ser notificados quando uma dada localização (que pedimos ao servidor) está vazia
- **Comunicar infeção**: utilizador comunica que está infetado;
- **Imprimir mapa de ocupações e doentes**: permite ao utilizador com autorização especial que imprima o mapa com a quantidade de utilizadores e doentes visitaram cada localização;
- **Verificar se está em risco de estar infetado**: permite ao utilizador saber se está em risco de estar infetado face às localizações que frequentou.

2.2.2 Stub

Classe que tem a função de enviar informação proveniente e que constitui os pedidos por parte do cliente para o servidor, através de um *socket* com a utilização de *DataInputStream* e *DataOutputStream*.

2.2.3 Barreira

A barreira foi introduzida como uma forma de sincronização entre os leitores e os escritores, de modo a estabelecer uma ordem de quem escreve e quem lê em cada pedido feito pelo cliente. Esta classe é fundamental para que cada thread só atue no momento exato em que deve atuar, sem interromper o funcionamento normal do programa. Tanto a barreira como as threads só são usadas após o *login*.

2.3 Servidor

Definiremos, a seguir, todas as funcionalidades e características do servidor implementado.

2.3.1 SimpleServerWithWorkers

Classe que cria o servidor *multi-threaded* de *sockets* a usar e que contém a lógica de negócio, de forma a poder executar as diferentes *queries*. De seguida este envia o resultado para o cliente, através do *stub*, que depois disponibiliza a informação no ecrã.

2.4 Exceptions

Por fim, vamos enumerar e descrever todas as excepções usadas para o melhor funcionamento do sistema.

2.4.1 FromServerException

Esta exceção foi utilizada no nosso sistema sempre que as operações do servidor não concluíssem como deveriam concluir num fluxo normal. Permitiu-nos enviar mensagens personalizadas a comunicar o erro.

3 Funcionalidades

3.1 Básicas

3.1.1 Login

Para efetuar o login, o cliente deve selecionar a opção 1 (Login) no menu inicial e posteriormente indicar o seu *username* e *password*. O login é feito através do método **login** da lógica de negócio e só acontece caso o utilizador já esteja registado e se este não estiver infetado.

3.1.2 Registar

Para efetuar o registo, o cliente deve selecionar a opção 2 (Registar) no menu inicial. O registo é feito através do método **register** que inicialmente verifica se o cliente já está registado e caso não esteja, regista-o no servidor.

3.1.3 Comunicar localização atual

A execução deste comando pede ao cliente que indique as suas coordenadas que terão de ser diferentes das suas atuais e faz com que o utilizador em questão fique com nova localização e guarde todos os *usernames* dos utilizadores que se encontram nessa localização, e que estão *loggados*, nos seus *contactos*, assim como estes guardam o seu. Além disso a célula correspondente a essa localização atualiza as suas variáveis de instância.

3.1.4 Verificar ocupação de uma localização

Este comando, através das coordenadas fornecidas pelo cliente, determina o número de utilizadores numa posição. Primeiro verifica se as coordenadas são válidas e depois invoca o método *getOcupacao* que conta o número de utilizadores com login feito que possuem uma localização atual equivalente à localização dada pelo utilizador.

3.1.5 Verificar se localização está vazia

Para realizar este comando, criamos uma nova thread que "adormece", até que o número de pessoas presentes nessa localização seja nulo, e quando isto acontece, ocorre um *signalAll* para notificar o cliente de que já pode avançar para lá.

3.1.6 Comunicar infeção

Quando o cliente comunica que está infetado com COVID-19 não terá mais acesso ao programa. Ainda, todos os outros utlizadores constituintes dos contactos do cliente são colocados como "em risco"e são notificados de tal, a partir do *signalAll*.

3.2 Adicionais

3.2.1 Imprimir mapa de ocupações e doentes

Esta opção permite aos utilizadores com autorização especial imprimir o mapa do número de utilizadores e o número de infetados que já estiveram em todas as posições, sendo ambos os números separados por uma '/'. Este invoca o método *getOcupacoes* que irá calcular para cada coordenada o tamanho de utilizadores na primeira posição e irá calcular o número de

utilizadores que pertencem à lista de infetados anteriormente obtida através do método *getInfetados* na segunda posição.

3.2.2 Verificar se está em risco de estar infetado

O cliente, mal faça *login* ativa automaticamente este comando, que faz com que uma *thread* fique à espera do momento em que este fique em risco de estar contagiado para o poder notificar.

4 Conclusão

Consideramos que a realização deste projeto foi bem conseguido, visto que implementamos os requisitos básicos para resolver o problema do rastreio de contactos e deteção de aglomerado de pessoas, bem como as funcionalidades adicionais propostas também no enunciado, respeitando sempre os conceitos aprendidos em sistemas distribuídos. Surgiram-nos algumas dificuldades no controlo dos *end of files* e no que poderíamos fazer quando esse tipo de mensagens era recebido, e tentámos melhorar o projeto nesses pequenos pormenores, para ter um funcionamento do programa mais simples e fácil de entender.