

## Relatório Projeto 2019 / 2020

### Projeto: Ambiente Cliente-Servidor

a21901796 – Miguel Pereira  
a21803823 – Afonso Rodrigues  
a21805522 – Muhammad Iqibal

Lisboa, Portugal  
Maio 2020

**Universidade Lusófona de Humanidades e Tecnologias**  
Licenciatura em Informática de Gestão / Redes de Computadores  
Orientador(es): Prof. Valderi Leithardt e Prof. José Faísca

# Índice

<b>Introdução</b>	<b>3</b>
<b>Arquitetura da aplicação</b>	<b>4</b>
<b>Descrição do protocolo de comunicação de nível aplicativo</b>	<b>5</b>
<b>Descrição Programas</b>	<b>6</b>
<b>Testes à aplicação</b>	<b>10</b>
<b>Conclusão</b>	<b>13</b>
<b>Referências Bibliográficas</b>	<b>14</b>

# Introdução

Este trabalho consiste em desenvolver uma Arquitetura Cliente-Servidor, aplicando a programação em Sockets na linguagem Java. O objetivo deste trabalho é desenvolver uma solução de rede Cliente-Servidor. Para encontrar a devida solução, utilizamos multi-threads, sockets TCP e UDP para enviar mensagens aos clientes, onde apenas os que têm os IP's válidos é que podem comunicar entre si. O software do Cliente, permite assim possibilitar a comunicação entre os clientes através do Server, utilizando um protocolo específico de comunicação.

# Arquitetura da aplicação

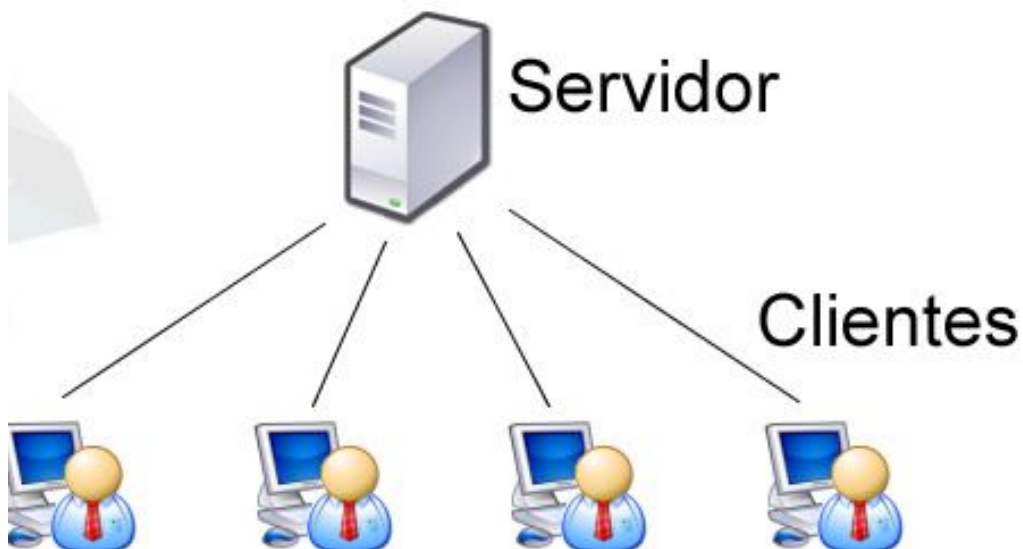
Este trabalho foi desenvolvido com uma arquitetura de Cliente-Servidor. Este tipo de arquitetura consiste num processamento da informação, sendo dividido em módulos ou processos distintos. Um processo é responsável pela manutenção da informação (servidores) e outros responsáveis pela obtenção dos dados (os clientes). Os processos cliente enviam pedidos para o processo servidor, e este por sua vez processa e envia os resultados dos pedidos.

## Características desta arquitetura:

- A interação desta arquitetura é assíncrona, ou seja, o cliente aguarda pela resposta do servidor.
- Descreve a relação de programas numa aplicação.

## Vantagens:

- Funciona com vários clientes diferentes de capacidades diferentes.
- Muitas tecnologias avançadas de cliente-servidor estão disponíveis e foram projetadas para garantir a segurança, facilidade de interface do usuário e facilidade de uso.



# Descrição do protocolo de comunicação de nível aplicativo

Neste projeto foram utilizados 2 tipos de protocolos de comunicação que são essenciais para este tipo de arquitetura: o protocolo TCP e o protocolo UDP.

**Protocolo UDP:** Usando o protocolo UDP, uma máquina emissor envia uma determinada informação e a máquina receptor recebe essa informação, não existindo qualquer confirmação dos pacotes recebidos. Se um pacote se perder não existe normalmente solicitação de reenvio, simplesmente deixa de existir para o destinatário. Este protocolo chega a ser mais rápido que o TCP.

**Protocolo TCP:** O TCP é o protocolo mais usado isto porque fornece garantia na entrega de todos os pacotes entre um PC emissor e um PC receptor. Verifica se o pacote chegou através do processo de checksum.

Em relação ao funcionamento destes protocolos no nosso projeto, utilizamos o UDP no servidor->cliente no porto 9031. Para o cliente->servidor a nossa aplicação utiliza a comunicação através do protocolo TCP, para assegurar que o pedido chega logo ao servidor. O porto que utilizamos neste caso foi o 6500.

Implementámos um série de funcionalidades ao nosso trabalho:

- Lista de utilizadores online
- Lista branca de utilizadores
- Lista negra de utilizadores
- Enviar mensagens, onde o cliente envia uma certa mensagem a outra pessoa, sendo que quem envia terá que especificar o respetivo IP.
- Mostrar o menu ao utilizador, implementamos esta função de modo a que o utilizador possa ver o menu durante a execução do programa.

# Descrição Programas

## Servidor

### [Classe Servidor]

#### addToWhitelist

Adiciona novos cliente (não presentes em nenhuma lista) à whiteList

#### checkOnFile

Verifica se o IP está na lista indicada por argumento.

#### checkIP

Verifica se o IP está em alguma das listas utilizando a função checkOnFile.  
Se não estiver em nenhuma das listas recorre à função addToWhiteList.

#### createLog

Cria o ficheiro log.

#### main

Responsável pela inicialização do servidor (sockets, etc).  
Utilização de thread para correr socket tcp e udp simultaneamente.

### [Classe ClientConnected]

#### ClientConnected

Responsável pela ligação TCP entre o cliente e o Servidor.

#### verifyIfAlreadyConnected

Verifica na lista de clientes online, se o cliente que se está a tentar conectar já está conectado.

Serve para evitar erros e clientes duplicados.

#### endConection

Serve para terminar a conexão entre o cliente e servidor.

### **[Classe ServerTcp]**

#### **sendMenu**

Enviar o menu para o cliente

#### **optionTwo**

Responsável pela opção 2 do menu cliente (Enviar mensagem ao utilizador) onde enviamos por UDP uma mensagem para o servidor e reenvia para o cliente pretendido.

#### **optionThree**

Envio de mensagens para todos os utilizadores. Utiliza UDP.

#### **start**

Recebe comandos dos clientes enquanto o servidor estiver ativo.

#### **run**

Função obrigatória por ser uma Thread. Inicializa a função start.

### **[Classe ServerUdp]**

#### **ServerUdp**

Iniciar servidor no porto 9031.

#### **Send**

Recebe argumentos mensagem, fromIP e toIP. Envia a mensagem recebida do cliente fromIP para o cliente toIP. Gere a troca de mensagens privadas (opção 2 do menu do cliente).

#### **SendToAll**

Envia mensagem recebida de um cliente para todos os clientes.

#### **run**

Recebe mensagens dos clientes enquanto o servidor está ativo.

# Cliente

## [Classe Cliente]

### runTcpClient

Envia comandos por TCP para o servidor. Responsável pela reconexão ao servidor caso esta seja perdida.

### connect

Cria a conexão ao servidor.

### runUdpClient

Cria thread para correr cliente UDP.

### main

Valida IP para ligação ao servidor e invoca funções runTcpClient e runUdpClient.

## [Classe ClientUdp]

### ClientUdp

Construtor. Criar DatagramSocket.

### sendEcho

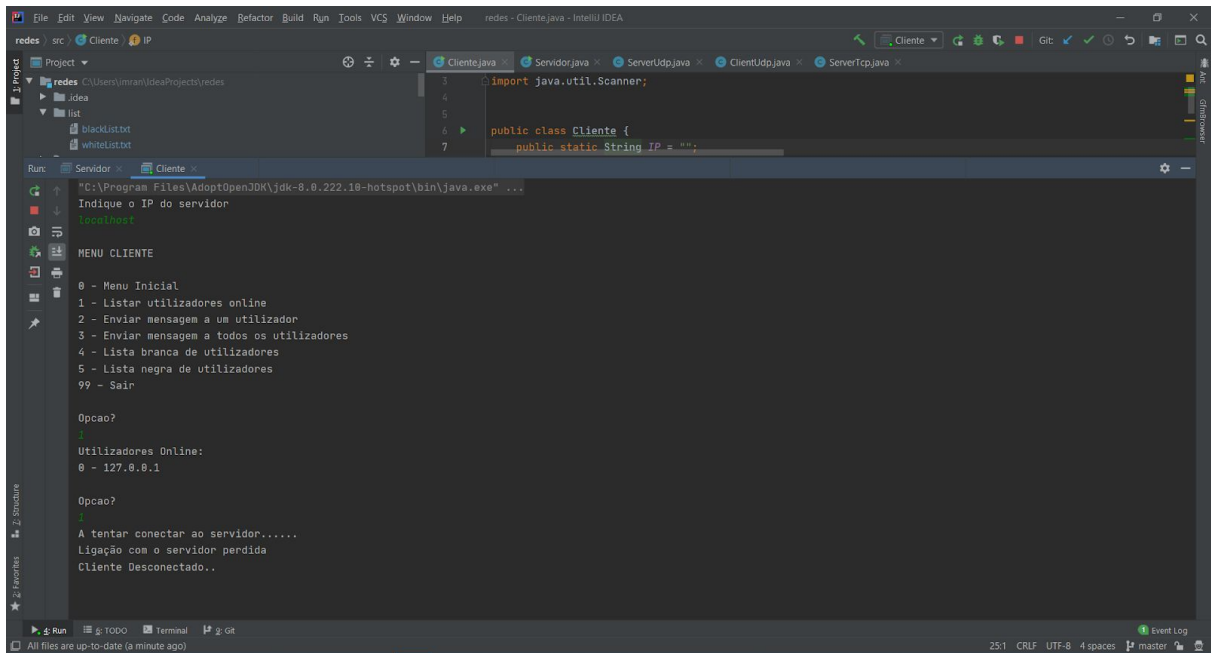
Enviar mensagem recebida por argumento.

### receiveEcho

Receber mensagens enquanto o cliente estiver ativo.



# Testes à aplicação

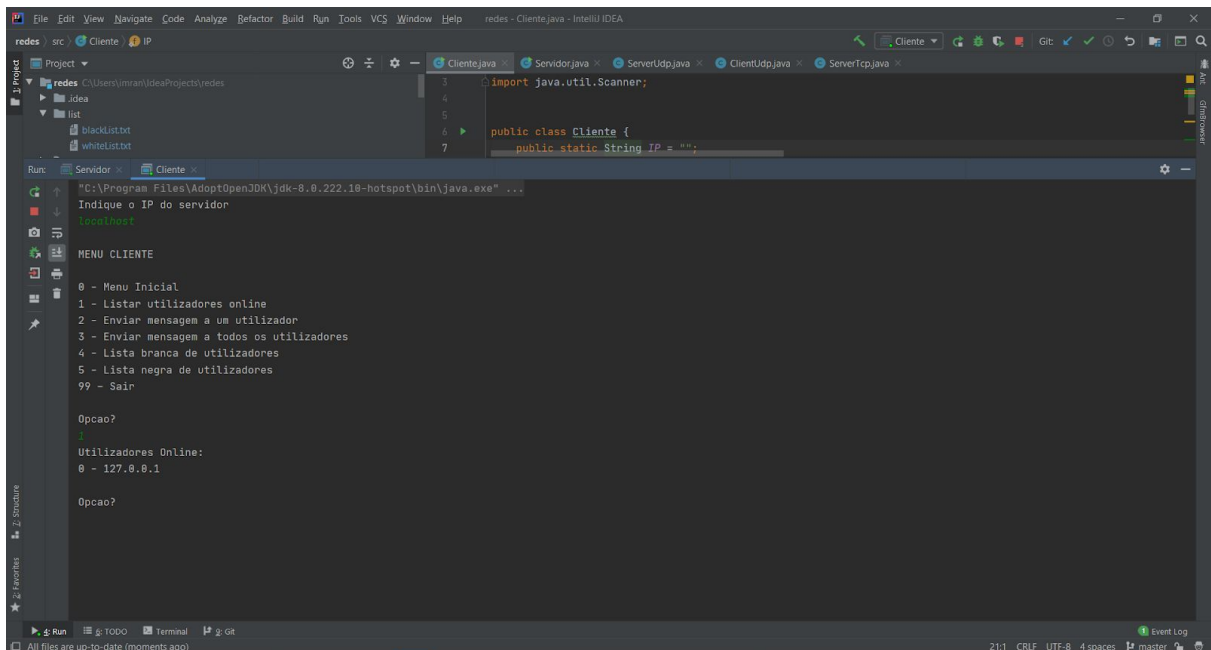


```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help redes - Cliente.java - IntelliJ IDEA
Project: redes
src \ Cliente \ IP
redes
  \ idea
    \ list
      \ blacklist.txt
      \ whitelist.txt
Run: Servidor x Cliente x
"C:\Program Files\AdoptOpenJDK\jdk-8.0.222.18-hotspot\bin\java.exe" ...
Indique o IP do servidor
127.0.0.1
MENU CLIENTE
0 - Menu Inicial
1 - Listar utilizadores online
2 - Enviar mensagem a um utilizador
3 - Enviar mensagem a todos os utilizadores
4 - Lista branca de utilizadores
5 - Lista negra de utilizadores
99 - Sair
Opcao?
1
Utilizadores Online:
0 - 127.0.0.1
Opcao?
A tentar conectar ao servidor.....
Ligação com o servidor perdida
Cliente Desconectado..
```

Neste caso verificamos que o cliente se desconecta quando o servidor se encontra offline.

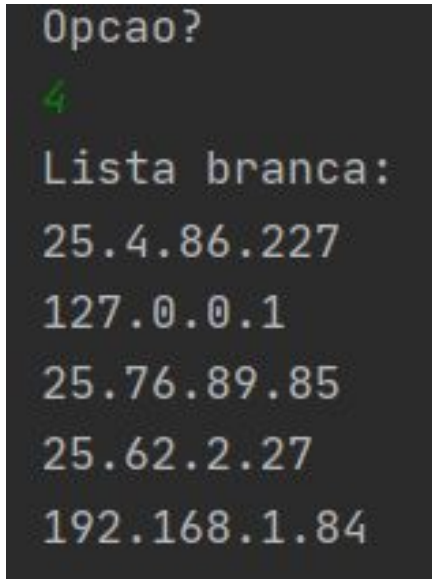
**Teste:** listar os utilizadores online

**Resultado:**



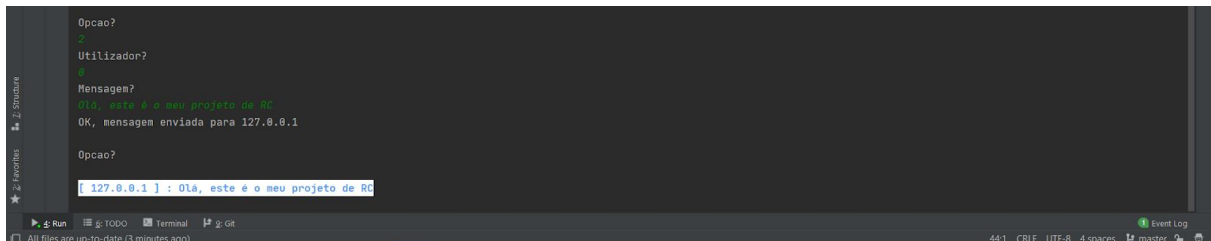
```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help redes - Cliente.java - IntelliJ IDEA
Project: redes
src \ Cliente \ IP
redes
  \ idea
    \ list
      \ blacklist.txt
      \ whitelist.txt
Run: Servidor x Cliente x
"C:\Program Files\AdoptOpenJDK\jdk-8.0.222.18-hotspot\bin\java.exe" ...
Indique o IP do servidor
127.0.0.1
MENU CLIENTE
0 - Menu Inicial
1 - Listar utilizadores online
2 - Enviar mensagem a um utilizador
3 - Enviar mensagem a todos os utilizadores
4 - Lista branca de utilizadores
5 - Lista negra de utilizadores
99 - Sair
Opcao?
1
Utilizadores Online:
0 - 127.0.0.1
Opcao?
```

**Resultado:**



**Teste:** Enviar mensagem ao utilizador

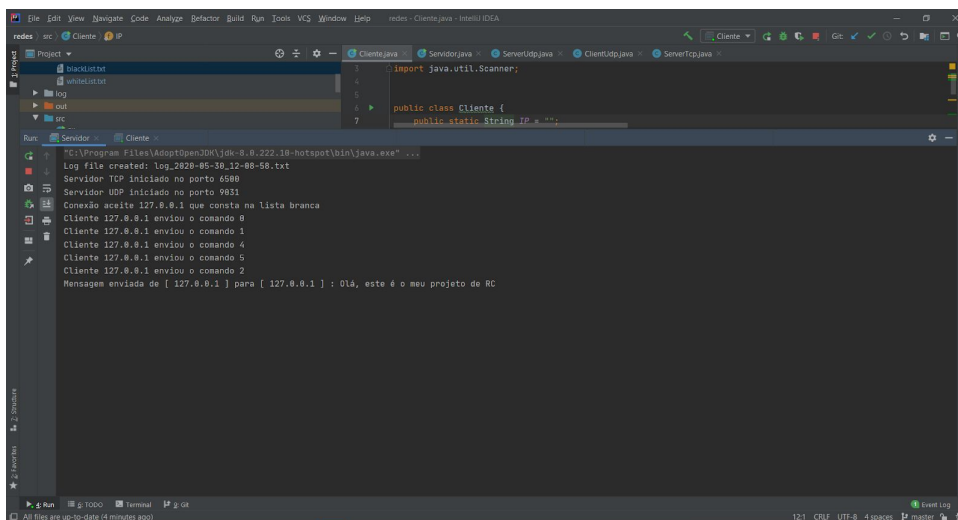
**Resultado:**



Neste caso verificamos que a mensagem foi enviada com sucesso para o utilizador.

**Teste:** Descrição dos comandos efetuados no servidor

**Resultado:**



Com este print, verificamos que o servidor descreve todos os controlos efetuados a partir do cliente.

**Teste:** Enviar mensagem a todos os utilizadores

**Resultado:**

```
3
Mensagem ?
Olá
OK, mensagem enviada a todos os utilizadores
```

```
Cliente 127.0.0.1 enviou o comando 3
Mensagem enviada de [ 127.0.0.1 ] para todos : Olá
|
```

A mensagem foi enviada com sucesso a todos os utilizadores.

**Teste:** Se o utilizador tiver um IP que se encontra na lista negra, o acesso ser-lhe-á negado.

**Resultado:**

```
Indique o IP do servidor
25.76.89.85
Acesso negado
Cliente Desconectado..
```

# Conclusão

Neste projeto, desenvolvemos uma aplicação num ambiente Cliente-Servidor na linguagem de programação Java. A conclusão que chegámos ao realizar este trabalho foram: conseguimos perceber os conceitos e os tópicos da cadeira de Redes de Computadores, nomeadamente a programação de Sockets e Threads, a utilização de protocolos TCP e UDP e entender melhor a arquitetura Cliente-Servidor. Para ver se estávamos a realizar o projeto bem e sem erros, fizemos vários testes que comprovaram que o projeto estava a ser bem efetuado.

# Referências Bibliográficas

1. Java Networking, Tutorialspoint  
[https://www.tutorialspoint.com/java/java\\_networking.htm](https://www.tutorialspoint.com/java/java_networking.htm)
2. Networking - Java Tutorials, Oracle  
<http://java.sun.com/docs/books/tutorial/networking/index.html>
3. Colored Output Console, Diogo Nunes  
<https://www.diigonunes.com/blog/java-colored-output-console/>