Universidade Paulista
I.C.E.T. – São José do Rio Preto – J.K.

Alexandre Ribeiro dos Santos

João Victor Tasifano de Paula Tagliari Brandão

Matheus Satake de Souza

Nathan Lucas Santos Nicolau

Paulo Henrique Gois de Padua

“DESENVOLVIMENTO DE UMA FERRAMENTA PARA COMUNICAÇÃO EM REDE”

4º Semestre – 2021
São José do Rio Preto, SP
2021

Discentes: Alexandre Ribeiro dos Santo **RA:** N638EC8 **Turma:** CC4Q28

Discentes: João Victor T. P. Tagliari Brandão **RA:** G14EBF0 **Turma:** CC4P28

Discentes: Matheus Satake de Souza **RA**: N579137 **Turma:** CC4Q28

Discentes: Nathan Lucas Santos Nicolau **RA:** F310FH5 **Turma:** CC4Q28

Discentes: Paulo Henrique Gois de Padua **RA:** F20FCF2 **Turma:** CC4P28

Docente: Fábio Renato de Almeida

"DESENVOLVIMENTO DE UMA FERRAMENTA PARA COMUNICAÇÃO EM REDE"

Trabalho Acadêmico, apresentado à UNIP – Universidade Paulista, como parte das exigências para conclusão da disciplina "Atividades Práticas Supervisionadas" do Curso de Graduação em Ciências da Computação.

São José do Rio Preto, SP

2021

SUMÁRIO

1 OBJETIVO E MOTICAÇÃO

Como universitários, durante o decorrido tempo de nossa esperada e dedicada graduação em Ciência da Computação, somos incumbidos da produção de trabalhos de cunho acadêmico, tanto individualmente, com uso de habilidades individuais, como, em maior importância para o crescimento intelectual, de trabalhos e projetos com desenvolvimento em grupo.

Dessa forma, o tema DESENVOLVIMENTO DE UMA FERRAMENTA PARA COMUNICAÇÃO EM REDE nos propôs o desafio de produção, como parte principal, de uma aplicação escrita em Java ou C# que se possibilita a comunicação de duas ou mais pessoas através do protocolo TCP/IP, e dessa forma o fizemos.

Dito isso, estando bem cientes todos os estudantes comprometidos com esse trabalho, da responsabilidade e influência dele para com a formação de cada um, podemos, pelo dedicado esforço como equipe, apresentar na sequência, todo um extenso projeto desenvolvido por nós, na proposta de tema da APS do 4º Semestre de 2021, com orientação do docente Fábio Renato de Almeida.

De uma forma simplificada, podemos dizer que o objetivo de tudo o que foi planejado e desenvolvido mediante a esse tema, com o uso de pesquisas e métodos de aprendizado e interpretação, foi de produzir uma ferramenta prática e de fácil usabilidade, que atende-se aos requisitos exigidos para uma boa avaliação da competência de um trabalho em grupo, e que pudesse, da melhor forma possível, transparecer o aprendizado que nos foi passado nos dois últimos semestres sobre a linguagem Java, com todos os seus recursos e aplicabilidades possíveis.

2 INTRODUÇÃO

Este trabalho acadêmico elaborado por nós, discentes do 4º (quarto) semestre do curso de Ciência da Computação, cuja proposta o desenvolvimento de uma ferramenta para comunicação em rede. Realizado com base em diversos estudos, pesquisas e experiências, para toda tecnologia utilizada, apresentamos informações fundamentais para seu entendimento, assim como informações complementares sobre essas tecnologias empregadas em seu desenvolvimento.

Mediante a essa proposta do tema da Atividade Prática Supervisionada, que em suma seria o desenvolvimento de uma ferramenta para comunicação em rede, o programa, desenvolvido totalmente na linguagem de programação orientada a objetos Java, fazendo o uso de suas bibliotecas e seus conceitos de programação orientada a objetos como encapsulamento, herança e polimorfismo, a utilização de sockets foi utilizada para realizar a comunicação de duas ou mais clientes através do protocolo TCP/IP, por onde eles constantemente efetuarão a troca de informações.

Portanto, para melhor conhecimento das tecnologias envolvidas em seu desenvolvimento, apresentamos os conceitos básicos dos fundamentos da comunicação de dados em rede, como o modelo cliente-servidor com suas vantagens e desvantagens, complementando com a explicação de dois protocolos de comunicação de rede TCP/IP e UDP.

Em seguida, focando no desenvolvimento do programa, apontamos os elementos e as ferramentas utilizadas, assim como, o seu projeto, as estruturas e os módulos utilizados. E então, o código utilizado na criação com o exemplo de sua funcionalidade.

3 FUNDAMENTOS DA COMUNICAÇÃO DE DADOS EM REDE

Com o surgimento dos computadores, criou-se a necessidade de trocar informações entre máquinas e, assim, começaram a desenvolver redes de comunicação de dados, que são redes em que a informação circula de forma binária. Essas redes, comumente chamadas de redes de computadores, desenvolveram-se em ritmo acelerado, sobretudo devido aos avanços tecnológicos. Essas redes disponibilizam aos usuários diversas vantagens, como a possibilidade de acessar um computador de forma remota, acesso a informação, facilidade de transferência de dados, entre outros.

As redes de computadores podem ser classificadas em três grupos, sendo eles: LANs (*Local Area Network* – Redes locais), MANs (*Metropolitan Area Network* – Redes metropolitanas), WANs (*Wide Area Network* – Redes grandes). As redes locais são utilizadas em pequenas áreas, como edifícios ou um conjunto de edifícios, e operam em velocidades elevadas. Já as redes metropolitanas são usadas em áreas maiores (uma cidade, por exemplo) e operam em velocidades das várias dezenas ou centenas de MB/s. As redes grandes abrangem grandes áreas, como um país, e operam nas velocidades baixas, limitadas pelos recursos de transmissão.

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

1. Entrega: o sistema precisa entregar os dados no destino correto e esses dados precisam ser recebidos apenas pelo dispositivo destino.
2. Confiabilidade: o sistema deve garantir a entrega dos dados, sem modificações ou dados corrompidos.
3. Tempo de atraso: o sistema deve entregar os dados em um tempo predeterminado e finito.

Ademais, um sistema básico de comunicação de dados possui cinco elementos: mensagem (informação a ser transmitida), transmissor (dispositivo que envia a mensagem), receptor (dispositivo que recebe a mensagem), meio (caminho físico que a mensagem percorre) e protocolo (conjunto de regras que determina como a comunicação vai ser feita).

A comunicação entre esses dispositivos pode ser feita de três formas, sendo elas, *simplex*, *half-duplex* ou *full-duplex*. No modo *simplex*, a comunicação

é unidirecional, somente um dos dispositivos é capaz de transmitir. Na *half-duplex*, cada dispositivo pode transmitir e receber, porém nunca ao mesmo tempo. Já na *full-duplex*, ambos dispositivos podem transmitir e receber simultaneamente.

Quanto ao tipo de conexão, pode ser feito de duas formas, ponto-a-ponto ou multi-ponto. A conexão ponto-a-ponto proporciona um *link* dedicado entre os dispositivos, enquanto a conexão multi-ponto dois ou mais dispositivos compartilham do mesmo *link*.

3.1 MODELO CLIENTE-SERVIDOR

A arquitetura cliente servidor é uma arquitetura de aplicação distribuída, ou seja, na rede existem os fornecedores de recursos ou serviços a rede, que são chamados de servidores, e existem os requerentes dos recursos ou serviços, denominados clientes.

O cliente não compartilha nenhum de seus recursos com o servidor, no entanto ele solicita alguma função do servidor, sendo ele, o cliente, responsável por iniciar a comunicação com o servidor, enquanto o mesmo aguarda requisições de entrada.

Após vários modelos estudados de cliente-servidor caracterizou-se chamar tecnicamente de arquitetura multicamada, inspirado nas camadas no Modelo OSI, o processo de dividir a arquitetura de cliente-servidor em várias camadas lógicas facilitando o processo de programação distribuída, existe desde o modelo mais simples de duas camadas, e o mais utilizado atualmente que é o modelo de três camadas que é paralelo ao modelo de arquitetura de software denominado MVC (*Model-view-controller*).

3.1.1 Vantagens:

- Na maioria dos casos, a arquitetura cliente-servidor permite que os papéis e responsabilidades de um sistema de computação possam ser distribuídos entre vários computadores independentes que são conhecidos por si só através de uma rede. Isso cria uma vantagem adicional para essa arquitetura: maior facilidade de manutenção.

Por exemplo, é possível substituir, reparar, atualizar ou mesmo realocar um servidor de seus clientes, enquanto continuam a ser a consciência e não afetado por essa mudança;

- Todos os dados são armazenados nos servidores, que geralmente possuem controles de segurança muito maiores do que a maioria dos clientes. Os servidores podem controlar melhor o acesso a recursos, para garantir que apenas os clientes com credenciais válidas possam aceder e alterar os dados;
- Como o armazenamento de dados é centralizado, as atualizações dos dados são muito mais fáceis de administrar em comparação com o paradigma P2P. Em uma arquitetura P2P, atualizações de dados podem precisar ser distribuídas e aplicadas a cada nó na rede, o que consome tempo e é passível de erro, já que pode haver milhares ou mesmo milhões de nós;
- Muitas tecnologias avançadas de cliente-servidor estão disponíveis e foram projetadas para garantir a segurança, facilidade de interface do usuário e facilidade de uso;
- Funciona com vários clientes diferentes de capacidades diferentes.

3.1.2 Desvantagens:

- Clientes podem solicitar serviços, mas não podem oferecê-los para outros clientes, sobrecarregando o servidor, pois quanto mais clientes, mais informações que irão demandar mais banda.
- Um servidor poderá ficar sobrecarregado caso receba mais solicitações simultâneas dos clientes do que pode suportar;
- Este modelo não possui a robustez de uma rede baseada em P2P. Na arquitetura cliente-servidor, se um servidor crítico falha, os pedidos dos clientes não poderão ser cumpridos. Já nas redes P2P, os recursos são normalmente distribuídos entre vários nós. Mesmo se uma ou mais máquinas falharem no momento de download de um arquivo, por exemplo, as demais ainda terão os dados necessários para completar a referida operação.

3.2 TCP E UDP

O TCP (*Transmission Control Protocol*) e o UDP (*User Datagram Protocol*) são protocolos de internet que determinam como os dados são compartilhados. Eles possuem características muito distintas, porém ambos são usados para a mesma finalidade; enviar pacotes para um determinado endereço IP, seja na internet ou na rede local. Ambos possuem vantagens e desvantagens e devem ser escolhidos caso a caso.

O protocolo TCP é considerado o mais confiável porque ele garante a entrega e a integridade dos dados pacote, enquanto o UDP não garante essa a entrega. O TCP funciona com o padrão de *Three-Way Handshake*, ou seja, primeiro entra em contato com a máquina de destino, sincroniza, envia o pacote de dados e recebe a confirmação de que o pacote foi enviado.

Esse processo de envio e recebimento de pacotes acontece sempre que você faz qualquer ação na internet que utilize o TCP, como acessar um site, clicar em um *link*, enviar uma mensagem, entre outros. Ele adota um sistema de envio que enumera os pacotes e os envia em ordem, quando um dos pacotes não é enviado corretamente, ele envia novamente e só segue o fluxo após receber confirmação de que os dados do pacote foram recebidos sem erros.

Uma característica importante do TCP é a checagem de erros realizada por ele, assegurando que as informações não sejam corrompidas durante o trajeto. Esse processo de checagem e o próprio processo de envio de dados do TCP fazem dele um protocolo bastante confiável e amplamente utilizado por todos.

Claro que todo esse processo de confirmação de recebimento e checagem dos pacotes faz com que o protocolo TCP seja um pouco mais lento. No UDP não existe o processo de verificação de erros e confirmação do pacote, isso faz com que ele seja mais ágil em comparação ao TCP, porém menos confiável. do que o TCP

Dessa forma, o UDP é muito utilizado em situações em que se exige uma conexão rápida como uma chamada de voz como em transmissões de vídeo ao vivo, ou, até mesmo, em jogos online. Isso porque o UDP manda diretamente a informação e, em caso de erro de algum pacote, ele simplesmente manda o

próximo pacote programado, priorizando a transmissão em tempo real, mesmo que alguns trechos saiam com distorções ou falhas.

O TCP é um protocolo orientado a conexão. A orientação da conexão significa que os dispositivos de comunicação devem estabelecer uma conexão antes de transmitir os dados e devem fechar a conexão após a transmissão dos dados. Já o UDP é o protocolo orientado a datagramas. Isso ocorre porque não há sobrecarga para abrir, manter e encerrar uma conexão. O UDP é eficiente para o tipo de transmissão de rede de broadcast e multicast, como já exemplificado anteriormente. O TCP é usado principalmente em HTTP, HTTPS, FTP, SMTP e Telnet. Enquanto o UDP é usado em DNS, DHCP, TFTP, SNMP, RIP e VoIP.

Apesar de serem os protocolos de transporte mais populares, o TCP e o UDP não são os únicos protocolos. Existem outros como o RTP, DCCP, SCTP e, mais recentemente o QUIC (*Quick UDP Internet Protocol*), criado pelo Google. O QUIC une a rapidez do UDP com a segurança e checagem do TCP. É um sistema que vem sendo utilizado para conexões HTTP de páginas do Google e Facebook.

As páginas que utilizam o QUIC conseguem usar o sistema de criptografia e TLS sem lentidão na transmissão de dados. Esse novo protocolo foi adotado pelo Internet Engineering Task Force (IETF) em novembro de 2018 e possibilitou o novo protocolo de internet chamado de HTTP/3, que vem sendo amplamente utilizado.

4 PLANO DE DESENVOLVIMENTO

- **Linguagem de programação Java**

Java linguagem de programação orientada a objetos (OOP) da Oracle Corporation, sendo uma das mais utilizadas para desenvolvimento de *softwares* em todo o mundo, gera um código intermediário que é executado em qualquer computador que possua a JVM. Ela fornece amplas possibilidades de utilização, sendo um bom exemplo o projeto aqui produzido.

- **Interface Gráfica Java Swing**

Representações gráficas responsáveis pela interatividade entre um usuário e um programa.

Java Swing uma API para interfaces gráficas (GUI) utilizado com a linguagem Java. É biblioteca que permite a criação de componentes gráficos como janelas, menus, botões, entre outros.

- **IDE – Netbeans 8.2**

Ambiente de Desenvolvimento Integrado (IDE) gratuito e de códgo aberto disponível em diversos sistemas operacionais, com suporte para as linguagems Java, JavaScript, HTML5, PHP, entre outros. fornece ferramentas muito úteis para a produção de aplicações com códigos-fonte mais bem produzidos, como o apontamento de erros de sintaxe, possíveis bugs, e até sugestões melhores do que as implementadas em código além de fornecer a possibilidade de trabalhar com interfaces gráficas, editando-as com gestos de clicar e arrastar.

- **SO – Windows 10**

Uma versão do Microsoft Windows, são sistemas operativos produzidos e comercializados pela Microsoft Corporation. Lançado oficialmente em 29 de julho de 2015.

Sistemas operacionais são responsáveis pelo gerenciamento de todo o *hardware* de um computador. Como o gerenciamento de processos (programas em execução) a fim de organiza-los para um melhor uso do processador, além de outros papéis importantes para a funcionalidade de um computador.

5 PROJETO DO PROGRAMA

O programa é um chat simples que utiliza o protocolo de envio e recebimento de dados TCP/IP para criar a conexão por soquete entre um aplicativo cliente e um servidor, permitindo a constante comunicação entre eles.

O aplicativo foi desenvolvido de forma completa utilizando a linguagem de programação Java, levando em conta sua interface gráfica que foi desenvolvida utilizando a API Swing, utilizando seus recursos como sua vasta biblioteca, que nos fornece diversas funções para facilitar a programação.

Logo, prosseguiremos explicando mais das tecnologias utilizadas na projeção do programa.

5.1 JAVA

Figura – Logo Comercial Java da Oracle Corporation.

Java, uma linguagem de programação orientada a objetos (OOP) de alto nível baseada em classes. É considerada uma linguagem de programação versátil, podendo ser utilizada em diversos propósitos. Assim como pretendido em seu slogan “*write once, run anywhere*” (“Escreva uma vez, execute em qualquer lugar”), os programadores que utilizam dessa linguagem podem criar aplicativos que executam em diversos lugares como uma TV ou um relógio de pulso inteligente, pois um código de Java é normalmente compilado em um *bytecode* que é executado em qualquer computador que tenha a máquina virtual Java, ou JVM (*Java Virtual Machine*, em inglês). Sua sintaxe é bastante similar ao C/C++, porém com menos funções de baixo-nível.

A originalidade do nome Java e de sua logo comercial, foi uma bebida, café, que naquela época café java (*java cooffee*) era considerado um café forte, pois os cafés cultivados na ilha de Java eram fortes.

Criado por James Gosling, na empresa Sun Microsystems, que em 2008 foi adquirida pela Oracle Corporation e se mantém até hoje com ela.

5.2 JAVA SWING

O Java Swing é ferramenta de interface gráfica para Java, que é parte da JFC (do inglês, *Java Foundation Classes*) da Oracle, uma API que fornece aos programas uma GUI, ou interface gráfica.

Ele foi desenvolvido para prover uma criação de interfaces gráficas mais completas e sofisticadas que a sua API para interface gráfica anterior, o AWT. O Swing fornece elementos como cores, formas, esboço ou *layout* e diversas fontes, e comportamento dinâmico de alguns elementos como botões, menus e janelas. Ele também fornece diversos componentes avançados para uma interface mais detalhada, sendo eles, mais eficientes e mais flexíveis se comparado com o AWT. Todos os seus componentes são todos escritos em Java, são multiplataformas .

Ainda sobre domínio da Sun Microsystems, em 2008 eles lançaram um *framework* baseado em CSS/FXML, que futuramente veio a ser o sucessor do Swing, chamado JavaFX, ainda mais completa.

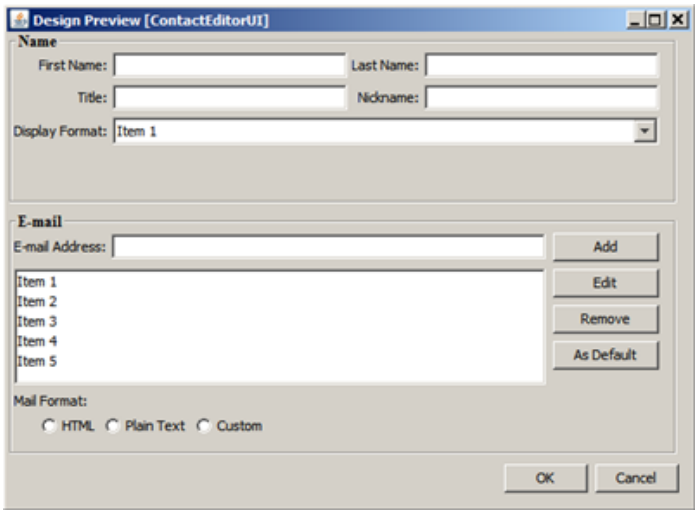


Figura - exemplo de janela feita em Java Swing

https://netbeans.apache.org/kb/docs/java/quickstart-gui_pt_BR.html

5.3 INTERFACE GRÁFICA

Interface gráfica do usuário (acrônimo GUI, do inglês *Graphical User Interface*) responsável por facilitar a interatividade entre um usuário e um programa por meio de quaisquer representações gráficas.

A interface de um programa, ou sua tela, é onde dispõe as coisas na tela, como uma calculadora, por exemplo, que exibe as informações em sua representação gráfica, ou a tela inicial do Windows com a sua barra de tarefas, menu iniciar e a área de trabalho. O usuário ao realizar a interação executará as funções do programa, seja fazer cálculos, ou abrir uma outra interface gráfica.

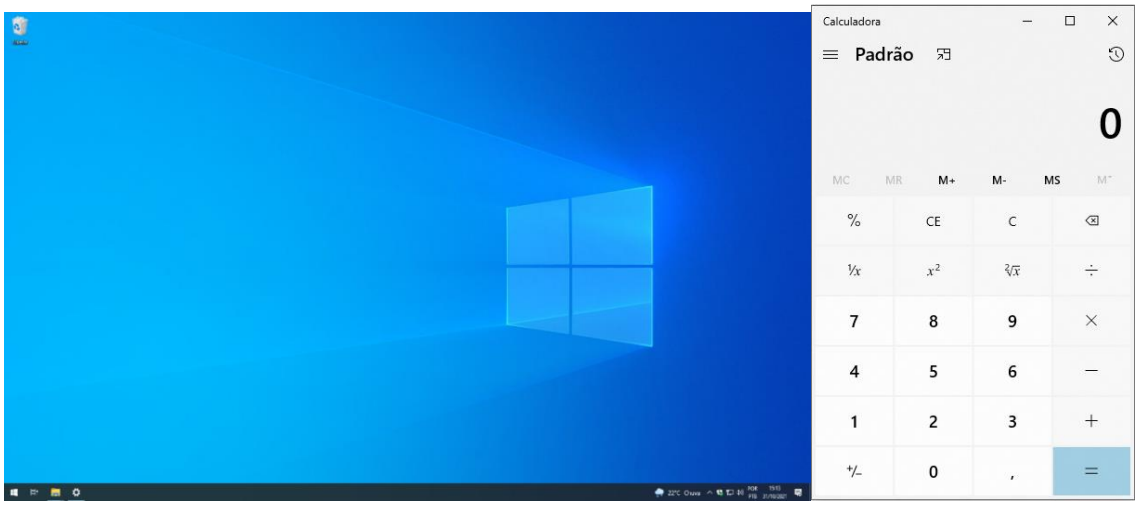


Figura – Tela inicial do Windows 10 e a interface gráfica de sua calculadora

Com a utilização de amplas tecnologias, a interface gráfica fornece ao usuário uma plataforma que qual ele possa interagir. E em computadores pessoais, por exemplo, existe a tecnologia conhecida como WIMP, que nada mais é do que uma combinação de janelas, ícones, menus e ponteiros, em inglês *window, icon, menu, pointing* respectivamente. As janelas são onde as informações são apresentadas de forma organizada, os ícones podem ser a representação visual de algo, por exemplo, poderíamos transformar um ponto de exclamação em um ícone de aviso, essas funções ficam organizadas em menus que realizam suas ações pela interação pelo ponteiro. Interligando o WIMP estão os gerenciadores de janelas. A unidade de processamento gráfico (GPU),

ou placa de vídeo, é a responsável pela geração, renderização e cálculo dos elementos visuais.

5.4 SOCKETS

Socket é o que faz a conexão para uma comunicação entre uma fonte e um destino de dois processos, tanto em máquina local quanto na rede. Os sockets de rede, TCP/IP, são representados por ip:porta, sendo o ip o responsável por localizar onde a máquina está dentro da rede e a porta para direcionar os dados recebidos pelo roteador para o aplicativo correspondente aquela porta. Por exemplo, um socket 127.0.0.1:2021, vai se conectar na máquina local (127.0.0.1 ou *localhost*) na porta 2021.

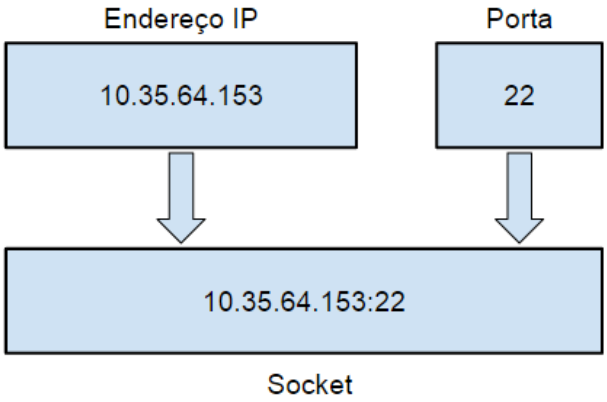


Figura – Exemplo de socket
(http://www.bosontreinamentos.com.br/redes-computadores/curso-de-redes-camada-de-transporte-da-pilha-tcpip/)

Na Internet a base para todas as comunicações é o protocolo TCP/IP, baseando-se no modelo OSI, possui um empilhamento de quatro camadas (4- Aplicação, 3- Transporte, 2- Rede, 1- Física).

Levando em conta esse modelo de quatro camadas do TCP/IP, os *sockets* são criados entre a camada da aplicação e a camada de transporte, para que ele possa realizar o intermédio entre a camada da aplicação e a camada da rede. Assim, sendo essa conexão estabelecida, os dados entram pela rede e chegam diretamente ao destinatário, permitindo assim a troca de informação de um com outro.

Para a utilização de *sockets*, uma das estruturações mais simples para um programa é utilizar o modelo cliente-servidor (em inglês *client/server model*). Essa estrutura distribui as tarefas e cargas de trabalho entre servidor e cliente.

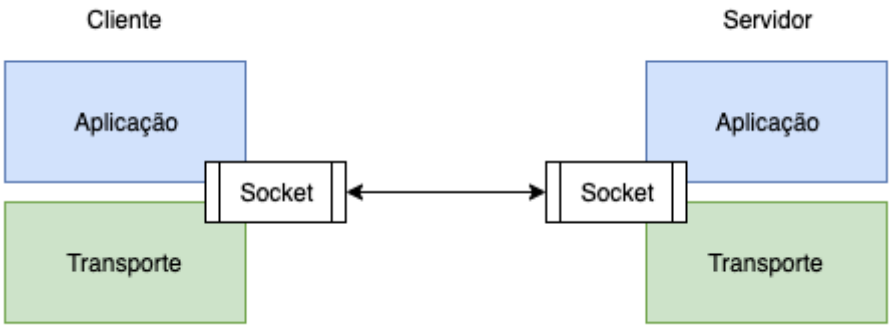


Figura - Exemplo da Conexão e Localização do Socket
(https://www.treinaweb.com.br/blog/uma-introducao-a-tcp-udp-e-sockets)

O servidor será o responsável por aguardar conexões de um ou mais programas clientes, para compartilhar de seus recursos, o cliente, portanto, não compartilhará recursos, somente solicitará o acesso as funções disponibilizadas pelo aplicativo servidor. Portanto, o cliente requisitará o acesso para o servidor nessa ip:porta específica, enquanto o servidor estará esperando as requisições feitas naquele ip:porta.

Um exemplo bastante utilizado no dia a dia é o navegador de internet, que é uma aplicação que utiliza sockets para requisitar dados de um servidor.

6 RELATÓRIO COM LINHAS DE CÓDIGO

O projeto desenvolvido, baseado no uso de *sockets*, descritos anteriormente, procura estabelecer conexão entre duas ou mais máquinas, como uma forma simples de *Chat* (rede local) sendo composto por dois executáveis Java.

Segue abaixo, as imagens dos códigos-fonte desenvolvidos:

6.1 NOMES DOS PROJETOS, PACOTES E CLASSES

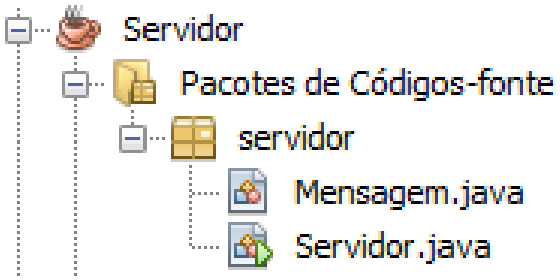


Imagem 1 – Projeto Servidor

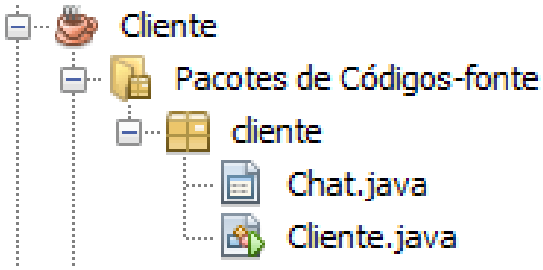


Imagem 2 - Projeto Cliente.

6.2 BIBLIOTECAS IMPORTADAS

```
package servidor;

import java.awt.Font;
import java.io.IOException;
import java.io.PrintStream;
import java.net.*;
import java.util.ArrayList;
import javax.swing.JFrame;
import javax.swing.JTextField;
```

Imagem 3 – Bibliotecas Java Utilizadas no Pacote Servidor.

```
package cliente;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.util.logging.*;
import javax.swing.*;
import static javax.swing.JOptionPane.*;
import java.time.*;
import java.time.format.DateTimeFormatter;
```

Imagem 4 – Bibliotecas Java Utilizadas no Pacote Cliente.

6.3 APLICAÇÃO SERVIDORA – CLASSE SERVIDOR

```
public class Servidor {
    private static JFrame tela_status = new JFrame();

    public Servidor(JFrame telaServidor, int campo) {
        Servidor.tela_status = telaServidor = new JFrame();
    }
    public static void main(String[] args) throws IOException, NullPointerException {
        ServerSocket server = new ServerSocket(5021);
        Socket socket;
        ArrayList<PrintStream> clientes = new ArrayList<>();
        try {
            int num_clientes_conectados;
            while (true) {
                socket = server.accept();
                clientes.add(new PrintStream(socket.getOutputStream())); // guarda o endereço do cliente
                Mensagem mensagem = new Mensagem(socket, clientes);
                num_clientes_conectados = clientes.size();
                JTextField num_clientes = setupComponents(num_clientes_conectados);
                startStatus(tela_status, num_clientes);
            }
        } catch (IOException ex) {

        }
    }
```

Imagem 5 – Classe Servidor.

logo nas primeiras linhas de código, dentro da classe Servidor, será definido um campo, o *JFrame* “tela_status” Ele foi definido como campo para ser utilizado dentro do *main.* Logo após é iniciado pelo construtor Servidor();

Com esse campo, será criada uma janela pertencente a aplicação. Ela se tornará responsável pelo encerramento do Servidor quando fechada, e mostrará a quantidade de clientes que foram conectados.

Inicialmente, dentro do *main*, são definidas as variáveis que tornaram a aplicação funcional.

O “server”, do tipo *ServerSocket*, o “socket” do tipo *Socket* e “clientes”, sendo um *ArrayList<PrintStream>.*

Após isso, as instruções (todas dentro de um *try-catch*, para captura de erros) determinadas em um laço de repetição infinito *while(true)*, serão as responsáveis por estabelecer a conexão entre o cliente e o Servidor e criar um objeto do tipo Mensagem “mensagem”, passando os parâmetros necessários da classe.

Vale ressaltar que dentro da classe Mensagem, será criada a *Thread*, que será executada em segundo plano e manterá a conexão aberta enquanto não for encerrada sua execução.

Por final serão chamados dois métodos que irão configurar a pequena tela com o número de clientes.

```
    private static JTextField setupComponents(int numero_clientes) {
        JTextField campo_clientes = new JTextField("N° de clientes conectados: " + numero_clientes);
        campo_clientes.setSize(tela_status.getPreferredSize().width, tela_status.getPreferredSize().height);
        campo_clientes.setHorizontalAlignment(JTextField.CENTER);
        campo_clientes.setToolTipText("Número de conexões que foram estabelecidas com o servidor");
        campo_clientes.setEditable(false);
        campo_clientes.setVisible(true);
        campo_clientes.setFont(new Font("Tahoma", Font.BOLD, 10));
        return campo_clientes;
    }

    private static void startStatus(JFrame tela, JTextField num_clientes) {
        tela.setSize(num_clientes.getPreferredSize().width + 20, 60);
        tela.setResizable(false);
        tela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        tela.add(num_clientes);
        tela.setLocation(950, 139);
        tela.setVisible(true);
    }

}
```

Imagem 6 – Componentes Gráficos Utilizados.

Aqui, estão os métodos mencionados anteriormente, o primeiro, cria a variável “campo_clientes” do tipo *JTextField* e seta suas configurações para retorná-lo a outra variável do mesmo tipo que o chamou, passando o parâmetro da quantidade de usuários.

O segundo método seta as características da “tela” que foi criada, como tamanho e localização. Ele recebe como parâmetros o *JFrame* já criado e *JTextField* criado no outro método.

6.4 APLICAÇÃO SERVIDORA – CLASSE MENSAGEM

```
import java.io.*;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.*;
import java.util.ArrayList;

public class Mensagem {
  private final Socket socket;
  private ArrayList<PrintStream> clientes;
  public Thread t;

  public Mensagem(Socket socket, ArrayList<PrintStream> clientes) throws IOException {
    this.socket = socket;
    this.clientes = clientes;
    Thread();
  }

private void Thread() throws IOException {
  t = new Thread(new Runnable() {
    @Override
    public void run() {
      String mensagem = "";
      try {
        InputStreamReader in = new InputStreamReader(socket.getInputStream());
        BufferedReader out = new BufferedReader(in);
        while ((mensagem = out.readLine()) != null) {
          enviarMensagem(mensagem);
            if(clientes.size() == 0){
              in.close();
              out.close();
              fechar(clientes.size());
            }
        }
      } catch (IOException ex) {
        ex.getStackTrace();
      }
    }
  });
  t.start();
}
```

Imagem 7 – Visualização da Classe Mensagem.

Sobre a classe Mensagem, são criados os campos para a conexão e iniciados pelo seu construtor e chamado o método Thread();

Nesse método, há o *@Override* do método *run( )* e nesse método (dentro de um bloco *try-catch*) há a criação, para cada usuário conectado, de um *InputStreamReader* e de um *BufferedReader*, respectivamente responsáveis pela leitura das entradas dos usuários e pela disponibilidade de leitura a quem estiver conectado com ele. Posteriormente, com o tratamento dentro de um *while*, o método “enviarMensagem” é chamado, e se não houver nenhum usuário, a *thread* é fechada com o método “fechar”.

```
  public int fechar(int value){
    if(value == 0){
        t.stop();
        System.exit(value);
    }
    return value;

   }
  private void enviarMensagem(String mensagem) {
    for (int i = 0; i < clientes.size(); i++) {
      clientes.get(i).println(mensagem);
      clientes.get(i).flush();
    }
  }

  public void setClientes(ArrayList<PrintStream> clientes) {
    this.clientes = clientes;
  }
}
```

Imagem 8 – Métodos utilizados na Classe Mensagem.

Aqui estão os métodos citados, com suas funcionalidades e um construtor para o *ArrayList* “clientes”.

6.5 APLICAÇÃO CLIENTE – CLASSE CLIENTE

```
package cliente;

import static javax.swing.JOptionPane.*;

public class Cliente {
    public static void main(String[] args) {
    String nome = showInputDialog(null, "Digite seu nome de Usuário: ", "Bem-Vindo ao OneMessage",PLAIN_MESSAGE);
    Chat chat = new Chat(nome);
    chat.setVisible(true);

  }


}
```

Imagem 10 – Programa do Inicializador Cliente

Nessa imagem, é visualizada a Classe com poucas instruções. É criada uma pequena janela *JOption*, que armazenará o nome que o usuário fornecerá em uma *String* “nome ”e após isso, irá criar uma interface gráfica, Chat chat (que é uma classe estendida de *JFrame*) que será a visualização gráfica das mensagens de dois ou mais usuários conectados simultaneamente.

6.5.1 Aplicação Cliente – Código Fonte do *JFrame Chat*

```
public class Chat extends javax.swing.JFrame {

    private Socket socket;
    private String nome;
    private BufferedReader out;
    private InputStreamReader in;
    private boolean rodar;
    private String horaConexao;
    private String horadaMensagem;

    public Chat(String nome) {
        this.nome = nome;
        rodar = true;
        try {
            socket = new Socket("localhost", 5021);
        } catch (IOException ex) {
            showMessageDialog(null, "Erro ao conectar!", "Erro", ERROR_MESSAGE);
            System.exit(0);
        }
        initComponents();
        setupComponents();
        Thread();
    }
```

Imagem 11 – Código Fonte da Classe Chat.

Logo no início da classe, são determinados os campos posteriormente utilizados para a funcionalidade da aplicação, como o nome, fornecido pelo usuário, o *socket*, para conexão, e os horários que serão armazenados nas suas respectivas variáveis para uso no *JFrame Chat*.

Com o início da do método construtor, já temos a inicialização do campo “nome” e do “socket”, dentro de um bloco *try-catch*, com uma mensagem em *JOptionPane* caso aconteça algum erro de conexão. Após isso, são chamados os métodos que serão explanados mais a frente e método da *Thread()*;

```
private void Thread() {
    setNomeCliente();
    Thread t = new Thread(new Runnable() {
        String msg;
        @Override
        public void run() {
            horaConexao = getHoraConexao();
            horadaMensagem = horaConexao; // em teoria, no momento em que a aplicação é iniciada,a primeira
            try {                         // mensagem será enviada no mesmo minuto.
                in = new InputStreamReader(socket.getInputStream());
                out = new BufferedReader(in);
              while ((msg = out.readLine()) != null) {
                  horadaMensagem = horaMensagem();
                  msgRecebida.setText(msgRecebida.getText() + msg + "\n");
                  if (!rodar) {
                      break;
                  }
              }
            } catch (IOException ex) {
                try {
                    showMessageDialog(null, "O Servidor foi desconectado...\n"
                            + "        " + "saindo da aplicação", "Aviso", WARNING_MESSAGE);
                    in.close();
                    out.close();
                    System.exit(0);
                } catch (IOException ex1) {
                    Logger.getLogger(Chat.class.getName()).log(Level.SEVERE, null, ex1);
                }
            } finally {
                try {
                    in.close();
                    out.close();
                    socket.close();
                } catch (IOException ex) {
                    showMessageDialog(null, "Erro ao Finalizar", "Erro", ERROR_MESSAGE);
                }
            }
        }
    });
    t.start();
}
```

Imagem 12 – Tratamento da *Thread*.

Assim como foi feito na classe Mensagem (imagem 7), o mesmo acontecerá aqui, com o @Override do método *run()*;

Com instruções seguidas, todas inclusas em blocos de captura de exceções, serão feitos os comandos para o tratamento das mensagens

recebidas e para os possíveis erros que ocorrerão. Por fim, é chamado o método que inicia a *Thread* “t”.

```
    @SuppressWarnings("unchecked")
    Generated Code

private void btnEnviarActionPerformed(java.awt.event.ActionEvent evt) {
    String msg = nome + " escreveu as " + horadaMensagem + ": ";
    try {
        PrintStream ps = new PrintStream(socket.getOutputStream());
        msg += msgEnviar.getText();
        ps.println(msg);
        ps.flush();
        msgEnviar.setText("");
    } catch (IOException ex) {
        showMessageDialog(null, "Erro ao enviar a mensagem!", "Erro", ERROR_MESSAGE);
    }
}

private void msgEnviarKeyPressed(java.awt.event.KeyEvent evt) {
    if (evt.getKeyCode() == KeyEvent.VK_ENTER) {
        String msg = nome + " escreveu as " + horadaMensagem + ": ";
        try {
            PrintStream ps = new PrintStream(socket.getOutputStream());
            msg += msgEnviar.getText();
            ps.println(msg);
            ps.flush();
            msgEnviar.setText("");
        } catch (IOException ex) {
            showMessageDialog(null, "Erro ao enviar a mensagem!", "Erro", ERROR_MESSAGE);
        }
    }
}
```


Imagem 12 – Programação de Componentes “btnEnviar” e “msgEnviar”.

Aqui estão programados dois eventos relacionados a aplicação, sendo ambos sobre o envio da mensagem escrita pelo usuário a tela de visualização do grupo, um através do clique do botão “btnEnviar” e o outro através do pressionar da tecla *ENTER*.

```
private void btnSairActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        if (btnSair.isEnabled() == true) {
            int resposta = showConfirmDialog(null, "Deseja realmente sair? ", "Sair do OneMessage", JOptionPane.OK_CANCEL_OPTI
            if (resposta == OK_OPTION) {
                socket.close();
                System.exit(0);
            } else {
                showMessageDialog(null, "Voltando...", "Cancelando saída", JOptionPane.INFORMATION_MESSAGE);
            }
        }

    } catch (IOException ex) {
        ex.getStackTrace();
    }
}
```

Imagem 12 – Programação de Componentes “btnSair”.

Nessa imagem, se encontra a funcionalidade do evento relacionado ao fechamento da aplicação por parte do usuário, através do clique do botão “btnSair”, com uma janela *JOption* para confirmação da decisão do usuário.

```
private void setNomeCliente() {
    nome_cliente.setToolTipText("Essa tela é do cliente " + nome);
    nome_cliente.setText(nome);
    nome_cliente.setFont(new Font("Tahoma", Font.BOLD, 10));
}

private void setupComponents() {
    setResizable(false);
    setLocationRelativeTo(null);
    setTitle("OneMessage");
}

private String getHoraConexao() {
    String horaDaConexao;
    LocalTime hora_atual = LocalTime.now(ZoneId.of("UTC-3"));
    DateTimeFormatter formatarHora = DateTimeFormatter.ofPattern("HH:mm");
    horaDaConexao = formatarHora.format(hora_atual);
    campo_hora.setText(horaDaConexao);
    return horaDaConexao;
}

private String setHoraMensagem() {
    while (true) {
        LocalTime hora_atual = LocalTime.now(ZoneId.of("UTC-3"));
        DateTimeFormatter formatterHora = DateTimeFormatter.ofPattern("HH:mm");
        String horaMensagemFormatada = formatterHora.format(hora_atual);
        return horaMensagemFormatada;
    }
}
```

Imagem 12 – Métodos Complementares para a Aplicação.

Como mencionado anteriormente, aqui se encontram os métodos que organizam algumas funcionalidades da aplicação, como o nome de cada usuário em um campo específico da tela, a hora da conexão na aplicação do usuário e hora que cada mensagem é enviada.

6.5.2 Aplicação Cliente – Interface Gráfica do *JFrame Chat.*

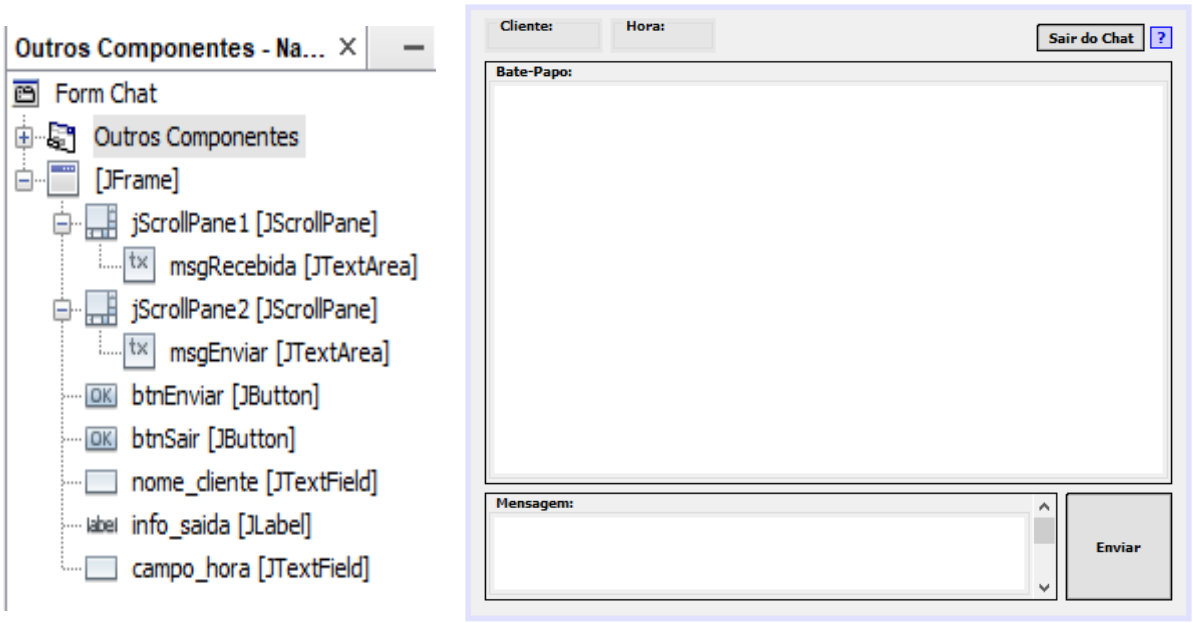


Imagem 12 – Interface Gráfica do Aplicativo e seus Componentes.

Aqui tratamos sobre a parte gráfica da aplicação, com os componentes utilizados (imagem 17) e suas nomeações, e do outro lado (imagem 18) a tela de visualização das ações e mensagens do(s) usuário(s).

Houve uma distribuição dos componentes visuais para uma fácil usabilidade da aplicação, como a aba principal tomando a maior parte da tela, que permite a visualização das mensagens enviadas por todos os conectados, a aba de escrita localizada abaixo, juntamente com o botão de envio, e, na parte superior, os campos que mostram ao usuário o nome fornecido por ele, que se torna visível a todos logo que ele envia alguma mensagem.

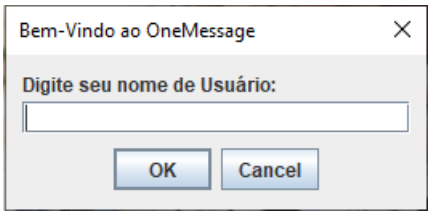
São inclusos também a hora em que o usuário se conectou, o botão para saída do Bate-Papo e uma janela de informação, que aponta para opção de fechamento total da aplicação, com encerramento do servidor.

Imagem 12 – Janelas dos Clientes que conectaram.

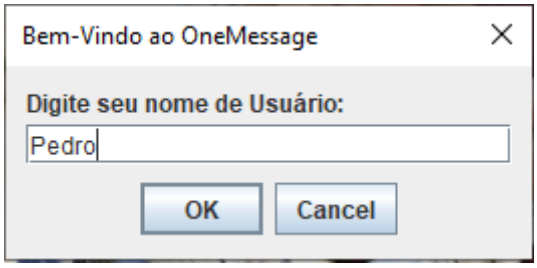
Essa janela mostra somente para quem executou o servidor, e apresenta quantidade de usuários que já se conectaram com o servidor.

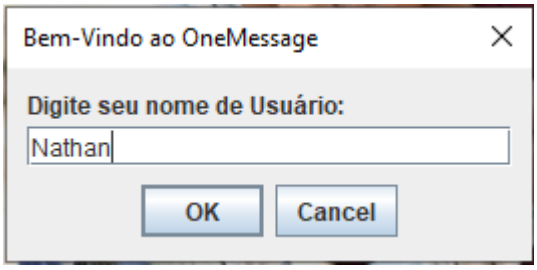
7 FUNCIONAMENTO DO PROGRAMA

Mostraremos o funcionamento do aplicativo em imagens e sua explicação, em sua ordem de execução, simulando exatamente a reação do programa ao ser acionado por um usuário.

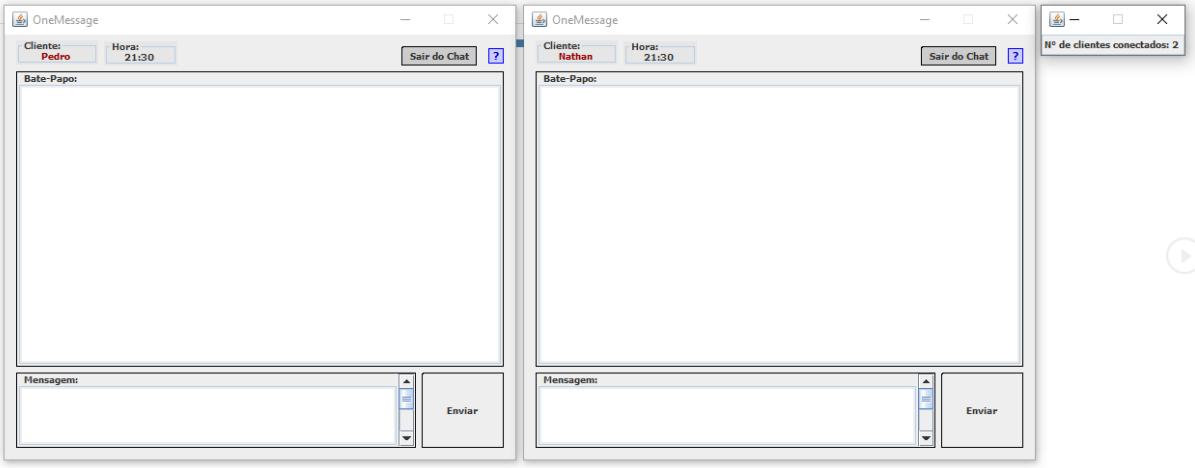


Janela Mostrada logo ao início do programa, o que for digitado nela será registrado como nome de usuário. Abaixo, verá exemplos de nomes usuários, utilizados nessa demonstração:

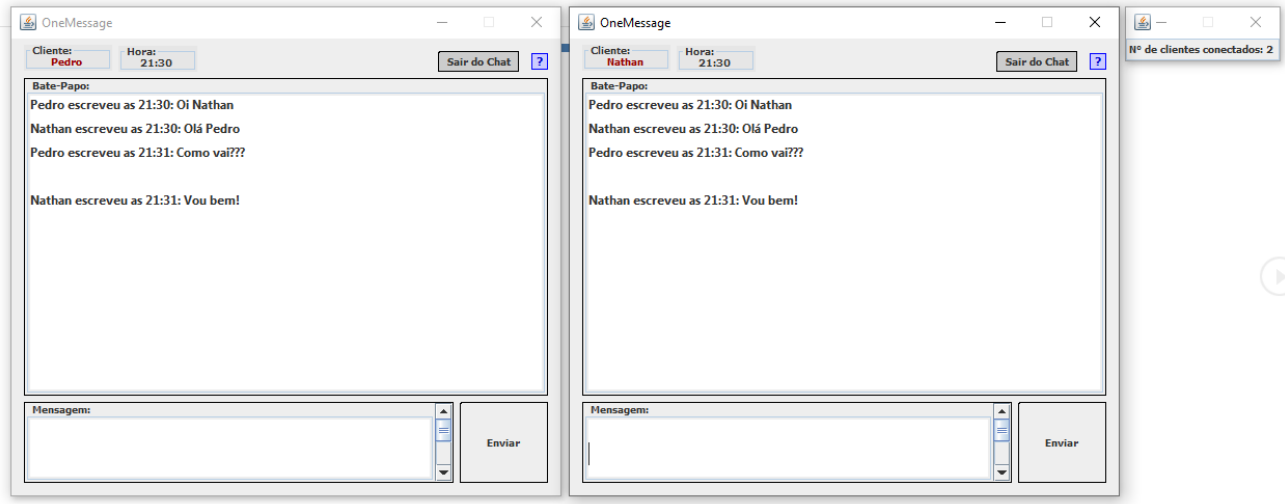


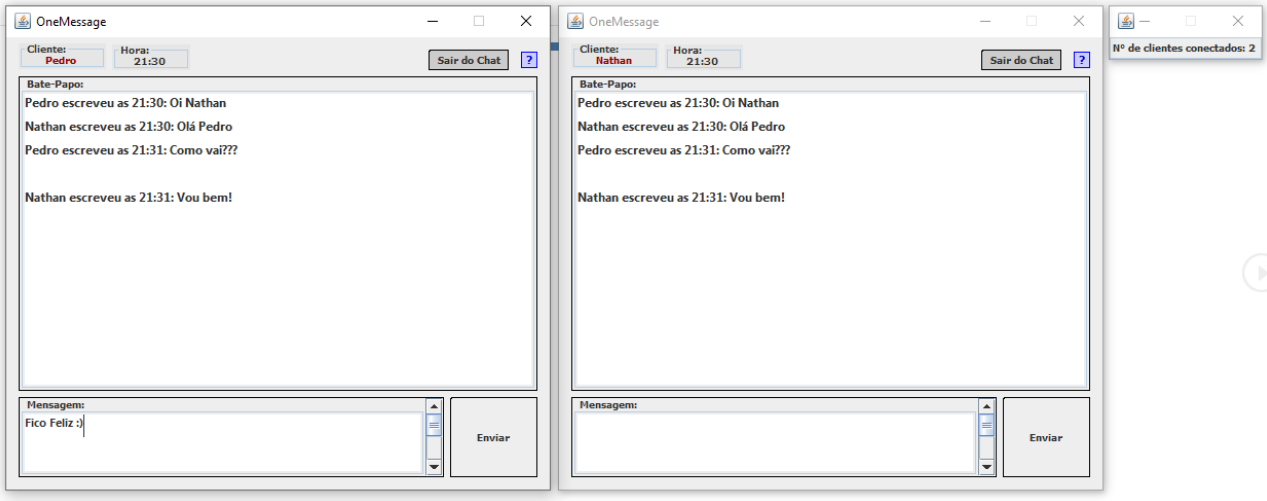


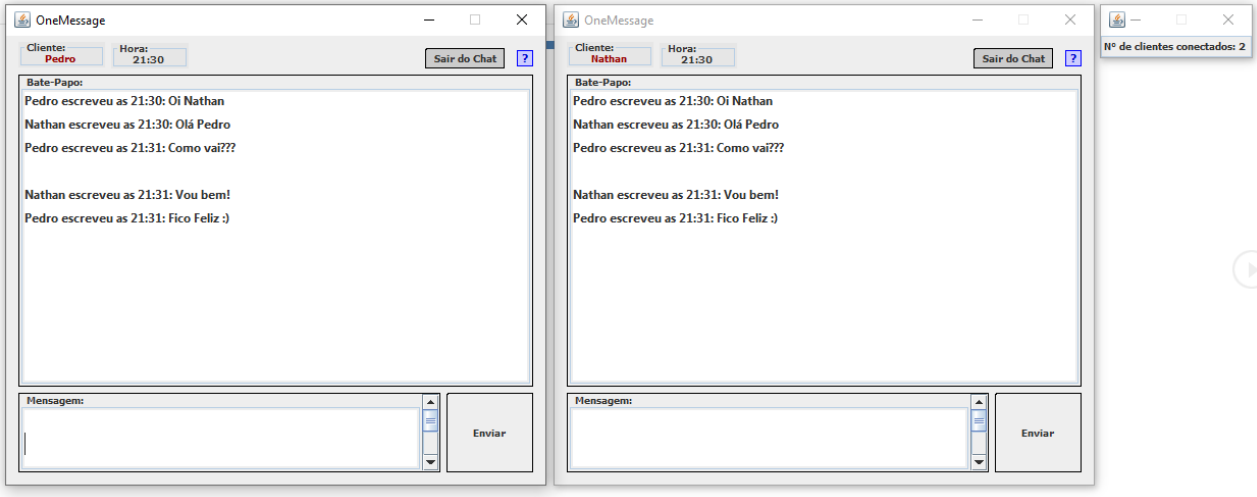
Assim que digitado o nome, temos a visualização da tela principal do aplicativo, com o nome e hora de conexão de ambos, no canto superior esquerdo, presente no canto superior direito o botão responsável pela finalização do chat. No Lado inferior, o campo onde se escreve a mensagem, e logo ao lado o botão para enviá-la.



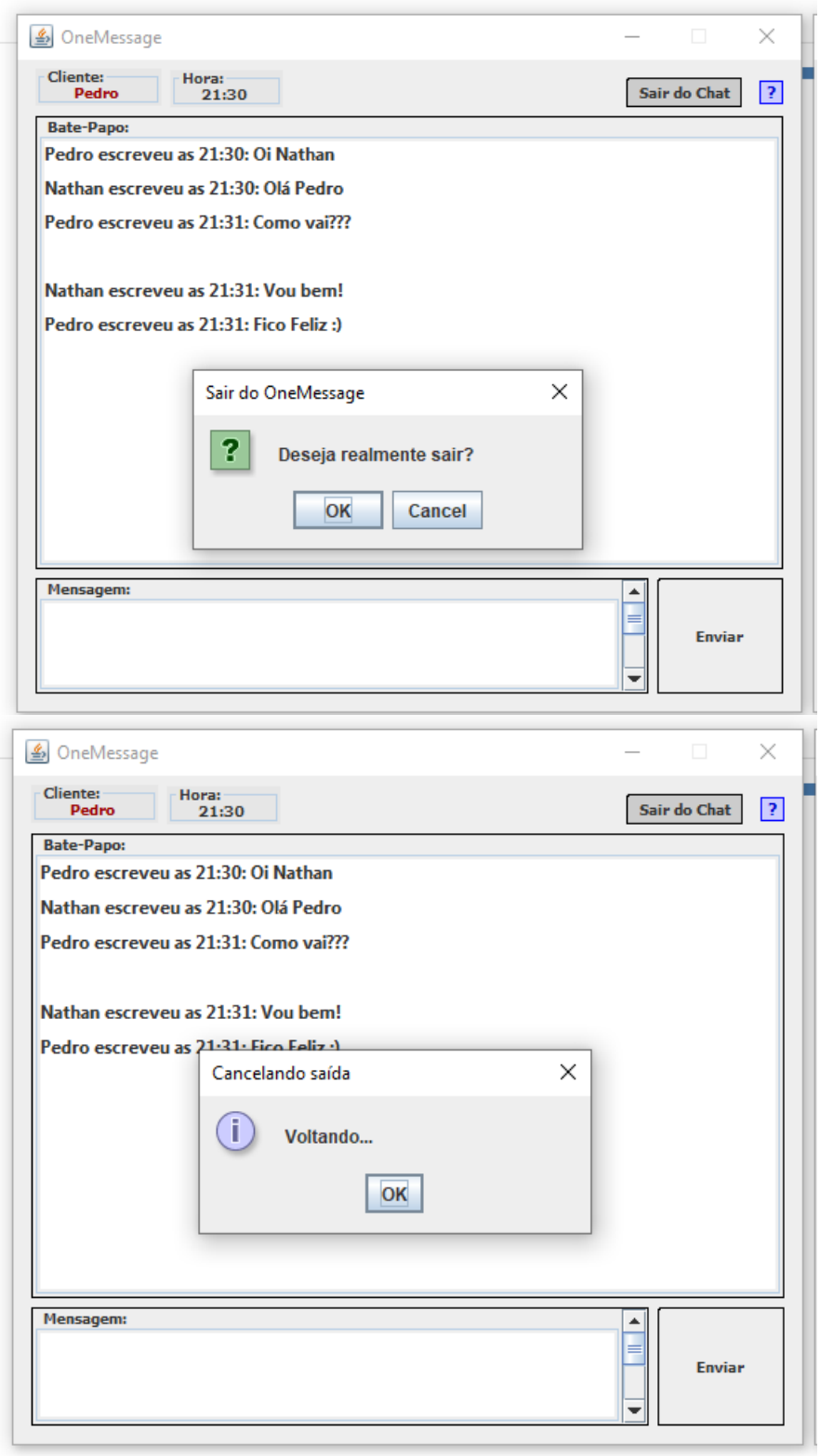
Então, um exemplo de troca de mensagens entre, mas não limitado à, dois usuários. Verifique que ao envio da mensagem, mostra o nome de usuário escolhido e a horário do envio da mensagem.



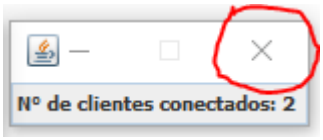




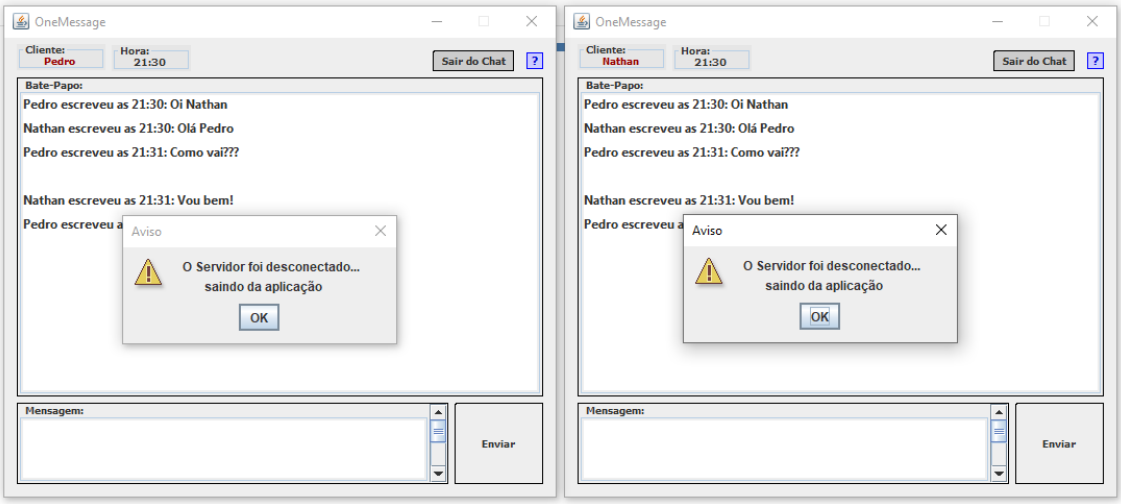
Ao finalizar o programa, temos o tratamento da escolha de saída do usuário pelo clique do botão **“Sair do Chat”.** Se o usuário desejar realmente sair, basta clicar “OK”, caso escolher “*Cancel*”, ele retornará a tela da aplicação, como mostrado nas duas próximas imagens.



Somente na execução do servidor, uma janela aparecerá, mostrando o total de conexões realizadas por ele.



Se houver a necessidade do desligamento, ao clicar no **X,** todas as conexões dos clientes serão encerradas, mostrando um aviso aos usuários ainda conectados.



8 CONSIDERAÇÕES FINAIS

Ao passo que fomos apresentados a linguagem Java nas aulas de ALPOO, podemos evoluir o aprendizado e compreender como produzir aplicações mais complexas e de forma mais correta, com otimizações, uso de sub-rotinas, e tratamentos de erros. Queremos, como inclusão em nossas considerações finais, dizer que, tanto individualmente como coletivamente, o desenvolvimento desse trabalho, com extensa pesquisa, dedicação a mudanças de código, debate de opiniões e divisão de tarefas, nos fez aprender um pouco mais sobre muitas das coisas que estão envolvidas no ramo da computação e desenvolvimento.

Procuramos dizer com toda modéstia que de certa forma, desde a ingressão no ensino superior, cada um aqui representado, pode obter evoluções em suas produções acadêmicas e no que diz respeito a competência de um trabalho em equipe. Queremos deixar, de forma explícita, que em todas as abordagens possíveis nos comprometemos a produzir o melhor de nós mesmos. Ao professor orientador, nossos agradecimentos.

REFERÊNCIAS BIBLIOGRÁFICAS

FERREIRA, Joaquim. " Entenda As Diferenças Entre Os Protocolos TCP E UDP"; Eletronet. Disponível em: https://www.eletronet.com/entenda-as-diferencas-entre-os-protocolos-tcp-e-udp/. Acesso em 28 de out. de 2021.

Interface gráfica do utilizador. Wikipédia: A enciclopédia livre. Disponível em: https://pt.wikipedia.org/wiki/Interface_gr%C3%A1fica_do_utilizador. Acesso em: 30 de out. de 2021.

Java (linguagem de programação). Wikipédia: A enciclopédia livre, 2021. Disponível em: https://pt.wikipedia.org/wiki/Java_(linguagem_de_programa%C3%A7%C3%A3o). Acesso em: 31 de out. de 2021.

Java (*programming language*). Wikipédia: A enciclopédia livre. Disponível em: https://en.wikipedia.org/wiki/Java_(programming_language). Acesso em: 31 de out. de 2021.

MENDES, Antonio. Arquitetura de Software: desenvolvimento orientado para arquitetura. Editora Campus. Rio de Janeiro - RJ, 2002.

PANTUZA, Gustavo. "O que são e como funcionam os sockets"; Blog Pantuza. Disponível em: https://blog.pantuza.com/artigos/o-que-sao-e-como-funcionam-os-sockets. Acesso em 28 de out. de 2021.

PERCíLIA, Eliene. "Comunicação de Dados"; Brasil Escola. Disponível em: https://brasilescola.uol.com.br/informatica/comunicacao-dados.htm. Acesso em 29 de out. de 2021.

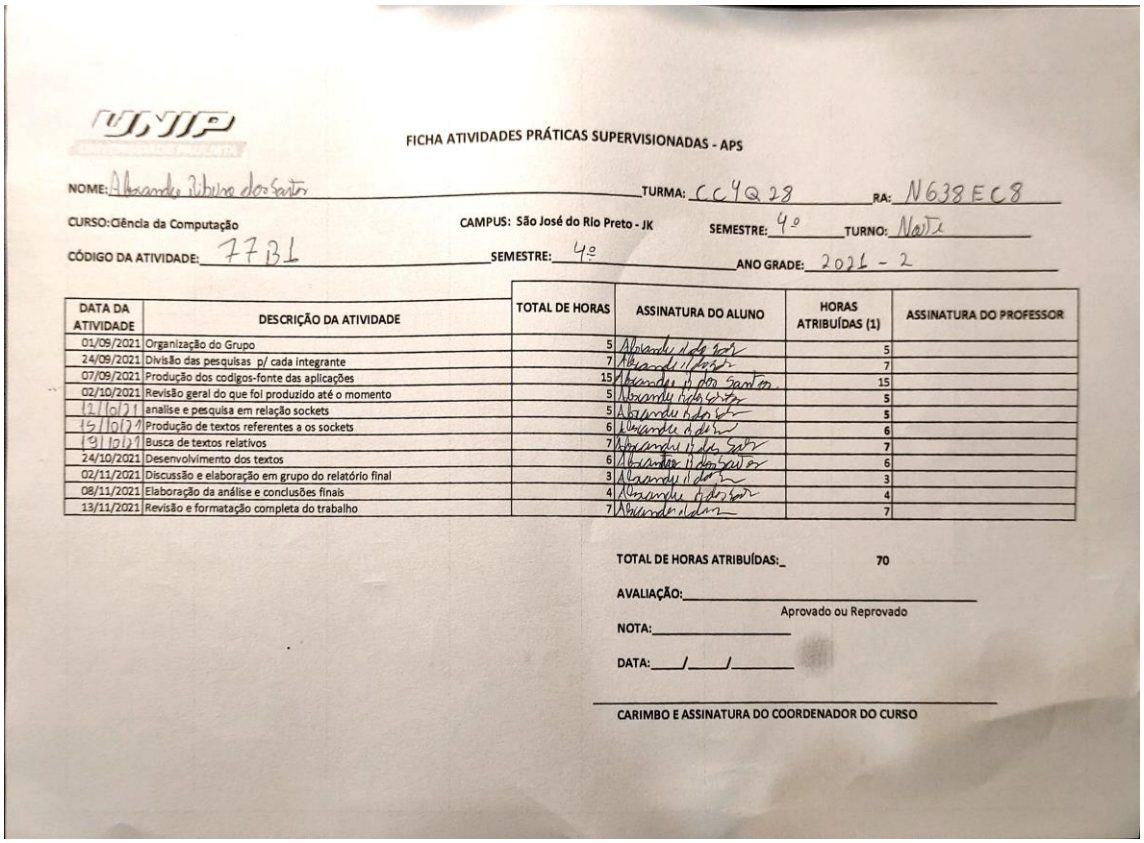
Porto Editora – redes de comunicação de dados na Infopédia [em linha]. Porto: Porto Editora. Disponível em https://www.infopedia.pt/$redes-de-comunicacao-de-dados. Acesso em 27 de out. de 2021.

Projetando uma GUI Swing no NetBeans IDE. Apache NetBeans. Disponível em: https://netbeans.apache.org/kb/docs/java/quickstart-gui_pt_BR.html. Acesso em: 01 de nov. de 2021.

REIS, Fábio. "Curso de Redes – Camadas de Transporte da pilha TCP/IP"; Bóson Treinamentos em Ciência e Tecnologia. Disponível em: http://www.bosontreinamentos.com.br/redes-computadores/curso-de-redes-camada-de-transporte-da-pilha-tcpip/. Acesso em 29 de out. de 2021.

Swing (Java). Wikipédia: A enciclopédia livre. Disponível em: https://en.wikipedia.org/wiki/Swing_(Java). Acesso em: 31 de out. de 2021.

TADESCO, Kennedy. "Uma introdução a TCP, UDP e Sockets"; Treinaweb. Disponível em: https://www.treinaweb.com.br/blog/uma-introducao-a-tcp-udp-e-sockets. Acesso em 29 de out. de 2021.



UNIP

FICHA ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Alexandre Ribeiro dos Santos TURMA: CC4Q28 RA: N638EC8

CURSO: Ciência da Computação CAMPUS: São José do Rio Preto - JK SEMESTRE: 4º TURNO: Norte

CÓDIGO DA ATIVIDADE: 77B1 SEMESTRE: 4º ANO GRADE: 2021 - 2

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
01/09/2021	Organização do Grupo	5		5	
24/09/2021	Divisão das pesquisas p/ cada integrante	7		7	
07/09/2021	Produção dos codigos-fonte das aplicações	15		15	
02/10/2021	Revisão geral do que foi produzido até o momento	5		5	
12/10/21	analise e pesquisa em relação sockets	5		5	
15/10/21	Produção de textos referentes a os sockets	6		6	
19/10/21	Busca de textos relativos	7		7	
24/10/2021	Desenvolvimento dos textos	6		6	
02/11/2021	Discussão e elaboração em grupo do relatório final	3		3	
08/11/2021	Elaboração da análise e conclusões finais	4		4	
13/11/2021	Revisão e formatação completa do trabalho	7		7	

TOTAL DE HORAS ATRIBUÍDAS: 70

AVALIAÇÃO: _____
Aprovado ou Reprovado

NOTA: _____

DATA: ___/___/_____

_____
CARIMBO E ASSINATURA DO COORDENADOR DO CURSO

UNIP
UNIVERSIDADE PAULISTA

FICHA ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: joão victor tasinafo de paula tagliari brandão TURMA: CC3P28 RA: G14EBF0

CURSO: Ciência da Computação CAMPUS: São José do Rio Preto - JK SEMESTRE: 4 TURNO: noturno

CÓDIGO DA ATIVIDADE: 77B1 SEMESTRE: 4 ANO GRADE: 2021

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
9/1/21	Organização do Grupo	5		5	
9/24/21	Divisão das pesquisas p/ cada integrante	7		7	
9/7/21	Produção dos codigos-fonte das aplicações	15		15	
10/2/21	Revisão geral do que foi produzido até o momento	5		5	
9/15/21	Fundamentos da comunicação de dados em rede	8		8	
10/24/21	Desenvolvimento dos textos	6		6	
11/2/21	Discussão e elaboração em grupo do relatório final	3		3	
11/8/21	Elaboração da análise e conclusões finais	4		4	
11/13/21	Revisão e formatação completa do trabalho	7		7	

TOTAL DE HORAS ATRIBUÍDAS: 60

AVALIAÇÃO: _____
Aprovado ou Reprovado

NOTA: _____

DATA: 18 / 11 / 120218

_____
CARIMBO E ASSINATURA DO COORDENADOR DO CURSO

UNIP UNIVERSIDADE PAULISTA

FICHA ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Matheus Satake de Souza **TURMA:** CC4Q28 **RA:** N579137

CURSO:Ciência da Computação **CAMPUS: São José do Rio Preto - JK** **SEMESTRE:** 4º **TURNO:** Noturno

CÓDIGO DA ATIVIDADE: 77B1 **SEMESTRE:** 4º **ANO GRADE:** 2021/2

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
01/09/2021	Organização do Grupo	5	Matheus Satake	3	
24/09/2021	Divisão das pesquisas p/ cada integrante	7	Matheus Satake	5	
07/09/2021	Produção dos codigos-fonte das aplicações	15	Matheus Satake	7	
02/10/2021	Revisão geral do que foi produzido até o momento	5	Matheus Satake	5	
05/10/2021	Pesquisa sobre fundamentos da comunicação de dados em rede	8	Matheus Satake	8	
11/10/2021	Pesquisa sobre modelo cliente-servidor	8	Matheus Satake	8	
17/10/2021	Pesquisa sobre TCP-DP	6	Matheus Satake	6	
24/10/2021	Desenvolvimento dos textos	6	Matheus Satake	6	
02/11/2021	Discussão e elaboração em grupo do relatório final	3	Matheus Satake	3	
08/11/2021	Elaboração da análise e conclusões finais	4	Matheus Satake	3	
13/11/2021	Revisão e formatação completa do trabalho	7	Matheus Satake	6	

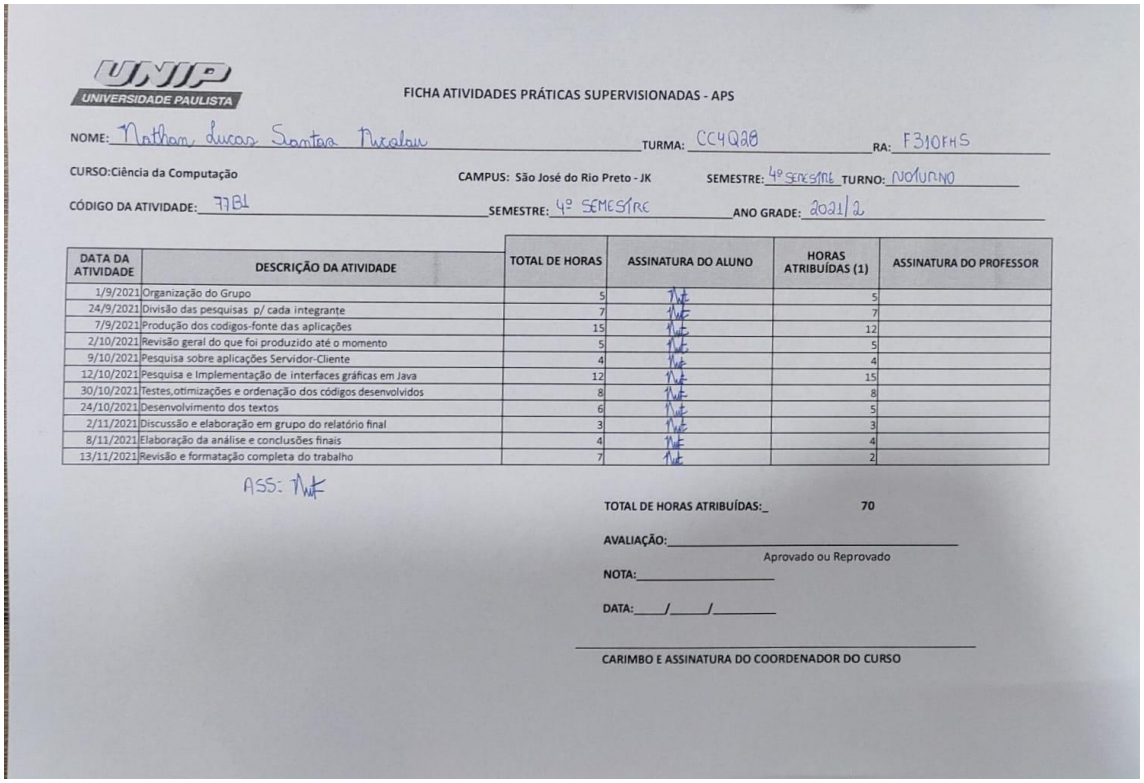
TOTAL DE HORAS ATRIBUÍDAS: 60

AVALIAÇÃO: Aprovado ou Reprovado

NOTA:

DATA: 15 / 11 / 2021

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO



UNIP UNIVERSIDADE PAULISTA

FICHA ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Matheus Lucas Santos Nicolau TURMA: CC4Q28 RA: F310FH5

CURSO:Ciência da Computação CAMPUS: São José do Rio Preto - JK SEMESTRE: 4º SEMESTRE TURNO: NOTURNO

CÓDIGO DA ATIVIDADE: 77B1 SEMESTRE: 4º SEMESTRE ANO GRADE: 2021/2

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
1/9/2021	Organização do Grupo	5		5	
24/9/2021	Divisão das pesquisas p/ cada integrante	7		7	
7/9/2021	Produção dos codigos-fonte das aplicações	15		12	
2/10/2021	Revisão geral do que foi produzido até o momento	5		5	
9/10/2021	Pesquisa sobre aplicações Servidor-Cliente	4		4	
12/10/2021	Pesquisa e Implementação de interfaces gráficas em Java	12		15	
30/10/2021	Testes,otimizações e ordenação dos códigos desenvolvidos	8		8	
24/10/2021	Desenvolvimento dos textos	6		5	
2/11/2021	Discussão e elaboração em grupo do relatório final	3		3	
8/11/2021	Elaboração da análise e conclusões finais	4		4	
13/11/2021	Revisão e formatação completa do trabalho	7		2	

ASS:

TOTAL DE HORAS ATRIBUÍDAS: 70

AVALIAÇÃO: Aprovado ou Reprovado

NOTA:

DATA: ____/____/____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO

UNIP UNIVERSIDADE PAULISTA

FICHA ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Paulo Henrique Reis de Padua TURMA: CC4P28 RA: F20FIF2

CURSO: Ciência da Computação CAMPUS: São José do Rio Preto - JK SEMESTRE: 4º TURNO: Noturno

CÓDIGO DA ATIVIDADE: 77131 SEMESTRE: 4º ANO GRADE: 2021

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
01/09/2021	Organização do Grupo	5	Paulo Henrique R. de Padua	3	
24/09/2021	Divisão das pesquisas p/ cada integrante	7	Paulo Henrique R. de Padua	4	
07/09/2021	Produção dos codigos-fonte das aplicações	15	Paulo Henrique R. de Padua	8	
02/10/2021	Revisão geral do que foi produzido até o momento	5	Paulo Henrique R. de Padua	5	
05/10/2021	Estudo sobre redes de computadores e GUIs	13	Paulo Henrique R. de Padua	13	
20/10/2021	Desenvolvimento do Projeto do Programa	8	Paulo Henrique R. de Padua	8	
10/11/2021	Manutenção do arquivo escrito da APS	10	Paulo Henrique R. de Padua	10	
24/10/2021	Desenvolvimento dos textos	6	Paulo Henrique R. de Padua	5	
02/11/2021	Discussão e elaboração em grupo do relatório final	3	Paulo Henrique R. de Padua	2	
08/11/2021	Elaboração da análise e conclusões finais	4	Paulo Henrique R. de Padua	2	
13/11/2021	Revisão e formatação completa do trabalho	7	Paulo Henrique R. de Padua	7	

TOTAL DE HORAS ATRIBUÍDAS: 67

AVALIAÇÃO: _____
Aprovado ou Reprovado

NOTA: _____

DATA: 17 / 11 / 2021

_____
CARIMBO E ASSINATURA DO COORDENADOR DO CURSO