

Universidade do Minho  
Escola de Engenharia  
Mestrado Integrado em Engenharia de Telecomunicações e Informática

**Sistemas Distribuídos**

# **Estimativa de infetados durante uma pandemia**

## **Relatório do projeto**

Grupo V

Bruno Santos A72122

Hugo Reynolds A83924

Manuel Mendes A77806

Guimarães  
21 de Maio de 2020

# Capítulo 1

## Introdução

Este trabalho, para a unidade curricular de Sistemas Distribuídos, consiste no desenvolvimento de um servidor que permite recolher uma proporção de infetados no decorrer de uma pandemia. Cada cliente deve registar-se e autenticar-se para se ligar ao servidor. Seguidamente, o servidor fica aguardando uma resposta de um cliente que consiste na comunicação de quantos casos de doença este conhece nos seus contactos. Sempre que algum cliente fornece informação ao servidor, este envia a todos os clientes ligados uma nova estimativa da proporção média. Os clientes podem indicar valores de casos de doença as vezes que entenderem até fecharem a conexão.

Para o respetivo projeto, foram elaborados os programas Servidor e Cliente em linguagem Java.

Como controle de versões, decidimos utilizar o *gitHub*. O repositório do presente projeto é: <https://github.com/BrunoXBSantos/Distributed-Systems-Java.git>.

Para a elaboração do presente relatório, optamos por utilizar o software  $\text{\LaTeX}$ . Sendo este software uma excelente forma de produzir trabalhos científicos, relatórios e textos académicos com excelente qualidade.

## Capítulo 2

# Programa Servidor

O cerne do presente projeto foi o desenvolvimento de um servidor que permitisse recolher estimativas da proporção de infectados numa pandemia.

O programa Servidor é composto por oito classes, sendo elas: a classe *Server* que contém o método *main()*, as classes *DataBaseServer* e *ClientData* responsáveis por armazenar informação relativa aos clientes, a classe *Connection*, a classe *LoggedClients* e as classes *ClientHandler* e *SendClient* que implementam a interface *Runnable* cujas instâncias são executadas por *threads*.

O servidor implementado comunica com os clientes através de *sockets TCP*, formando uma conexão segura e viável para a troca de mensagens.

No método *main()* e através da classe *ServerSocket* e do seu construtor, é criado o Servidor. O construtor desta classe recebe como parâmetro o número da porta que ficará à escuta (tem que ser um número superior a 2000), cria o socket e realiza o *bind* e o *listen*, ficando o servidor pronto para aceitar conexões.

Depois de aceite a conexão de um cliente, o método *accept()* devolve um socket para que a comunicação entre o servidor e o cliente seja possível. Uma vez que o servidor pode comunicar com mais do que um cliente ao mesmo tempo, a comunicação entre o servidor e o respetivo cliente aceite é processada numa *thread*. A classe cuja instância é executada na thread designa-se *ClientHandler* e recebe como argumento o socket devolvido pelo método *accept()*, a base de dados onde os clientes estão registados, uma lista com os clientes autenticados e o id do próximo cliente. Depois de iniciada a respetiva thread, a thread principal do programa (que executa o método *main()*) fica bloqueada, passivamente, no método *accept()* à espera de novas conexões.

Um dos requisitos do protejo consiste no registo prévio de um cliente antes de começar a enviar estimativas para o servidor. As classes que permitem o registo de clientes e a manipulação dos seus dados são a classe *DataBaseServer* e a classe *ClientData*.

A classe *ClientData* contém as informações de um cliente, sendo as suas variáveis de instância; a password, o seu id, o número de contactos conhecidos (por defeito *int contacts = 150*), o número de casos reportados pelo cliente e um lock explícito. Os métodos de instância desta classe permitem aceder e/ou modificar o conteúdo das variáveis de instância. O método *void lock()* e *void unlock()* permitem obter o lock e o unlock, respetivamente, do objeto da classe *ClientData*.

A classe *DataBaseServer* tem como variáveis de instância:

---

```
private HashMap<String, ClientData> clientsList; // lista de clientes
private String serverName;
private ReentrantLock dataBaseServer;
```

---

A variável *clientsList* armazena os clientes registados num *HashMap*. A cada nome registado faz a correspondência ao objeto *ClientData* que guarda as informações do respetivo cliente. A variável *dataBaseServer* é um lock explícito.

Nesta classe, tem como métodos mais relevantes: *boolean createClient( ... )*, *boolean checkPassword( ... )* e *double proportionCasesReported()*.

O método *boolean createClient( ... )* permite criar um cliente novo e inseri-lo no *HashMap* que

contem todos os clientes registados. Para verificar se o cliente a ser criado já existe ou inseri-lo no HashMap caso contrário, é necessário fazer o lock do objeto *DataBaseServer* no início do método e, no fim, fazer o unlock.

O método *double proportionCasesReported()* permite obter a estimativa atual e encontra-se a seguir:

---

```
public double proportionCasesReported(){
    this.dataBaseServer.lock();
    // after locking the dataBaseServer we know that no account can be created or
    // deleted, and thus 'locked' is all existing accounts
    Set<String> locked = this.clientsList.keySet();
    // lock all accounts
    for(String name : locked){
        this.clientsList.get(name).lock();
    }
    this.dataBaseServer.unlock();
    // all accounts are locked: we can release the bank lock
    // compute the total cases
    int totalCases = 0;
    int totalContacts = 0;
    for(String name : locked){
        ClientData clientData = this.clientsList.get(name);
        totalCases += clientData.getReportedCases();
        totalContacts += clientData.getContacts();
        clientData.unlock();
    }
    return (double) totalCases / totalContacts;
}
```

---

Inicialmente é feito o lock do objeto para, durante a obtenção das respetivas contas de todos os clientes, o HashMap não seja manipulado. De seguida é feito o lock de cada conta de cada cliente e guardado em *Set(String) locked*. Liberta-se o objeto que contem o HashMap. De seguida para cada cliente é verificado o número de casos reportados e o número de contactos conhecidos e calculada a respectiva proporção e libertando as respetivas contas.

Para o controlo de concorrência nas classes *DataBaseServer* e *ClientData*, decidimos utilizar locks explícitos em detrimento da primitiva *synchronized*, uma vez que os locks explícitos permitem que os respetivos objetos possuem mais concorrência.

Como supramencionado, quando uma conexão é aceite pelo servidor, é executada uma *thread* com uma instância da classe *ClientHandler*. A presente thread fica, passivamente, à espera que o cliente escolha uma opção (1 - registar, 2 - login ou 0 - Sair). Se o cliente escolher a opção 2, através de uma troca de mensagens entre o cliente e o servidor é realizado o registo do cliente. Se o cliente escolher a opção 1, é feito o respetivo login. No caso do login: depois de o servidor receber a opção 1, fica novamente à espera, passivamente, que o cliente introduza o nome e a sua password. Recebida essa informação no servidor, através da de métodos da classe *StringTokenizer*, é separado o nome e a password e, através de métodos atrás mencionados, da classe *DataBaseServer*, é verificado o login. Se o nome do cliente não existir no HashMap, é enviado para o cliente a string "NE" (Não existe). Se existe o nome mas a password é incorreta, é enviado a string "PI". Se a password for a correta é enviada a string "PC" e o cliente é autenticado com sucesso.

Para que o servidor tivesse um comportamento assíncrono, ou seja, quando o servidor recebe uma estimativa de um cliente a envia para todos os clientes ligados, foi desenvolvida a classe *LoggedClients*. Esta classe tem como variável de instância *HashMap<Integer, Connection> logged*. Este HashMap guarda o id de todos os clientes conectados. Para cada id de um cliente ligado, associa a extremidade de escrita (out) do socket do respetivo cliente. Os métodos *void connect(...)* e *void disconnect(...)* permitem adicionar um cliente "online" ou remove-lo.

O método apresentado de seguida, *void sendAll(...)* permite enviar uma estimativa para todos os clientes ligados.

---

```
public synchronized void sendAll(Integer id, String message) {  
    for(Integer idClient: this.logged.keySet()) {  
        Connection connection = this.logged.get(idClient);  
        SendClient sendClient = new SendClient(connection, id, message);  
        Thread thread = new Thread(sendClient);  
        thread.start();  
    }  
}
```

---

Este método recebe o id do cliente que envia a estimativa e a estimativa. Para cada id dos clientes ligados (no HashMap logged), é criada uma thread que executa uma instância da classe *SendClient*. Cada thread recebe a extremidade de escrita no socket (out) de cada cliente ligado e a estimativa e procede ao envio da mesma para o cliente. Esta foi a solução encontrada para o envio assíncrono das estimativas para os clientes ligados. Deste modo, nenhum cliente é afetado se algum cliente tiver uma ligação fraca ou, por qualquer outro motivo, uma latência bastante superior em relação aos outros, uma vez que o envio das estimativas é realizado concorrentemente para todos os clientes.

Para garantir a exclusão mútua na classe *LoggedClients*, foi utilizado a primitiva *synchronized* em todos os métodos.

Um cliente autenticado com sucesso pode enviar várias estimativas. As estimativas enviadas têm de ser um número inteiro não negativo e inferior ao número de contactos conhecidos. Se alguma estimativa não corresponder a estes parâmetros, são enviadas exceções nos respetivos métodos e, posteriormente apanhadas e tratadas, deste modo é assegurada a integridade dos dados armazenados.

Quando um cliente digita "quit", é feito o logout do cliente e este é removido do hashMap logged (contem todos os clientes autenticados), mas o seu registo e os dados manipulados ficam armazenados no programa no HashMap clientsList.

## Capítulo 3

# Programa Cliente

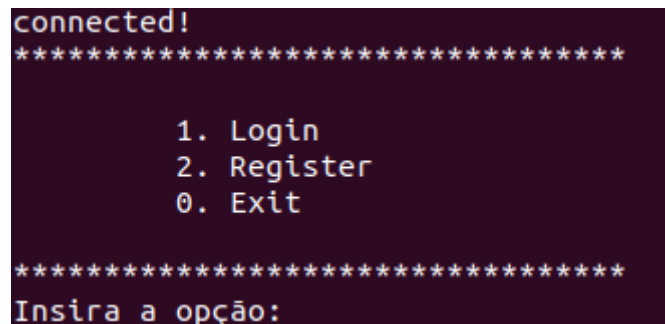
O programa cliente é constituído pelas classes `SocketReader`, `Client` e `UtilClient`.

A classe `Client` contém o método `void main()`.

A classe `SocketReader` através do método `public void run()`, permite que o cliente possa ler mensagens do servidor.

A classe `UtilClient` tem vários métodos, como por exemplo: `public static void menu()`, `public static int verifyNumberOption` e `public static boolean verifyCases`, que têm como objetivo assegurar que os dados introduzidos são corretos.

O programa cliente é iniciado com o par: endereço de IP e o número da porta do servidor. De seguida, e através do construtor da classe `Socket` é criado o socket e feito a conexão com o servidor. Com a comunicação TCP estabelecida, na interface do cliente, é apresentado o seguinte menu:



```
connected!
*****

    1. Login
    2. Register
    0. Exit

*****
Insira a opção:
```

Figura 3.1: Menu - programa cliente)

Para um cliente enviar estimativas, é necessário fazer o registo previamente. Para o registo ou o login, é efetuada uma troca de mensagens com o utilizador, exemplificada no capítulo anterior.

Após a autenticação efetuada com sucesso, e como o servidor é assíncrono, é iniciada uma thread sobre uma variável de instancia da classe `SocketReader`. Esta thread, tem como objetivo receber mensagens enviadas do servidor. Enquanto o servidor não envia mensagens, a presente thread faz uma espera passiva na instrução `String msg = this.in.readLine();`.

---

```
if (connected) {
    Thread reader = new Thread(new SocketReader(in));
    reader.start();
}
```

---

A qualquer momento, o cliente pode enviar uma nova estimativa para o servidor.

De seguida, é apresentado algumas imagens dos programas Cliente e Servidor em execução.

## Capítulo 4

# Análise de resultados

Na figura seguinte apresenta o programa servidor em execução com dois clientes autenticados.

```
Server initialized: 0.0.0.0/0.0.0.0:3001  
  
New connection  
BRUNO autenticado com sucesso!  
  
New connection  
PEDRO autenticado com sucesso!
```

Figura 4.1: Servidor - Autenticações

Na figura 4.2 mostra o registo no programa cliente de uma utilizador cujo nome é *Bruno*. De seguida recebe duas estimativas do servidor. Cada estimativa contém a proporção de infetados atual e a data calculada da respetiva estimativa.

```
Insira dados para o registo (username password): Bruno Santos  
Utilizador registado com sucesso!  
  
2: Average Proportion Atual: 50,00%          2020/05/22 00:56:23  
3: Average Proportion Atual: 66,67%          2020/05/22 00:57:09
```

Figura 4.2: Cliente - 2 estimativas

Na figura 4.3 apresenta algumas mensagens trocadas entres os clientes e o servidor. Por exemplo, os utilizadores pedro e flavia comunicaram que todos os contactos conhecidos estão infetados e o utilizador Bruno fez *logout*.

```
pedro: 150/150  
Average Proportion Atual: 50,00%  
  
New connection  
FLAVIA autenticado com sucesso!  
  
flavia: 150/150  
Average Proportion Atual: 66,67%  
  
BRUNO Disconnected
```

Figura 4.3: Servidor - 2 estimativas