



**Curso de Ciência da Computação
UNIVERSIDADE PAULISTA**

**Gabriel Macedo Ramos RA:F281060
Guilherme Augusto Sbizero Correa RA:F235289
Matheus Moreira Dias RA: F2265B8**

DESENVOLVIMENTO DE UMA FERRAMENTA PARA COMUNICAÇÃO EM REDE

**SÃO PAULO – SP
2022**

RESUMO

Redes de computadores é um conjunto de dispositivos (normalmente conhecido como nós) conectados por links de comunicação. Um nó pode ser um computador, uma impressora ou outro dispositivo de envio e/ou recepção de dados, que estejam a outros nós da rede. Uma rede de computadores permite que o usuário compartilhe uma enorme quantidade de informações e enviem mensagens uns aos outros, incluindo textos, imagens, áudios e vídeos. A operacionalização de uma rede de computadores tem como objetivos básicos prover a comunicação confiável entre os vários sistemas de informação, melhorar o fluxo e o acesso às informações, bem como agilizar a tomada de decisões administrativas facilitando a comunicação entre seus usuários. O objetivo deste trabalho é avaliar de forma clara o conceito de redes, sua utilização e sua aplicabilidade, por meio de uma ferramenta de comunicação em rede desenvolvida na linguagem Java para atender as necessidades do Conselho Nacional do Meio Ambiente (CONAMA) que deseja saber quais atividades industriais estão gerando poluição do Rio Tietê desde sua nascente em Salesópolis (SP) até a sua passagem pela região da grande São Paulo. Para tal ela precisa trocar informações das equipes de inspetores treinados e capacitados que se revezarão dentro de cada indústria, controlando os processos e passando informações online para a Secretaria. O trabalho realizado teve caráter qualitativo, fundamentado em obras publicadas por estudiosos especialistas, que já apontaram os conceitos, usos, aplicações, desempenho e falhas de certos tipos de redes.

Palavras-chave: comunicação rede; Berkeley; TCP/IP; sockets.

Sumário

| | |
|--|-----------|
| 1 Introdução..... | 4 |
| 2 Objetivo e motivação do trabalho..... | 6 |
| 3 Fundamentos da Comunicação em rede..... | 7 |
| 3.1 Arpanet | 8 |
| 3.2 O que é TCP/IP | 8 |
| 3.3 O que são e como funcionam os sockets | 8 |
| 3.4 Comunicação de Dados | 11 |
| 3.5 Componentes | 12 |
| 3.6 Tipos de Conexão | 14 |
| 4 Plano de desenvolvimento da Aplicação | 15 |
| 5 Projeto | 20 |
| 6 Relatório com as linhas de código do programa | 23 |
| 7 Apresentação do programa em funcionamento em computador | 49 |
| 8 Bibliografia | 55 |
| 9 Ficha de Atividades Práticas Supervisionadas | 56 |

1. Introdução

A comunicação é uma das maiores necessidades da sociedade humana desde os primórdios de sua existência. Conforme as civilizações se espalhavam, ocupando áreas cada vez mais dispersas geograficamente, a comunicação a longa distância se tornava, cada vez mais, uma necessidade e um desafio. Formas de comunicação através de sinais de fumaça ou pombos-correios foram as maneiras encontradas por nossos ancestrais para tentar aproximar as comunidades distantes.

A invenção do telégrafo por Samuel F. B. Morse, em 1838, inaugurou uma nova época nas comunicações. Nos primeiros telégrafos utilizados no século XIX, mensagens eram codificadas em cadeias de símbolos binários (código Morse) e então transmitidas manualmente por um operador através de um dispositivo gerador de pulsos elétricos. Desde então, a comunicação através de sinais elétricos atravessou uma grande evolução, dando origem à maior parte dos grandes sistemas de comunicação que temos hoje em dia, como o telefone, o rádio e a televisão.

A evolução no tratamento de informações não aconteceu somente na área da comunicação. Equipamentos para processamento e armazenamento de informações também foram alvo de grandes invenções ao longo do nosso desenvolvimento. A introdução de sistemas de computadores na década de 1950 foi, provavelmente, o maior avanço do século nesse sentido.

Inicialmente, os computadores eram máquinas caríssimas que centralizaram em um único ponto o processamento das aplicações de vários usuários e, muitas vezes, de toda uma organização. Com a redução de custos do hardware e a introdução dos microcomputadores no cenário da informática, a estrutura centralizada cedeu lugar a uma estrutura totalmente distribuída, na qual, diversos equipamentos dos mais variados portes, processam informações de formas isoladas, o que acarreta uma série de problemas. Dentre eles destaca-se a duplicação desnecessária de recursos de hardware (impressoras, discos etc.) e de software (programas, arquivos de dados, etc.).

A fusão dos computadores e das comunicações teve uma profunda influência na forma como os sistemas computacionais eram organizados. Está

totalmente ultrapassado o conceito de um “centro de computadores”, como uma sala para onde os usuários levam os programas a serem processados. O velho modelo de um computador atendendo a todas às necessidades computacionais de organização foi substituído pelas chamadas redes de computadores, nas quais os trabalhos são realizados por uma série de computadores interconectados.

Forouzan e Mosharraf (2003, p. 01) descrevem a rede como um meio de transmissão de dados com ou sem fio.

Olifer & Olifer (2008, p. 04) comprovam que as redes de computadores também podem ser consideradas um meio de transmissão de informações a longa distância. Para isso as redes de computadores implementam diversos métodos de codificação de dados e multiplexação amplamente adotados nos sistemas de telecomunicações. Segundo Forouzan (2008, p. 07), redes de computadores é um conjunto de dispositivos (normalmente conhecido como nós) conectados por links de comunicação. Um nó pode ser um computador, uma impressora ou outro dispositivo de envio e/ou recepção de dados, que estejam a outros nós da rede. Scrimger, LaSalle, Parihar e Gupta (2002, p. 03) titubeiam a rede como dois ou mais computadores que compartilham informações. Contudo, as redes podem ser bastante diversificadas. Uma rede de computadores permite que o usuário compartilhe uma enorme quantidade de informações e enviem mensagens uns aos outros, incluindo textos, imagens, áudios e vídeos. Olifer & Olifer (2008, p. 04) provem a influência das redes de computadores sobre outros tipos de redes de telecomunicações, onde resultou em uma convergência de redes, um processo que começou muito antes da internet.

Independentemente do tamanho e do grau de complexidade, o objetivo básico de uma rede de computadores é garantir que todos os recursos de informação sejam compartilhados rapidamente, com segurança e de forma confiável. Para tanto, a rede deve possuir meios de transmissão eficientes, regras básicas (protocolos) e mecanismos capazes de garantir o transporte das informações entre os seus elementos constituintes.

2. Objetivo e motivação do trabalho

O objetivo do trabalho desse trabalho consistente em estudar e desenvolver uma ferramenta para comunicação em rede e toda teoria computacional envolvida. Após pesquisa bibliográfica sobre o assunto em questão o grupo de deverá criar uma aplicação que permita que duas ou mais pessoas possam se comunicar em uma rede, utilizando o protocolo TCP/IP. A utilização para a comunicação de dados implementando a ferramenta que foi utilizada as primitivas dos sockets de Berkeley, os quais serão implementados na linguagem de programação JAVA.

3 Fundamentos da comunicação em rede

A palavra rede tem várias definições. Aplicada aos computadores, rede é uma maneira de conectar computadores para que eles tenham consciência um do outro e possam unir e compartilhar seus recursos. (Nobrega Filho, 2016).

Entre os anos 70 e 80 ocorreu uma fusão dos campos de Ciência da Computação e Comunicação de Dados e isto trouxe vários fatos relevantes. Nas empresas, escolas e em muitos outros tipos de organização, as redes de comunicação de dados, em seus diversos tipos oferecem vários benefícios:

Permitir acesso simultâneo a programas e dados importantes;

Permitir às pessoas compartilhar dispositivos periféricos;

Facilitar o processo de realização de cópias de segurança (backup);

Agilizar as comunicações pessoais com correio eletrônico ou mensagens instantâneas;

Na área comercial, por exemplo, as redes revolucionaram o uso da tecnologia dos computadores. Muitas empresas que costumavam depender de um sistema centralizado em mainframe com uma série de terminais agora usam redes de computadores, em que cada empregado tem um computador em sua mesa. A tecnologia e a perícia dos computadores não estão mais centralizadas no mainframe de uma companhia de e em seu departamento de tecnologia, elas estão distribuídas por toda a organização, entre uma rede de computadores e usuários preparados. (Nobrega Filho, 2016).

A indústria da computação, por assim dizer, teve um processo espetacular em um curto período e as redes de comunicação também fazem parte desse crescimento tendo um papel fundamental nos processos de comunicação para o mundo globalizado. As redes de comunicação, cada vez mais rápidas e eficiente, permitiram a comunicação e o acesso rápido a qualquer parte do globo de forma praticamente instantânea.

3.1 Arpanet

A Arpanet foi a primeira rede computadores, construída em 1969 como um meio robusto para transmitir dados militares sigilosos e para interligar os departamentos de pesquisa de todos os Estados Unidos. Arpanet primeiro executou NCP (Network Control Protocol em português Protocolo de Controle de Rede) e posteriormente a primeira versão do protocolo de internet ou conjuntos de protocolos TCP/IP.

3.2 O que é TCP/IP

De uma forma simples, o TCP/IP é o principal protocolo de envio e recebimento de dados. TCP significa Transmission Control Protocol (Protocolo de Controle Transmissão) e o IP Internet Protocol (Protocolo de Internet).

Protocolo é uma espécie de linguagem utilizada para que dois computadores consigam se comunicar. Por mais que duas máquinas estejam conectadas à mesma rede, se não “falarem” a mesma língua, não há como estabelecer uma comunicação. Então, o TCP/IP é uma espécie de idioma que permite às aplicações conversarem entre si.

3.3 O que são e como funcionam os sockets

Formalmente falando os sockets foram a forma de permitir que dois processos se comuniquem. Esses processos podem ou não estar na mesma máquina.

Diversas aplicações que utilizamos no dia a dia fazem uso de sockets para se comunicar. Nosso navegador web utiliza sockets para requisitar páginas. Quando um sistema se integra com um banco de dados abre um socket. Quando fazemos um SSH em um servidor estamos abrindo e utilizando um socket.

Originalmente a implementação dos sockets foi feita no BSD em 1983. Essa implementação foi portada para o Linux com poucas modificações. Na forma de uma

API, os sockets abstraem a camada de rede para a que uma aplicação possa se comunicar com outra sem ter que se preocupar com detalhes da pilha TCP/IP que gere a rede abaixo dessa aplicação.

A internet utiliza como base para todas as suas comunicações a pilha TCP/IP que possui apenas 4 camadas. Baseado no modelo OSI, o TCP/IP é, atualmente, o padrão de comunicação em redes. A imagem abaixo apresenta uma comparação de modelo OSI com sete camadas em relação as respectivas 4 camadas do modelo TCP/IP:

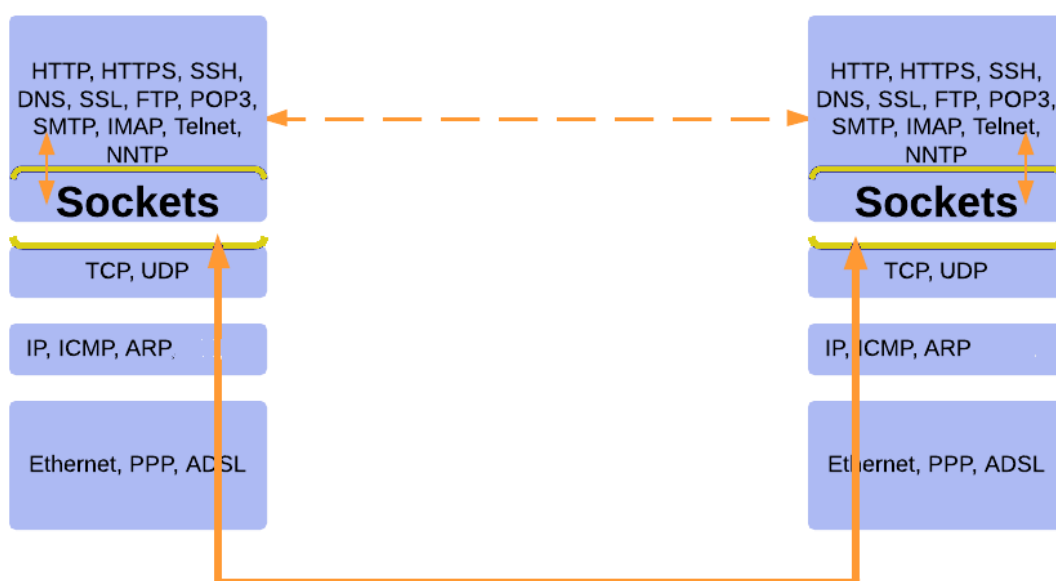


Fonte: Blog Pantuza, 2017

Considerando a internet e o TCP/IP, os sockets estão entre a camada de transporte e a aplicação. Estando nesse ponto de intercessão, eles conseguem fazer

uma interface entre aplicações e rede de maneira bem transparente. Assim, aplicações são implementadas através de uma comunicação lógica. Lógica no sentido de que para esses programas, eles estão se comunicando diretamente um com o outro, mas na prática eles estão passando pela rede para trocar mensagens.

A imagem a seguir mostra essa divisão lógica e física de comunicação dentro da pilha TCP/IP e onde exatamente entre as camadas de transporte e aplicação se encaixa a API de sockets.



Fonte: Blog Pantuza, 2017

Até agora vimos que os sockets foram criados na forma de uma API que possibilita aplicações/processos se comunicarem. Abaixo contém algumas das principais funções ao criar um programa utilizando sockets.

```

/**
 * Principais funções para escrever programas com sockets
 */

getaddrinfo() // Traduz nomes para endereços sockets
socket()      // Cria um socket e retorna o descritor de arquivo
bind()        // Associa o socket a um endereço socket e uma porta
connect()     // Tenta estabelecer uma conexão com um socket
listen()      // Coloca o socket para aguardar conexões
accept()      // Aceita uma nova conexão e cria um socket
send()        // caso conectado, transmite mensagens ao socket
recv()        // recebe as mensagens através do socket
close()       // desaloca o descritor de arquivo
shutdown()    // desabilita a comunicação do socket

```

fonte: DevMedia,2008.

Quando se programa utilizando sockets, uma arquitetura muito comum para esses programas, é utilizar o Cliente/Servidor (client/server). Para essa arquitetura temos que implementar um programa cliente e um programa servidor. Ambos fazem uso da mesma API sockets.

3.4 Comunicação de Dados

A comunicação de dados é uma disciplina da ciência da computação que se trata da comunicação entre computadores e dispositivos de calculadoras analógicas antigas sem utilização de nenhum protocolo do modelo OSI ou da arquitetura TCP/IP diferentes através de um meio de transmissão incomum.

A eficiência de um sistema de comunicação de dados depende fundamentalmente de três características:

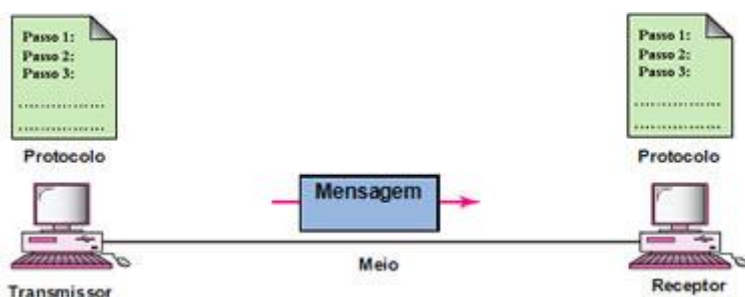
- 1- Entrega: o sistema deve entregar os dados ao destino correto. Os dados devem ser recebidos somente pelo dispositivo ou usuário de destino.
- 2- Confiabilidade: o sistema deve garantir a entrega dos dados. Dados modificados ou corrompidos em uma transmissão são pouco úteis.
- 3- Tempo de atraso: o sistema deve entregar dados em um tempo finito e predeterminado. Dados entregues tardiamente são pouco úteis. Por exemplo, no caso de transmissões multimídia, como vídeo, os atrasos não são

desejáveis, de modo que eles devem ser entregues praticamente no mesmo instante em que foram produzidos, isto é, sem atrasos significativos.

3.5 Componentes

Um sistema básico de comunicação de dados é composto de cinco elementos

- 1. Mensagem:** é a informação a ser transmitida. Pode ser constituída de texto, números, figuras, áudio e vídeo.
- 2. Transmissor:** é o dispositivo que envia a mensagem de dados. Pode ser um computador, uma estação de trabalho, um telefone, uma câmera de vídeo, etc.
- 3. Receptor:** é o dispositivo que recebe a mensagem. Pode ser um computador, uma estação de trabalho, uma câmera de vídeo e assim por diante.
- 4. Meio:** é o caminho físico por onde viaja uma mensagem originada e dirigida ao receptor.
- 5. Protocolo:** é um conjunto de regras que governa a comunicação de dados. Ele representa um acordo entre os dispositivos que se comunicam.



Fonte: Brasil Escola, 2022.

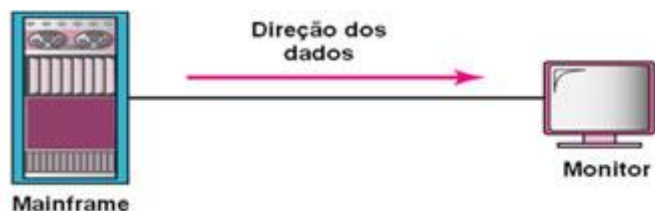
Direção do fluxo de dados: Uma comunicação entre dois dispositivos pode acontecer de três maneiras diferentes: simplex, Half-duplex ou full duplex.

Simplex: no modo simplex, a comunicação é unidirecional, como em uma rua de mão única. Somente um dos dois dispositivos no link é capaz de transmitir; logo o outro só será capaz de receber.



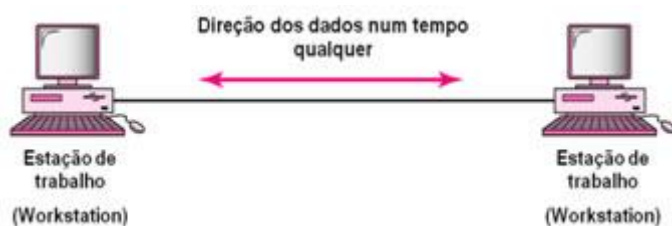
Fonte: Brasil Escola,2022.

Half-duplex: Neste modo, cada estação pode transmitir e receber, mas nunca ao mesmo tempo. Quando um dispositivo está transmitindo o outro está recebendo e vice-versa. Em uma transmissão Half-duplex, toda a capacidade do canal é dada ao dispositivo que estive transmitindo no momento.



Fonte: Brasil Escola,2022.

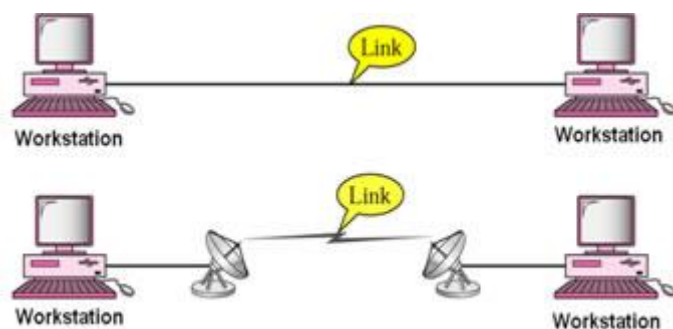
Full-duplex: Neste modo, ambas as estações podem transmitir e receber simultaneamente. Sinais em direções opostas compartilham a capacidade do link ou canal.



Fonte: Brasil Escola,2022.

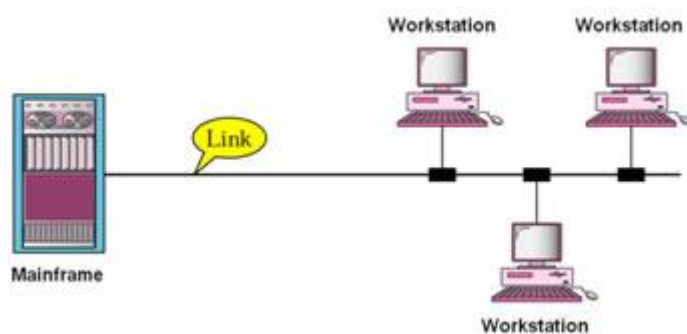
3.6 Tipos de Conexão

Ponto a ponto: proporciona um link dedicado entre dispositivos.



Fonte: Brasil Escola,2022.

Multi-ponto: É aquela na qual mais de dois dispositivos compartilham um único link

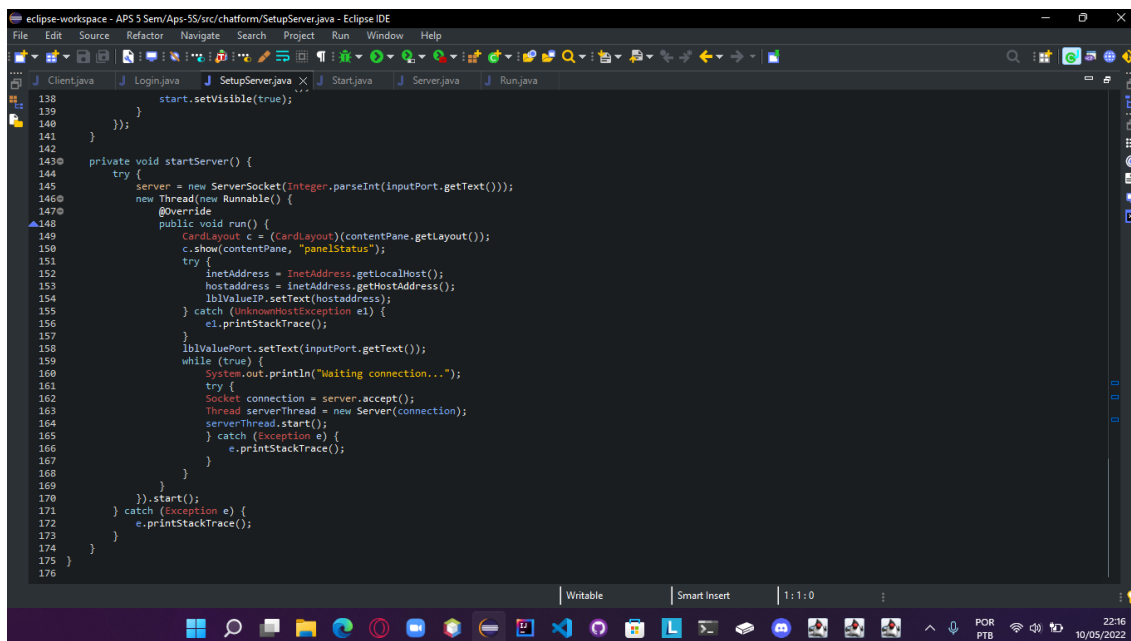


Fonte: Brasil Escola,2022.

4 Plano de desenvolvimento da aplicação

Para criarmos a estrutura básica de nossa aplicação nós utilizamos o Sockets, que normalmente é composta por um servidor e vários clientes. Que basicamente funciona com o cliente solicitando um certo serviço para o servidor, o servidor analisa o que foi pedido e entrega a informação que o cliente pediu.

Podemos ver na imagem abaixo como foi implementada ao nosso programa.



fonte: Guilherme Augusto,2022.

Foi utilizado a biblioteca JAVA.IO que contém uma série de classes para a manipulação de dados de entrada e saída. Com esse pacote nós tivemos então uma liberdade executar as ações que o programa exigia.

Conseguimos visualizar na imagem abaixo como foi implementado em nossa aplicação.

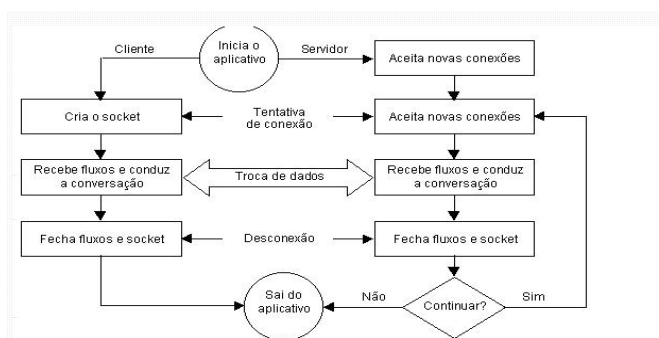
```

1 package connection;
2
3 import java.io.*;
4
5
6
7 public class Server extends Thread {
8     private static ArrayList<BufferedWriter> clients = new ArrayList<BufferedWriter>();
9     private static ArrayList<String> users = new ArrayList<String>();
10    private Socket connection;
11    private BufferedReader bufferReader;
12    private String currentUser = "";
13
14    public Server(Socket connection) {
15        this.connection = connection;
16
17        try {
18            // Reader
19            bufferReader = new BufferedReader(new InputStreamReader(connection.getInputStream()));
20        } catch (IOException e) {
21            e.printStackTrace();
22        }
23    }
24
25    @Override
26    public void run() {
27        try {
28            BufferedWriter bufferWriter = new BufferedWriter(new OutputStreamWriter(this.connection.getOutputStream()));
29            clients.add(bufferWriter);
30            currentUser = this.bufferReader.readLine();
31            users.add(currentUser);
32
33            String msg = "Text" + currentUser + " Conectado";
34
35            while (!("Text@Disconnect " + currentUser).equalsIgnoreCase(msg) && msg != null) {
36                System.out.println(currentUser + " [Server(run)] " + msg);
37                msg = this.bufferReader.readLine();
38            }
39        }
40    }
41 }

```

fonte: Guilherme Augusto,2022.

A Aplicação funcionará da seguinte maneira, basicamente falando, cada software que irá ficar nas máquinas conterà um servidor e um cliente de socket (java.net.socket) e com isso elas irão se comunicar entre si.



fonte: DevMedia,2008.

Todas as máquinas conectadas a uma rede possuem um endereço IP para que dessa maneira possa ser identificado na rede.

A classe InetAddress permite que obtenhamos informações de um computador conectado à rede.

Na Imagem abaixo podemos ver que ele pega o endereço IP e o Localhost. Implementado no código de nossa aplicação.


```

49 contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
50 setContentPane(contentPane);
51 contentPane.setLayout(null);
52 setLocationRelativeTo(null);
53 setTitle("Aplicacao de Conversa (Cliente)");
54
55 JButton btnLogin = new JButton("Login");
56 btnLogin.addActionListener(new ActionListener() {
57     public void actionPerformed(ActionEvent arg0) {
58         login.this.dispose();
59         setFocusableWindowState(false);
60
61         connect();
62     }
63 });
64
65 btnLogin.setFont(new Font("Arial", Font.PLAIN, 15));
66 btnLogin.setBounds(5, 198, 355, 39);
67 contentPane.add(btnLogin);
68
69 JLabel lblServerIp = new JLabel("IP do servidor:");
70 lblServerIp.setFont(new Font("Arial", Font.PLAIN, 15));
71 lblServerIp.setBounds(5, 11, 151, 39);
72 contentPane.add(lblServerIp);
73
74 JTextField inputIP = new JTextField();
75 inputIP.setText("127.0.0.1");
76 inputIP.setFont(new Font("Arial", Font.PLAIN, 15));
77 inputIP.setBounds(138, 15, 222, 39);
78 contentPane.add(inputIP);
79 inputIP.setColumns(10);
80
81 JLabel lblPortaDoServidor = new JLabel("Porta do servidor:");
82 lblPortaDoServidor.setFont(new Font("Arial", Font.PLAIN, 15));
83 lblPortaDoServidor.setBounds(5, 72, 151, 39);
84 contentPane.add(lblPortaDoServidor);
85
86 JTextField inputPort = new JTextField();
87 inputPort.setText("45454");
88

```

fonte: Guilherme Augusto,2022.

O Protocolo TCP não poderia faltar de maneira alguma pois é vital para um programa de troca de informações, e quando precisamos de uma troca confiável de informações (Mensagens) é com esse protocolo que conseguimos isso.

O Protocolo TCP é o responsável por estabelecer uma conexão entre dois pontos interligados. Um exemplo seria uma mensagem sendo passada de uma máquina conectada a uma rede (Host) para outra máquina que intercepta a mensagem indicando o correto recebimento da informação.

Uma informação pode ser enviada em diversos pacotes diferentes, o protocolo TCP cuida para que ao chegar a mensagem remontá-las de forma certa para que a informação seja entregue de forma correta. Caso algum pacote não chegue ao destino o TCP requisita o envio novamente, pois não há sentido em passar uma informação incompleta.

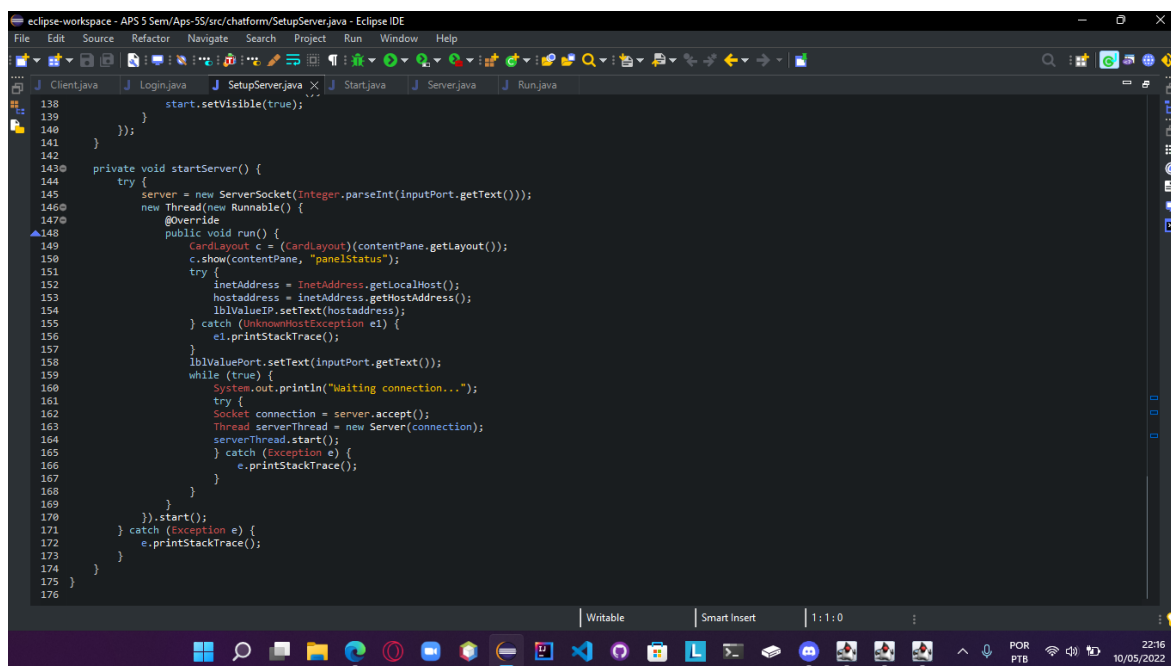
E Somente depois da montagem de todos esses pacotes enviados, a mensagem fica disponível para a nossa aplicação.

Na programação de sockets com o TCP é utilizado streams, o que simplifica muito os processos de leitura e envio de informações através da rede.

As Streams são objetos Java que permitem as aplicações obterem dados de qualquer fonte de entrada, como por exemplo teclado, ou até um fluxo de bytes recebido pela rede. Isso nos dá uma liberdade maior para a manipulação de dados

de redes como se fossem arquivos. É como se literalmente estivéssemos lendo um arquivo e ao enviar estamos gravando dados em um arquivo.

Podemos ver o protocolo TCP sendo implementado aqui em nossa aplicação.



```

138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
    start.setVisible(true);
    });
    });
}
private void startServer() {
    try {
        server = new ServerSocket(Integer.parseInt(inputPort.getText()));
        new Thread(new Runnable() {
            @Override
            public void run() {
                CardLayout c = (CardLayout) contentPane.getLayout();
                c.show(contentPane, "panelStatus");
                try {
                    InetAddress = InetAddress.getLocalHost();
                    hostAddress = InetAddress.getHostAddress();
                    lblValueIP.setText(hostAddress);
                } catch (UnknownHostException e1) {
                    e1.printStackTrace();
                }
                lblValuePort.setText(inputPort.getText());
                while (true) {
                    System.out.println("Waiting connection...");
                    try {
                        Socket connection = server.accept();
                        Thread serverThread = new Server(connection);
                        serverThread.start();
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        }).start();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

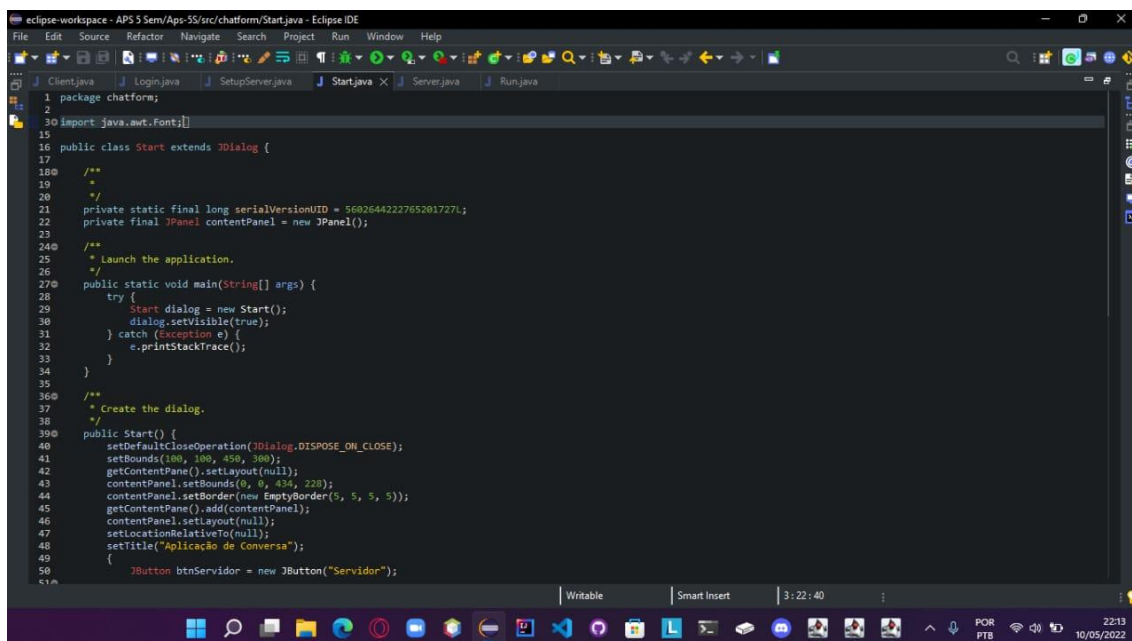
```

fonte: Guilherme Augusto, 2022.

E por fim, mas não menos importante temos a nossa interface da aplicação. Para a interface do programa em modo geral, foi utilizado duas bibliotecas, uma delas foi o Java.fx, que nos proporcionou diversos recursos para que deixássemos a interface do jeito que está, foi escolhida também pela facilidade de manuseio da mesma, nos proporcionando diversas maneiras de trabalhar com ela.

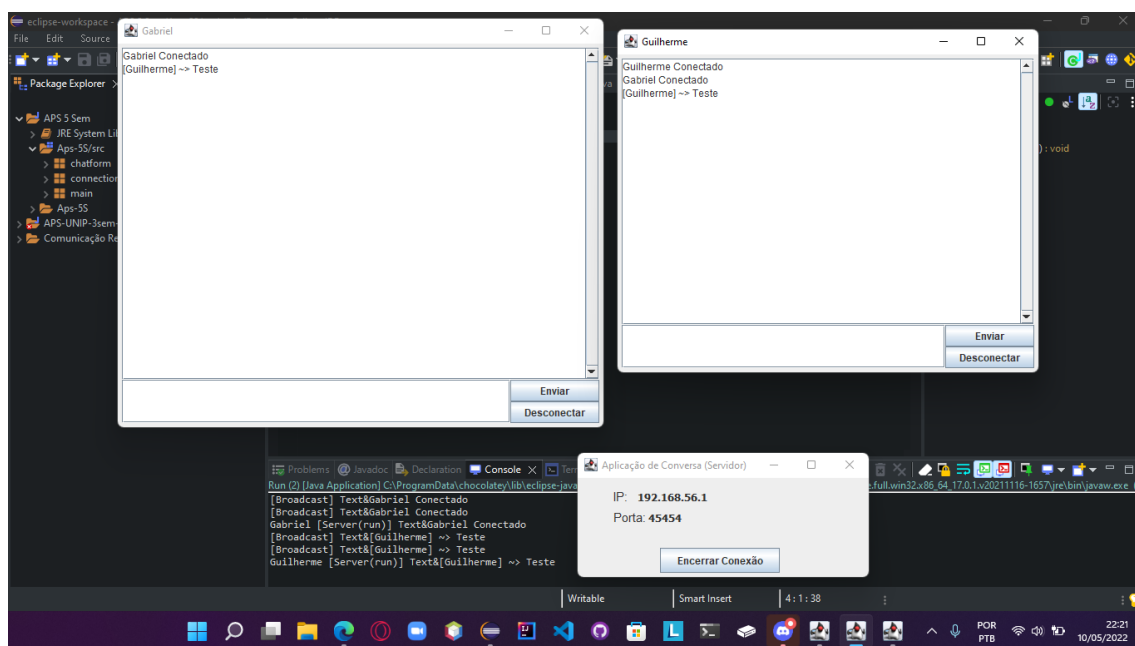
A outra biblioteca utilizada para a interface foi o Java.Swing, nós decidimos utilizá-la também pois diferente das outras API'S a SWING busca renderizar por si própria, todos os componentes, diferente mente de outras API'S que deixam essa tarefa para o sistema operacional, isso nos deu mais liberdade para trabalhar na interface, pois ela é mais completa que a maioria das outras API'S, nos proporcionando diversos componentes.

Na imagem abaixo conseguimos ver como foi implementado e como codificamos a interface, as ferramentas utilizadas nos deram uma liberdade maior para que pudéssemos trabalhar com a interface.



fonte: Guilherme Augusto,2022.

Podemos ver aqui também como ficou a interface de nossa aplicação ao ser finalizada.



fonte: Guilherme Augusto,2022.

5 Projeto

Precisávamos de uma ferramenta que fosse possível realizar a comunicação dos membros do Conselho Nacional do Meio Ambiente (CONAMA), com o objetivo de trocar informações via rede local. Com isto, foi desenvolvido um software que tornou possível a troca de mensagens a partir do localhost, sendo possível interagir com outros usuários.

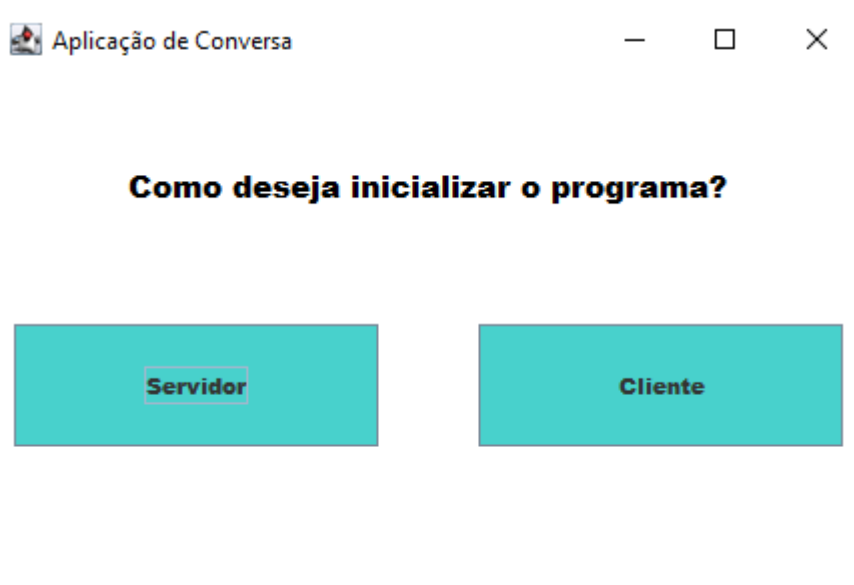
Para estabelecermos um servidor local, foi utilizado sockets que são compostos por um conjunto de primitivas do sistema operacional e foram originalmente desenvolvidos para o BSD Linux.

Foi concebida uma interface gráfica onde o usuário poderá interagir e iniciar o servidor ou se conectar a um servidor que esteja ativo. Ao iniciar um servidor é possível definir qual porta TCP/IP estará disponível para aceitar novas conexões. Após a inicialização do servidor, ele estará disponível para que os usuários façam login.

Para que seja feita a conexão com sucesso, o usuário que deseja se conectar, deverá informar o IP do servidor da rede local e informar a porta que deseja se conectar, do contrário não será possível estabelecer a conexão, inviabilizando a troca de mensagens na ferramenta. O usuário poderá definir um usuário de identificação na ferramenta, que será exibido na troca de mensagens.

Dito isto, foi desenvolvido diversas classes para manter o devido funcionamento da ferramenta, como também, tratar os devidos erros que possam surgir, além de tornar a ferramenta intuitiva a partir da interface gráfica. Foi utilizado a classe JFrame do pacote Swing do JAVA para construção da interface do usuário, possibilitando criar as janelas, botões e as devidas interações na ferramenta.

A estrutura da ferramenta acontece da seguinte maneira, no início da execução, o usuário é apresentado a duas possibilidades, Servidor e Cliente:



fonte: Gabriel Macedo,2022.

Seguindo por Servidor, o programa irá chamar pela classe SetupServer que definirá os parâmetros para inicialização do servidor local. Após a definição desses parâmetros, iniciará o método responsável por receber esses parâmetros e partir dos sockets mencionados anteriormente, iniciar o servidor e deixá-lo disponível para receber novas conexões na porta definida.

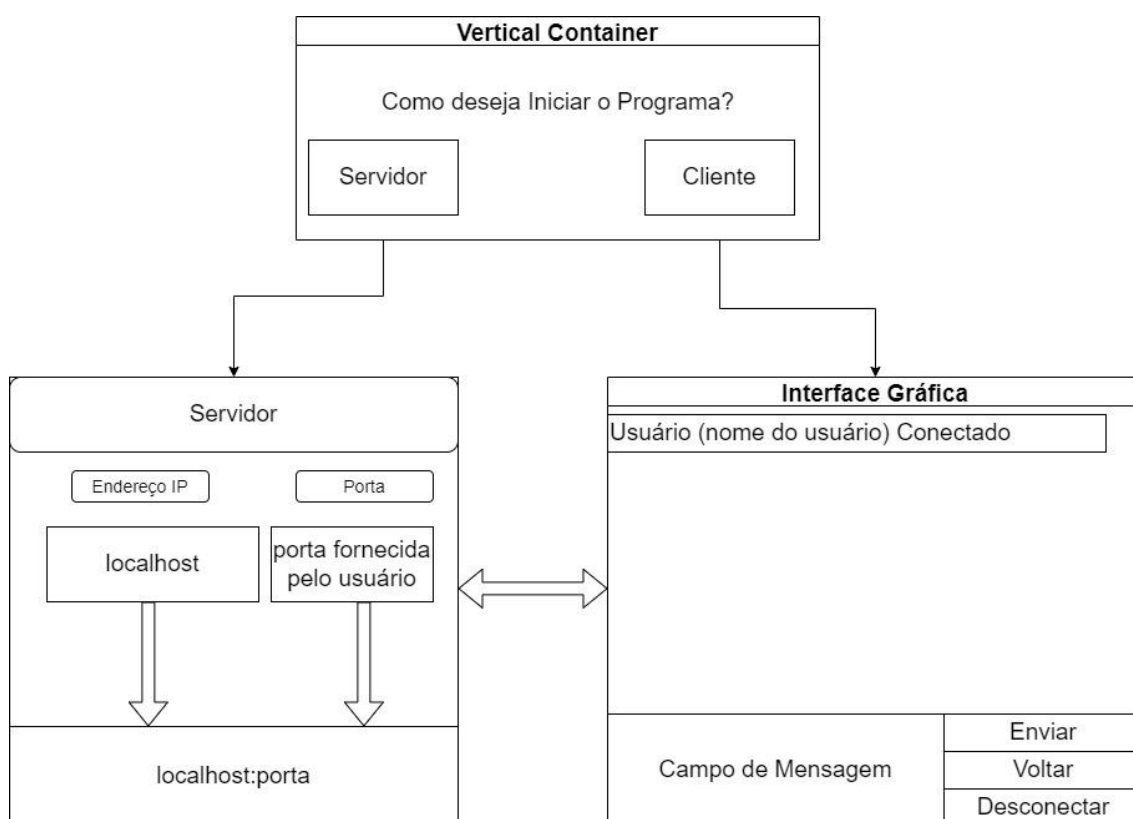
Após esse processo de inicialização do servidor, é possível retornar a tela inicial da ferramenta e seguir para a opção Cliente, o programa ficará responsável por chamar pela classe Login onde será feito o login do Cliente para iniciar a troca de mensagens. Será necessário informar o servidor ativo e a respectiva porta, além de um usuário de identificação na ferramenta. Feito o login, o usuário estará apto a trocar mensagens com os demais usuários, em tempo real. Nesse ambiente, é possível a conexão e troca de mensagens de diversos usuários diferentes ao mesmo tempo. Não há um limite de conexões simultâneas definido, até que o servidor seja devidamente encerrado, continuará a aceitar novas conexões.

Após realizado o login, será iniciado a classe Cliente responsável pela conexão do usuário ao servidor e pela para troca de mensagens entre usuários. Nesse momento, o usuário poderá trocar mensagens com os demais usuários conectados na ferramenta. Caberá ao usuário a encerrar a conexão com o servidor e encerrar o bate-papo. Como mencionado anteriormente, o encerramento da conexão com o servidor por parte dos usuários não afetará na disponibilidade do

servidor, até que ele seja encerrado novos usuários poderão se conectar e utilizar da ferramenta para trocar dados e informações.

Essa conexão entre usuário e servidor, se deve aos sockets do tipo TCP/IP que viabilizam essa comunicação entre processos através de uma rede computadores. Sendo possível garantir a ordem dos pacotes, considerado confiável e sem perdas.

Abaixo temos uma representação desse fluxo:



fonte: Guilherme Augusto,2022.

Nessa representação é possível notar a constante conexão do Cliente com o Servidor, o que possibilita a troca de dados e informações com os demais usuários conectados a mesma rede interna, garantindo a segurança e integridade dos dados e usuários.

6 Relatório com as linhas de código do programa

Run.java

```
package main;

import chatform.Start;

public class Run {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Start.main(null);
    }
}
```

Server.java

```
package connection;

import java.io.*;
import java.net.Socket;
import java.util.ArrayList;

public class Server extends Thread {
    private static ArrayList<BufferedWriter> clients = new
ArrayList<BufferedWriter>();
    private static ArrayList<String> users = new ArrayList<String>();
    private Socket connection;
    private BufferedReader bufferReader;
    private String currentUser = "";

    public Server(Socket connection) {
        this.connection = connection;

        try {
```

```

        // Reader
        BufferedReader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
    } catch (IOException e) {
        e.printStackTrace();
    }
}

@Override
public void run() {
    try {

        BufferedWriter bufferWriter = new BufferedWriter(new
OutputStreamWriter(this.connection.getOutputStream()));

        clients.add(bufferWriter);
        currentUser = this.bufferedReader.readLine();
        users.add(currentUser);

        String msg = "Text&" + currentUser + " Conectado";

        while (!("Text&Disconnect " + currentUser).equalsIgnoreCase(msg) &&
msg != null) {
            broadCast(msg);
            System.out.println(currentUser + " [Server(run)] " + msg);
            msg = this.bufferedReader.readLine();
        }

        removeUser(currentUser);

        broadCast("Text&Usuário " + currentUser + " Desconectado");
    } catch (Exception e) {
        e.printStackTrace();
        removeUser(currentUser);
    }
}

```



```

    }
}

private void broadCast(String msg) {
    for (BufferedWriter bufferClient : clients) {
        try {
            bufferClient.write(msg + "\r\n");
            System.out.println("[Broadcast] " + msg);
            bufferClient.flush();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
}

```

```

private void removeUser(String user) {
    int index = users.indexOf(user);
    clients.remove(index);
    users.remove(index);
}
}

```

Start.java

```

package chatform;

import java.awt.Font;

import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import java.awt.event.ActionListener;

```

```

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import java.awt.event.ActionEvent;
import java.awt.Color;

public class Start extends JFrame {

    /*
     * Variáveis da classe Start
     */

    private static final long serialVersionUID = 5602644222765201727L;
    private final JPanel contentPanel = new JPanel();

    /*
     * Executa a aplicação
     */
    public static void main(String[] args) {
        try {
            Start dialog = new Start();
            dialog.setVisible(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /*
     * Definido as propriedades dos componentes utilizado e ação dos botões
     */
    public Start() {
        getContentPane().setBackground(Color.BLACK);
        setForeground(Color.BLACK);
        setBackground(Color.WHITE);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

```

```

setBounds(100, 100, 450, 300);
getContentPane().setLayout(null);
contentPanel.setBackground(Color.WHITE);
contentPanel.setBounds(0, 0, 434, 261);
contentPanel.setBorder(new EmptyBorder(5, 5, 5, 5));
getContentPane().add(contentPanel);
contentPanel.setLayout(null);
setLocationRelativeTo(null);
setTitle("Aplicação de Conversa");
{
    JButton btnServidor = new JButton("Servidor");

    btnServidor.setBackground(new Color(72, 209, 204));
    btnServidor.setFont(new Font("Arial Black", Font.PLAIN, 11));
    btnServidor.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
            SetupServer server = new SetupServer();
            server.setVisible(true);
            Start.this.dispose();
        }
    });
    btnServidor.setBounds(10, 126, 182, 61);
    contentPanel.add(btnServidor);
}

{
    JButton btnCliente = new JButton("Cliente");

    btnCliente.setBackground(new Color(72, 209, 204));
    btnCliente.setFont(new Font("Arial Black", Font.PLAIN, 11));
    btnCliente.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            SetupServer s1 = new SetupServer();
            Login login = new Login();

```

```

        login.setVisible(true);

    }
});
btnCliente.setBounds(242, 126, 182, 61);
contentPanel.add(btnCliente);
}
{
    JLabel lblIniciar = new JLabel("Como desea inicializar o
programa?");
    lblIniciar.setForeground(Color.BLACK);
    lblIniciar.setBounds(67, 40, 305, 34);
    contentPanel.add(lblIniciar);
    lblIniciar.setFont(new Font("Arial Black", Font.PLAIN, 15));
}
}
}

```

SetupServer.java

```

package chatform;

import connection.Server;

import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import java.awt.Font;
import javax.swing.JTextField;
import javax.swing.JButton;

```

```

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.CardLayout;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.UnknownHostException;
import java.awt.Color;

public class SetupServer extends JFrame {
    /**
     * Server variáveis
     */
    private static ServerSocket server;

    /**
     * Form variáveis
     */
    private static final long serialVersionUID = 4998717362394143017L;
    private JPanel contentPane;
    private JTextField inputPort;
    private JButton btnOk;
    private JButton btnBack;
    private JPanel panelConfig;
    private JPanel panelStatus;
    private JLabel lblIp;
    private JLabel lblPort;
    private JLabel lblValueIP;
    private JLabel lblValuePort;
    private JButton btnStopConnection;
    private InetAddress inetAddress;
    private String hostaddress;
    private JButton btnBack_1;
    private String localhost = "";

```

```

/**
 * Executa a aplicação
 */
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                SetupServer frame = new SetupServer();
                frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}

```

tela

```

/**
 * Aqui é definido as propriedades dos componentes utilizados nessa
 * além de definir a ação dos botões
 */
public SetupServer() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 366, 158);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
}

```

```

contentPane.setLayout(new CardLayout(0, 0));
this.setLocationRelativeTo(null);
this.setTitle("Aplicação de Conversa (Servidor)");

panelConfig = new JPanel();
contentPane.add(panelConfig, "panelConfig");
panelConfig.setLayout(null);

JLabel lblNumeroDaPorta = new JLabel("Número da porta:");
lblNumeroDaPorta.setBounds(10, 35, 113, 18);
panelConfig.add(lblNumeroDaPorta);
lblNumeroDaPorta.setFont(new Font("Arial Black", Font.PLAIN,
11));

inputPort = new JTextField();
inputPort.setBounds(135, 35, 86, 20);
panelConfig.add(inputPort);
inputPort.setText("");
inputPort.setColumns(10);

btnOk = new JButton("Ok");
btnOk.setBackground(new Color(72, 209, 204));
btnOk.setFont(new Font("Arial Black", Font.PLAIN, 11));
btnOk.setBounds(90, 86, 73, 23);
panelConfig.add(btnOk);
btnOk.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        startServer();
    }
});

btnBack = new JButton("Voltar");
btnBack.setFont(new Font("Arial Black", Font.PLAIN, 11));

```

```

btnBack.setBackground(new Color(72, 209, 204));
btnBack.setBounds(177, 86, 73, 23);
panelConfig.add(btnBack);

```

```

panelStatus = new JPanel();
contentPane.add(panelStatus, "panelStatus");
panelStatus.setLayout(null);

```

```

lblIp = new JLabel("IP:");
lblIp.setFont(new Font("Arial Black", Font.PLAIN, 15));
lblIp.setBounds(38, 11, 46, 14);
panelStatus.add(lblIp);

```

```

lblPort = new JLabel("Porta:");
lblPort.setFont(new Font("Arial Black", Font.PLAIN, 15));
lblPort.setBounds(38, 36, 57, 14);
panelStatus.add(lblPort);

```

```

lblValueIP = new JLabel();
lblValueIP.setForeground(new Color(0, 100, 0));
lblValueIP.setFont(new Font("Arial Black", Font.PLAIN, 13));
lblValueIP.setBounds(67, 11, 171, 14);
panelStatus.add(lblValueIP);

```

```

lblValuePort = new JLabel();
lblValuePort.setForeground(new Color(0, 100, 0));
lblValuePort.setFont(new Font("Arial Black", Font.PLAIN, 13));
lblValuePort.setBounds(97, 36, 121, 14);
panelStatus.add(lblValuePort);

```

```

btnStopConnection = new JButton("Encerrar Conexão");
btnStopConnection.setBackground(new Color(72, 209, 204));
btnStopConnection.setFont(new Font("Arial Black", Font.PLAIN,

```

```

11));

```



```

btnStopConnection.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});
btnStopConnection.setBounds(38, 79, 144, 30);
panelStatus.add(btnStopConnection);

btnBack_1 = new JButton("Voltar");
btnBack_1.setFont(new Font("Arial Black", Font.PLAIN, 11));
btnBack_1.setBackground(new Color(72, 209, 204));
btnBack_1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        SetupServer.this.dispose();

        Start start = new Start();
        start.setVisible(true);
    }
});
btnBack_1.setBounds(212, 79, 75, 30);
panelStatus.add(btnBack_1);
btnBack.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        SetupServer.this.dispose();
        Start start = new Start();
        start.setVisible(true);
    }
});
}

```

/*

* Aqui é iniciado o servidor, o inetadress pega as informações do
localhost

* e inicia o servidor, deixando disponível para novas conexões

```

        */
        private void startServer() {
            try {
                server = new
ServerSocket(Integer.parseInt(inputPort.getText()));
                new Thread(new Runnable() {
                    @Override
                    public void run() {
                        CardLayout c
(CardLayout)(contentPane.getLayout());
                        c.show(contentPane, "panelStatus");
                        try {
                            InetAddress
InetAddress.getLocalHost();
                            hostaddress
inetAddress.getHostAddress();
                            localhost = hostaddress;
                            lblValueIP.setText(hostaddress);
                        } catch (UnknownHostException e1) {
                            e1.printStackTrace();
                        }
                        lblValuePort.setText(inputPort.getText());
                        while (true) {
                            System.out.println("Waiting connection...");
                            try {
                                Socket connection = server.accept();
                                Thread serverThread = new Server(connection);
                                serverThread.start();
                            } catch (Exception e) {
                                e.printStackTrace();
                            }
                        }
                    }
                }).start();
            }
        }
    }
}

```

```

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

Login.java

```

package chatform;

import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JButton;
import javax.swing.JLabel;
import java.awt.Font;
import javax.swing.JTextField;
import java.awt.event.ActionListener;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.awt.event.ActionEvent;
import java.awt.Color;

public class Login extends JFrame {

    /**
     *
     */
    private static final long serialVersionUID = 948747724372712259L;
    private JPanel contentPane;
    private JTextField inputIP;
    private JTextField inputPort;

```

```

private JTextField inputUser;
SetupServer s1 = new SetupServer();
private static InetAddress inetAddress;
    private static String hostaddress;

/**
 * Executa a aplicação
 */
public static void main(String[] args) {

    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                Login frame = new Login();
                frame.setVisible(true);

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}

/**
 * Cria o frame
 */
public Login() {

    setResizable(false);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 374, 277);
    contentPane = new JPanel();

```

```

contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
setContentPane(contentPane);
contentPane.setLayout(null);
setLocationRelativeTo(null);
setTitle("Aplicacao de Conversa (Cliente)");
try {
    inetAddress = InetAddress.getLocalHost();
    hostaddress = inetAddress.getHostAddress();
} catch (UnknownHostException e1) {
    e1.printStackTrace();
}

```

```

JButton btnLogin = new JButton("Login");
btnLogin.setBackground(new Color(72, 209, 204));
btnLogin.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        Login.this.dispose();
        setFocusableWindowState(false);

        connect();
    }
});

```

```

btnLogin.setFont(new Font("Arial Black", Font.PLAIN, 12));
btnLogin.setBounds(5, 198, 83, 39);
contentPane.add(btnLogin);

```

```

JButton btnBack = new JButton("Voltar");
btnBack.setFont(new Font("Arial Black", Font.PLAIN, 12));
btnBack.setBackground(new Color(72, 209, 204));
btnBack.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Login.this.dispose();
        Start start = new Start();
    }
});

```

```
start.setVisible(true);

    }
});
btnBack.setBounds(98, 198, 83, 39);
contentPane.add(btnBack);

JLabel lblServerIp = new JLabel("IP do servidor:");
lblServerIp.setFont(new Font("Arial Black", Font.PLAIN, 12));
lblServerIp.setBounds(5, 11, 151, 39);
contentPane.add(lblServerIp);

inputIP = new JTextField();
inputIP.setText(hostaddress);
inputIP.setFont(new Font("Arial Black", Font.PLAIN, 15));
inputIP.setBounds(138, 15, 222, 30);
contentPane.add(inputIP);
inputIP.setColumns(10);

JLabel lblPortaDoServidor = new JLabel("Porta do servidor:");
lblPortaDoServidor.setFont(new Font("Arial Black", Font.PLAIN, 12));
lblPortaDoServidor.setBounds(5, 72, 151, 39);
contentPane.add(lblPortaDoServidor);

inputPort = new JTextField();
inputPort.setText("");
inputPort.setFont(new Font("Arial Black", Font.PLAIN, 15));
inputPort.setColumns(10);
inputPort.setBounds(138, 76, 222, 30);
contentPane.add(inputPort);

JLabel lblUsuario = new JLabel("Nome de usuário:");
lblUsuario.setFont(new Font("Arial Black", Font.PLAIN, 12));
lblUsuario.setBounds(5, 134, 151, 39);
contentPane.add(lblUsuario);
```

```

        inputUser = new JTextField();
        inputUser.setText("unip1");
        inputUser.setFont(new Font("Arial Black", Font.PLAIN, 15));
        inputUser.setColumns(10);
        inputUser.setBounds(138, 138, 222, 30);
        contentPane.add(inputUser);
    }

    private void connect() {
        try {
            Thread thread = new Thread(new Client(new String[] {getUser(),
getIP(), getPort()}));
            thread.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private String getIP() {
        return inputIP.getText();
    }

    private String getPort() {
        return inputPort.getText();
    }

    private String getUser() {
        return inputUser.getText();
    }
}

```

Client.java

```
package chatform;
```

```
import java.awt.BorderLayout;
import java.awt.event.*;

import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JTextField;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JButton;
import javax.swing.ScrollPaneConstants;
import javax.swing.text.DefaultCaret;

import java.io.*;

import java.net.ConnectException;
import java.net.Socket;
import java.awt.Font;
import java.awt.Color;

public class Client extends JFrame implements Runnable {

    /**
     * Variáveis do Socket
     */
    private Socket socket;
    private BufferedWriter bufferWriter;

    /**
     * Variáveis da classe Client
     */
    private String user;
    private String serverIP;
```



```

private int serverPort;
Login login = new Login();

/**
 * Variáveis do Form
 */
private static final long serialVersionUID = 5391582161763137020L;
private JTextField inputText;
private JTextArea output;

/**
 * Execução da aplicação
 */
public void run() {
    try {
        this.setTitle(this.user);
        this.establishConnection();
        this.listenConnection();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

/**
 * Cria o frame, aqui é verificado a conexão do socket,
 * como também, se necessário encerra o programa ou retorna para tela
anterior
 */
public Client(String[] args) {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.addWindowListener(new WindowAdapter() {
        @Override
        public void windowClosing(WindowEvent e) {

```

```

        if(socket == null)
            dispose();
        else if(!socket.isClosed()) {
            try {
                disconnect();
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }

        Start start = new Start();
        start.setVisible(true);
        dispose();
    }
}

);

/* Aqui é definido as propriedades da interface e seus componentes
 *
 */

setBounds(100, 100, 600, 500);
JPanel contentPane = new JPanel();
contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
setContentPane(contentPane);
contentPane.setLayout(new BorderLayout(0, 0));
setLocationRelativeTo(null);
setTitle("Aplicação de Conversa (Cliente)");

output = new JTextArea();
output.setBackground(Color.WHITE);
output.setFont(new Font("Arial Black", Font.PLAIN, 12));
output.setEditable(false);
output.setLineWrap(true);

```

```

JScrollPane messages = new JScrollPane(output);

messages.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR
_ALWAYS);

DefaultCaret caretOutput = (DefaultCaret) output.getCaret();
caretOutput.setUpdatePolicy(DefaultCaret.ALWAYS_UPDATE);

contentPane.add(messages, BorderLayout.CENTER);

JPanel interactive = new JPanel();
contentPane.add(interactive, BorderLayout.SOUTH);
interactive.setLayout(new BorderLayout(0, 0));

inputText = new JTextField();
inputText.setFont(new Font("Arial Black", Font.PLAIN, 12));
interactive.add(inputText, BorderLayout.CENTER);
inputText.setColumns(10);

JPanel buttons = new JPanel();
interactive.add(buttons, BorderLayout.EAST);
buttons.setLayout(new BorderLayout(0, 0));

JButton btnSend = new JButton("Enviar");
btnSend.setBackground(new Color(72, 209, 204));
btnSend.setFont(new Font("Arial Black", Font.PLAIN, 11));
btnSend.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String text = inputText.getText();
        if(!text.equals("")) {
            sendMessage(concatMsg(text));
            inputText.setText("");
        }
    }
}

```

```

});

buttons.add(btnSend, BorderLayout.NORTH);

inputText.addKeyListener(new KeyListener() {
    @Override
    public void keyTyped(KeyEvent e) {
        // TODO Auto-generated method stub

    }

    @Override
    public void keyReleased(KeyEvent e) {
        // TODO Auto-generated method stub

    }

    @Override
    public void keyPressed(KeyEvent e) {
        if (e.getKeyCode() == KeyEvent.VK_ENTER) {
            String text = inputText.getText();
            if(!text.equals("")) {
                sendMessage(concatMsg(text));
                inputText.setText("");
            }
        }
    }
});

JButton btnExit = new JButton("Desconectar");
btnExit.setBackground(new Color(72, 209, 204));
btnExit.setFont(new Font("Arial Black", Font.PLAIN, 11));
btnExit.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {

```

```

        disconnect();Client.this.dispose();
        login.setVisible(true);
    } catch (Exception e1) {
        e1.printStackTrace();
    }
}

});
buttons.add(btnExit, BorderLayout.SOUTH);

JButton btnBack = new JButton("Voltar");
btnBack.setBackground(new Color(72, 209, 204));
btnBack.setFont(new Font("Arial Black", Font.PLAIN, 11));
btnBack.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Client.this.dispose();
        login.setVisible(true);
    }
});
buttons.add(btnBack, BorderLayout.CENTER);

setLocationRelativeTo(null);
setVisible(true);
setClientInfo(args[0], args[1], args[2]);
}

private void setClientInfo(String user, String serverIP, String serverPort) {
    this.user = user;
    this.serverIP = serverIP;
    this.serverPort = Integer.parseInt(serverPort);
}

private void sendMessage(String msg) {
    try {
        bufferWriter.write(msg);
    }
}

```

```

        bufferWriter.flush();
    } catch (Exception e) {
        writeOutput("Desconectado");
    }
}

private String concatMsg(String msg) {
    return "Text&" + "[" + user + "]" + " ~> " + msg + "\r\n";
}

private void writeOutput(String phrase) {
    output.append(phrase + "\r\n");
}

```

/*Aqui verifica o IP e a Porta informados no login e estabelece a conexão com o servidor,

* caso contrário, apresenta a mensagem de erro

*

*/

```

private void establishConnection() {
    try {
        socket = new Socket(this.serverIP, this.serverPort);
        bufferWriter = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
        bufferWriter.write(user + "\r\n");
        bufferWriter.flush();
    } catch (ConnectException e) {
        JOptionPane.showMessageDialog(null, "Nao foi possível criar a
conexão, servidor indisponível na porta e IP indicados.");
        Client.this.dispose();
        login.setVisible(true);
    } catch (Exception e) {

```

```

        e.printStackTrace();
        disconnect();
    }
}

/*
 * Aqui é verificado a mensagem, se ocorrer algum problema na
mensagem, apresenta erro e desconeta o usuário
 */
private void listenConnection() {
    try {
        BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        String msg = "";
        String command;
        String textMsg;

        do {
            if (bufferedReader.ready()) {
                msg = bufferedReader.readLine();
                command = msg.split("&", 2)[0];
                textMsg = msg.split("&", 2)[1];

                if(command.equals("Text")) {
                    writeOutput(textMsg);
                } else {
                    writeOutput("Algo está errado na mensagem recebida do
servidor");
                }
            }
        } while (!("Disconnect " + user).equalsIgnoreCase(msg));

    } catch (Exception e) {

```

```
        System.out.println("Impossível escutar servidor. O mesmo  
possivelmente está indisponível.");
```

```
    }
```

```
}
```

```
private void disconnect() {
```

```
    writeOutput("Desconectado");
```

```
    sendMessage("Disconnect " + this.user);
```

```
    try {
```

```
        bufferWriter.close();
```

```
        socket.close();
```

```
    } catch (Exception e) {
```

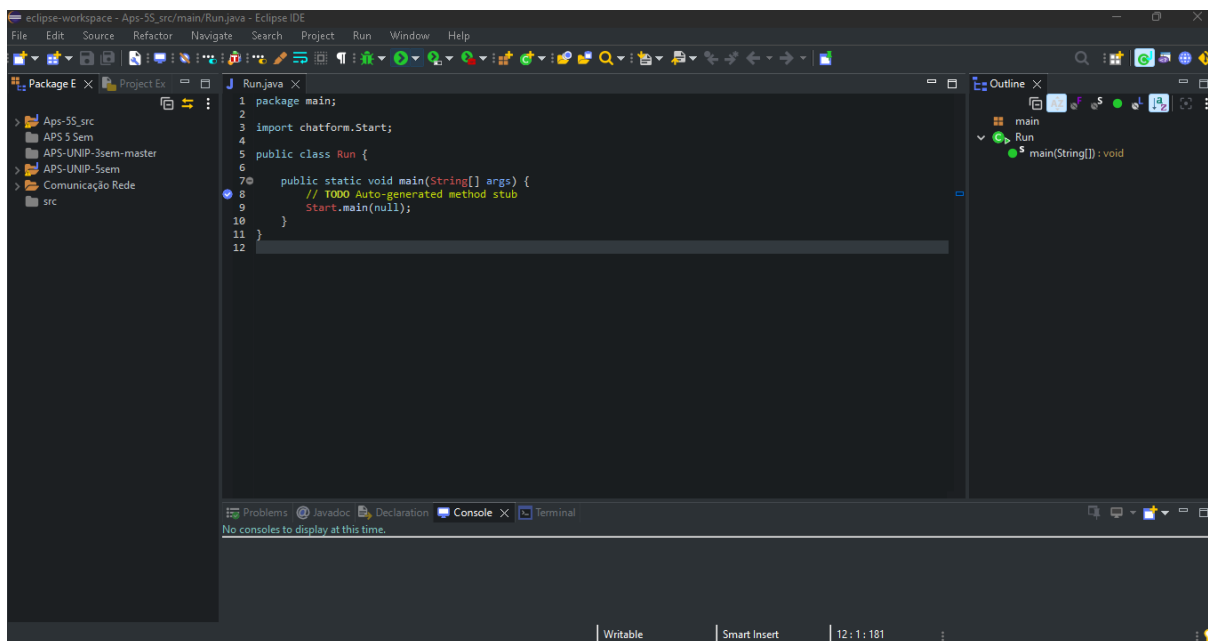
```
        System.out.println("Nao é possível fechar conexão.");
```

```
    }
```

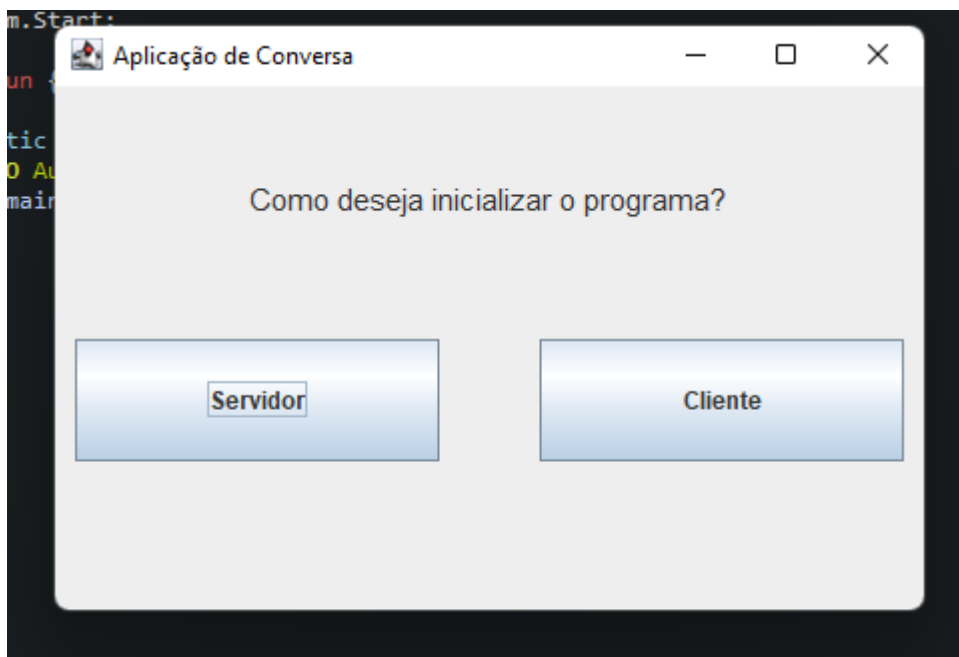
```
}
```

```
}
```

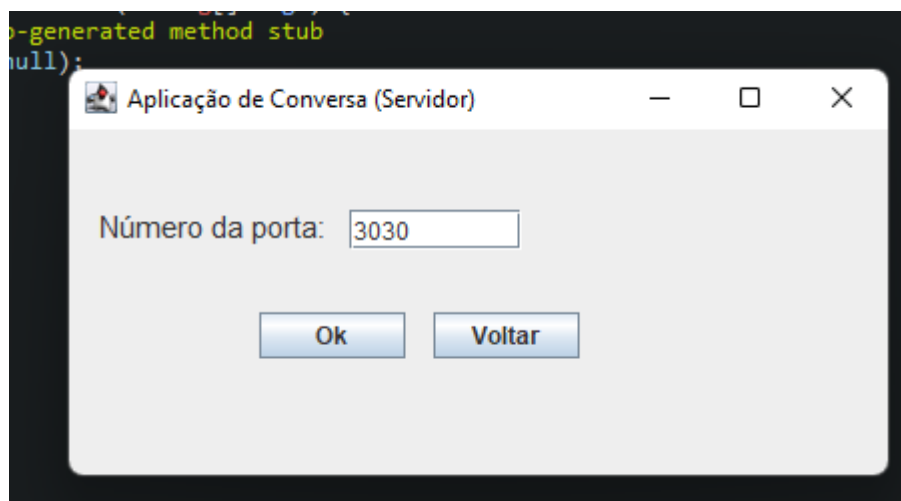

7 Apresentação do programa funcionando em um computador



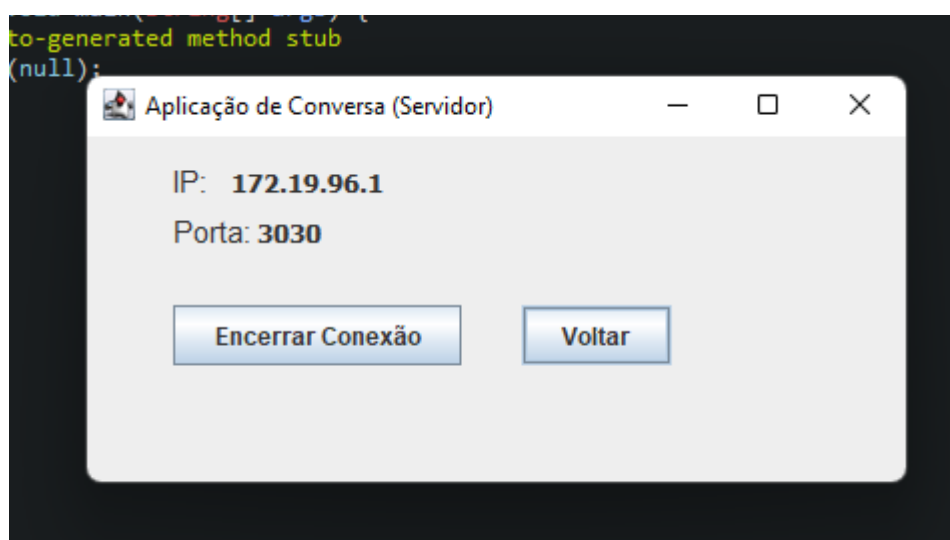
fonte: Guilherme Augusto,2022.



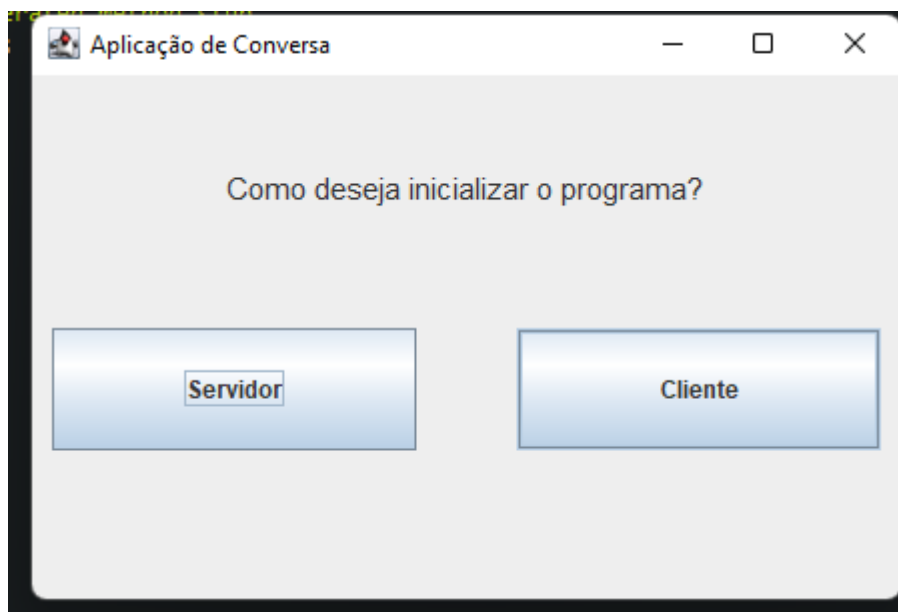
fonte: Guilherme Augusto,2022.



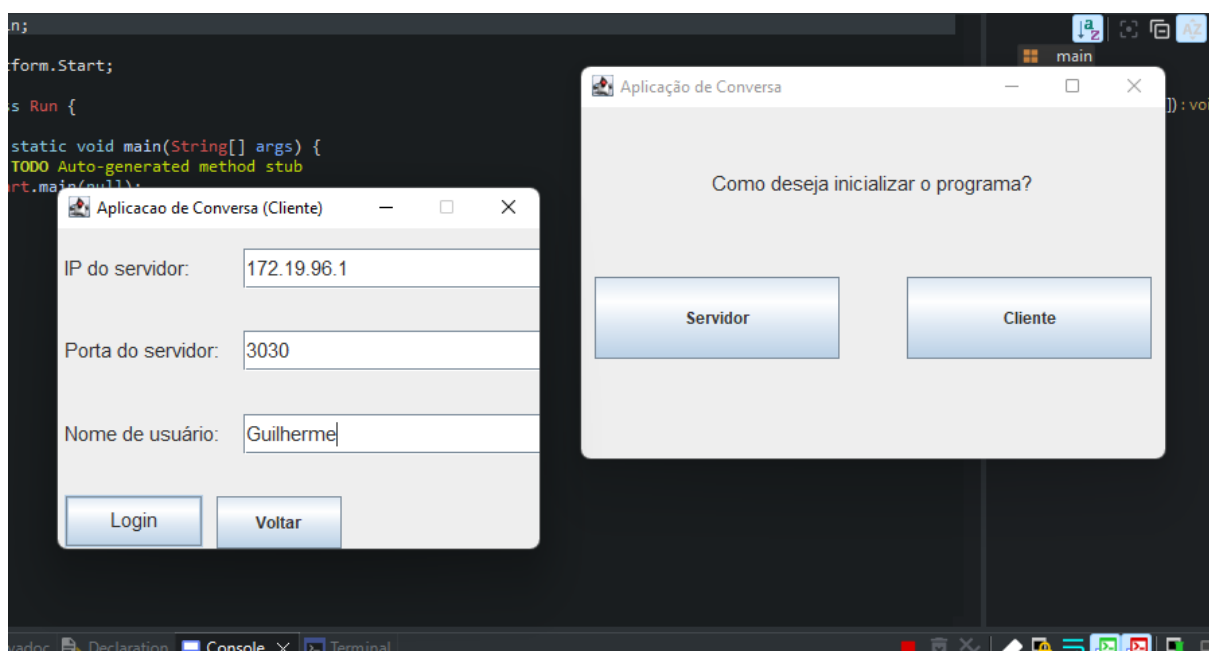
fonte: Guilherme Augusto,2022.



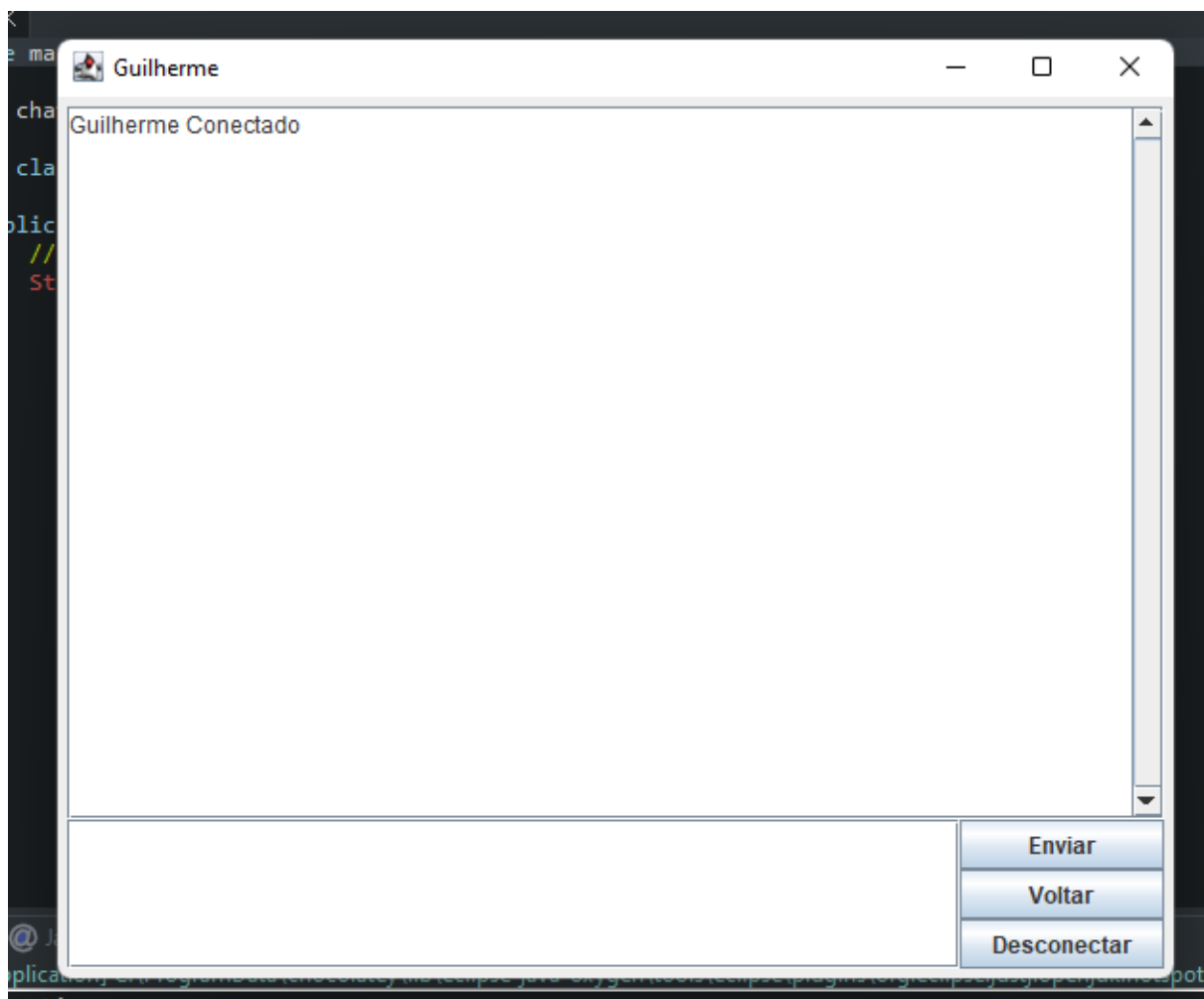
fonte: Guilherme Augusto,2022.



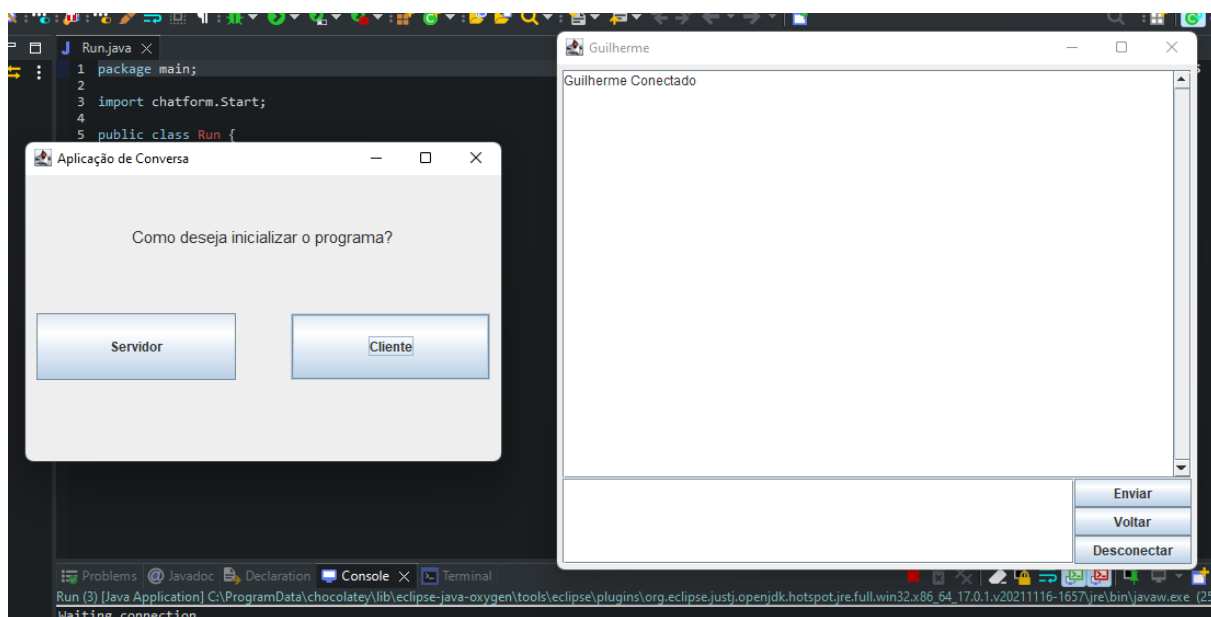
fonte: Guilherme Augusto,2022.



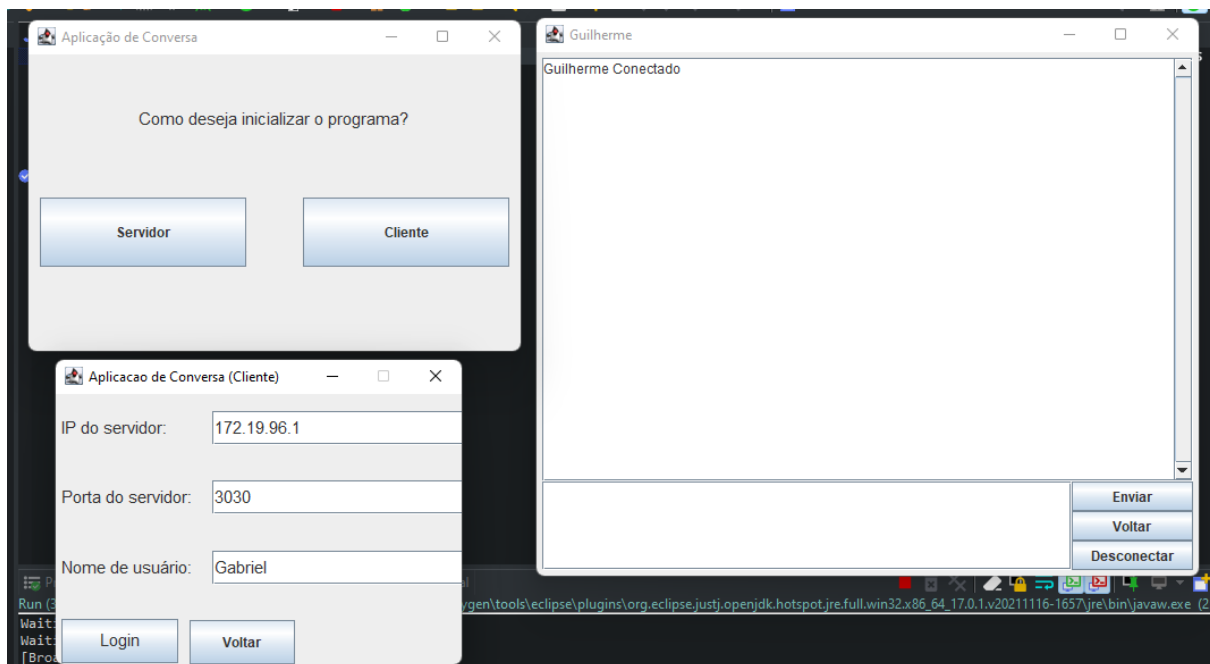
fonte: Guilherme Augusto,2022.



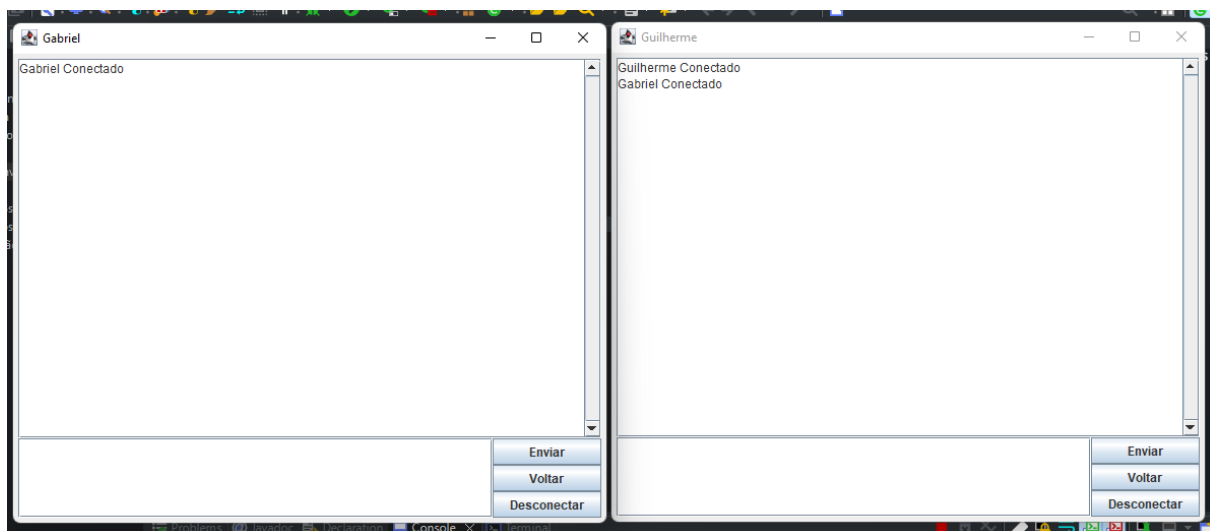
fonte: Guilherme Augusto,2022.



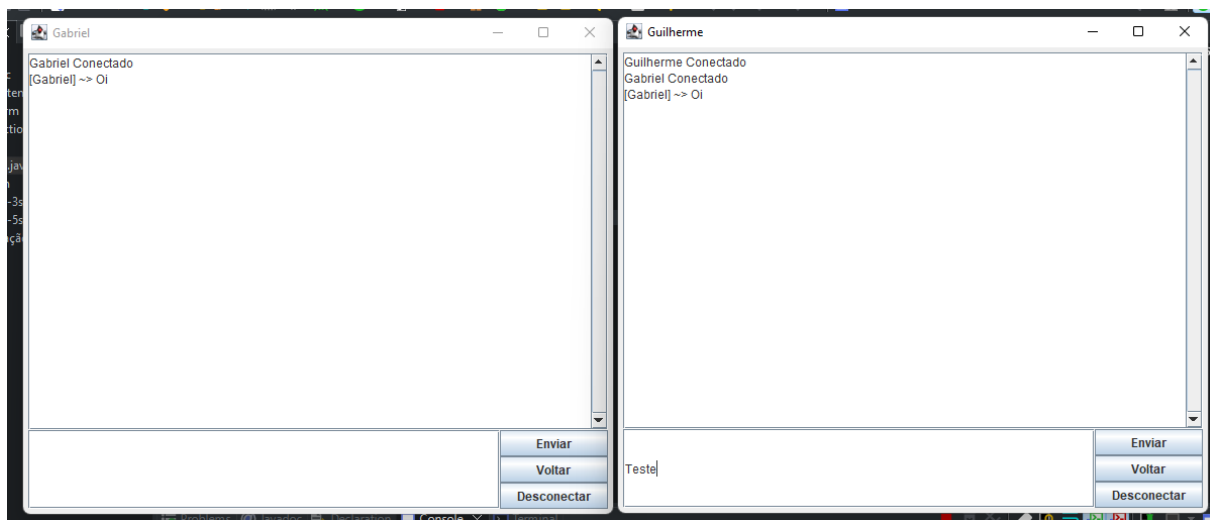
fonte: Guilherme Augusto,2022.



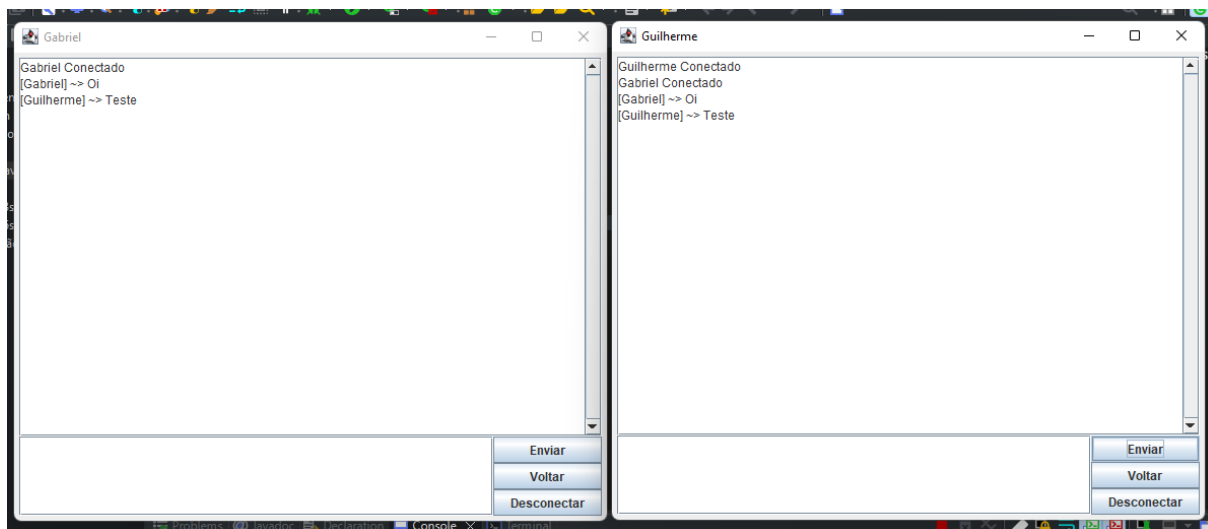
fonte: Guilherme Augusto,2022.



fonte: Guilherme Augusto,2022.



fonte: Guilherme Augusto,2022.



fonte: Guilherme Augusto,2022.

8 Bibliografia

Mozilla Developer 2022. Disponível em:

[Arpanet - | MDN \(mozilla.org\)](#) Acesso em 28/04/2022

Tecmundo 2012 Disponível em:

[O que é TCP/IP? - TecMundo](#) Acesso em 28/04/2022

Pantuza Blog 2017 Disponível em:

[O que são e como funcionam os Sockets \(pantuza.com\)](#) Acesso em 28/04/2022

GOV CRTRJ 2021 Disponível em:

[Redes de Comunicação de Dados | Principais Conceitos – CRT-RJ \(crtrj.gov.br\)](https://www.crtrj.gov.br/Redes-de-Comunicacao-de-Dados-Principais-Conceitos-CRT-RJ)

Acesso em 29/04/2022

ESCOLA, Brasil 2020 Disponível em:

[Comunicação de Dados - Brasil Escola \(uol.com.br\)](https://www.uol.com.br/comunicacao-de-dados-brasil-escola) Acesso em 07/05/2022

Bibliografias consultadas

COMER, Douglas E. Internet Working with TCP/IP. 4a. Ed. Prentice Hall, 2000.

FOROUZAN, Behrouz A. Comunicação de dados e redes de computadores. 3a. ed. Porto Alegre: Bookman, 2006.

STALLINGS, William. Data and computer communications. 8a. ed. Upper Saddle River, NJ : Prentice Hall, 2006.

STALLINGS, William. Redes e sistemas de comunicação de dados. 5ª. ed. Rio de Janeiro: Elsevier, 2005."

9 Ficha de Atividade Prática Supervisionada

FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Gabriel Macedo Ramos TURMA: CC5A33 RA: F281060
CURSO: Ciência da Computação CAMPUS: Tatuapé SEMESTRE: 5º Semestre
TURNO: Diurno

CÓDIGO DA ATIVIDADE: 77B2 SEMESTRE: 5º Semestre ANO GRADE: 2022/23

| DATA DA ATIVIDADE | DESCRIÇÃO DA ATIVIDADE | TOTAL DE HORAS | ASSINATURA DO ALUNO | HORAS ATRIBUÍDAS (1) | ASSINATURA DO PROFESSOR |
|-------------------|--------------------------------------|----------------|----------------------|----------------------|-------------------------|
| 19/05/2022 | Confecção do projeto destinado a APS | 75h | Gabriel Macedo RAMOS | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: _____

AValiação: _____

Aprovado ou Reprovado

NOTA: _____

DATA: ____/____/____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO

FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Matheus Moreira Dias TURMA: CC5A33 RA: F2265B8a
CURSO: Ciência da Computação CAMPUS: Tatuapé SEMESTRE: 5º Semestre
TURNO: Diurno

CÓDIGO DA ATIVIDADE: 77B2 SEMESTRE: 5º Semestre ANO GRADE: 2022/23

| DATA DA ATIVIDADE | DESCRIÇÃO DA ATIVIDADE | TOTAL DE HORAS | ASSINATURA DO ALUNO | HORAS ATRIBUÍDAS (1) | ASSINATURA DO PROFESSOR |
|-------------------|--------------------------------------|----------------|----------------------|----------------------|-------------------------|
| 19/05/2022 | Confecção do projeto destinado a APS | 75h | Matheus Moreira Dias | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: _____

AValiação: _____

Aprovado ou Reprovado

NOTA: _____

DATA: ____/____/____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO

FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Guilherme Augusto Sbizeiro Correa TURMA: CC5A33 RA: F23528CURSO: Ciência da Computação CAMPUS: Tatuapé SEMESTRE: 5º SemestreTURNO: DiurnoCÓDIGO DA ATIVIDADE: 77B2 SEMESTRE: 5º Semestre ANO GRADE: 2022/23

| DATA DA ATIVIDADE | DESCRIÇÃO DA ATIVIDADE | TOTAL DE HORAS | ASSINATURA DO ALUNO | HORAS ATRIBUÍDAS (1) | ASSINATURA DO PROFESSOR |
|-------------------|---|----------------|---------------------|----------------------|-------------------------|
| 19/05/2022 | Confecção do projeto <u>destinado a APS</u> | 75h | Sbizeiro | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: _____

AVALIAÇÃO: _____

Aprovado ou Reprovado

NOTA: _____

DATA: ____/____/____

Link para repositório da APS no Google Drive e no GitHub

Google Drive:

<https://drive.google.com/drive/folders/1y0JGdIWIkx1jXWLdQyzD5Hhpvb13oR0U?usp=sharing>

GitHub: [Guilherme-Sbizeiro/APS-5-Semestre: Repositório com os arquivos e programa da aps de 5º semestre de Ciência da Computação \(github.com\)](#)