

**UNIVERSIDADE PAULISTA
CIÊNCIA DA COMPUTAÇÃO**

DIOGO RAMOS LOPES DA SILVEIRA – C173398

RICARDO FERNANDES SOUTO – C13CHI5

WANDERSON MARTINS OLIVEIRA – C202754

YURY RODRIGUES ANUNCIAÇÃO – C203360

DESENVOLVIMENTO DE UMA FERRAMENTA PARA COMUNICAÇÃO EM REDE

**SÃO PAULO
2016**

SUMÁRIO

1 OBJETIVOS.....	2
1.1 Geral.....	2
1.2 Específicos.....	2
2 INTRODUÇÃO.....	3
3 FUNDAMENTOS DA COMUNICAÇÃO DE DADOS EM REDE.....	6
3.1 Hardware de rede.....	6
3.1.1 Classificação das redes.....	6
3.1.2 Topologias físicas.....	7
3.1.3 Meios de transmissão.....	7
3.1.4 Tecnologias de transmissão.....	7
3.2 Software de rede.....	8
3.2.1 Modos de transmissão.....	8
3.2.2 Tipos de serviços.....	9
3.2.3 Qualidade dos serviços.....	9
3.3 Modelos de referência.....	10
3.3.1 OSI.....	10
3.3.2 TCP/IP.....	11
4 PLANO DE DESENVOLVIMENTO DA APLICAÇÃO.....	13
4.1 Linguagem de programação.....	13
4.2 Ambiente de desenvolvimento.....	14
4.3 Interface gráfica.....	15
4.4 HtmlCleaner.....	16
4.5 Sockets.....	16
4.6 Emojis.....	17
4.7 Logomarca.....	18
5 PROJETO DO PROGRAMA.....	19
5.1 Servidor.....	19
5.1.1 ServidorGUI.....	19
5.1.2 Servidor.....	20
5.1.3 EscutaCliente.....	21
5.1.4 Mensagem.....	22
5.2 Cliente.....	22

5.2.1 ClienteGUI.....	23
5.2.2 Cliente.....	25
5.2.3 RecebeMensagem.....	25
5.2.4 MensagemParaEnviar.....	25
5.2.5 ExibeMensagem.....	26
5.2.6 CaixaMensagem.....	26
5.2.7 AlterarServidorGUI e SobreGUI.....	26
6 RELATÓRIO COM AS LINHAS DE CÓDIGO DO PROGRAMA.....	27
7 APRESENTAÇÃO DO PROGRAMA EM FUNCIONAMENTO EM UM COMPUTADOR.....	37
REFERÊNCIAS.....	49

1 OBJETIVOS

1.1 Geral

Desenvolver, utilizando a linguagem de programação Java, um chat com interface gráfica, para comunicação em rede, entre duas ou mais pessoas, através do protocolo TCP/IP.

Tendo como plano de fundo a troca online de informações de equipes de inspetores, espalhadas pelas indústrias do estado de São Paulo, com a Secretaria do Meio Ambiente.

1.2 Específicos

São objetivos específicos deste trabalho:

- Pesquisar e dissertar sobre os conceitos gerais da comunicação de dados em rede;
- Definir o modelo de chat que será desenvolvido;
- Definir a estrutura do projeto;
- Criar a interface gráfica;
- Implementar o chat utilizando a linguagem de programação Java.

2 INTRODUÇÃO

A informática foi uma das áreas da tecnologia que tiveram os maiores avanços durante as últimas décadas. Junto a ela, as redes de computadores cresceram em uma proporção incrível, cuja parte mais visível e imprescindível na vida cotidiana, empresarial, e governamental, é a internet.

A comunicação sempre foi um fator de extrema importância, desempenhando papel fundamental na sobrevivência e evolução das diversas sociedades que nasceram desde o surgimento do homem. Sem ela, o próprio conceito de sociedade seria impossível.

No início as informações eram transmitidas de forma oral e pessoal, que além de ser um método demorado para levar a informação, através da movimentação dos mensageiros, é extremamente inseguro, seja pelo perigo do sequestro dos mensageiros ou ainda pela adulteração das informações transmitidas, de maneira intencional ou não(como o esquecimento de determinados fatos).

A partir do desenvolvimento da escrita, e sua utilização para transmissão de informações, ocorre uma maior garantia da integridade da mensagem transmitida, além da comunicação ser melhor desenvolvida em termos de conteúdo e de quantidade de pessoas alcançadas. Seja, por exemplo, através dos atuais jornais e revistas ou das trocas de cartas privadas.

A escrita ainda se mantém, indiscutivelmente, como uma das maiores e mais utilizadas formas de comunicação, mesmo após milênios de seu advento. Embora a criação dos meios de comunicação sonoros(como o rádio e o telefone) e audiovisuais(televisão e cinema) trouxeram uma revolução nas suas respectivas épocas de difusão, a maior revolução sofrida na comunicação foi sem dúvida, ocasionada pela internet.

Mesmo que o telefone tenha levado informações em tempos e distâncias anteriormente inimagináveis, foi o advento da internet que transformou o mundo como um todo, eliminando barreiras físicas como nenhuma outra, fazendo do planeta uma única aldeia. Comunicação instantânea entre um japonês e um inglês não fica mais restrito aos contos de ficção mais imaginativos de nossas décadas passadas, fica a apenas um clique do mouse ou um toque na tela.

O surgimento da internet só foi possível graças a evolução que os computadores, e junto com eles as redes de computadores, tiveram durante as

décadas anteriores. As redes de computadores podem ser descritas como dois ou mais computadores interligados(seja por fio, infravermelho, bluetooth, ondas de rádio, satélite, etc), que podem ou não trocar informações e compartilhar recursos(como arquivos e impressoras) entre si.

Os computadores, as suas redes, e a internet, são hoje as tecnologias imprescindíveis da vida moderna. E isto é ainda mais importante para empresas, instituições e governos. Quanto mais moderna tecnologicamente for sua estrutura, de um modo geral, mais a frente da concorrência estará.

Com um clique as organizações podem obter informações sobre suas filias espalhadas pelo mundo. Trocar relatórios de estoque, de produção, de pesquisas, dados sobre clientes, realizar reuniões a partir de qualquer localidade, resumidamente, ter acesso instantaneamente a todos os dados que são necessários para manter a organização funcionando e expandido.

Com o advento da internet surgiram diversos aplicativos para a realização de comunicação instantânea. Dentre eles, alguns se destacaram e conseguiram ser disseminados entre os usuários da rede mundial, e durante algum tempo reinaram como os maiores e mais utilizados programas de comunicação.

No início da década de 90 os principais eram as salas de bate-papo, logo em seguida surgiu o mIRC(1995), o ICQ(1996), o Yahoo! Messenger(1998), o MSN(1999), um dos mais famosos de todos, o Skype em 2003, o Facebook(2008), e finalmente o WhatsApp(2011), que até o momento está reinando como o principal aplicativo de comunicação.

Se a comunicação instantânea é importante para a população em geral, para as organizações é imprescindível. Gerenciar equipes e coordenar trabalhos remotamente não é algo a mais, é essencial nesta era. Não utilizar os mecanismos disponíveis para tal, é o mesmo que utilizar pá enquanto a escavadeira está a uma mão de distância. Algo completamente ilógico.

Neste sentido, a atitude da Secretaria do Meio Ambiente, de adotar uma ferramente de comunicação em rede para coordenar suas equipes de inspetores espalhadas pelo estado de São Paulo, é a melhor ação a ser tomada, seja para tornar o serviço mais rápido, aumentar a produtividade, diminuir os custos do serviço, gerenciar melhor os colaboradores, e principalmente, melhorar a qualidade do serviço prestado.

A tecnologia é a maior aliada do desenvolvimento sustentável, principalmente

no âmbito dos serviços públicos, garantindo uma administração pública eficiente, rápida em executar serviços, com menor desperdício de dinheiro público, e acima de tudo, focada na resolução dos problemas da sociedade e no melhoramento da qualidade de vida da população. No caso da Secretaria do Meio Ambiente, através da despoluição de um dos principais rios do estado de São Paulo, o Rio Tietê.

Baseando-se neste cenário, este trabalho terá como um de seus objetivos desenvolver uma ferramenta para comunicação em redes, utilizando a linguagem de programação Java e o protocolo TCP/IP. Sendo que esta aplicação poderá ser utilizada na troca de informações entre os inspetores e a Secretaria do Meio Ambiente do Estado de São Paulo.

3 FUNDAMENTOS DA COMUNICAÇÃO DE DADOS EM REDE

A comunicação de dados em rede é feita por uma série de dispositivos e processos que propiciam a troca de informações entre os diversos dispositivos que estão conectados.

Deve-se deixar claro que toda e qualquer informação pode ser transmitida em uma rede por meio de sinais eletromagnéticos enviados por meios físicos ou não.

3.1 Hardware de rede

As redes de computadores podem ser descritas como dois ou mais computadores interligados(seja por fio, infravermelho, bluetooth, ondas de rádio, satélite, etc), que podem ou não trocar informações e compartilhar recursos(como arquivos e impressoras) entre si.

Uma rede é composta por **hosts** e **sub-rede**. **Host** é qualquer máquina, geralmente um computador, conectada a uma determinada rede. Já a **sub-rede**, é uma subdivisão da rede principal. Ela contém os roteadores e os meios de comunicação(cabos, por exemplo) que interligam as máquinas daquela rede. A operação de uma sub-rede fica, geralmente, sob responsabilidade da empresa operadora da rede, como por exemplo, empresas que fornecem acesso à internet.

Os **roteadores** são computadores especializados em intermediar a comunicação entre os dispositivos da rede, recebendo dados por uma determinada entrada e os encaminhando por uma determinada saída, por onde os dados prosseguirão em direção ao dispositivo destinatário.

3.1.1 Classificação das redes

Uma rede pode ser classificada em três principais grupos, utilizando-se como critério de classificação a sua abrangência geográfica.

- **LAN** (Local Area Network): é a classificação dada a uma rede de abrangência local, que permite alta velocidade de comunicação entre dispositivos próximos;
- **MAN** (Metropolitan Area Networks): são redes de abrangência metropolitana, que tem a função de interligar as diversas LAN's em uma determinada região;

- **WAN** (Wide Area Network): nestes grupos estão as redes de abrangência ampla, que interligam diversas MAN's, podendo abranger continentes.

3.1.2 Topologias físicas

As topologias físicas referem-se a maneira como os diversos dispositivos de uma rede estão interligados. Abaixo segue alguns de seus principais tipos.

- **Topologia de barramento**: uma única barra, cabo, percorre todos os hosts da rede conectando-os. Por isto, enquanto um computador estiver transmitindo, nenhum outro computador pode transmitir;
- **Topologia em anel**: os hosts são conectados em um percurso fechado, formando um anel. Pelo seu formato circular, os dados podem ser transmitidos em qualquer sentido, passando por cada computador da rede até atingir o destinatário;
- **Topologia em estrela**: todos as máquinas da rede estão conectados a um mesmo dispositivo central, não havendo interação direta entre elas;
- **Topologia em árvore**: de forma simples, seria a união de diversas redes de topologia estrela, ligadas através de seus dispositivos centrais.

3.1.3 Meios de transmissão

As informações na rede podem ser transmitidas por basicamente dois meios:

- **Guiado**: através de cabos e conectores. Alguns exemplos são o par trançado, o cabo coaxial e a fibra óptica;
- **Não guiado**: os dados são transmitidos sem a utilização de fios, sendo propagados pela atmosfera. Isto pode ser feito, por exemplo, através de bluetooth, infravermelho ou as ondas de rádio.

3.1.4 Tecnologias de transmissão

As duas principais tecnologias de transmissão de dados dentro de uma rede são a **broadcasting** e a **peer to peer**.

- **Links de difusão (broadcasting)**: nesta tecnologia, a mensagem é enviada

não somente para o destinatário, e sim para todos os hosts da rede. A máquina de destino recebe a mensagem e executa alguma ação, enquanto que as demais ignoram a mensagem recebida;

- **Links ponto a ponto (peer to peer):** ao contrário da *broadcasting*, na *peer to peer* a mensagem é transmitida somente para o host destinatário.

3.2 Software de rede

Uma rede é formada não somente pela parte física, mas também pela parte não física, o *software*. O *software* de uma rede é altamente estruturado. Para reduzir a complexidade do seu projeto, passou-se a organizar as redes como uma pilha de camadas, colocadas umas sobre as outras, onde cada camada tem por função fornecer serviços à camada superior, em uma hierarquia.

Cada camada da máquina transmissora somente se comunica com a sua camada correspondente na máquina receptora. Isto se deve ao fato de que cada camada contém protocolos diferentes.

Desta forma, a arquitetura de uma rede é o conjunto de camadas e protocolos que a constituem.

Protocolo – Conjunto de regras que define os modos de comunicação entre camadas equivalentes em hosts diferentes. Seriam, basicamente, a “linguagem” utilizada na comunicação entre os computadores.

Interface – Define os serviços que uma camada inferior fornece à superior.

3.2.1 Modos de transmissão

As transmissões de dados dentro de uma rede, podem ser de três modos:

- **Simplex:** a comunicação é realizada de forma unidirecional. Somente um dispositivo transmite as informações;
- **Half-duplex:** comunicação bidirecional não-simultânea. Os dois dispositivos podem transmitir, mas, enquanto um transmite dados o outro não pode transmitir, apenas receber;
- **Full-duplex:** comunicação bidirecional simultânea. Os dispositivos podem transmitir e receber dados simultaneamente.

3.2.2 Tipos de serviços

- **Orientados a conexões:** Neste tipo de serviço, tanto a máquina transmissora quanto a máquina receptora enviam pacotes uma para a outra antes da transmissão dos dados reais ser iniciada. Desta forma, prepara a máquina destinatária para o recebimento das mensagens. Além de possibilitar que seja verificado se todos os pacotes enviados chegaram corretamente na máquina destinatária, podendo pedir o reenvio caso não tenha chegado.
 - A transmissão dos dados é realizada de forma sequencial, similarmente a um “tubo”, onde os dados saem de uma extremidade e chegam na outra na mesma ordem em que entraram.
 - O principal protocolo orientado a conexão é o TCP(Transmission Control Protocol).
- **Sem conexão:** Neste tipo de serviço não existe uma comunicação inicial entre as máquinas transmissora e receptora. O transmissor simplesmente envia os pacotes que deseja. Isto aumenta a velocidade de transmissão, no entanto, não permite a confirmação de entrega dos pacotes.
 - O UDP(User Datagram Protocol) é o principal protocolo não orientado a conexão.

3.2.3 Qualidade dos serviços

A qualidade de um serviço é algo extremamente importante na comunicação de dados em rede, podendo ser de dois tipos:

- **Serviços confiáveis:** Sua principal característica é que este tipo de serviço garante total integridade dos dados, assim se sabe que eles nunca serão perdidos numa comunicação. Para que isso ocorra o receptor sempre envia uma confirmação para o transmissor para cada pacote de bit enviado, caso isso não ocorra o remetente reenvia o pacote perdido. Este tipo de serviço é amplamente utilizado para a transmissão de dados em geral;
- **Serviços não-confiáveis:** Diferentemente dos serviços confiáveis, o receptor não envia a confirmação de recebimento do pacote. Este serviço é recomendado onde é mais importante os dados chegarem ao destino do que

sua integridade, como, por exemplo, comunicação em vídeo em tempo real.

3.3 Modelos de referência

3.3.1 OSI

Com o objetivo de facilitar a interconexão de computadores em rede, foi desenvolvido o modelo de referência OSI(Open Systems Interconnection). Os protocolos criados pelas fabricantes seriam baseados neste modelo. Permitindo, assim, que computadores de diferentes fabricantes, que utilizem este modelo, pudessem trocar informações.

O modelo OSI descreve como o *hardware* e o *software* podem ser organizados para funcionarem na rede. Isto é feito através de 7 camadas:

1. **Física:** é a camada que define os meios físicos para a transmissão de informações, como os cabos de cobre, cabo de fibra óptica, conectores, etc. Além de pegar os dados vindos da camada superior, de **Enlace**, e os converter nos sinais que serão transmitidos pela rede. Por exemplo, se o meio for elétrico, converte em sinais elétricos, se óptico converte em sinais luminosos. Fazendo o processo inverso com os sinais recebidos, transformando eles em 0s e 1s que serão enviados para a camada de **Enlace**.
2. **Enlace:** esta camada tem por função principal transformar os dados em quadros para a transmissão na rede. Ela também contem meios de reconhecer, e, opcionalmente, corrigir erros de transmissão.
3. **Rede:** tem como responsabilidade o endereçamento dos pacotes. Além das decisões de roteamento, controle de congestionamento e definição da rota mais adequada. É onde encontramos o protocolo IP.
4. **Transporte:** fornece serviços como: estabelecimento e fechamento de conexão, controle de fluxo de dados, recuperação de erros, etc. Nesta camada temos protocolos como o TCP, UDP e SCTP.
5. **Sessão:** estabelece e mantém uma sessão de comunicação.
6. **Apresentação:** camada onde são realizadas as conversões dos dados. Como a conversão de sintaxe, a criptografia e a compressão.
7. **Aplicação:** camada que permite que os programas se comuniquem na rede,

fazendo a intermediação entre o programa e a pilha de protocolos. Um programa de e-mail, por exemplo, se comunicará com esta camada para enviar ou receber e-mails. Exemplos de protocolos nesta camada são o HTTP, HTTPS, FTP, SMTP e POP3.

3.3.2 TCP/IP

O TCP/IP(Transmisson Control Protocol/Internet Protocol) é o modelo mais utilizado atualmente, tendo sido projetado com 4 camadas:

1. **Interface de rede(acesso à rede):** a camada que envia os dados pelo meio de transmissão. Engloba as camadas *Física* e de *Enlace* do modelo OSI.
2. **Inter-Rede(Internet):** tem as mesmas responsabilidades que a camada *Rede* do modelo OSI. Contém o endereçamento dos pacotes, decisões de roteamento, controle de congestionamento e definição da rota mais adequada. É onde encontramos o protocolo IP.
3. **Transporte:** assim como a camada de *Transporte* do modelo OSI, fornece serviços como: estabelecimento e fechamento de conexão, controle de fluxo de dados, recuperação de erros, etc. Nesta camada temos protocolos como o TCP, UDP e SCTP.
4. **Aplicação:** engloba as três últimas camadas do modelo OSI(*Aplicação*, *Apresentação* e *Sessão*). Todos os programas que desejam enviar ou receber dados através da rede se comunicam com esta camada. Exemplos de protocolos nesta camada são o HTTP, HTTPS, FTP, SMTP e POP3.

Na Figura 1 estão mostradas as camadas do modelo OSI e do modelo TCP/IP.

Quando um programa deseja se comunicar na rede ele se comunica com a camada de *Aplicação*, que envia os dados para camada abaixo dela, e esta por sua vez para a sua camada inferior, até o dado chegar a última camada(*Física* ou *Interface de rede*), que enviará os dados pela rede até a máquina destinatária.

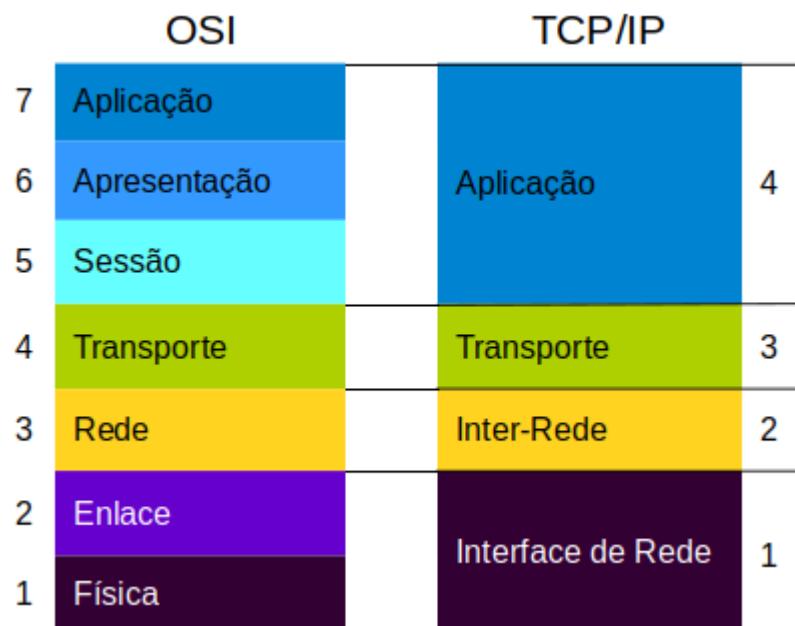
Conforme os dados vão passando pelas camadas, cada camada adiciona as informações que considera relevantes para a comunicação. Somente a camada equivalente na máquina destinatária conseguirá ler estas modificações.

O processo inverso é realizado quando um dado chega na máquina, começando pela camada inferior(*Física* ou *Interface de rede*) ele sobe até chegar a

camada de *Aplicação* que enviará os dados para o programa destinatário. Sendo que conforme passa pelas camadas, cada camada lê as informações gravadas pela camada equivalente na máquina remetente, remove estas informações do dado e o envia para a camada superior.

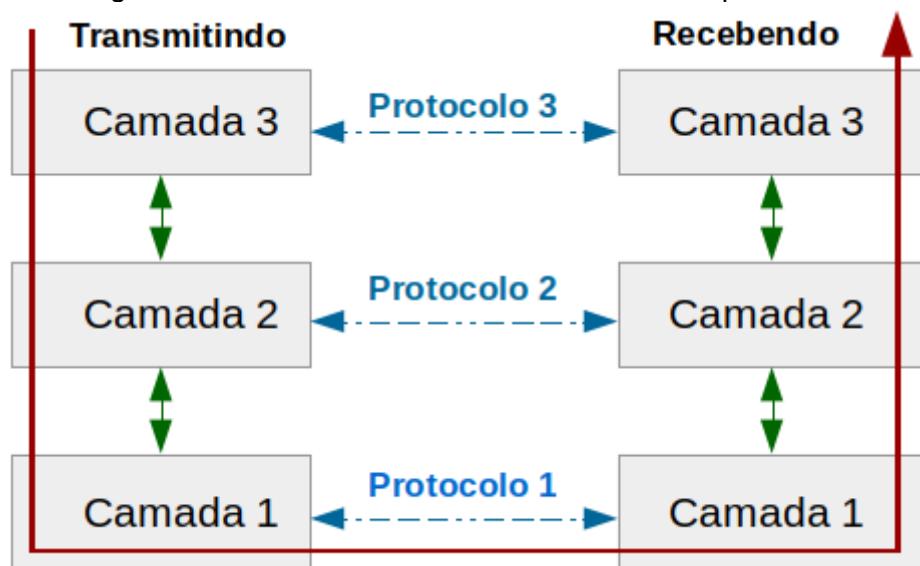
Na Figura 2 está exemplificado o envio de dados entre dois computadores. Lembrando que os dados são transmitidos, de uma máquina para outra, através do meio de transmissão, seja ele guiado ou não.

Figura 1 – Camadas dos modelos OSI e TCP/IP



Fonte: Elaborada pelos autores

Figura 2 – Transmissão de dados entre dois computadores



Fonte: Elaborada pelos autores

4 PLANO DE DESENVOLVIMENTO DA APLICAÇÃO

A primeira etapa do projeto de desenvolvimento de uma aplicação é definir o que será feito. No caso deste trabalho, será desenvolvida uma ferramenta para comunicação em rede, entre duas ou mais pessoas, utilizando o protocolo TCP/IP.

Esta aplicação deverá ter uma interface gráfica através da qual os usuários se comunicarão. Devendo ser o mais simples e direta possível, deixando de lado quaisquer elementos que possam tirar o foco do objetivo do chat, que é a comunicação rápida entre seus usuários.

As mensagens enviadas e recebidas serão única e exclusivamente através de textos, que podem ou não ter emojis. Estes emojis terão representação gráfica, cada qual com sua imagem correspondente, seja ela o rosto de um bonequinho expressando uma emoção, ou algum objeto qualquer.

Além disto, o chat permitirá conversas privadas entre dois usuários. Trazendo segurança e privacidade para a discussão de assuntos sigilosos, característica essencial, principalmente, pelo fato de que a ferramenta poderá ser utilizada em um órgão governamental, onde todos os dados devem estar protegidos.

A aplicação será formada por dois programas, o *servidor*, que receberá e aceitará ou não as conexões dos usuários, além de transmitir as mensagens(recebidas de um cliente) para o usuário destinatário(se for privada) ou para todos os usuários(se for pública), e o *cliente*, que conectará ao servidor e enviará e receberá mensagens dos demais usuários do chat.

Outra característica da aplicação será o fato de que o programa servidor permitirá a troca do número da porta utilizada no estabelecimento das conexões com os usuários. E o programa cliente, por sua vez, terá a opção de trocar o endereço IP e a porta do servidor com o qual se conectará.

Estes serão os principais recursos implementados na aplicação.

4.1 Linguagem de programação

O chat será desenvolvido utilizando a linguagem de programação Java e o paradigma da Orientação a Objetos. Sendo construído com foco na **JRE versão 1.8**.

A escolha desta linguagem se deu pelo fato de ser multiplataforma, podendo os programas criados nela rodarem em qualquer sistema operacional que tenha a

máquina virtual do Java instalada. Uma característica imprescindível, pois, geralmente, o sistema utilizado em organizações governamentais é o Linux, principalmente em servidores, onde será instalado, provavelmente, o programa *servidor* do chat.

Os usuários do chat têm a mesma liberdade de escolherem o sistema operacional que usarão. O mais utilizado pela população é o Windows, sendo assim, pode-se supor que este será o utilizado pela grande maioria deles.

De qualquer forma, não importa em qual sistema rodará a aplicação, o programa *servidor* e o programa *cliente* serão multiplataforma, rodando tanto em Linux, Windows e MacOS, não importando a versão do sistema operacional utilizada.

4.2 Ambiente de desenvolvimento

O IDE(Integrated Development Environment, ou, em português, Ambiente de Desenvolvimento Integrado) utilizado no desenvolvimento da aplicação foi o **Netbeans**. Um IDE gratuito, de código aberto e que pode ser utilizado não somente para o desenvolvimento de programas na linguagem Java, mas também em C, C++, PHP, Ruby e muitas outras. Sendo construído utilizando a linguagem Java, pode ser executado tanto em Windows, Linux, Solaris e MacOS, em sistemas operacionais 32bits e 64bits.

Outro IDE empregado neste projeto foi o **Eclipse**. Tendo sido utilizado na geração dos arquivos executáveis do chat, pois realiza esta tarefa de forma mais rápida e simples do que o Netbeans.

Para controlar as versões do chat que foram sendo desenvolvidas, foi empregado o GIT. Um sistema de controle de versões de arquivos, que permite que diversas pessoas possam simultaneamente editar, criar e excluir arquivos do projeto, sem que as alterações de uma pessoa sobrescrevam as alterações de outra.

Desta forma, tem-se a segurança de que as modificações de um membro da equipe não prejudique as funcionalidades já desenvolvidas ou em desenvolvimento por outros membros. Sendo que todas, ou algumas, das modificações criadas, podem ser unidas em um só projeto quando for necessário. Permitindo assim, um controle total sobre as alterações realizadas no projeto, trazendo maior segurança, velocidade e produtividade no desenvolvimento da aplicação.

O Netbeans já vem por padrão com um plugin para o GIT. Já no Eclipse foi

necessário instalar o plugin **Egit** para poder utilizar este sistema de controle de versões.

Para a criação dos arquivos UML e diagramas do projeto, foi instalado o *plugin ObjectAid* no Eclipse. Todos os arquivos destes tipos, apresentados neste trabalho, foram criados através deste *plugin*.

Outro programa empregado no desenvolvimento do projeto, foi o **GIMP**. Um editor de imagens, gratuito, *open source* e multiplataforma, que contém a grande maioria das funcionalidades e características do Photoshop. Sendo utilizado exclusivamente para a criação da logomarca do chat.

Na Figura 3 está as logomarcas dos programas e *plugins* utilizados.

Figura 3 – Logomarcas dos programas e *plugins* utilizados



Fonte: Das respectivas empresas e projetos.

4.3 Interface gráfica

Para o desenvolvimento da interface gráfico do chat, foi utilizada a biblioteca gráfica **Swing**. Esta biblioteca vem por padrão em todas as JRE, desde a versão 1.2 do Java. A principal vantagem dela vem do fato de ser multiplataforma. Sendo assim, uma vez que a interface do chat foi desenvolvida, ela terá os mesmos componentes,

mesmas cores, mesmos tamanhos de botões e textos, exatamente a mesma interface, não importando se o sistema operacional onde está sendo executada é o Windows, o MacOS ou Linux. Algo essencial neste projeto.

4.4 HtmlCleaner

A mensagem de texto enviada pelo usuário e as mensagens recebidas que serão exibidas na tela, são formatadas através de códigos HTML. Para manipular corretamente o código da mensagem, principalmente para adicionar os emojis na posição correta(por exemplo, no meio de uma palavra), será utilizada a biblioteca **HtmlCleaner**.

Com esta biblioteca escrita em Java, é possível formatar, ordenar, inserir, remover, encontrar, substituir e adicionar *tags* junto com seus atributos, entre tantas outras funcionalidades disponíveis para manipular códigos HTML. Muito simples e leve, mas poderosa no que se destina a fazer.

Disponibilizada sob a licença BSD, permite a livre utilização, modificação e distribuição.

4.5 Sockets

Para que os programas possam se comunicar na rede, serão utilizados sockets.

A interface de sockets foi criada pela Universidade da Califórnia em Berkeley, para o sistema operacional Unix BSD(Berkeley Software Distribution), com o objetivo de fornecer suporte a comunicação em rede. Ficando conhecida como interface de sockets de Berkeley ou sockets de Berkeley.

Os sockets podem ser definidos como os pontos finais de uma comunicação entre dois programas em uma rede de computadores. Para que dois computadores se comuniquem, é preciso que cada um deles crie um socket. Sendo as máquinas classificados em dois tipos: servidor e cliente.

O servidor é o computador onde o programa cria um socket e fica esperando pelas conexões, com o objetivo de fornecer algum tipo de serviço ao cliente. Sendo o cliente a máquina onde o programa cria um socket e envia o pedido de conexão ao servidor, para receber algum serviço.

Para efetuar a comunicação, é preciso saber o endereço de socket do programa com o qual deseja se comunicar. Este endereço é único na rede, sendo a combinação do IP da máquina e o número da porta com a qual o programa enviará e receberá mensagens.

Através deste mecanismo é possível identificar um programa na rede. Mesmo que mais de uma instância do programa esteja rodando na mesma máquina, dois chats, por exemplo, cada um deles terá seu próprio endereço de socket na rede, pois mesmo que tenham o mesmo IP eles terão portas diferentes.

Para que o programa servidor possa se comunicar com diversos clientes ao mesmo tempo, assim que aceita uma conexão o servidor redireciona o cliente para outra porta, deixando a porta inicial livre para receber novas conexões. Este mecanismo é implementado no Java através de *threads*.

O Java fornece classes para trabalhar com comunicação em rede. Sendo divididas em cliente e servidor. No caso dos sockets, elas são, respectivamente, **Socket** e **ServerSocket**, que fazem parte do pacote java.net.

4.6 Emojis

Uma das partes essenciais da comunicação por texto hoje em dia são os emojis. Emoções em forma de símbolos, que deixaram muito mais expressiva e divertida a troca de mensagens.

Criados e popularizados no Japão, teve a Apple como principal disseminadora fora do país asiático. Foi a partir do iOS 4, sistema operacional do iPhone, que os emojis foram disponibilizados para todos os usuários, independente do país. Logo após isto, o Google adotou os mesmos no Android e a Microsoft no Windows Phone.

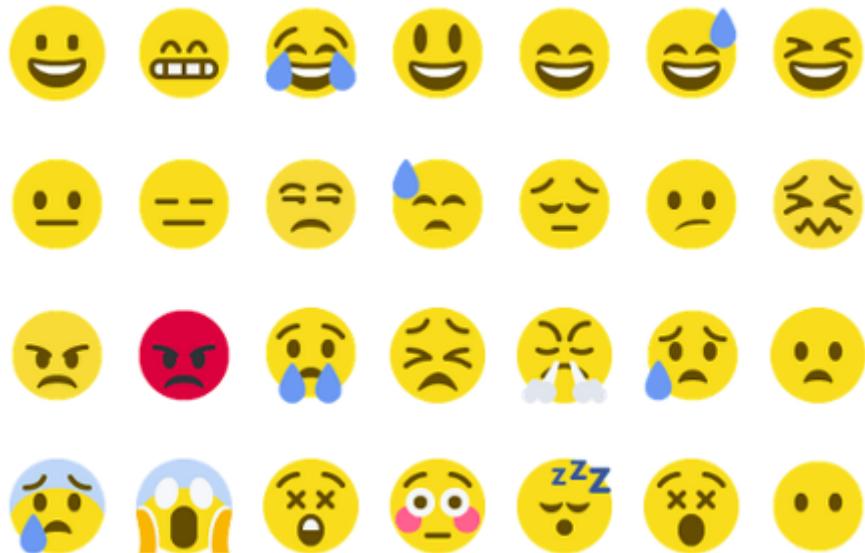
Um recurso muito útil e agradável, que também será adicionado ao sistema de mensagens do chat desenvolvido neste projeto. Porém, apenas alguns emojis, os principais, estarão disponíveis, cerca de 60.

Existem diversos conjuntos de emojis, desde os proprietários até os de uso livre. Neste projeto será utilizado o conjunto desenvolvido pelo Twitter, que é *open source* e distribuído através da licença CC-BY 4.0. Que permite a livre utilização e adaptação, mesmo para fins comerciais, contanto que o crédito pela criação original seja atribuído ao respectivo autor.

A Figura 4 mostra a representação gráfica de alguns emojis do pacote

Twemoji.

Figura 4 – Alguns dos emojis do pacote Twemoji



Fonte: Twitter Emoji (Twemoji)

4.7 Logomarca

Foi criada uma logomarca extremamente simples para o chat. Basicamente, a sigla da Secretaria do Meio Ambiente do Estado de São Paulo, **SMASP**, dentro de um balão de conversa. Como visto na Figura 5.

Figura 5 – Logomarca do chat



Fonte: Elaborada pelos autores

5 PROJETO DO PROGRAMA

Os arquivos da aplicação foram divididos em 3 pacotes: **service**, **view** e **img**. O pacote **img** contém a logomarca do chat e um subpacote chamado **emojis2**, que armazena as imagens dos emojis utilizados nas trocas de mensagens.

Já o pacote **view** contém todas as classes referentes a interface gráfica e a exibição de mensagens. Por sua vez, no pacote **service** estão as classes responsáveis por estabelecer as conexões entre os clientes e o servidor, além de enviarem e receberem as mensagens do chat. Nestes dois pacotes estão distribuídas todas as classes do programa servidor e do programa cliente.

5.1 Servidor

O programa servidor é formado por 4 classes: ***ServidorGUI***, ***Servidor***, ***EscutaCliente*** e ***Mensagem***.

5.1.1 ServidorGUI

Esta classe, pertencente ao pacote **view**, é a principal do programa servidor. Toda vez que o programa for executado, será invocada esta classe, que iniciará o servidor e mostrará sua interface gráfica.

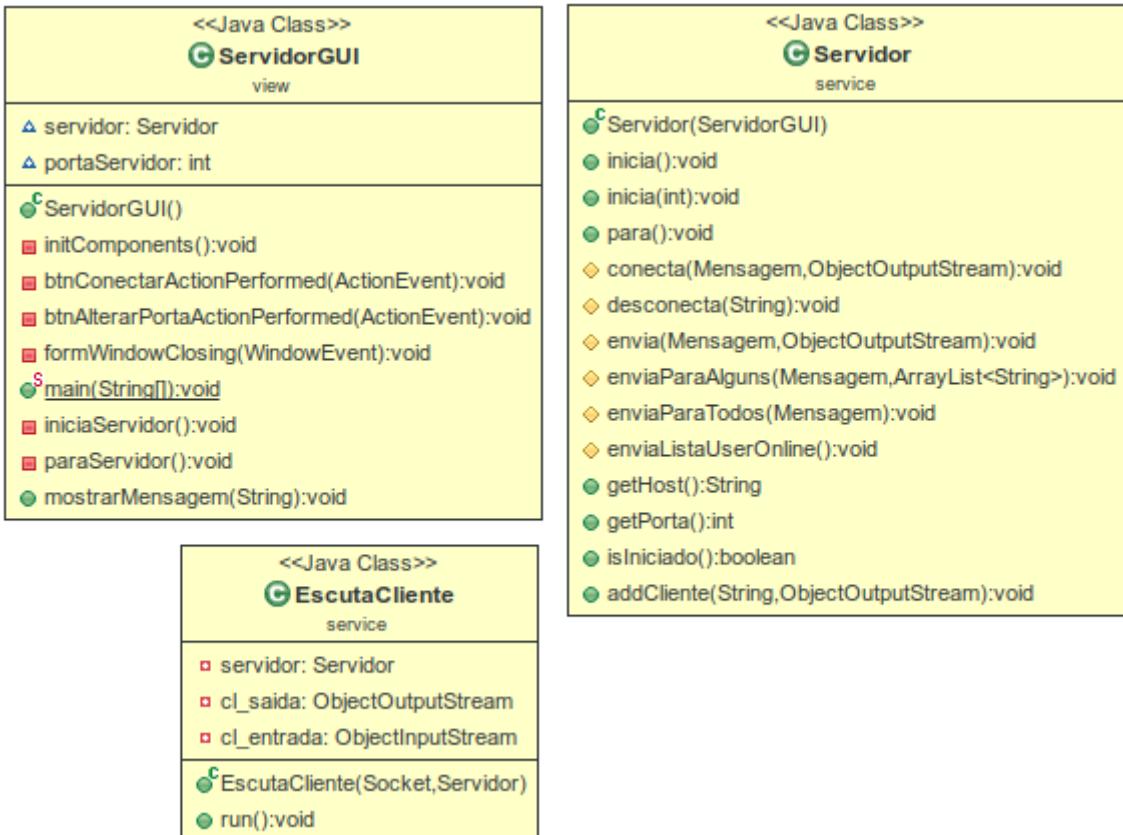
A interface gráfica exibirá a porta na qual o servidor receberá as conexões dos usuários, uma opção para parar/iniciar o servidor e outra para alterar o número da porta. Alguns métodos desta classe estão mostrados na Figura 6.

O método **main** instancia um objeto do tipo ***ServidorGUI***, ou seja, da própria classe. O objeto então inicializa os componentes da interface gráfica, **initComponents()**, inicia o servidor, **iniciaServidor()**, e por fim, mostra a interface gráfica.

Os métodos **btnConectarActionPerformed** e **btnAlterarPortaActionPerformed**, são executados quando o usuário deseja iniciar/parar o servidor e alterar a porta pela qual receberá as conexões dos clientes, respectivamente. Já o método **formWindowClosing** é invocado sempre que a janela do servidor é fechada, e tem como objetivo parar o servidor e enviar para todos os clientes do chat uma mensagem informando que o servidor foi encerrado.

Para realizar as operações do servidor, como iniciar, parar e enviar mensagens, é instanciado um objeto do tipo **Servidor**.

Figura 6 – Classes do programa servidor, com alguns de seus métodos e atributos



Fonte: Elaborada pelos autores

5.1.2 Servidor

O objeto desta classe, que está no pacote **service**, será instanciado exclusivamente pela classe **ServidorGUI**. Sendo o responsável pela comunicação com os programas clientes, através da criação de um *servidor socket*.

O *servidor socket*, criado a partir do método **inicia**, esperará pelas conexões dos usuários, assim que recebê-las, criará uma *thread*(do tipo **EscutaCliente**) que ficará responsável pela comunicação com o cliente que enviou o pedido de conexão. Desta forma o servidor trabalhará com vários clientes ao mesmo tempo, cada qual com sua própria *thread*.

Quando o método **para** é invocado, envia uma mensagem para todos os clientes que estão online, informando que o servidor foi encerrado. Logo após, finaliza todas as conexões ativas e encerra o servidor.

O método **conecta**, invocado a partir da *thread* do cliente, verifica se o nome de usuário escolhido já está em uso, caso não esteja mantém a conexão ativa e associa o nome do usuário com a conexão. Se o nome já estiver em uso por outro usuário, informa para o cliente sobre este fato e encerra a conexão.

Já o método **desconecta**, como o próprio nome indica, serve para desconectar um cliente do chat. Quando, por algum motivo, a conexão de um usuário deve ser encerrada ou o próprio usuário enviou o pedido de desconexão, este método fecha a conexão do respectivo usuário, e envia uma mensagem para os demais clientes informando sobre este fato.

Para manter a lista dos usuários online sempre atualizada, foi criado o método **enviaListaUserOnline**, que envia para todos os clientes do chat uma lista com os usuários que estão online naquele momento.

Além destes, existem mais 3 métodos principais nesta classe, todos com a função de enviar mensagens para os clientes, seja mensagens do próprio servidor ou a retransmissão de mensagens recebidas dos clientes.

O método **envia**, é utilizado para enviar uma mensagem do servidor para um cliente específico. Já o **enviaParaAlguns** é responsável por enviar mensagens privadas de um usuário para outro. Por último, tem o **enviaParaTodos**, que envia as mensagens de um usuário para todos os outros. Estas mensagens serão exibidas publicamente, na sala pública do chat.

5.1.3 EscutaCliente

Esta classe implementa a interface **Runnable**, sendo instanciada por um objeto **Servidor**, que executa os objetos instanciados dentro de uma *thread*. Sua única responsabilidade é “escutar” o cliente, recebendo as mensagens que são enviadas por ele.

Para cada mensagem recebida é verificado qual a ação que o usuário deseja. Se é para se conectar no chat(enviando como texto o seu nome de usuário), se desconectar, enviar uma mensagem de forma privada, ou enviar uma mensagem pública para todos os usuários online. Cada ação executa um método correspondente, no objeto **Servidor**.

5.1.4 Mensagem

Esta classe faz parte do pacote **`service`** e é utilizada tanto no programa cliente quanto no programa servidor. Os objetos instanciados da classe **Mensagem**, conterão as mensagens que serão enviadas pelos usuários e pelo servidor.

Além do texto da mensagem, os atributos desta classe armazenam diversos outros dados de igual importância. O primeiro, e principal deles, é o tipo de mensagem que está sendo enviada. Desta forma, o destinatário e o servidor sabem qual ação tomar com relação aquela mensagem.

A ação pode ser de 4 tipos:

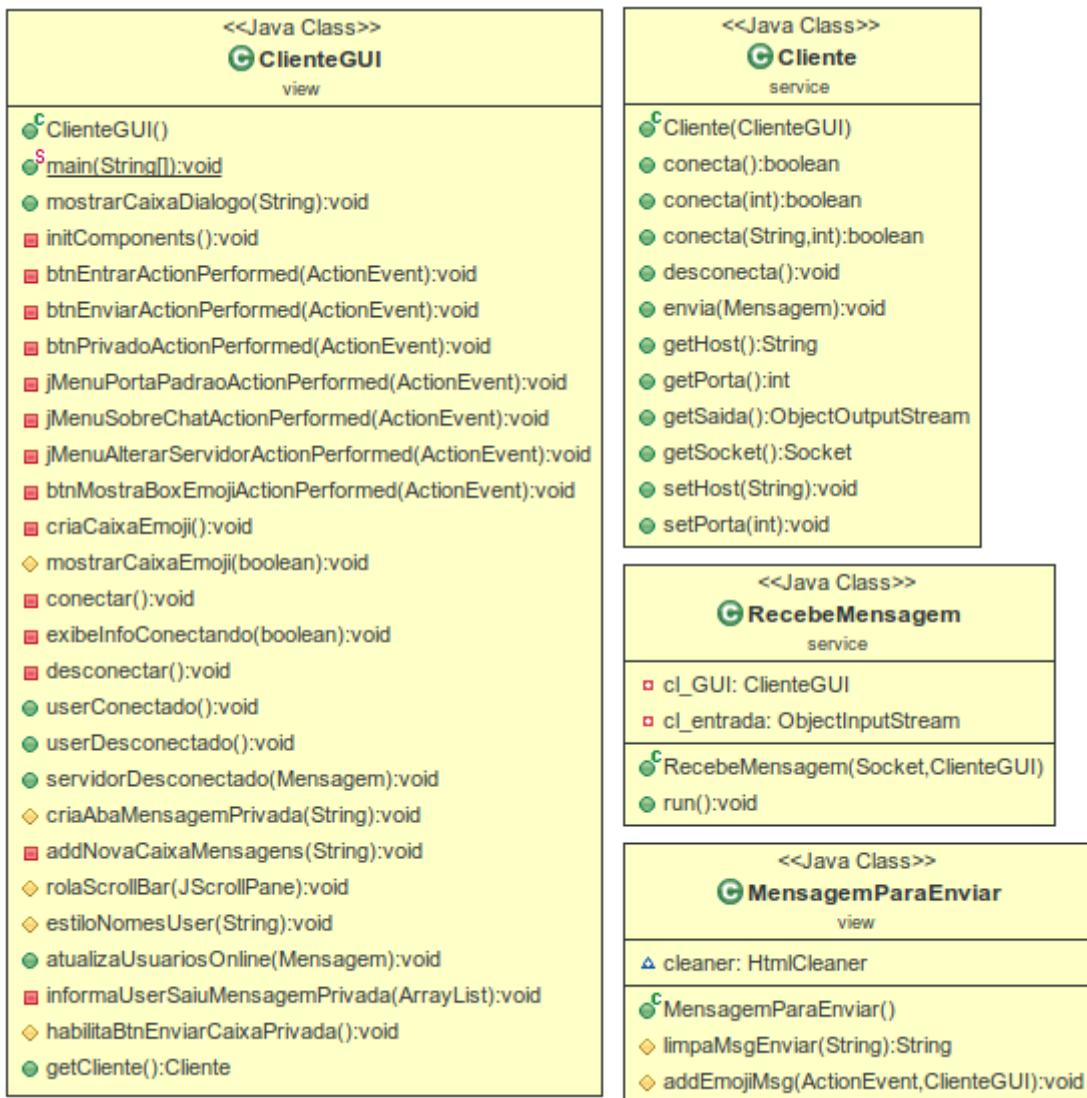
- **`conectar`**: o cliente pede para se conectar ao servidor através de um nome de usuário, e o servidor confirma se aceitou ou não o pedido de conexão;
- **`desconectar`**: o cliente informa ao servidor que está se desconectando do chat. Ou, o servidor informa aos clientes que está encerrando o chat;
- **`enviar`**: o cliente ou o servidor envia uma mensagem privada para um usuário específico;
- **`enviar_para_todos`**: o cliente ou o servidor envia uma mensagem para todos os usuários;
- **`usuarios_online`**: o servidor envia a lista de usuários online para todos os clientes;

Outras informações enviadas junto com o texto da mensagem são o nome do remetente(seja ele o servidor ou um usuário qualquer), o nome do usuário destinatário, caso a mensagem seja privada, e a lista de usuários online, caso seja uma mensagem enviada pelo servidor para os clientes com a finalidade de informá-los sobre quais usuários estão online.

5.2 Cliente

O programa cliente é formado por 9 classes: **`ClienteGUI`**, **`Cliente`**, **`RecebeMensagem`**, **`MensagemParaEnviar`**, **`CaixaMensagem`**, **`ExibeMensagem`**, **`Mensagem`**, **`AlterarServidorGUI`**, **`SobreGUI`**. Sendo que a classe **Mensagem** é a mesma utilizada no programa servidor.

Figura 7 – Algumas classes do programa cliente, com alguns de seus métodos e atributos



Fonte: Elaborada pelos autores

5.2.1 ClienteGUI

Esta classe pertence ao pacote **view**, e é a principal do programa cliente. Toda vez que o programa for executado, será invocada esta classe, que mostrará a interface gráfica do chat.

Alguns métodos desta classe estão mostrados na Figura 7. O método **main** instancia um objeto do tipo **ClienteGUI**, ou seja, da própria classe. O objeto então inicializa os componentes da interface gráfica através do método **initComponents**, logo após instancia um objeto do tipo **Cliente**, finalizando com a exibição da interface gráfica.

A classe contém diversos métodos, sendo os principais deles listados abaixo:

- **btnEntrarActionPerformed**: este método é invocado quando o usuário clica no botão de entrar/sair do chat, invocando dois outros métodos, o **conectar** quando o chat está desconectado e o **desconectar** quando o chat está conectado;
- **btnEnviarActionPerformed** e **btnPrivadoActionPerformed**: os dois são invocados quando seus respectivos botões são clicados. O primeiro envia a mensagem escrita pelo usuário e o segundo invoca o método **criaAbaMensagemPrivada**;
- **criaCaixaEmoji**: cria uma caixa de botões contendo 60 emojis que podem ser selecionados pelo usuário e inseridos na mensagem que será enviada;
- **conectar**: invoca o método **conecta** do objeto do tipo **Cliente**, e aguarda a confirmação do estabelecimento da conexão. Caso seja confirmada, instancia um objeto do tipo **RecebeMensagem** e executa ele em uma *thread*. Logo em seguida, envia uma mensagem para o servidor(através do método **envia** do objeto **Cliente**) informando o nome do usuário e a ação do tipo CONECTAR.
 - Este método também contém um mecanismo de segurança, que caso o tempo de resposta do servidor para a confirmação do estabelecimento da conexão ultrapasse 10 segundos, encerra a tentativa de conexão e informa o usuário;
- **desconectar**: invoca o método **desconecta** do objeto do tipo **Cliente**, e em seguida o método **userDesconectado** da própria classe;
- **userConectado**: quando o servidor confirmar o estabelecimento da conexão com base no nome do usuário, este método será invocado. Ele ativa os campos e botões do chat que permitem a comunicação do usuário com os demais clientes que estejam online;
- **userDesconectado**: ele desativa os campos e botões do chat que permitem a comunicação do usuário com os demais clientes que estejam online;
- **servidorDesconectado**: caso o servidor venha a ficar desligado, exibe uma mensagem informando que o servidor não está ativo e invoca o método **userDesconectado**;
- **criaAbaMensagemPrivada**: cria uma sala privada para o usuário conversar com outro usuário de forma privada;
- **estiloNomesUser**: responsável por escolher a cor com a qual os nomes dos

- usuários serão exibidos no chat;
- **atualizaUsuariosOnline**: atualiza a lista de usuários online;

5.2.2 Cliente

O objeto desta classe, que está no pacote **service**, será instanciado exclusivamente pela classe **ClienteGUI**. Sendo o responsável pela comunicação com o programa servidor, através da criação de um *socket*.

Esta classe contém 3 métodos **conecta**. O primeiro conecta ao servidor padrão, o segundo conecta ao servidor padrão só que com o número de sua porta alterado(informado pelo usuário), e o terceiro conecta a um servidor diferente, especificado pelo usuário.

Além destes existem mais 2 métodos. O primeiro é o **desconecta** que encerra a conexão com o servidor, e o segundo é o **envia**, que envia a mensagem do usuário para o servidor.

5.2.3 RecebeMensagem

Esta classe tem basicamente, as mesmas responsabilidades da classe **EscutaCliente**, do programa servidor. Esta classe implementa a interface **Runnable**, sendo instanciada por um objeto **ClienteGUI**, que executa o objeto instanciado dentro de uma *thread*. Sua única responsabilidade é receber as mensagens enviados pelo servidor.

Para cada mensagem recebida é verificado qual tipo de informação que a mensagem contém. Se é a confirmação da conexão com o servidor, se é informando que o servidor foi desconectado, se é uma mensagem enviada por um usuário de forma privada, ou se é uma mensagem pública vinda de um usuário qualquer. Para cada tipo é executado um método correspondente, no objeto **ClienteGUI**.

5.2.4 MensagemParaEnviar

Esta classe manipulará o código da mensagem que será enviada pelo usuário. Existem dois métodos, o primeiro é o **limpaMsgEnviar**, que através de um objeto do tipo **HtmlCleaner** remove todo código HTML desnecessário da mensagem

e substitui as imagens dos emojis por seus respectivos códigos, retornando uma String com a mensagem formatada, que será enviada para o servidor.

O segundo método é o **addEmojiMsg**, que adiciona o emoji escolhido pelo usuário(através da interface gráfica) na posição desejada na mensagem.

5.2.5 ExibeMensagem

Esta classe é responsável por exibir as mensagens recebidas. Ela contém três métodos. O primeiro, **exibeMensagemPrivada**, é responsável por selecionar a caixa de mensagens privada onde será exibida a mensagem, caso não exista invocará um método que criará uma. O segundo, **exibeMensagemGlobal**, selecionará a caixa de mensagens pública que deve ser utilizada para exibir a mensagem.

Tanto o primeiro quanto o segundo, invocam o método **exibeMsgChat**, que efetivamente exibirá a mensagem na caixa selecionada por eles. Formatando a mensagem e trocando os códigos dos emojis por suas respectivas imagens.

5.2.6 CaixaMensagem

Esta classe criará as caixas de mensagens do chat, tanto a pública como as privadas, em seu método construtor. Tendo dois principais atributos, o **ultimoRemetente**, que guarda o nome do último usuário que teve sua mensagem exibida na caixa, e o segundo, **usuarioOnline**, utilizado nas caixas de mensagens privadas, define se o usuário com o qual o cliente está trocando mensagens está online.

5.2.7 AlterarServidorGUI e SobreGUI

Estas duas classes, **AlterarServidorGUI** e **SobreGUI**, tem suas interfaces gráficas exibidas quando o usuário seleciona as opções de “Alterar servidor” e “Sobre/Chat”, respectivamente, na barra de menus do chat. A primeira permite que o usuário informe o endereço IP e a porta do servidor com o qual deseja se conectar, e a segunda mostra os dados dos desenvolvedores e dos autores das bibliotecas e das imagens utilizadas nesta aplicação.

6 RELATÓRIO COM AS LINHAS DE CÓDIGO DO PROGRAMA

Nas próximas páginas estão expostos os códigos fontes das principais classes da aplicação. Algumas partes do código foram ocultadas, como, por exemplo, as linhas de importação de bibliotecas e classes(**import**).

Programa 1 – Classe Servidor

```
package service;
public class Servidor {
    private String host;
    public static int porta = 12345;
    private ServerSocket servidor;
    private boolean iniciado;
    private Map<String, ObjectOutputStream> clientes = new HashMap<String,
ObjectOutputStream>();
    private ServidorGUI obView;
    private String nomeServer = "servidorChat";
    public Servidor(ServidorGUI obView) {
        this.obView = obView;
    }
    public void inicia() {
        inicia(Servidor.porta);
    }
    public void inicia(int porta) {
        Servidor.porta = porta;
        try {
            servidor = new ServerSocket(Servidor.porta);
            host = servidor.getInetAddress().getHostAddress();
            iniciado = true;
            Servidor sv = this;
            new Thread(new Runnable() {
                @Override
                public void run() {
                    Socket cl_socket;
                    EscutaCliente ec;
                    while (iniciado) {
                        try {
                            cl_socket = servidor.accept();
                            ec = new EscutaCliente(cl_socket, sv);
                            new Thread(ec).start();
                        } catch (IOException ex) {}
                    }
                }
            }).start();
        } catch (IOException ex) {
            this.iniciado = false;
            obView.mostrarMensagem("Não foi possível iniciar o
servidor. \nTente uma porta diferente.");
        }
    }
    public void para() {
        try {
            this.iniciado = false;
            if(!clientes.isEmpty()){


```

```

        Mensagem msgServer = new Mensagem();
        msgServer.setRemetente(this.nomeServer);
        msgServer.setTexto("O servidor não está disponível. Entre
novamente mais tarde.");
        msgServer.setAcao(Mensagem.Acao.DESCONETAR);
        enviaParaTodos(msgServer);
    }
    clientes.clear();
    this.servidor.close();
} catch (IOException ex) {}
}

protected void conecta(Mensagem msg_cl, ObjectOutputStream cl_saida) {
    if (!msg_cl.getRemetente().equals(this.nomeServer) &&
(this.clientes.isEmpty() || !
this.clientes.containsKey(msg_cl.getRemetente())))
    {
        addCliente(msg_cl.getRemetente(), cl_saida);
        enviaListaUserOnline();
        Mensagem msgServer = new Mensagem();
        msgServer.setRemetente(this.nomeServer);
        msgServer.setTexto("YES");
        msgServer.setAcao(Mensagem.Acao.CONECTAR);
        envia(msgServer, cl_saida);
    } else {
        Mensagem msgServer = new Mensagem();
        msgServer.setRemetente(this.nomeServer);
        msgServer.setTexto("NO");
        msgServer.setAcao(Mensagem.Acao.CONECTAR);
        envia(msgServer, cl_saida);
    }
}
protected void desconecta(String cliente) {
    this.clientes.remove(cliente);
    Mensagem msgServer = new Mensagem();
    msgServer.setRemetente(this.nomeServer);
    msgServer.setTexto(cliente + " saiu do chat");
    msgServer.setAcao(Mensagem.Acao.ENVIAR_PARA_TODOS);
    enviaParaTodos(msgServer);
    enviaListaUserOnline();
}
protected void envia(Mensagem msg, ObjectOutputStream cliente) {
    try {
        cliente.writeObject(msg);
    } catch (IOException ex) {}
}
protected void enviaParaAlguns(Mensagem msg, ArrayList<String>
cl_enviar) {
    for (int i = 0; i < cl_enviar.size(); i++) {
        if (this.clientes.containsKey(msg.getUsuariosEnviar().get(i)))
    {
        try {
            clientes.get(msg.getUsuariosEnviar().get(i)).writeObject(msg);
        } catch (IOException ex) {}
    }
}
protected void enviaParaTodos(Mensagem msg) {
    for (Map.Entry<String, ObjectOutputStream> cl :
this.clientes.entrySet()) {
        try {

```

```

        cl.getValue().writeObject(msg);
    } catch (IOException ex) {}
}
protected void enviaListaUserOnline() {
    ArrayList<String> userOnline = new ArrayList<>();
    for (Map.Entry<String, ObjectOutputStream> cl_online :
clientes.entrySet()) {
        userOnline.add(cl_online.getKey());
    }
    Mensagem msgServer = new Mensagem();
    msgServer.setRemetente(this.nomeServer);
    msgServer.setAcao(Mensagem.Acao.USUARIOS_ONLINE);
    msgServer.setUsuariosOnline(userOnline);
    for (Map.Entry<String, ObjectOutputStream> cl :
clientes.entrySet()) {
        try {
            cl.getValue().writeObject(msgServer);
        } catch (IOException ex) {}
    }
}
public String getHost() {
    return host;
}
public int getPorta() {
    return porta;
}
public boolean isIniciado() {
    return iniciado;
}
public void addCliente(String cl, ObjectOutputStream obOs) {
    this.clientes.put(cl, obOs);
}
}

```

Fonte: Elaborado pelos autores

Programa 2 – Classe EscutaCliente

```

package service;
public class EscutaCliente implements Runnable {
    private Servidor servidor;
    private ObjectOutputStream cl_saida;
    private ObjectInputStream cl_entrada;
    public EscutaCliente(Socket cl_socket, Servidor servidor) {
        this.servidor = servidor;
        try {
            this.cl_saida = new
ObjectOutputStream(cl_socket.getOutputStream());
            this.cl_entrada = new
ObjectInputStream(cl_socket.getInputStream());
        } catch (IOException ex) {}
    }
    @Override
    public void run() {
        Mensagem msg_cliente = null;
        try {
            while ((msg_cliente = (Mensagem) cl_entrada.readObject()) !=
null) {

```

```

        Acao acao = msg_cliente.getAcao();
        switch (acao) {
            case CONECTAR:
                servidor.conecta(msg_cliente, cl_saida);
                break;
            case DESCONECTAR:
                servidor.desconecta(msg_cliente.getRemetente());
                break;
            case ENVIAR:
                servidor.enviaParaAlguns(msg_cliente,
msg_cliente.getUsuariosEnviar());
                break;
            case ENVIAR_PARA_TODOS:
                servidor.enviaParaTodos(msg_cliente);
                break;
            default:
                break;
        }
    }
} catch (IOException ex) {
    servidor.desconecta(msg_cliente.getRemetente());
} catch (ClassNotFoundException ex) {}
}
}

```

Fonte: Elaborado pelos autores

Programa 3 – Classe Mensagem

```

package service;
public class Mensagem implements Serializable {
    private String remetente;
    private String texto;
    private ArrayList<String> usuariosEnviar;
    private Acao acao;
    private ArrayList<String> usuariosOnline;
    public enum Acao {
        CONECTAR, DESCONECTAR, ENVIAR, ENVIAR_PARA_TODOS, USUARIOS_ONLINE;
    }
    public String getRemetente() { return remetente; }
    public void setRemetente(String nome) { this.remetente = nome; }
    public String getTexto() { return texto; }
    public void setTexto(String texto) { this.texto = texto; }
    public ArrayList<String> getUsuariosEnviar() { return usuariosEnviar; }
    public void setUsuariosEnviar(ArrayList<String> usuariosEnviar) {
        this.usuariosEnviar = usuariosEnviar;
    }
    public Acao getAcao() { return acao; }
    public void setAcao(Acao acao) {
        this.acao = acao;
    }
    public ArrayList<String> getUsuariosOnline() {
        return usuariosOnline;
    }
    public void setUsuariosOnline(ArrayList<String> usuariosOnline) {
        this.usuariosOnline = usuariosOnline;
    }
}

```

Fonte: Elaborado pelos autores

Programa 4 – Classe Cliente

```

package service;
public class Cliente {
    public static String HOST_PADRAO      = "127.0.0.1";
    private String host                   = HOST_PADRAO;
    public static int PORTA_PADRAO        = 12345;
    private int porta                    = PORTA_PADRAO;
    private Socket socket;
    private ObjectOutputStream saida;
    private ClienteGUI obView;
    public Cliente(ClienteGUI obView) {
        this.obView = obView;
    }
    public boolean conecta() {
        return conecta(host, porta);
    }
    public boolean conecta(int porta) {
        return conecta(this.host, porta);
    }
    public boolean conecta(String host, int porta) {
        this.host = host;
        this.porta = porta;
        try {
            socket = new Socket(this.host, this.porta);
            saida = new ObjectOutputStream(socket.getOutputStream());
            return true;
        } catch (IOException ex) {
            return false;
        }
    }
    public void desconecta() {
        try {
            saida.close();
            socket.close();
        } catch (IOException ex) {}
    }
    public void envia(Mensagem msg) {
        try {
            saida.writeObject(msg);
        } catch (IOException ex) {}
    }
    public String getHost() { return host; }
    public int getPorta() {
        return porta;
    }
    public ObjectOutputStream getSaida() {
        return saida;
    }
    public Socket getSocket() {
        return socket;
    }
    public void setHost(String host) {
        this.host = host;
    }
    public void setPorta(int porta) {
        this.porta = porta;
    }
}

```

Fonte: Elaborado pelos autores

Programa 5 – Classe RecebeMensagem

```

package service;
public class RecebeMensagem implements Runnable {
    private ClienteGUI cl_GUI;
    private ObjectInputStream cl_entrada;
    public RecebeMensagem(Socket cl_socket, ClienteGUI cl_GUI) {
        this.cl_GUI = cl_GUI;
        try {
            this.cl_entrada = new
ObjectInputStream(cl_socket.getInputStream());
        } catch (IOException ex) {}
    }
    public void run() {
        Mensagem msg_servidor = null;
        try {
            while ((msg_servidor = (Mensagem) cl_entrada.readObject()) !=
null) {
                Mensagem.Acao acao = msg_servidor.getAcao();
                switch (acao) {
                    case CONECTAR:
                        if(msg_servidor.getTexto().equals("NO")){
                            cl_GUI.mostrarCaixaDialogo("Não foi possível
entrar no chat. "
                                + "\nDigite outro nome de usuário");
                        } else {
                            cl_GUI.userConectado();
                        }
                        break;
                    case DESCONECTAR:
                        cl_GUI.servidorDesconectado(msg_servidor);
                        break;
                    case ENVIAR:
                        cl_GUI.getExibeMensagem().exibeMensagemPrivada(msg_servidor);
                        break;
                    case ENVIAR_PARA_TODOS:
                        cl_GUI.getExibeMensagem().exibeMensagemGlobal(msg_servidor);
                        break;
                    case USUARIOS_ONLINE:
                        cl_GUI.atualizaUsuariosOnline(msg_servidor);
                        break;
                    default:
                        break;
                }
            }
        } catch (IOException | ClassNotFoundException ex) {}
    }
}

```

Fonte: Elaborado pelos autores

Programa 6 – Classe MensagemParaEnviar

```

package view;
public class MensagemParaEnviar {
    HtmlCleaner cleaner;;
    public MensagemParaEnviar(){
        this.cleaner = new HtmlCleaner();
    }
}

```

```

    }
    protected String limpaMsgEnviar(String msg){
        TagNode rootNode = cleaner.clean(msg);
        TagNode[] elementosImg = rootNode.getElementsByName("img", true);
        for(int i=0; elementosImg != null && i < elementosImg.length; i++){
        {
            String codigoEmoji =
elementosImg[i].getAttributeByName("alt");
            cleaner.setInnerHTML(elementosImg[i].getParent(),
": "+codigoEmoji+":");
            elementosImg[i].removeFromTree();
        }
        TagNode[] styleTag = rootNode.getElementsByName("style", true);
        if(styleTag.length > 0){
            styleTag[0].removeFromTree();
        }
        TagNode[] elementosP = rootNode.getElementsByName("p", true);
        for(int i=0; elementosP != null && i < elementosP.length; i++){
            elementosP[i].removeAttribute("style");
        }
        String htmlMsg =
cleaner.getInnerHTML(rootNode.getElementsByName("body", true)[0]).trim();
        htmlMsg = htmlMsg.replaceAll("<p>", "<span style='text-align: center;'>");
        htmlMsg = htmlMsg.replaceAll("</p>", "</span><br />");
        return htmlMsg.trim();
    }
    protected void addEmojiMsg(java.awt.event.ActionEvent evt, ClienteGUI cliGUI) {
        cliGUI.mostrarCaixaEmoji(false);
        JButton btn = (JButton) evt.getSource();
        URL urlEmoji =
getClass().getResource("/img/emojis2/"+btn.getName()+".png");
        JEditorPane caixaMsgUser = cliGUI.txtMensagemUser;
        HTMLDocument docCaixaMsgUser = (HTMLDocument)
caixaMsgUser.getDocument();
        try {
            StyleContext styleContextChat = new StyleContext();
            Style style = styleContextChat.addStyle("EMOJI-MSG-USER",
null);
            StyleConstants.setFontFamily(style, "EMOJI-IMG");
            StyleConstants.setFontSize(style, 0);
            StyleConstants.setForeground(style, Color.WHITE);
            StyledDocument docStyled = (StyledDocument)
caixaMsgUser.getDocument();
            docStyled.insertString(caixaMsgUser.getCaretPosition(), ".",
styleContextChat.getStyle("EMOJI-MSG-USER"));
            TagNode rootNode = cleaner.clean(caixaMsgUser.getText());
            TagNode localInserirEmoji =
rootNode.findElementByAttValue("face", "EMOJI-IMG", true, true);
            cleaner.setInnerHTML(localInserirEmoji, "<span
class='"+System.nanoTime()+"'><img src='"+urlEmoji+"' width='25'
height='25' alt='"+btn.getName()+"' /></span>");
            localInserirEmoji.removeAttribute("face");
            localInserirEmoji.removeAttribute("size");
            localInserirEmoji.removeAttribute("color");
            String htmlMsg = cleaner.getInnerHTML(rootNode).trim();
            htmlMsg = htmlMsg.replaceAll("<font>", "");
            htmlMsg = htmlMsg.replaceAll("</font>", "&#32;");
        }
    }
}

```

```
        caixaMsgUser.setText("<html>" + htmlMsg + "</html>");  
    } catch (BadLocationException ex) {}  
}
```

Fonte: Elaborado pelos autores

Programa 7 – Classe ExibeMensagem

```
package view;
public class ExibeMensagem {
    private ClienteGUI cGUI;
    public ExibeMensagem(ClienteGUI clienteGUI){
        this.cGUI = clienteGUI;
    }
    public void exibeMensagemPrivada(Mensagem msg){
        String nomeAba;
        if(msg.getRemetente().equals(cGUI.getNomeUsuario())){
            nomeAba = msg.getUsuariosEnviar().get(0);
        } else if(msg.getRemetente().equals(cGUI.getNomeServidor())){
            nomeAba = msg.getUsuariosEnviar().get(0);
        } else{
            nomeAba = msg.getRemetente();
        }
        if(!cGUI.getCaixasMensagens().containsKey(nomeAba)){
            cGUI.criaAbaMensagemPrivada(nomeAba);
        }
        CaixaMensagem caixaMsgPrivada =
cGUI.getCaixasMensagens().get(nomeAba);
        JTabbedPane jtpPainelAbasMensagem =
cGUI.getJtpPainelAbasMensagem();
        JScrollPane jSP = (JScrollPane)
jtpPainelAbasMensagem.getComponentAt(jtpPainelAbasMensagem.indexOfTab(nomeAba));
        boolean deveRolarCaixa = deveRolarCaixa(jSP);
        exibeMsgChat(msg, caixaMsgPrivada);
        if(deveRolarCaixa){
            cGUI.rolaScrollBar(jSP);
        }
        if(jtpPainelAbasMensagem.getSelectedIndex() !=
jtpPainelAbasMensagem.indexOfTab(nomeAba)){
            jtpPainelAbasMensagem.setBackgroundAt(jtpPainelAbasMensagem.indexOfTab(nomeAba), Color.yellow);
        }
    }
    public void exibeMensagemGlobal(Mensagem msg){
        JScrollPane jSP;
        CaixaMensagem caixaMsgGlobal =
cGUI.getCaixasMensagens().get(cGUI.getNomeAbaGeral());
        JTabbedPane jtpPainelAbasMensagem =
cGUI.getJtpPainelAbasMensagem();
        if(jtpPainelAbasMensagem.isVisible()){
            if(jtpPainelAbasMensagem.getSelectedIndex() !=
jtpPainelAbasMensagem.indexOfTab(cGUI.getNomeAbaGeral())){
                jtpPainelAbasMensagem.setBackgroundAt(jtpPainelAbasMensagem.indexOfTab(cGUI.getNomeAbaGeral()), Color.yellow);
            }
        }
    }
}
```

```

        jSP = (JScrollPane)
jtpPainelAbasMensagem.getComponentAt(jtpPainelAbasMensagem.indexOfTab(cGUI
.getNomeAbaGeral()));
    } else{
        jSP = cGUI.getjSPanelChat();
    }
    boolean deveRolarCaixa = deveRolarCaixa(jSP);
    exibeMsgChat(msg, caixaMsgGlobal);
    if(deveRolarCaixa){
        cGUI.rolaScrollBar(jSP);
    }
}
public void exibeMsgChat(Mensagem msg, CaixaMensagem caixaMensagem){
    HTMLDocument documentCaixaMsg = (HTMLDocument)
caixaMensagem.getCaixaMensagem().getDocument();
    if(cGUI.getStyleContextChat().getStyle(msg.getRemetente()) == null
&& !msg.getRemetente().equals(cGUI.getNomeServidor())){
        cGUI.estiloNomesUser(msg.getRemetente());
    }
    try {
        URL file;
        for(int i=0; i<cGUI.getEmojisCodigo().length;i++){
            file =
getClass().getResource("/img/emojis2/"+cGUI.getEmojisCodigo()[i]+".png");
msg.setTexto(msg.getTexto().replaceAll(": "+cGUI.getEmojisCodigo()[i]+":",
"<img src='"+file+"' width='25' height='25'>"));
        }
        if(!caixaMensagem.getUltimoRemetente().isEmpty()){
            if((msg.getRemetente().equals(cGUI.getNomeServidor()) && !
caixaMensagem.getUltimoRemetente().equals(cGUI.getNomeServidor())) || (!
msg.getRemetente().equals(cGUI.getNomeServidor()) &&
caixaMensagem.getUltimoRemetente().equals(cGUI.getNomeServidor())) ||

msg.getAcao().equals(Mensagem.Acao.DISCONETAR) ||
msg.getAcao().equals(Mensagem.Acao.CONECTAR)){
                documentCaixaMsg.insertAfterEnd(documentCaixaMsg.getCharacterElement(docum
entCaixaMsg.getLength()), "<br />");
            }
        }
        if(msg.getRemetente().equals(cGUI.getNomeServidor())){
            documentCaixaMsg.insertAfterEnd(documentCaixaMsg.getCharacterElement(docum
entCaixaMsg.getLength()), msg.getTexto()+"<br />");
        } else{
            StyledDocument docStyled = (StyledDocument)
documentCaixaMsg;
            docStyled.insertString(docStyled.getLength(),
msg.getRemetente()+": ",
cGUI.getStyleContextChat().getStyle(msg.getRemetente()));
        }
        documentCaixaMsg.insertAfterEnd(documentCaixaMsg.getCharacterElement(docum
entCaixaMsg.getLength()), msg.getTexto());
    }
    catch (BadLocationException | IOException ex) {}
}
private boolean deveRolarCaixa(JScrollPane jSPane)
{

```

```

        JScrollBar sb = jSPane.getVerticalScrollBar();
        int min = sb.getValue() + sb.getVisibleAmount();
        int max = sb.getMaximum();
        return min == max;
    }
}

```

Fonte: Elaborado pelos autores

Programa 8 – Classe CaixaMensagem

```

package view;
public class CaixaMensagem {
    private JTextPane caixaMensagem;
    private String ultimoRemetente;
    private boolean usuarioOnline;
    public CaixaMensagem(){
        caixaMensagem = new JTextPane();
        caixaMensagem.setEditable(false);
        caixaMensagem.setEnabled(true);
        caixaMensagem.setBackground(new java.awt.Color(255, 255, 255));
        caixaMensagem.setMargin(new java.awt.Insets(3, 5, 3, 3));
        caixaMensagem.setContentType("text/html");
        caixaMensagem.setDocument(new HTMLDocument());
        caixaMensagem.setEditorKit(new HTMLEditorKit());
        caixaMensagem.setText(
            "<html>" +
            "+ "<style type='text/css'>" +
            "+ "body{color:black; font-family: Dialog; font-size: 10px}" +
            "+ "</style>" +
            "+ "<body>" +
        );
        ultimoRemetente = "";
        usuarioOnline = true;
    }
    public JTextPane getCaixaMensagem() {
        return caixaMensagem;
    }
    public String getLastRecipient() {
        return ultimoRemetente;
    }
    public void setLastRecipient(String ultimoRemetente) {
        this.ultimoRemetente = ultimoRemetente;
    }
    public boolean isUsuarioOnline() {
        return usuarioOnline;
    }
    public void setUsuarioOnline(boolean usuarioOnline) {
        this.usuarioOnline = usuarioOnline;
    }
}

```

Fonte: Elaborado pelos autores

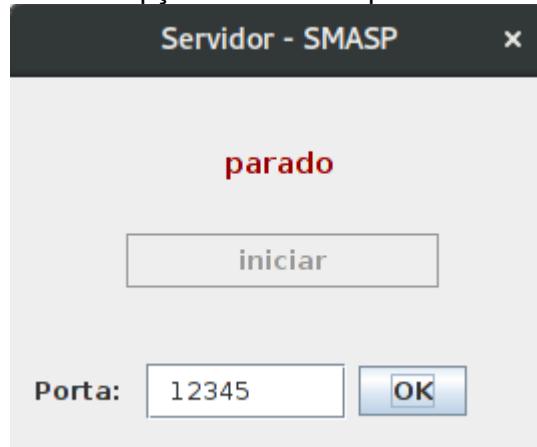
7 APRESENTAÇÃO DO PROGRAMA EM FUNCIONAMENTO EM UM COMPUTADOR

Figura 8 – Servidor iniciado(tela inicial do servidor) e servidor parado



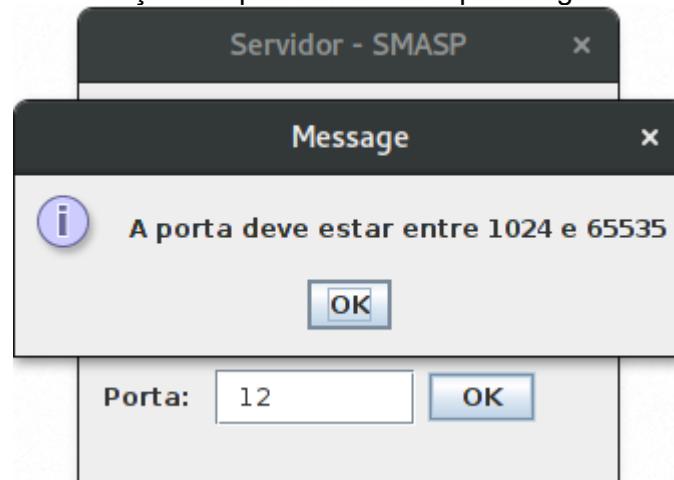
Fonte: Elaborada pelos autores

Figura 9 – Quando a opção de alterar a porta do servidor é clicada



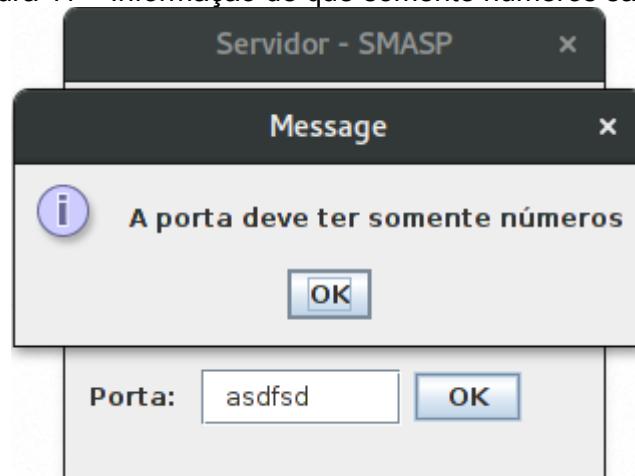
Fonte: Elaborada pelos autores

Figura 10 – Informação de que o número da porta digitado está incorreto



Fonte: Elaborada pelos autores

Figura 11 – Informação de que somente números são permitidos



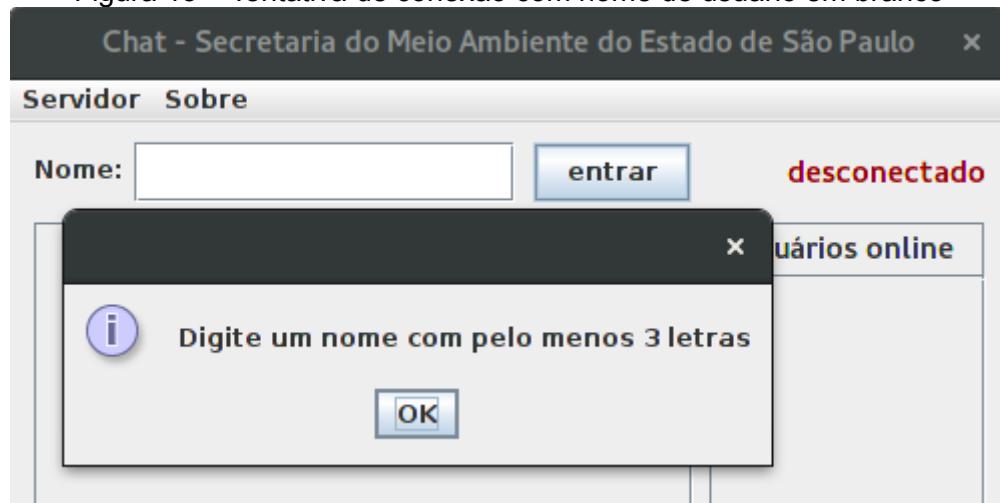
Fonte: Elaborada pelos autores

Figura 12 – Chat desconectado / tela inicial do chat



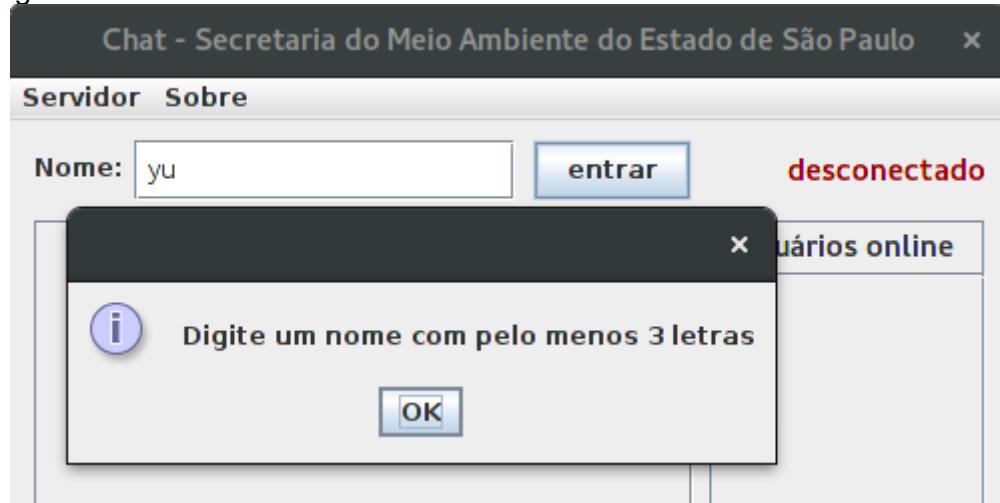
Fonte: Elaborada pelos autores

Figura 13 – Tentativa de conexão com nome de usuário em branco



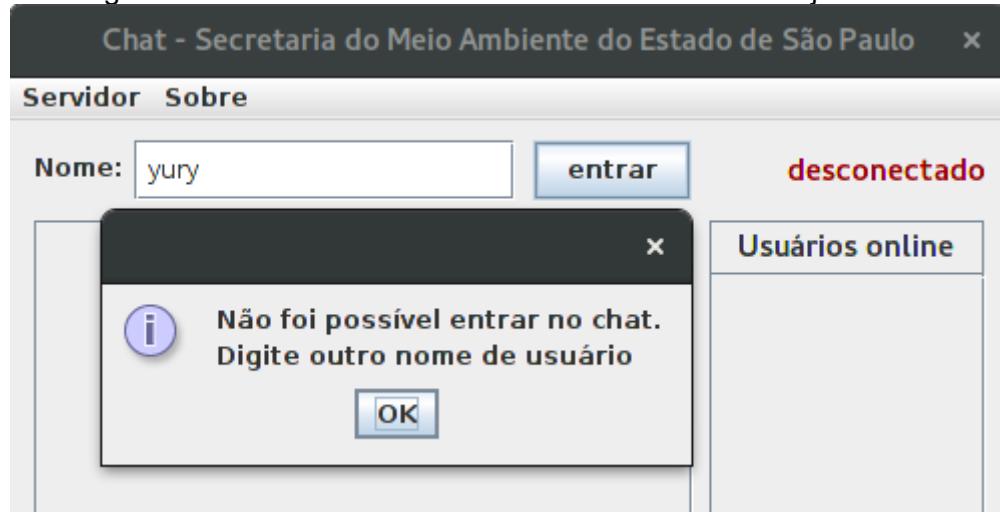
Fonte: Elaborada pelos autores

Figura 14 – Tentativa de conexão com nome de usuário com menos de 3 letras



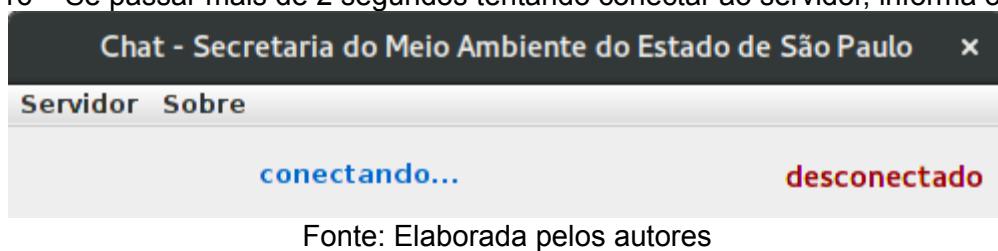
Fonte: Elaborada pelos autores

Figura 15 – Tentativa de conexão com nome de usuário já em uso



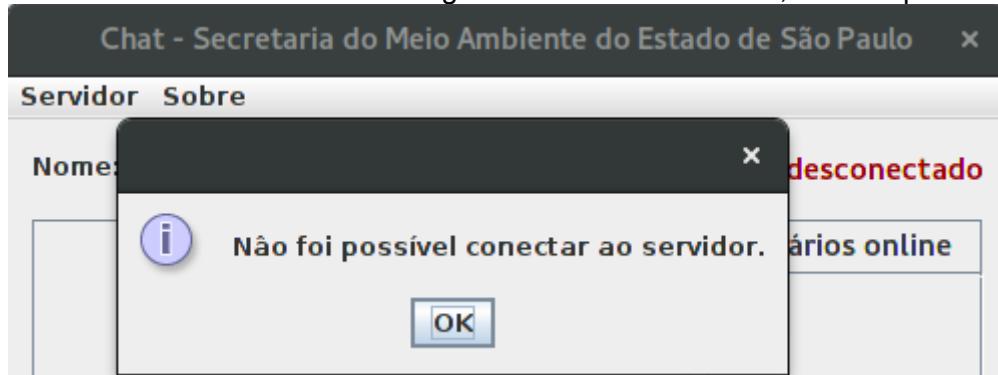
Fonte: Elaborada pelos autores

Figura 16 – Se passar mais de 2 segundos tentando conectar ao servidor, informa o usuário



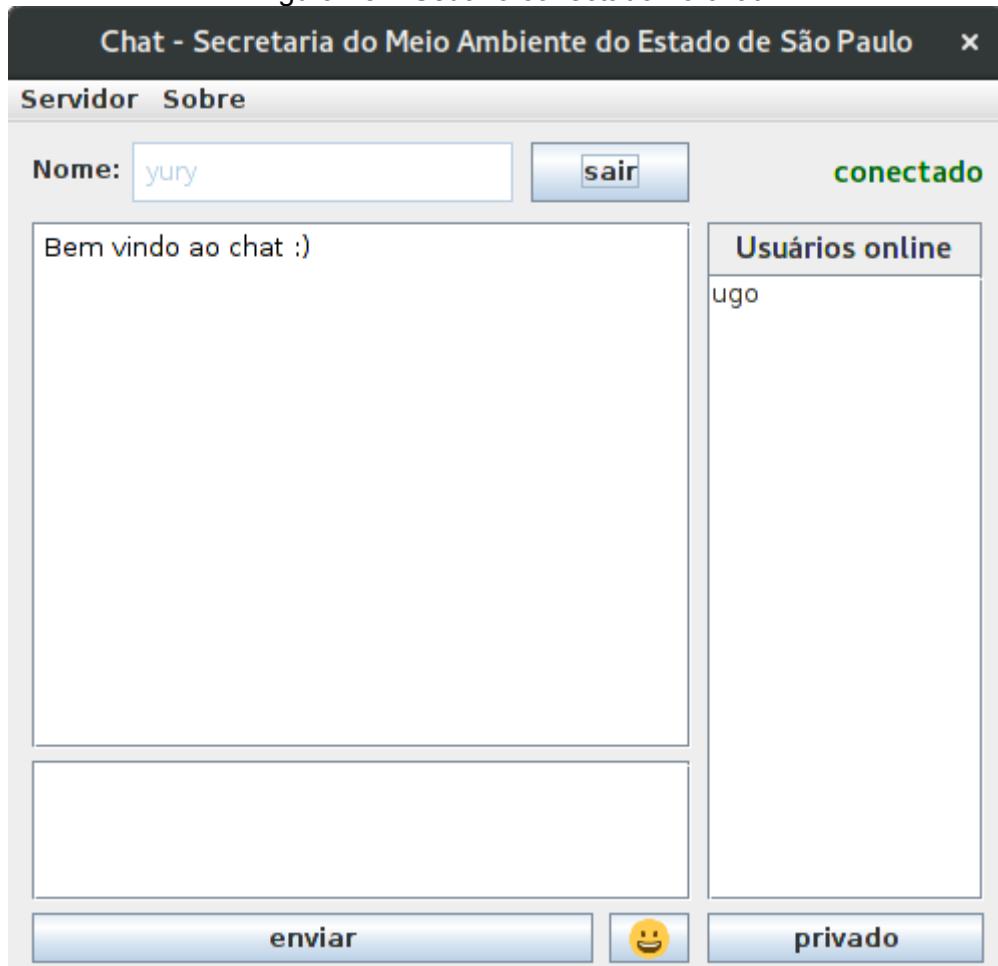
Fonte: Elaborada pelos autores

Figura 17 – Se demorou mais de 10 segundos tentando conectar, interrompe a tentativa



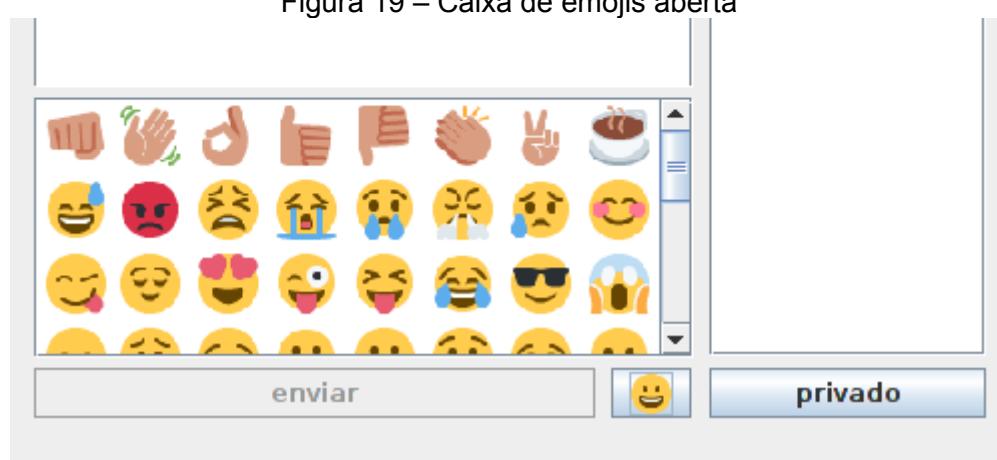
Fonte: Elaborada pelos autores

Figura 18 – Usuário conectado no chat



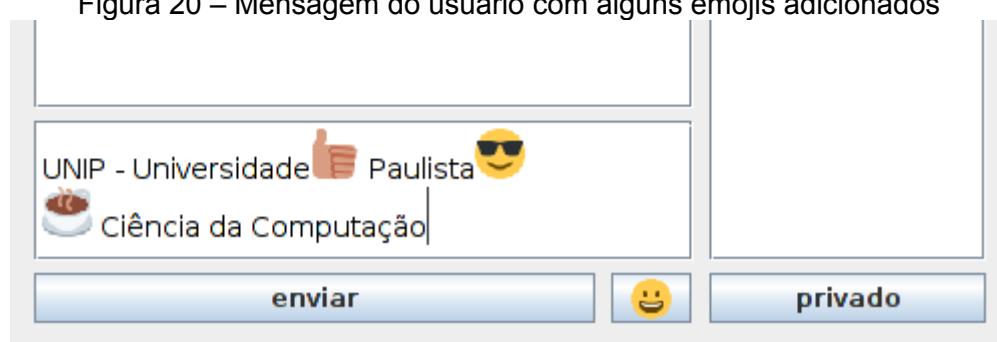
Fonte: Elaborada pelos autores

Figura 19 – Caixa de emojis aberta



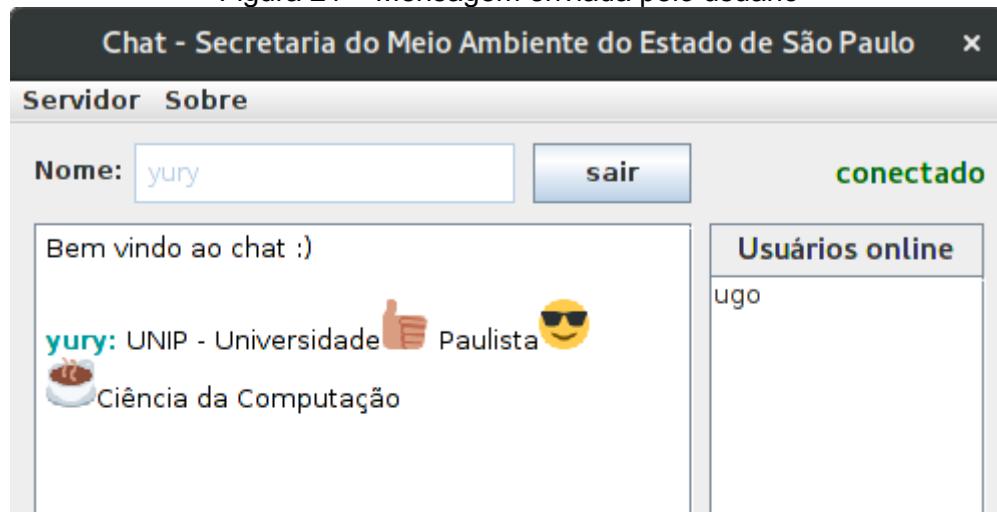
Fonte: Elaborada pelos autores

Figura 20 – Mensagem do usuário com alguns emojis adicionados



Fonte: Elaborada pelos autores

Figura 21 – Mensagem enviada pelo usuário



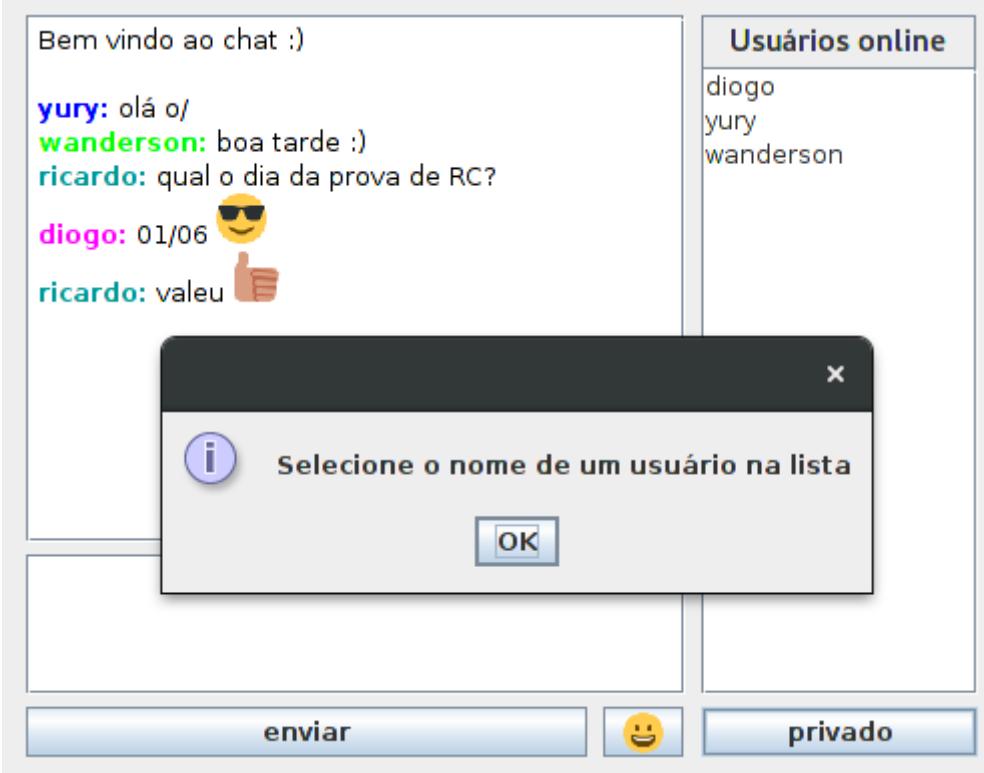
Fonte: Elaborada pelos autores

Figura 22 – Algumas mensagens enviadas e recebidas



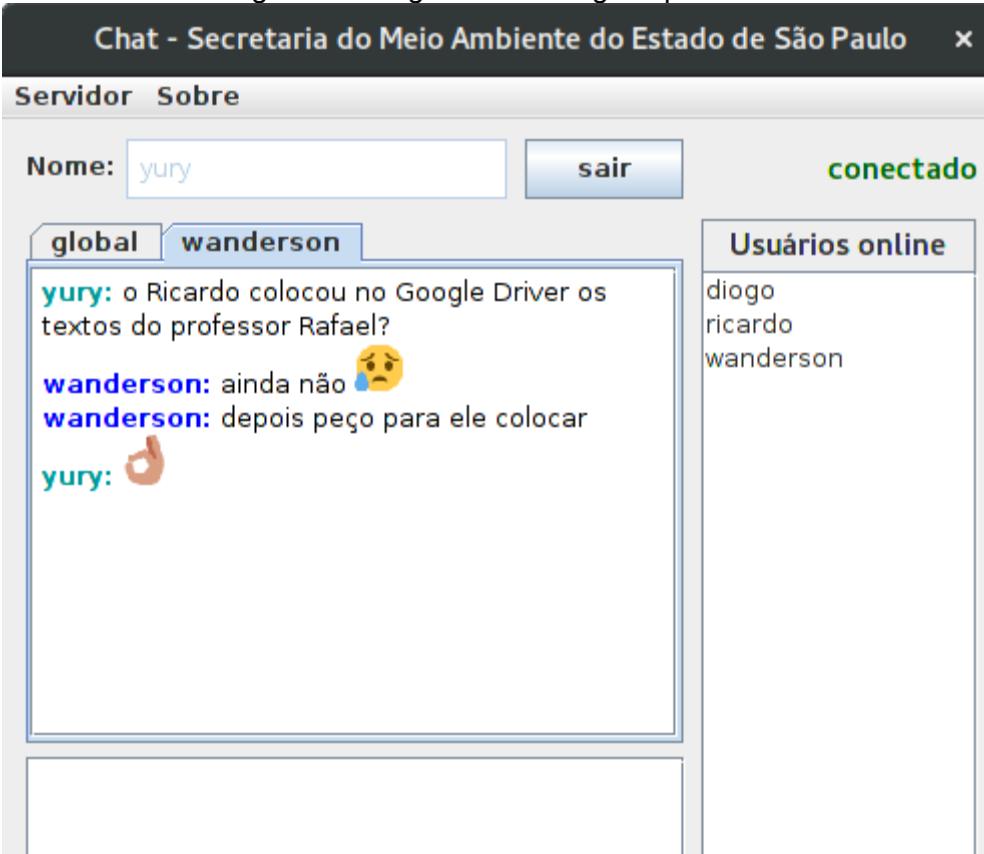
Fonte: Elaborada pelos autores

Figura 23 – Tentativa de abrir uma conversa privada sem ter selecionado algum usuário



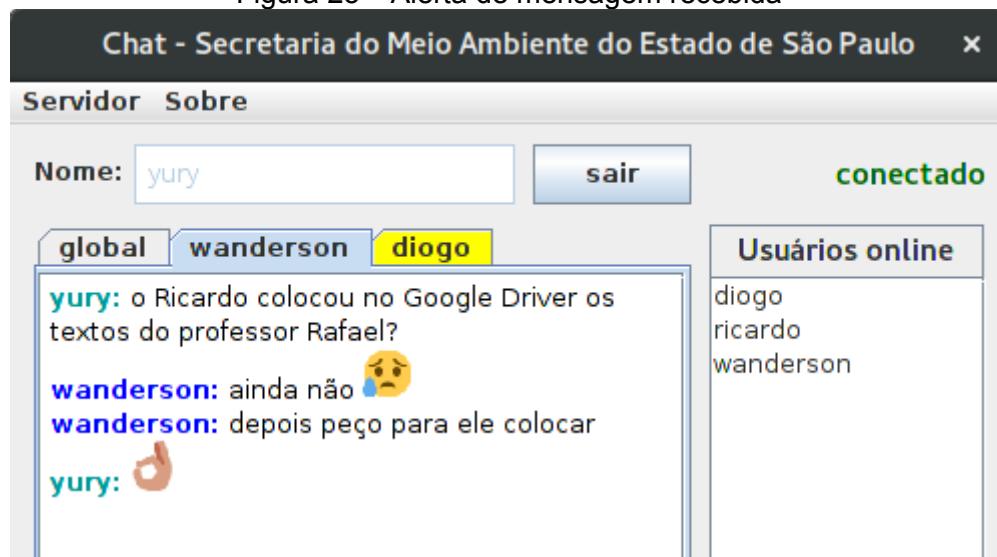
Fonte: Elaborada pelos autores

Figura 24 – Algumas mensagens privadas



Fonte: Elaborada pelos autores

Figura 25 – Alerta de mensagem recebida



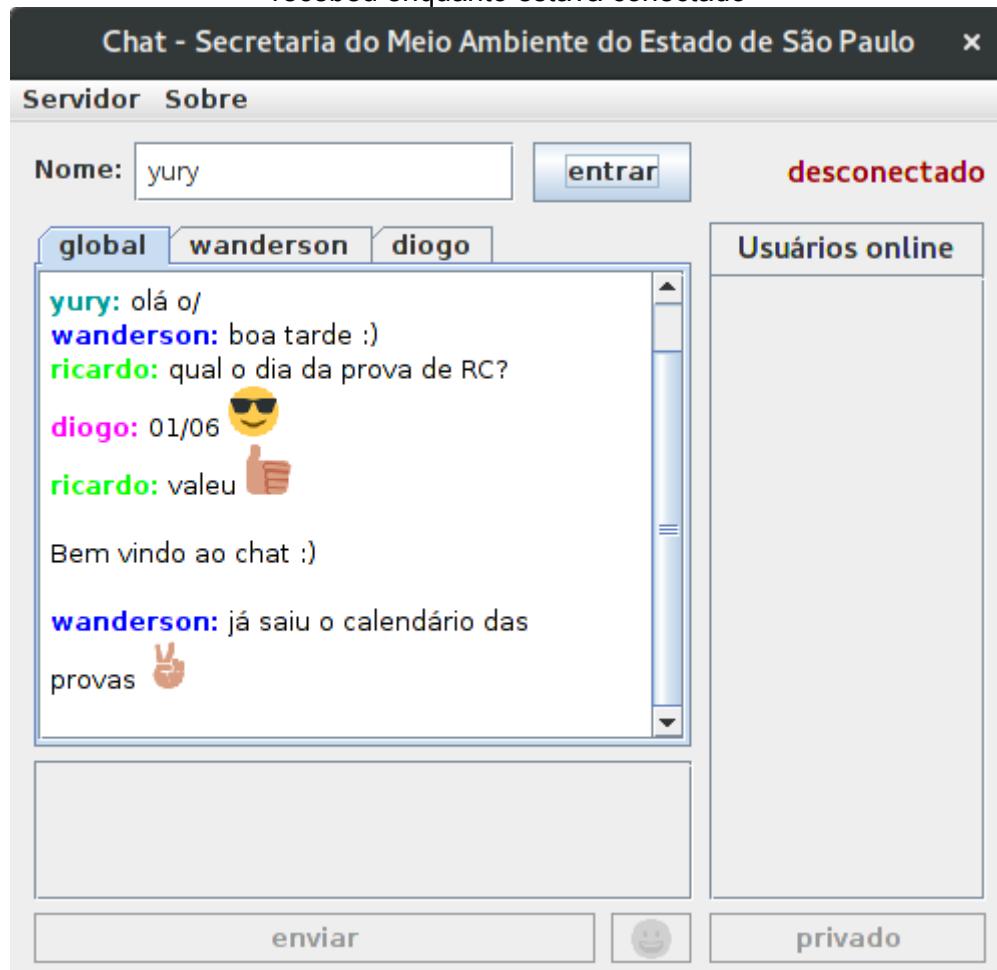
Fonte: Elaborada pelos autores

Figura 26 – Desativa o botão de enviar mensagens nas caixas privadas onde o usuário saiu do chat



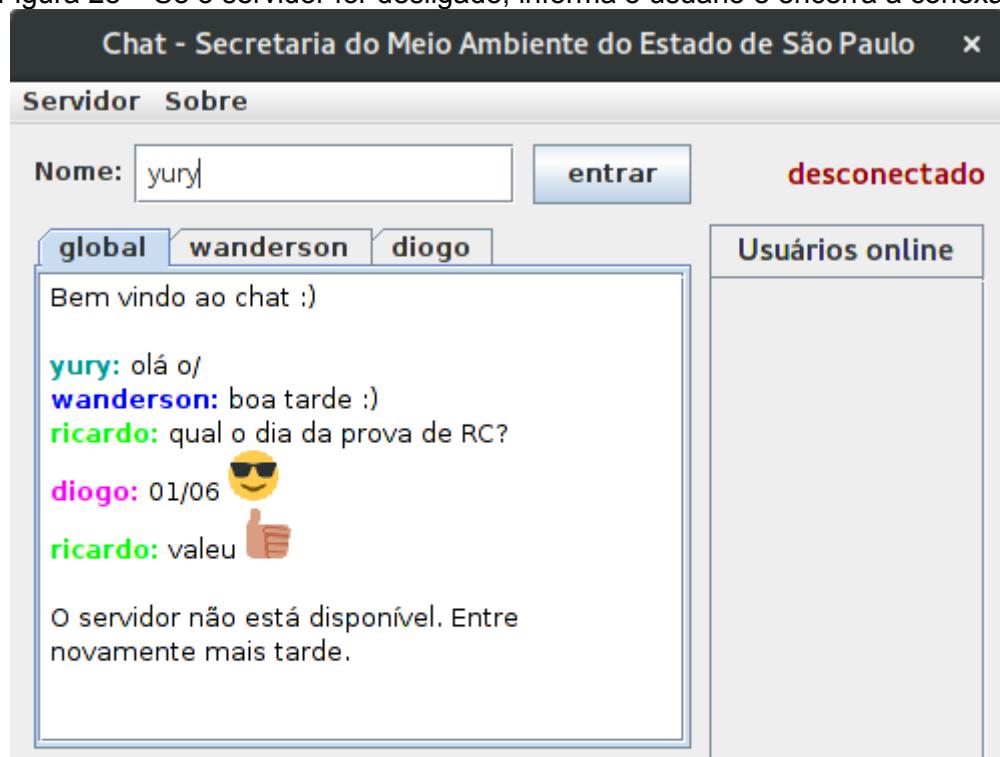
Fonte: Elaborada pelos autores

Figura 27 – Mesmo depois do usuário se desconectar, ele ainda pode ler as mensagens que recebeu enquanto estava conectado



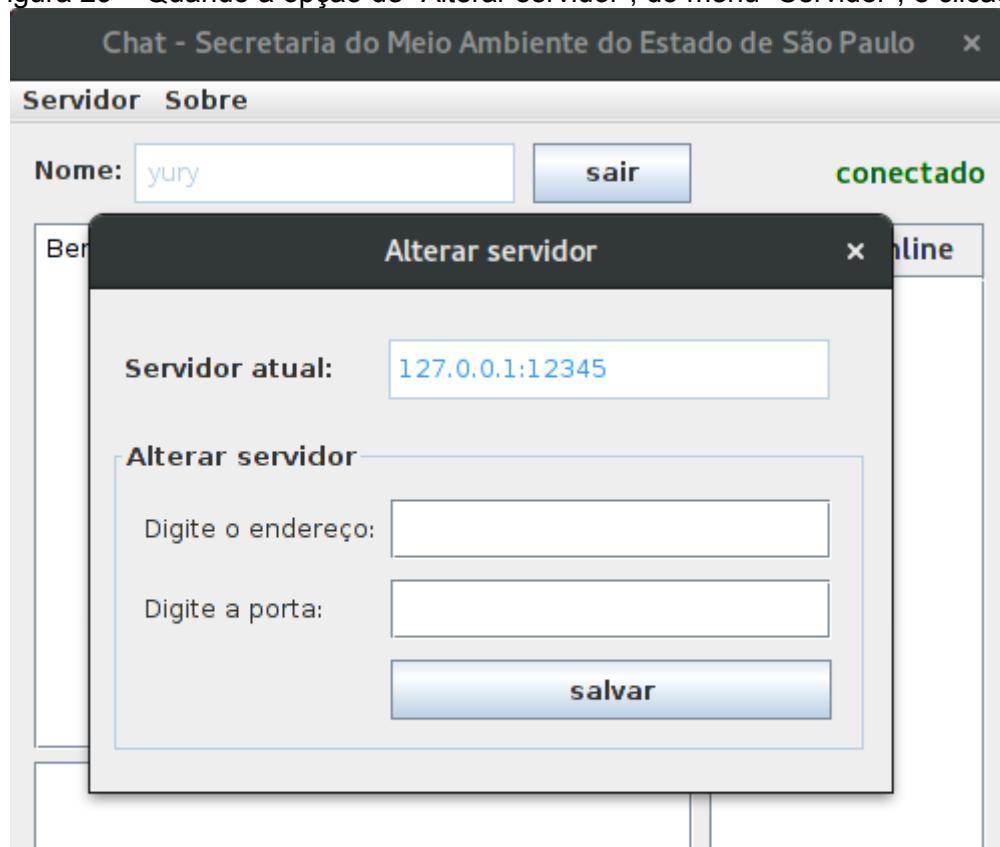
Fonte: Elaborada pelos autores

Figura 28 – Se o servidor for desligado, informa o usuário e encerra a conexão



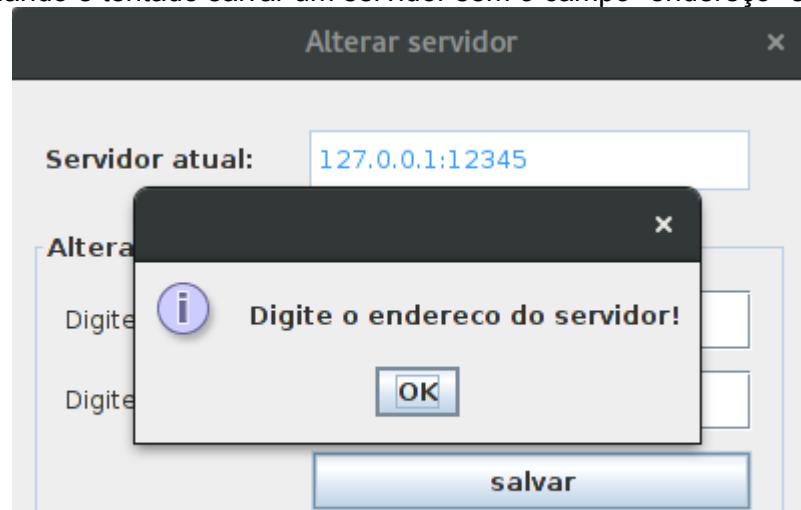
Fonte: Elaborada pelos autores

Figura 29 – Quando a opção de “Alterar servidor”, do menu “Servidor”, é clicada



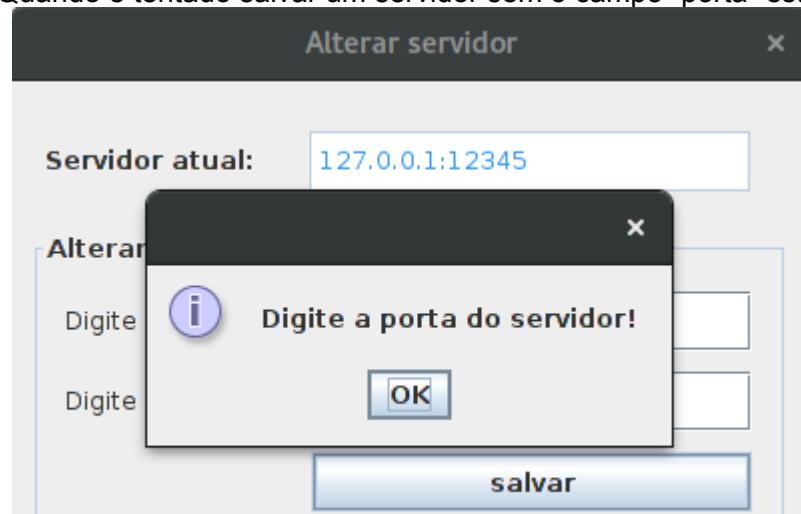
Fonte: Elaborada pelos autores

Figura 30 – Quando é tentado salvar um servidor sem o campo “endereço” estar preenchido



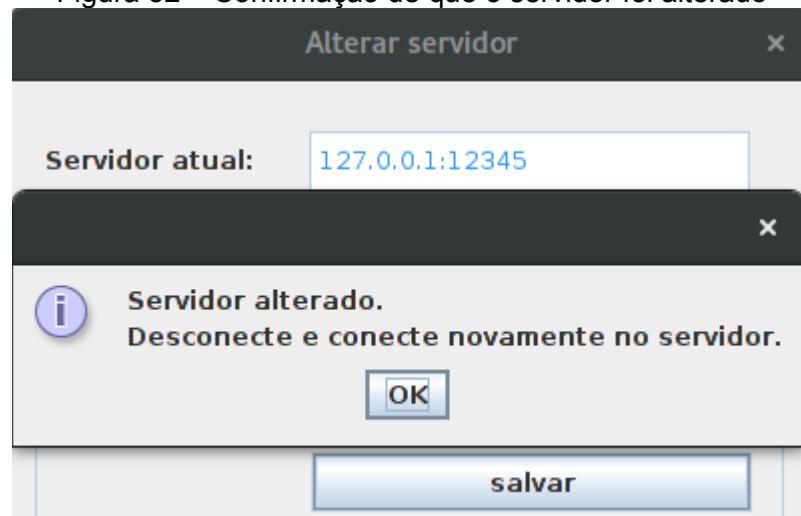
Fonte: Elaborada pelos autores

Figura 31 – Quando é tentado salvar um servidor sem o campo “porta” estar preenchido



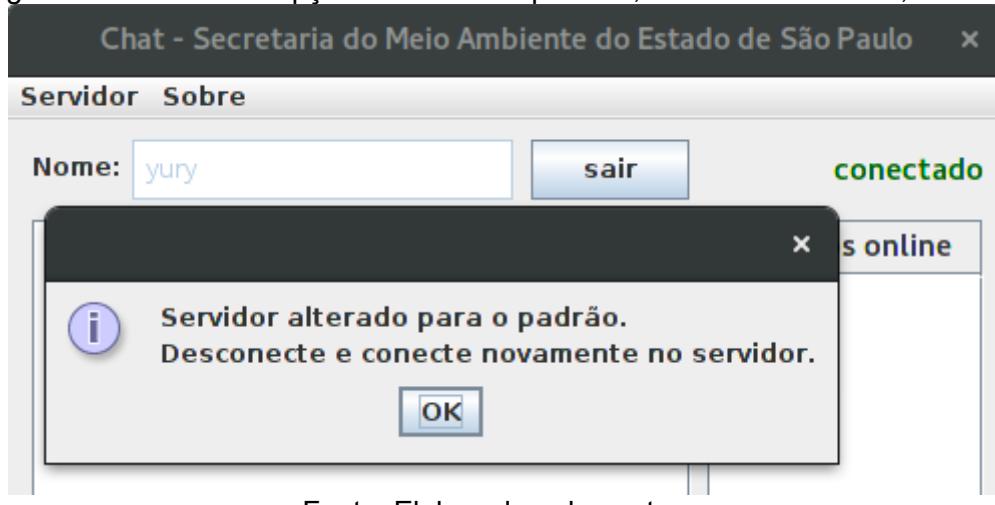
Fonte: Elaborada pelos autores

Figura 32 – Confirmação de que o servidor foi alterado



Fonte: Elaborada pelos autores

Figura 33 – Quando a opção de “Servidor padrão”, do menu “Servidor”, é clicada



Fonte: Elaborada pelos autores

Figura 34 – Quando a opção de “Chat”, do menu “Sobre”, é clicada



Fonte: Elaborada pelos autores

REFERÊNCIAS

ALENCAR, Márcio Aurélio dos Santos. **Fundamentos de Redes de Computadores**. Disponível em: <http://redeetec.mec.gov.br/images/stories/pdf/eixo_infor_comun/tec_man_sup/081112_fund_redes_comp.pdf>. Acesso em: 22 de maio de 2016.

ALMEIDA, Helóra. **Linha do tempo da comunicação virtual**. Disponível em: <<http://heloraalmeida.com.br/tecnologia/6368/>>. Acesso em: 22 de maio de 2016.

CAELUM. **Apêndice – Sockets**. Disponível em: <<https://www.caelum.com.br/apostila-java-orientacao-objetos/apendice-sockets/>>. Acesso em: 22 de maio de 2016.

CASTRO, Maria Cristina Felippetto de. **Fundamentos de Comunicação de Dados**. Disponível em: <http://www.feng.pucrs.br/~decastro/TPI/TPI_Cap3.pdf>. Acesso em: 22 de maio de 2016.

FERREIRA, Tadeu. **Redes I - EJA**. Disponível em: <<https://docente.ifrn.edu.br/tadeuferreira/disciplinas/2012.1/redes-i-eja>>. Acesso em: 22 de maio de 2016.

HTMLCLEANER. **HtmlCleaner**. Disponível em: <<http://htmlcleaner.sourceforge.net/index.php>>. Acesso em: 22 de maio de 2016.

JUNIOR, Almeida. **Redes de Difusão vs Redes Ponto a Ponto**. Disponível em: <<http://meubizu.com.br/redes-de-difus-o-vs-redes-ponto-ponto>>. Acesso em: 22 de maio de 2016.

OFICINA DA NET. **Quais são os meios físicos de transmissão de dados?** Disponível em: <<https://www.oficinadanet.com.br/artigo/redes/quais-sao-os-meios-fisicos-de-transmissao-de-dados>>. Acesso em: 22 de maio de 2016.

OK CONCURSOS. **Introdução à Redes de Computadores**. Disponível em: <<http://www.okconcursos.com.br/apostilas/apostila-gratis/130-informatica-para-concursos/1658-introducao-a-redes-de-computadores>>. Acesso em: 22 de maio de 2016.

RAULINO, Filipe. **Arquitetura e Protocolos de Rede TCP/IP**. Disponível em: <http://docente.ifrn.edu.br/filiperaulino/disciplinas/arquitetura-de-redes-de-computadores-e-tecnologia-de-implementacao-de-redes-info3v/copy_of_aulas/4.Arquitetura%20OSI%20-%20TCP/IP.pdf>. Acesso em: 22 de maio de 2016.

SECRETARIA DO MEIO AMBIENTE. **Secretaria do Meio Ambiente**. Disponível em: <<http://www.ambiente.sp.gov.br/quem-somos/o-sistema/secretaria-do-meio-ambiente/>>. Acesso em: 22 de maio de 2016.

TORRES, Gabriel. **O Modelo de Referência OSI para Protocolos de Rede**. Disponível em: <<http://www.clubedohardware.com.br/artigos/o-modelo-de-referencia-osi-para-protocolos-de-rede/1349/4>>. Acesso em: 22 de maio de 2016.

TRINDADE, Cristiano. **Conexões – O mundo dos sockets**. Disponível em: <<http://imasters.com.br/artigo/473/java/conexoes-o-mundo-dos-sockets>>. Acesso em: 22 de maio de 2016.

TWEMOJI. **Twitter Emoji for Everyone**. Disponível em: <<https://github.com/twitter/twemoji>>. Acesso em: 22 de maio de 2016.

WIKIVERSITY. **Introdução às Redes de Computadores/Programação com sockets**. Disponível em: <https://pt.wikiversity.org/wiki/Introdu%C3%A7%C3%A3o_%C3%A0s_Redes_de_Computadores/Programa%C3%A7%C3%A3o_com_sockets>. Acesso em: 22 de maio de 2016.

FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

Atividades Práticas Supervisionadas (laboratórios, atividades em biblioteca, Iniciação Científica, trabalhos individuais e em grupo, práticas de ensino e outras)

NOME: Diogo Henrique Soárez da Silveira

RA: 687339-8 **CURSO:** Ensino de computação

CAMPUS: Tatucapé

SEMESTRE:

SEMESTRE: 5º Semestre TURNO: Noite

TOTAL DE HORAS: _____

FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

Atividades Práticas Supervisionadas (laboratórios, atividades em biblioteca, Iniciação Científica, trabalhos individuais e em grupo, práticas de ensino e outras)

NOME: Ricardo Fernandes Souto
RA: 213CH15 CURSO: Ciéncia da computaçao
CAMPUS: Tatuapé SEMESTRE: 5º Semestre TURNO: Noite

DATA	ATIVIDADE	TOTAL DE HORAS	ASSINATURA	PROFESSOR
			ALUNO	
	levantamento bibliográfico			
	Testes			
	Desenvolvimentos			

TOTAL DE HORAS:

FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

Atividades Práticas Supervisionadas (laboratórios, atividades em biblioteca, Iniciação Científica, trabalhos individuais e em grupo, práticas de ensino e outras)

NOME: Wanderson Martins Oliveira

RA: C 20275-4

CAMPUS: Tatuaí

CURSO: Ciéncia da Computacão

50 *Symmetries*

SEMESTRE: 5º Semestre TURNO: Noite

TOTAL DE HORAS: _____

FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

atividades Práticas Supervisionadas (laboratórios, atividades em biblioteca, Iniciação Científica, trabalhos individuais e em grupo, práticas de ensino e outras)

Nome: Yury Rodrigues Anunciação

RA: C 203336 - 0

CAMPUS: Tatwāpē

SEMESTRE: SIS SEMESTRE: TURNO: Notas

CURSO: Ciéncias da Computação

SEMESTRE: 5º Semestre

TOTAL DE HORAS: _____