Sistemas Distribuídos
Trabalho Prático - AlarmeCovid
Grupo 36

University of Minho, Department of Informatics,
4710-057 Braga, Portugal

2021

Margarida Faria Ana César
Angélica Cunha

e-mail: {a71924,86038,84398}@alunos.uminho.pt

25 de janeiro de 2021

Conteúdo

1 Introdução

O presente documento visa a esclarecer todo o procedimento realizado na abordagem do problema que nos foi proposto: a implementação de uma aplicação inspirada no problema do rastreio de contactos e deteção de concentração de pessoas, sob a forma de Cliente/Servidor em Java utilizando sockets e threads.

A plataforma pretende auxiliar o combate à Pandemia Covid-19 que mudou o Mundo como o conhecíamos, no ano de 2020.

Como o desenvolvimento da plataforma é em contexto académico, não foi considerável a proteção dos dados pessoais dos utilizadores.

Inicialmente iremos descrever a arquitetura Cliente/Servidor e a maneira como os dois interagem, e posteriormente separada e de forma detalhada o comportamento de cada um dos seus componentes.

2 Arquitetura Servidor/Cliente

Para o desenvolvimento da plataforma foi escolhida o modelo Servidor/Cliente, que consiste numa estrutura de aplicação distribuída onde são repartidas as tarefas entre os fornecedores de um recurso ou um serviço, designados como **Servidores**, e os requerentes desses serviços, utilizadores da aplicação, identificados como **Clientes**.

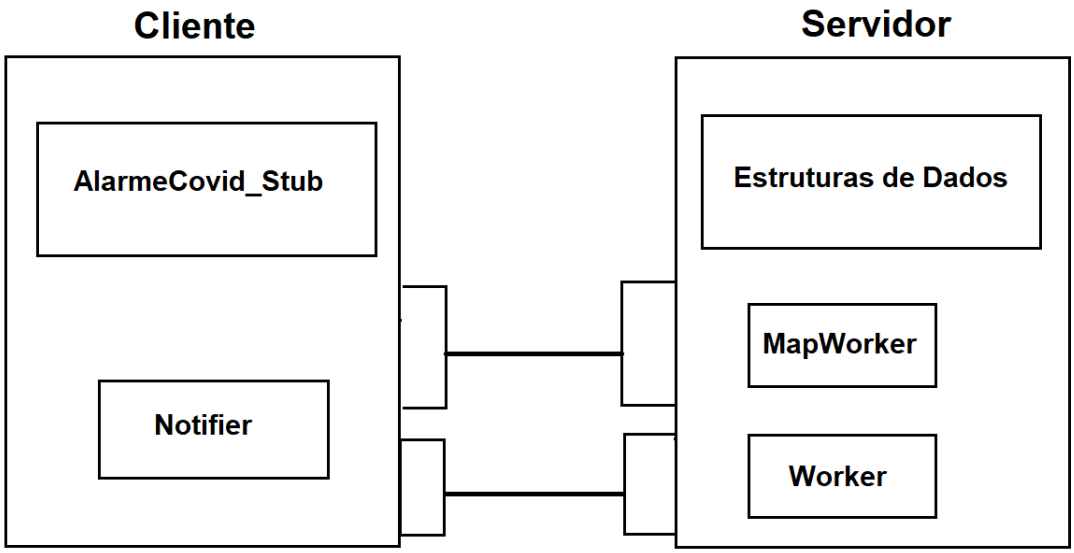

Figura 1: Arquitetura Servidor/Cliente

2.1 Cliente

A interação do Cliente com o Servidor é realizada através de uma aplicação local, responsável por enviar pedidos e receber as devidas respostas, da parte do Servidor. A aplicação torna a comunicação remota com o Servidor invisível aos utilizadores da aplicação.

Comparativamente ao Servidor, o Cliente possui menos grau de complexidade, uma vez que a sua função como utilizador da plataforma é enviar pedidos e trabalhar sobre as respostas recebidas, na forma de dados.

2.2 Servidor

Como já referido na secção anterior referente ao Cliente, o Servidor tem a função de responder aos pedidos efetuados pelos clientes, e além disso, fazer uma gestão dos clientes conectados.

É ainda essencial que a arquitetura do servidor conceder uma logística adequada dos diferentes recursos disponíveis.

As características do Servidor revêm-se nos seguintes pontos:

- Ficar à escuta de pedidos de conexão da parte dos Clientes;
- Aceitar os pedidos de conexão;
- Receber pedidos e responder aos mesmos de modo adequado;
- Controlo de Concorrência entre os clientes.

2.3 Comunicação

A comunicação utilizada entre os Clientes e o Servidor é feita através de **Sockets TCP**. O Protocolo TCP/IP, é um protocolo de rede, presente na camada de Aplicação do *Modelo Osi*, e é o mais indicado para permitir a comunicação na nossa plataforma, na medida em que, contrariamente a outros Protocolos, garante que tanto o Servidor como o Cliente enviem e recebam dados (*Full-Duplex*, é orientado à conexão, e tem um mecanismo de controlo de erros que permite que os dados sejam entregues sem erros.

2.4 Serialização de Dados

Sendo a comunicação entre Servidor/Cliente feita através da rede, a forma de serialização dos dados veio a demonstrar-se fundamental para o sistema ser o mais coeso possível.

O protocolo Servidor/Cliente é em formato binário, e recorremos a classes do Java que nos permitissem ler tipos de dados nessa representação, num modo independente da máquina. Sao estas classes **Data[Input/Output]Stream**.

3 Descrição do Sistema

Nesta parte do documento, iremos proceder a uma explicação detalhada de todas as partes que compõem o nosso Sistema.

Na fase inicial de recolha de requisitos, foi-nos proposto implementar as seguintes funcionalidades:

- Autenticação e registo de utilizador, dado o seu nome e palavra-passe;
- Manter o servidor informado sobre a localização atual do utilizador;
- O utilizador saber o número de pessoas numa dada localização;
- O utilizador saber quando não houver ninguém numa dada localização;
- Comunicação ao servidor de que o utilizador está doente.

Adicionalmente, o servidor deve notificar todos os utilizadores potencialmente contagiados por um doente, e um utilizador com autorização especial poderá descarregar um mapa indicando quantos utilizadores e quantos doentes visitaram cada localização.

Para conseguir responder aos pedidos, o Servidor terá que implementar estas funcionalidades, bem como o Cliente, portanto foi criada uma interface a ser implementada no lado do Cliente, e no lado do Servidor.

No lado do Cliente, a implementação da interface traduz-se na simulação da invocação dos métodos, garantindo uma comunicação remota com o Servidor.

No lado do Servidor, o objetivo é a implementação da lógica de acesso, feita de forma concorrente, às estruturas de dados.

```
public interface AlarmCovidInterface {

    void registration(String username, String password, String special_password)
            throws AlreadyRegistedException, SpecialPasswordInvalidException;
    boolean authentication(String username, String password)
            throws InvalidLoginException, QuarantineException;
    void notify_positive(String username);
    int nr_people_location(int node)
            throws InvalidLocationException;
    void notify_empty_location(String username, int node)
            throws InvalidLocationException;
    void update_location(String username, int new_location)
            throws InvalidLocationException;

    //special client
    void download_map(String username);
}
```

Figura 2: Interface Desenvolvida na Plataforma

3.1 Arquitetura Cliente

A imagem seguinte ilustra a abordagem da arquitetura do cliente.

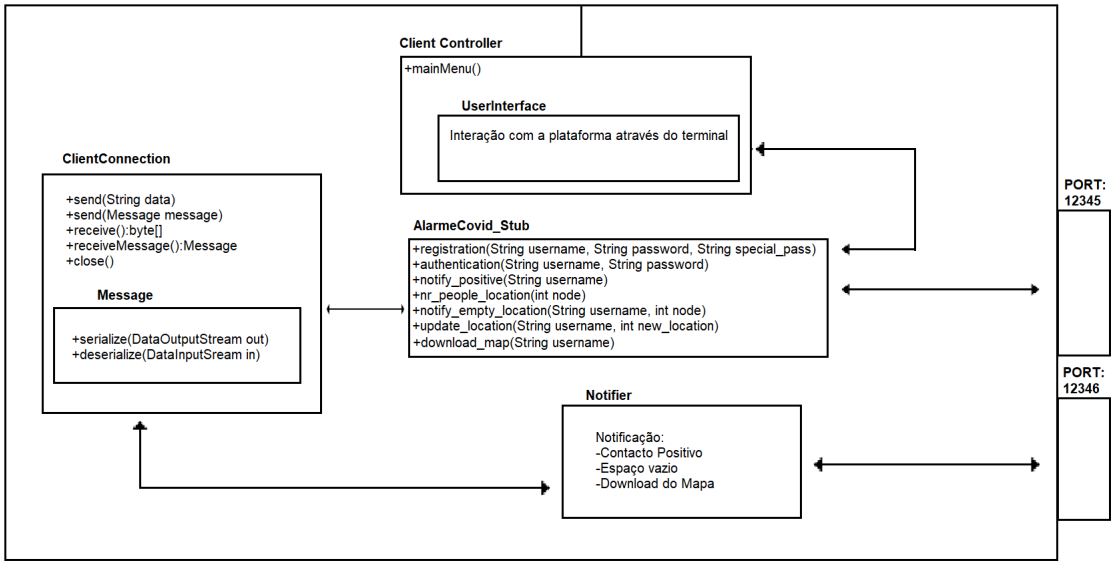

Figura 3: Arquitetura do Cliente

Quando é estabelecida uma conexão com um Cliente, é gerado um socket para a comunicação remota com o Servidor, cujo protocolo de comunicação é *request-reply*, cujo objetivo é enviar os pedidos, e receber as respostas correspondentes Após a autenticação, é criado outro socket que tem como funcionalidade receber as notificações.

3.1.1 AlarmeCovid Stub

A implementação da interface da plataforma AlarmeCovid, apresentada anteriormente como *AlarmeCovidInterface*, é feita através desta classe, onde é implementada a lógica de receção e envio de dados.

3.1.2 Notifier

O objetivo desta classe é que as notificações sejam apresentadas mesmo durantes processos com a plataforma, de uma forma independente do outro socket de comunicação. De tal modo, esta classe é usada por uma nova thread para lidar com as notificações. Após a autenticação do utilizador, é esta thread que fica à escuta das mesmas.

3.1.3 Client Connection

É nesta classe que é definida a estrutura *Message*, desenvolvida para tratar as notificações, e cujas variáveis de instância são a *tag*, cuja correspondência é que tag = 0 representa uma notificação de doença, tag = 1 refere-se a uma notificação de espaço vazio, e tag = 2 corresponde a um download do mapa, e *um array de bytes* onde são armazenados os dados.

3.2 Arquitetura Servidor

A seguinte figura demonstra a arquitetura do Servidor, assim como dos seus componentes.

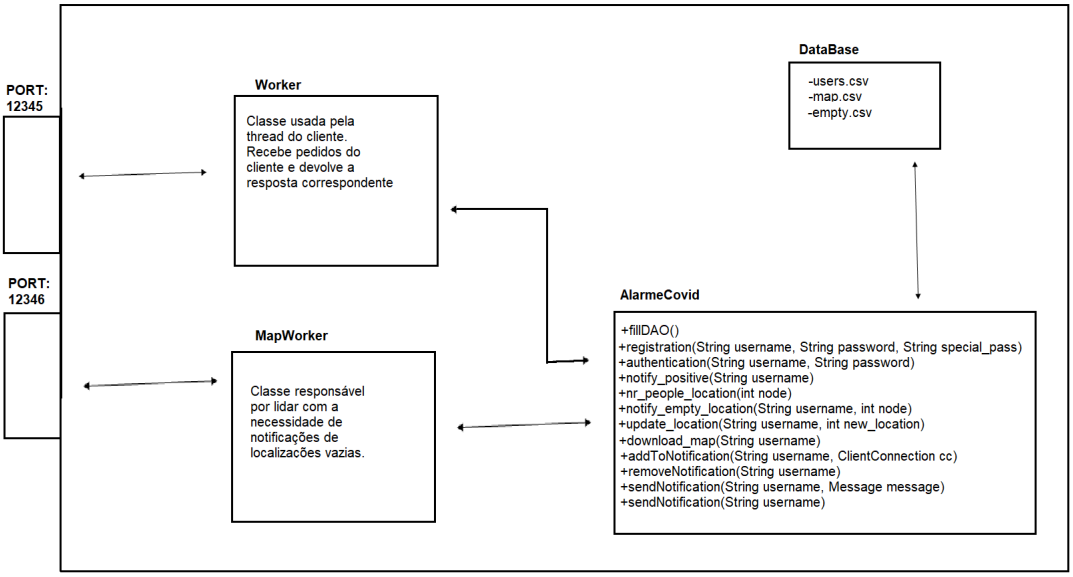

Figura 4: Arquitetura do Servidor

No momento em que o Servidor é inicializado, são criadas três threads:

- Para lidar com os pedidos de conexão ao socket cuja Port é *12345*, responsável por lidar com a interação com cada utilizador;
- É responsável por aceitar pedidos de conexão ao socket de notificações, cuja Port é *12346*.
- Para garantir controlo de concorrência ao nível do mapa, mais especificamente para lidar com a espera passiva de uma localização estar vazia.

3.2.1 AlarmeCovid

Analogamente ao que se verificou na Arquitetura do Cliente, esta classe implementa a interface *AlarmeCovidInterface*, e toda a lógica dos métodos para fornecer as devidas respostas aos pedidos correspondentes do Cliente. Também contém a funcionalidade de fazer a gestão das notificações dos utilizadores do Sistema.

3.2.2 Worker

A classe Worker implementa a interface *Runnable* e é usada pela thread reservada para cada Cliente. Inicialmente espera por pedidos de autenticação e de registo. É responsável por assegurar uma interação correta e fiável do sistema com o utilizador.

3.2.3 MapWorker

Similarmente à classe Worker, esta classe implementa a interface *Runnable* e é usada pelas threads que trabalham sobre cada localização no mapa.

3.2.4 DataBase

A abordagem utilizada foi a de guardar a informação em ficheiro de texto, em formato *csv*. É esta classe que é o caminho do Mapa de utilizadores e doentes por localização, que o utilizador especial tem a permissão de efetuar o download.

4 Conclusão

Embora a necessidade da realização deste projeto não seja de todo a mais desejável, foi uma boa oportunidade para pormos em prática a utilização dos recursos leccionados durante o semestre na Unidade Curricular de Sistemas Distribuídos.

O desenvolvimento da arquitetura Servidor/Cliente pode ser descrito como complexo, pois é composto por dois componentes distintos, que interagem entre si sem saber propriamente da existência de cada um.

Na parte do Cliente, começamos por uma abordagem idêntica à das aulas práticas mas a utilização do Stub veio-se a demonstrar mais relevante para a nossa abordagem ao problema, e consequentemente repensamos a solução inicial.

Na parte do Servidor, revelou-se mais trabalhosa o tratamento do controlo de concorrência em clientes do Sistema, mas a nossa abordagem conseguiu de algum modo desmistificar esse problema tão comum nos Sistemas Distribuídos.

Com a realização deste trabalho concluimos que o desenvolvimento de uma aplicação em sistemas distribuídas, com boa escalabilidade e gestão dos recursos fornecidos, é na verdade um problema com elevado grau de complexidade e intricações.