UNIVERSIDADE PAULISTA

CIÊNCIA DA COMPUTAÇÃO

DESENVOLVIMENTO DE UMA FERRAMENTA DE COMUNICAÇÃO EM REDE

SÃO PAULO

2018

CAÍQUE DOS SANTOS PINHEIRO - N892BD8

JOÃO VICTOR SEMEÃO MENA - D1375I-9

THIAGO OLIVEIRA SILVA - C77HBG6

DESENVOLVIMENTO DE UMA FERRAMENTA DE COMUNICAÇÃO EM REDE

SÃO PAULO

2018

Índice

1. Objetivo e motivação

O objetivo geral do trabalho, consiste em colocar em pratica conceitos que foram lecionados em sala de aula, a partir da criação de uma ferramenta multiplataforma de comunicação em rede, que permita que duas ou mais pessoas possam se comunicar em uma rede, utilizando o protocolo TCP/IP. Esta ferramenta será capaz de atender uma empresa fictícia do ramo ambiental, sendo mais especifico na área de poluição dos rios. Tal aplicação será composta por componentes gráficos, emoticons, transferência de arquivos e correio de voz. Será desenvolvido utilizando sockets em Berkeley e implementado em linguagem JAVA.

A motivação para o trabalho é resultante de um assunto intrínseco na sociedade em geral, que passa despercebido para a maioria das pessoas, como a poluição do rio tiete por conta das indústrias.

A Secretaria de Estado do Meio Ambiente deseja saber e controlar o nível das atividades industriais e domesticas que estão gerando poluição do Rio Tietê desde sua nascente em Salesópolis (SP) até a sua passagem pela região da grande São Paulo. Para tal ela precisa trocar informações das equipes de inspetores treinados e capacitados que estarão se revezando dentro de cada indústria, controlando os processos e passando informações online para a Secretaria.

Conscientizar e desenvolver o costume, nos usuários, de sempre buscar mais conhecimento de cunho ambiental e usar isso em prol de nosso próprio planeta.

2. Introdução

Rede de computadores é um conjunto de computadores interligados entre si através de um cabo, rádio, satélite, entre muitos outros, de forma que se comuniquem e troquem informações entre si. Todo equipamento conectado a uma rede de computadores que tenha autonomia para transmitir e receber dados são chamados de Host ou hospedeiro. O host de origem prepara o pacote e envia para o host de destino que recebe o pacote da rede, assim criando uma comunicação entre host ou entre usuários de locais diferentes. Uma rede não se compõe sempre de apenas dois usuários, ela pode partir de uma simples rede doméstica a vários continentes.

Uma rede de computadores ela pode ser:

- LAN (Local Area Network), que é uma rede local;
- MAN (Metropolitan Area Network), este tipo de rede abrange o espaço físico de uma cidade;
- WLAN (Wireless Local Area Network), rede através de uma onda de rádio de alta frequência;
- CAN (Campus Area Network), rede composta de vários edifícios;
- SAN (Storage Area Network), para recursos de armazenamento;
- PAN (Personal Area Network), pequena rede local (Domestica);
- WANs (Wide AreaNetwork), este tipo de rede abrange vários municípios e países.

A comunicação de dados entre locais remotos pode ser realizada através de um processo denominado conectividade, que envolve desde a conexão de computadores, meios e dispositivos de rede.

Em nosso trabalho, apresentaremos um exemplo de comunicação entre duas ou várias outras máquinas, através da arquitetura Cliente-servidor, utilizando protocolos TCP/IP e utilizando a API de Sockets de Berkeley.

A camada cliente, neste trabalho, terá a função de prover a interface para que os usuários possam manipular as informações, ou seja, através dela realiza-se a interação entre o usuário e o sistema. É desenvolvida para se conectar diretamente ao banco de dados, tendo como responsabilidade fazer as solicitações dos dados necessários ao servidor, sendo que este os processa e

devolve o resultado.

A Secretaria de Estado do Meio Ambiente, que deseja saber e controlar o nível das atividades industriais e domesticas que estão gerando poluição do Rio Tietê, utilizarão o programa no qual desenvolvemos, para trocar informações das equipes de inspetores dentro de cada indústria. O cliente (client), será responsável por prover a interface para que as equipes e dar às equipes todo o tipo de informação necessária e disponível. O servidor será responsável por armazenar as mensagens, cadastros, transmitir e receber informações das diversas equipes de inspetores que controlam os processos de poluição do Tietê e passando informações online para a Secretaria de Estado do Meio Ambiente.

3. Fundamentos da comunicação de dados em rede

A comunicação de dados é a teoria da ciência da computação que trata da comunicação entre computadores e dispositivos diferentes através de um meio de transmissão comum. A comunicação de dados necessita de uma infraestrutura de rede, composta de computadores, hosts, roteadores, switches, hubs, cabeamento estruturado, protocolos de comunicação, etc.

Uma rede de computadores é formada por um conjunto de módulos processadores capazes de trocar informações e compartilhar recursos, interligados por um subsistema de comunicação, isto é, quando há pelo menos dois ou mais computadores e outros dispositivos interligados de modo compartilhar recursos físicos e lógicos. Tal rede é constituída de três elementos básicos: hosts, meios de transmissão e roteadores. Ao conjunto dos roteadores e dos meios de transmissão dá-se a denominação de sub-rede. A sub-rede normalmente é operada por uma concessionária de serviços de telecomunicações como, por exemplo, a Embratel (dentre outras).

- Hosts são os equipamentos que geram os dados a serem transmitidos e os equipamentos que recepcionam estes dados. Serão designados por transmissores e receptores respectivamente.
- Meios físicos são todos aqueles por meio dos quais os bits transmitidos trafegarão: cabos de cobre, cabos de fibra óptica, links de micro-ondas, links de satélite, etc.
- Roteadores são computadores especializados que, ao receber dados por uma linha de entrada, devem encaminhá-los a determinada linha de saída. É um equipamento de comutação e cada um deles se constitui num nó da rede.

O software de rede é altamente estruturado, criando hierarquias de protocolos. Para reduzir a complexidade do projeto do software, a maioria das redes é organizada como uma pilha de camadas (ou níveis), colocadas uma sobre as outras. O objetivo de cada camada é oferecer determinados serviços às camadas superiores, isolando essas camadas dos problemas a serem resolvidos por ela. De certo modo, cada camada pode ser entendida como uma máquina

virtual que oferece serviços à camada acima dela. Uma rede, por sua vez, pode apresentar dois tipos principais de serviços para a transmissão dos dados: os serviços orientados à conexão e os serviços sem conexão. Os primeiros são baseados no sistema telefônico, os quais seguem três fases distintas: Conectar (teclar o número desejado), Transmitir (conversa) e desconectar (colocar o fone no gancho). Os segundos assemelham-se ao serviço postal, isto é, os bits são agrupados em pacotes os quais contém o endereço completo do destino e o endereço completo da origem. Além disso, cada pacote é roteado através da rede independentemente dos outros pacotes e a ordem de entrega dos pacotes não é garantida no receptor. Pela analogia com o serviço postal estes pacotes de bits são denominados “datagramas” (pela semelhança com o que ocorre com os telegramas).

Serão utilizados para alcançar o objetivo deste projeto, comunicação local em rede dos funcionários do SEMA, sockets de Berkeley, que são como “um ponto-final” de um fluxo de comunicação entre dois aplicativos através de uma rede. Com o uso de Sockets, podemos identificar unicamente um aplicativo na rede de comunicação IP. Um socket de rede pode ser utilizado em ligações de redes para se estabelecer um elo de comunicação bidirecional entre dois programas que utilizam o mesmo protocolo e que estão ligados na mesma rede. A interface padronizada dos sockets surgiu originalmente no sistema operacional Unix BSD (Berkeley Software Distribution); por isso eles costumam ser chamados de Berkeley Sockets. Por fim, o projeto será feito criando quatro projetos: Server, Cliente e Share. O projeto Server será responsável por fornecer os serviços de uma rede de computador, o projeto do Cliente realiza a interação dos usuários com o servidor desde a parte do login até tela principal (e das outras funções também, como envio de arquivos, entre outros), e o projeto Share, auxilia na organização das funções do projeto cliente.

3.1 Servidor

Modelo de servidor de chat, o serviço do computador que funciona como base deve, primeiro, abrir uma porta e ficar ouvindo até alguém tentar se conectar, como é exemplificado a baixo na linguagem Java.

```
import java.net.*;

public class Servidor {

    public static void main(String[] args) throws IOException {

        ServerSocket servidor = new ServerSocket(12345);

        System.out.println("Porta 12345 aberta!");

        // a continuação do servidor deve ser escrita aqui

    }

}
```

Com o objeto criado, a porta 12345 estava fechada e foi aberta. Se outro programa possui o controle desta porta neste instante, tal porta não funcionará, pois não é possível que dois programas utilizem uma mesma porta ao mesmo tempo.

Após abrir a porta, precisamos esperar por um cliente através do método accept da ServerSocket. Assim que um cliente se conectar, o programa continuará, por isso dizemos que esse método é “blocante”, pois segura a thread até que algo o notifique.

```
Socket cliente = servidor.accept();

System.out.println("Nova conexão com o cliente " +

cliente.getInetAddress().getHostAddress()

); // imprime o ip do cliente
```

Por fim, basta ler todas as informações que o cliente nos enviar:

```
Scanner scanner = new Scanner(cliente.getInputStream());

while (scanner.hasNextLine()) {

    System.out.println(scanner.nextLine());

}
```

Fechamos as conexões, começando pelo fluxo:

```
in.close();
cliente.close();
servidor.close();
```

O resultado é a classe a seguir:

```
public class Servidor {
    public static void main(String[] args) throws IOException {
        ServerSocket servidor = new ServerSocket(12345);
        System.out.println("Porta 12345 aberta!");
        Socket cliente = servidor.accept();
        System.out.println("Nova conexão com o cliente " +
            cliente.getInetAddress().getHostAddress()
        );
        Scanner s = new Scanner(cliente.getInputStream());
        while (s.hasNextLine()) {
            System.out.println(s.nextLine());
        }
        s.close();
        servidor.close();
        cliente.close();
    }
}
```

3.2 Cliente

O programa cliente envia mensagens para o servidor. O código a seguir é a parte principal e tenta se conectar a um servidor no IP 127.0.0.1 (máquina local) e porta 12345:

```
Socket cliente = new Socket("127.0.0.1",12345);
System.out.println("O cliente se conectou ao servidor!");
```

Queremos ler os dados do cliente, da entrada padrão (teclado):

```
Scanner teclado = new Scanner(System.in);
while (teclado.hasNextLine()) {
    // lê a linha e faz algo com ela
}
```

Basta ler as linhas que o usuário digitar através do buffer de entrada (in), e jogá-las no buffer de saída:

```
PrintStream saida = new PrintStream(cliente.getOutputStream());
Scanner teclado = new Scanner(System.in);
while (teclado.hasNextLine()) {
    saida.println(teclado.nextLine());
}
saida.close();
teclado.close();
```

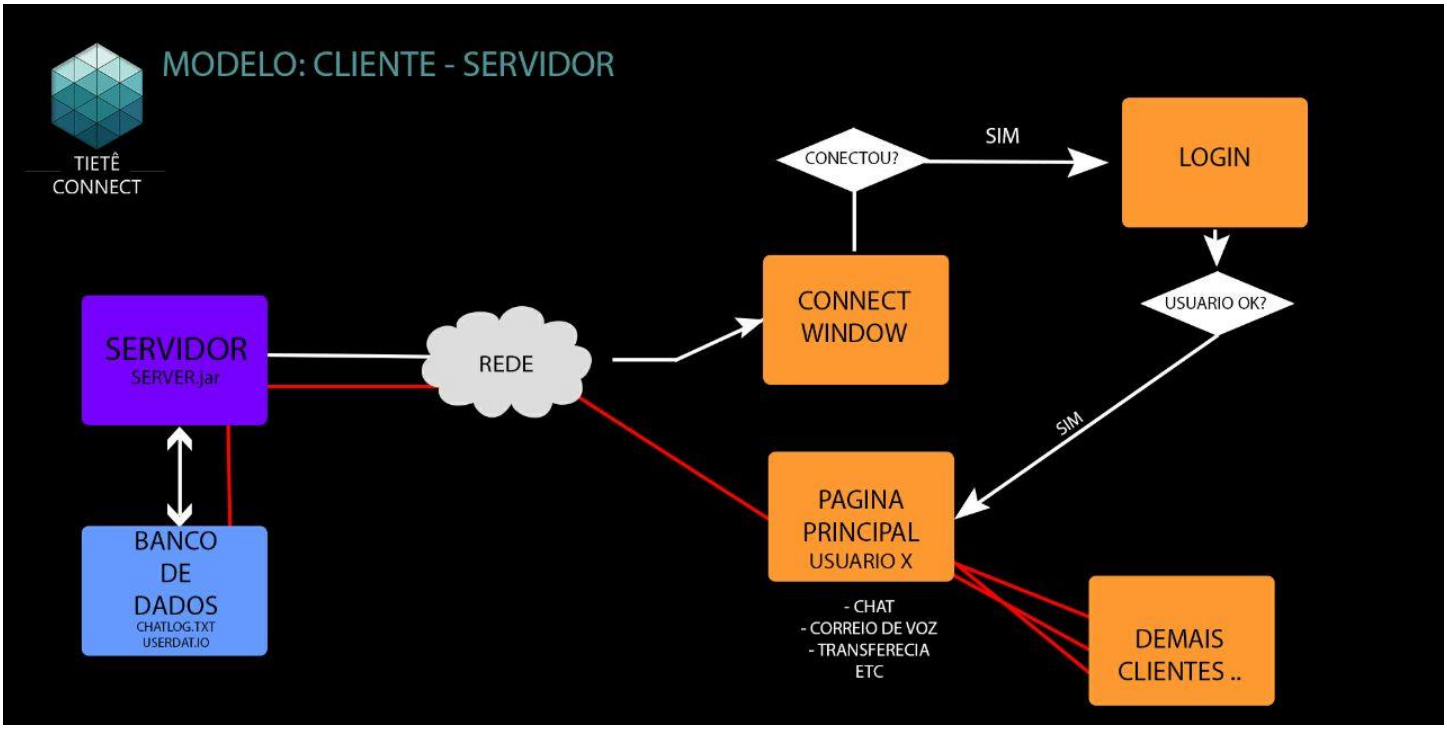
Para as classes Scanner e PrintStream, não há diferença de onde que se lê ou escreve os dados, o importante é que esse stream seja um InputStream / OutputStream.

Compilando todos as linhas de códigos:

```
public class Cliente {
```

```
    public static void main(String[] args)
                throws UnknownHostException, IOException {
        Socket cliente = new Socket("127.0.0.1", 12345);
        System.out.println("O cliente se conectou ao servidor!");
        Scanner teclado = new Scanner(System.in);
        PrintStream saida = new PrintStream(cliente.getOutputStream());
        while (teclado.hasNextLine()) {
            saida.println(teclado.nextLine());
        }
        saida.close();
        teclado.close();
        cliente.close();
    }
}
```

Para testar este sistema que apenas exemplifica os métodos utilizados neste trabalho. Primeiro deve-se rodar o servidor e, logo depois, o cliente. Tudo o que for digitado no cliente será enviado para o servidor.



(A imagem acima é uma demonstração de como funciona a arquitetura do programa.)

3.3 Multithreading

Thread é um pequeno programa que trabalha como um subsistema, sendo uma forma de um processo se autodividir em duas ou mais tarefas. É o termo em inglês para Linha ou Encadeamento de Execução. Essas tarefas múltiplas podem ser executadas simultaneamente para rodar mais rápido do que um programa em um único bloco ou praticamente juntas, mas que são tão rápidas que parecem estar trabalhando em conjunto ao q mesmo tempo.

Para que o servidor seja capaz de trabalhar com dois clientes ao mesmo tempo é necessário criar um thread logo após executar o método accept. O thread criada será responsável pelo tratamento dessa conexão, enquanto o laço do servidor disponibilizará a porta para uma nova conexão:

```
while (true) {
    Socket cliente = servidor.accept();
    // cria um objeto que vai tratar a conexão
    TratamentoClass tratamento = new TratamentoClass(cliente);
```

```
    // cria a thread em cima deste objeto
    Thread t = new Thread(tratamento);
    // inicia a thread
    t.start();
}
```

4. Plano de desenvolvimento da aplicação

Seguindo os tópicos de desenvolvimento do trabalho, onde primeiramente pesquisamos sobre os sockets de Berkeley. Essa API foi concebida para suportar diversos tipos de protocolos, logo ela suporta diversos formatos de dados, onde são armazenados em uma estrutura especial genérica. Para cada família de protocolos existem estruturas mais especificas e adequadas, como por exemplo para o protocolo IP, que vai usar uma estrutura que pertence à família AF_NET, que contem também os protocolos UDP, TCP e ICMP, facilitando assim o desenvolvimento do programa.

Pesquisamos também sobre os protocolos que utilizaríamos no desenvolvimento do trabalho, pesquisamos sobre o protocolo TCP, que é um dos principais protocolos da camada da internet de modelo TCP/IP, ele permite gerenciar os dados vindo da camada inferior, ou seja, o protocolo IP. Quando os dados são liberados ao protocolo IP, este guarda em datagramas IP (datagrama e o nome dado a uma mensagem sem conexão e sem confirmação). Ele funciona entregando em ordem (já que utiliza a transmissão de socket baseado em conexões) os datagramas do protocolo IP, ele verifica o fluxo de dados para evitar uma saturação na rede e, permitindo o início e o fim de uma comunicação de maneira correta, onde ele é um dos principais protocolos que consiste esse programa. Temos também o protocolo IP, que foi bem pesquisado, já que assim como o TCP faz parte dos principais protocolos da camada de internet de modelo TCP/IP, um dos protocolos mais importante da internet, pois permite a composição e o transporte dos datagramas, assegurando a entrega dos pacotes de dados, onde determina o destinatário da mensagem graças a três campos que são, o de endereço IP, máscara de sub rede e gateway, sendo um dos principais protocolos que fizeram parte do trabalho.

Pesquisamos também sobre diversos códigos que viriam ajudar a trazer um trabalho mais completo, trazendo assim todo o potencial da comunicação entre usuários na mesma rede, onde não ocorresse nenhuma falha. Essa etapa do trabalho foi realizada por todos.

4.1 Cronograma

DESENVOLVIMENTO DO PROJETO				
Tópicos	Datas entrega	Integrantes do Grupo		
		Caique	João Victor	Thiago
Objetivo e Motivação	19/mar			
Introdução	26/mar			
Fundamentos da Comunicação de dados em rede	02/abr			
Plano de Desenvolvimento da Aplicação	16/abr			
Projeto (estrutura) do Programa. Apresentação dos Grupos	23/abr			
relatório final	07/mai			
Desenvolvimento do software	A partir de 05/03			

Revisão
Responsável

4.2 Testes

A maior parte dos testes foi realizada pelo integrante do grupo: Thiago Oliveira, alguns foram realizados no próprio laboratório da instituição de ensino UNIP, campus marques, onde estabelecemos um computador como cliente e o outro como cliente/servidor, assim testamos as mensagens de texto, onde funcionaram com êxito, testamos o correio de voz, e o banco de dados que vai ter a função de guardar a mensagem de voz que será enviada pelo usuário do

programa, testamos a conexão entre um computador e outro onde também funcionou com êxito. Os testes seguirão até a entrega do trabalho, para evitarmos futuras falhas, etapa do trabalho realizada por todos.

4.3 Parte escrita

A parte escrita segue o cronograma que foi passado para cada grupo, onde consiste em 6 etapas, objetivo, motivação, introdução, fundamentos, desenvolvimento e por fim o projeto. Já foram entregues 4 etapas da parte escrita sendo desenvolvida com base no programa. etapa do trabalho está sendo realizada pelos integrantes Caíque Pinheiro e Joao Victor.

A conclusão será feita assim que todas os tópicos do trabalho forem concluídos e todos os testes do programa serem realizados tendo previsão para o dia de entrega da APS, dia 31/05/2018.

5. Projeto (estrutura) do programa

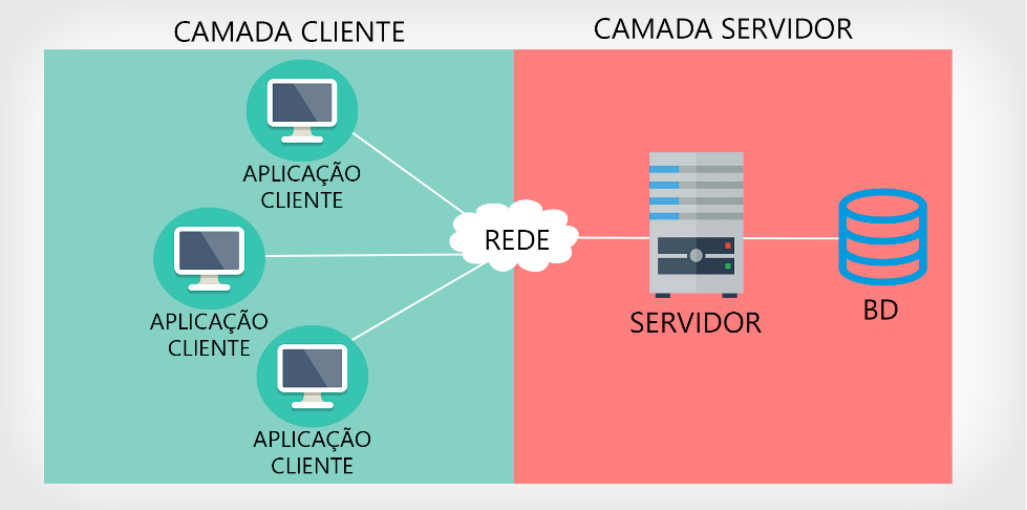
O programa Tietê Connect utiliza a arquitetura Cliente-Servidor utilizando o protocolo TCP/IP. Esta arquitetura é uma arquitetura na qual o processamento da informação é dividido em módulos ou processos distintos. Um processo é responsável pela obtenção dos dados (os clientes), e outro responsável pela manutenção da informação (servidor). Normalmente o servidor é uma máquina que atua como depósito de dados e funciona como um sistema gerenciador de banco de dados (SGBD), que é o caso de nosso projeto.

Os clientes e o servidor desta arquitetura se comunicam através de uma rede. A comunicação entre o cliente e o servidor é transacional e cooperativa.

A característica transacional indica que o servidor envia apenas os resultados relevantes do pedido do cliente. Assim, a quantidade de dados transferida tende a ser a menor quantidade necessária para o cliente executar seu trabalho.

A natureza cooperativa do paradigma cliente/servidor significa que ocorre um processamento significativo e colaborativo nos extremos cliente e servidor.

A imagem abaixo representa a estrutura do projeto:



5.1 Módulo Cliente

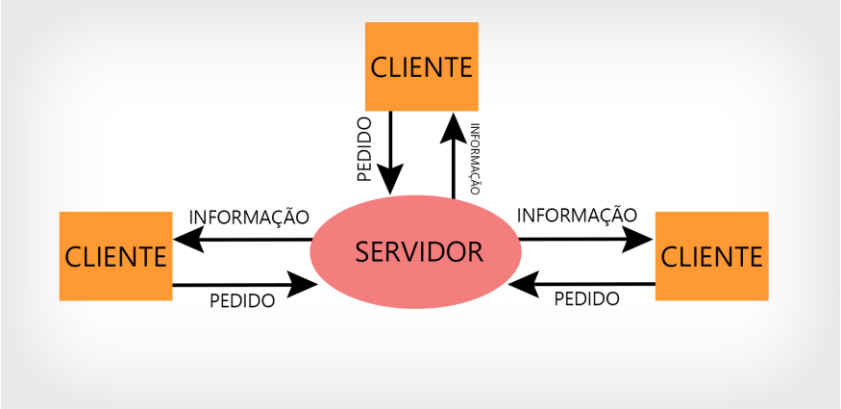
O cliente é a parte que interage com o usuário, é considerada a estação de trabalho em nosso projeto e possui a interface que o usuário utiliza para requisitar as tarefas ao servidor, sendo chamado de front-end da aplicação.

A principal característica do módulo Cliente, é de que são solicitados serviços ao servidor. Os principais serviços em nosso projeto são:

- Permissão de Login – Cabe ao servidor decidir se o usuário pode se conectar à rede ou não;
- Permissão de novo registro - Cabe ao servidor decidir se pode haver um novo registro de usuário ou não;
- Ao realizar um Login – Logo após a conexão ser estabelecida do servidor para o cliente, a aplicação cliente requisita ao servidor o serviço de mostrar quais usuários estão conectados na rede e quais estão online ou off-line;
- Atualizar informações – Caso o usuário queira alterar a senha, mudar de nome ou o status;
- Trocar mensagens com outros usuários – Caso o usuário queira iniciar um chat com qualquer outro usuário;
- Iniciar comunicação através de chamada de voz – Este serviço só está disponível quando o servidor permitir.

5.2 Módulo Servidor

Servidor é o processo que responde a uma mensagem solicitando a realização de alguma tarefa por parte do cliente, sendo assim é o de back-end do nosso projeto. A imagem abaixo demonstra como o módulo funciona:



O servidor oferece serviços a diversos clientes, pois nosso projeto permite a conexão de múltiplos usuários simultaneamente. Também é responsável por armazenar todos os dados dos usuários em um banco de dados criado pelo próprio servidor.

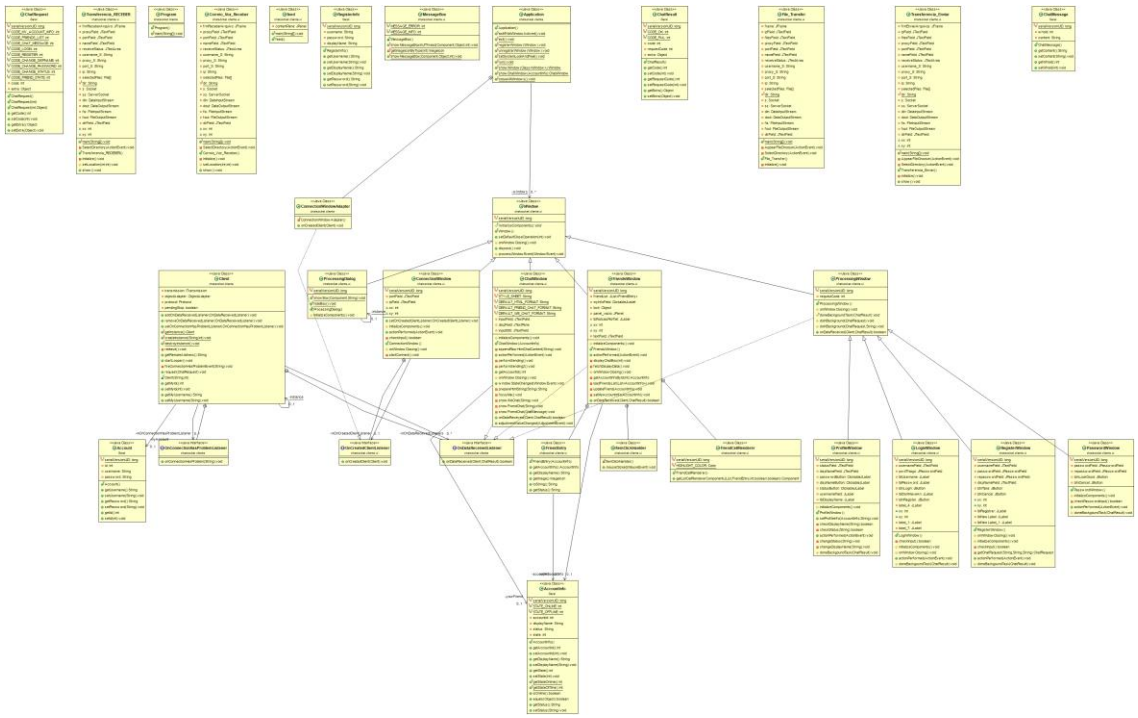
Algumas das características do servidor:

- Sua execução é contínua;
- Recebe e responde solicitações dos clientes;
- Não se comunica com outros servidores;
- Atende a diversos clientes simultaneamente.

5.3 Diagramas UML

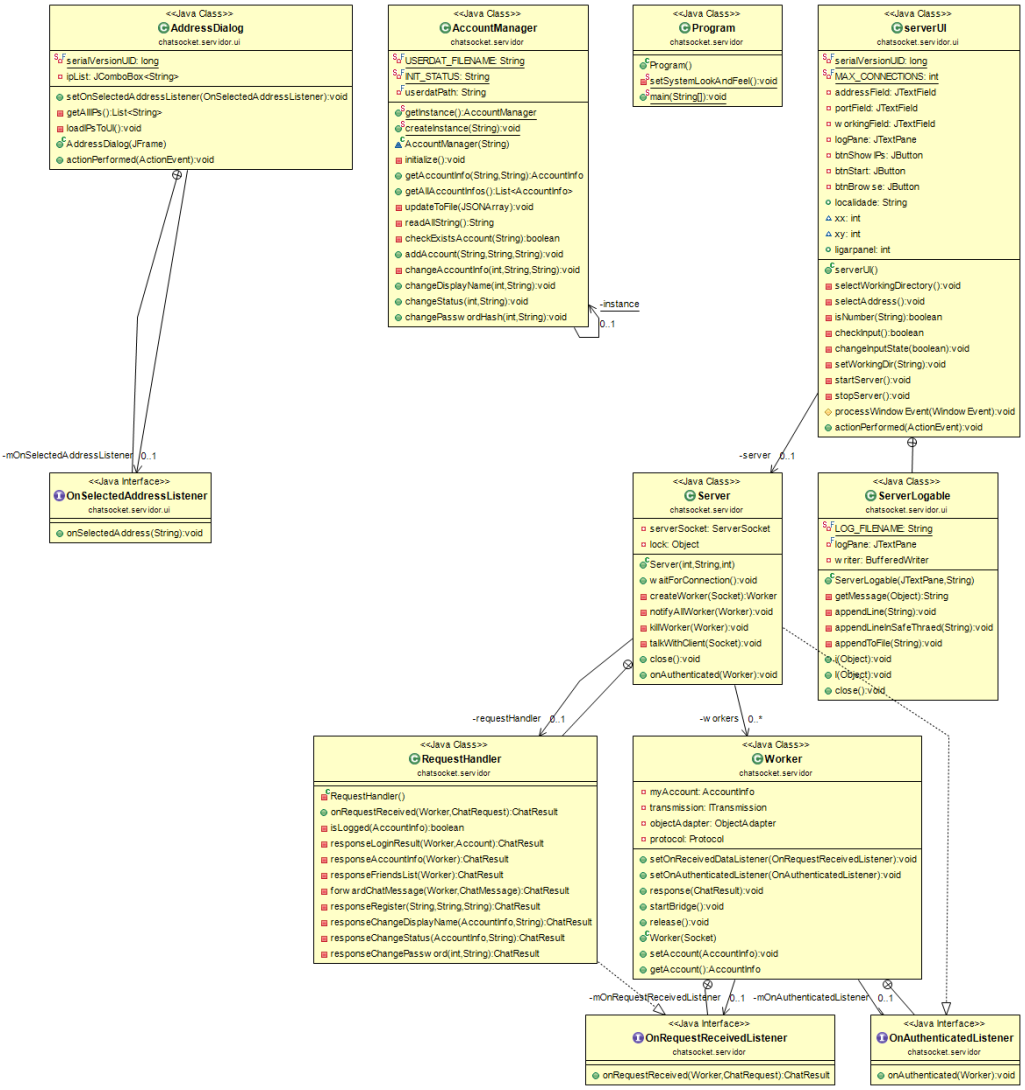
Para melhor entendimento, abaixo mostraremos alguns diagramas UML da aplicação.

Diagrama UML: Cliente:



A classe principal do programa Cliente (Window) faz diversas interconexões com as demais classes.

Diagrama UML: Servidor:



A classe principal do programa Servidor (Server) faz diversas interconexões com as demais classes.

5.4 Protótipo das telas do software: Cliente

Tela inicial do cliente até o login:

Tela de cadastro:

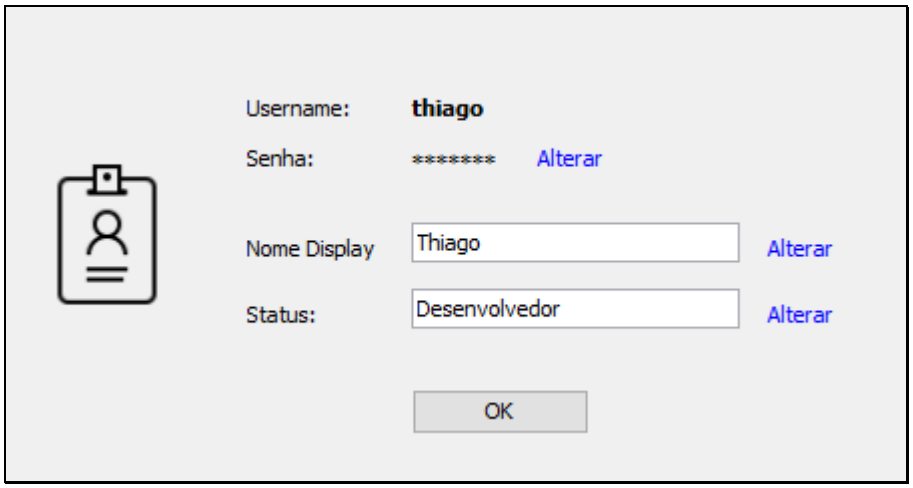
É nesta tela que novos usuários são registrados.

Tela Principal da aplicação após realizar o login:

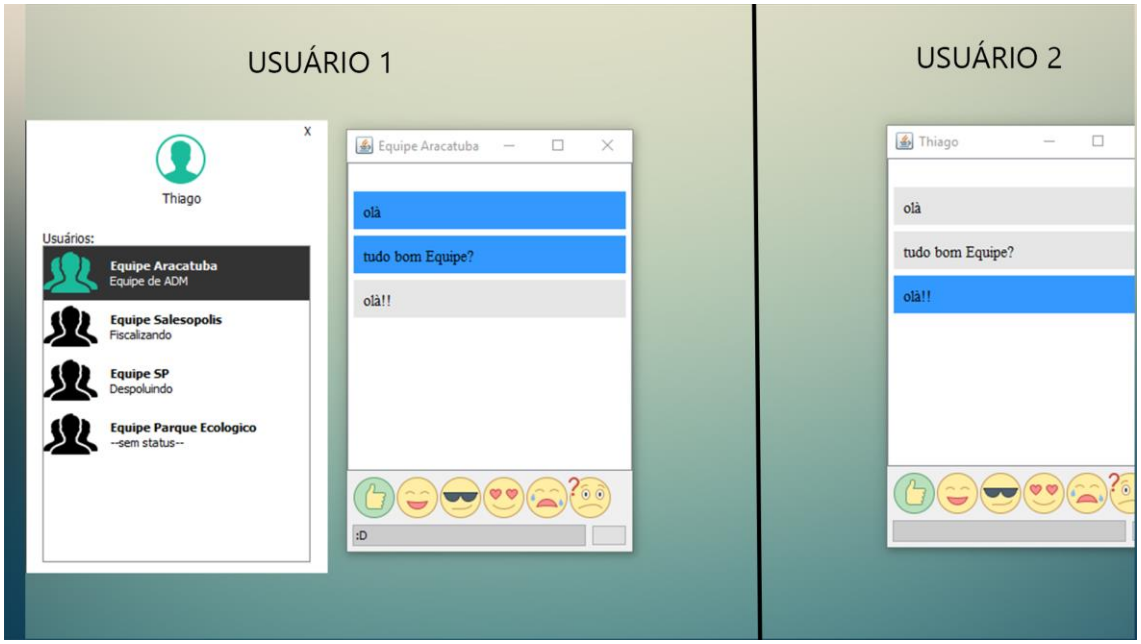
Identificação de cada ícone:

Páginas abertas ao clicar nos ícones da barra esquerda:

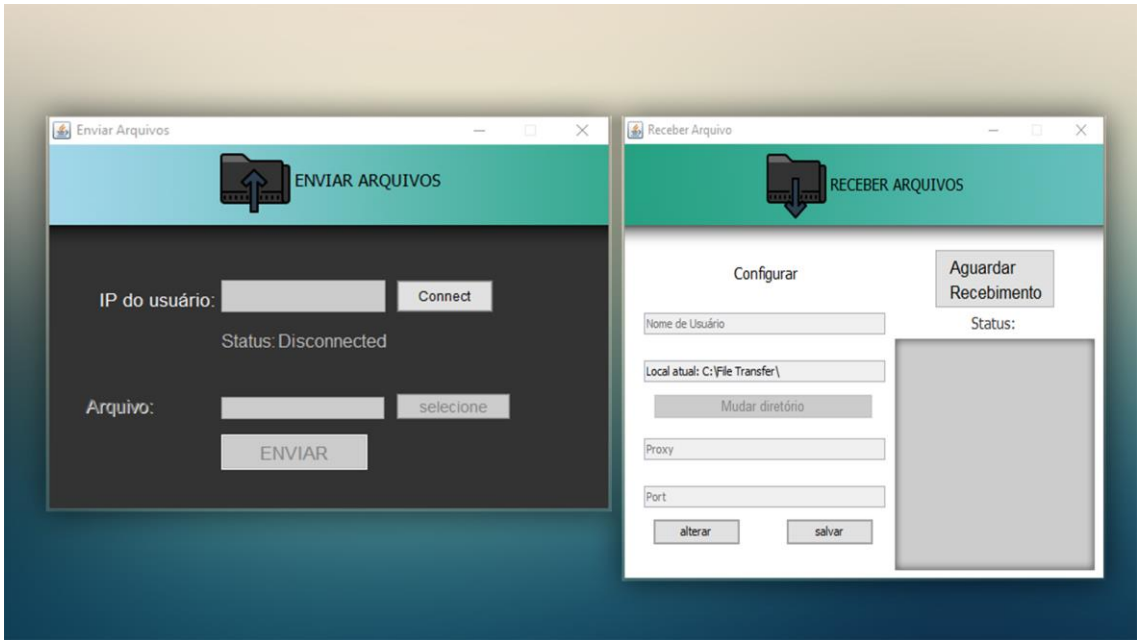
Demonstração de tela para alterar informações do usuário:



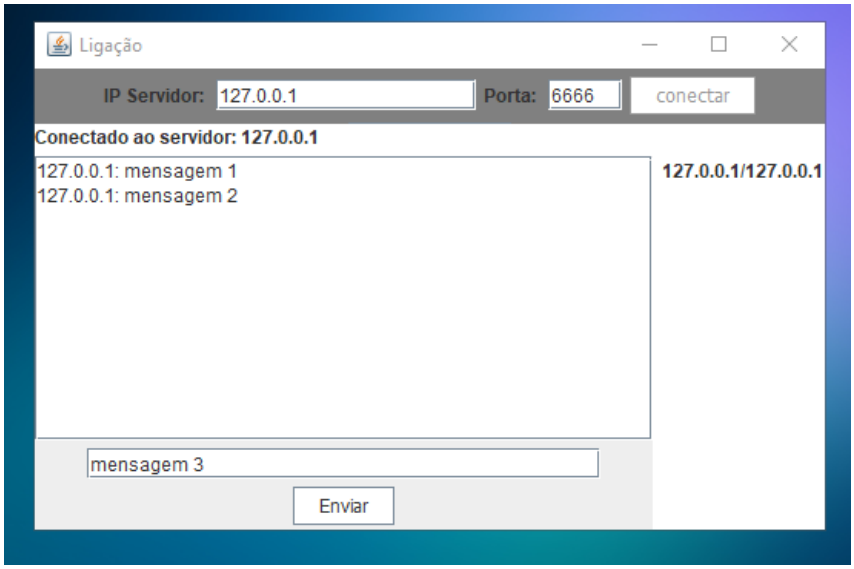
Demonstração do chat:



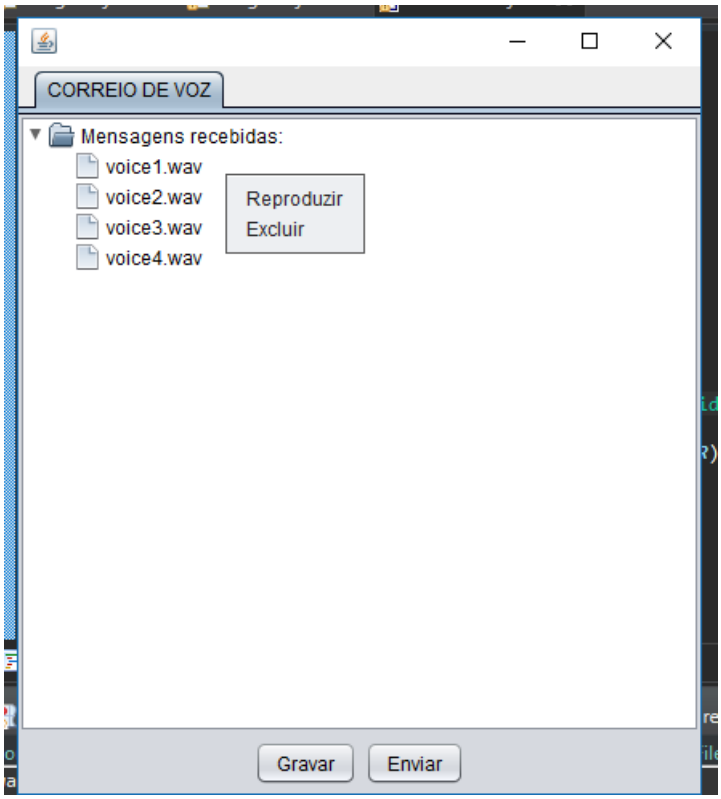
Demonstração da tela de Transferência de Arquivos:



Demonstração de tela para Chamada de voz:

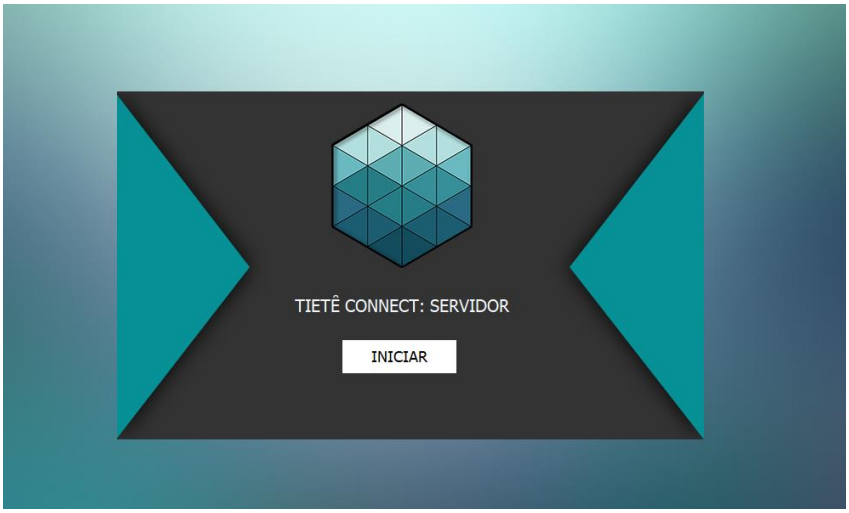


Demonstração de tela para Correio de voz:

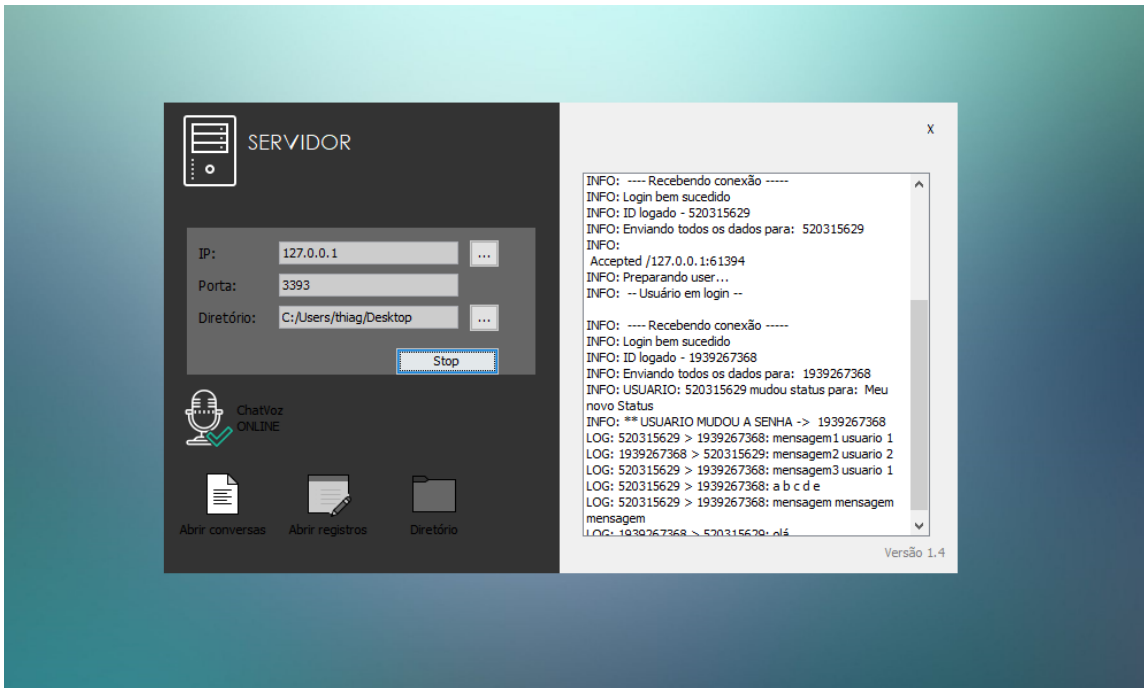


5.5 Protótipo das telas do software: Servidor

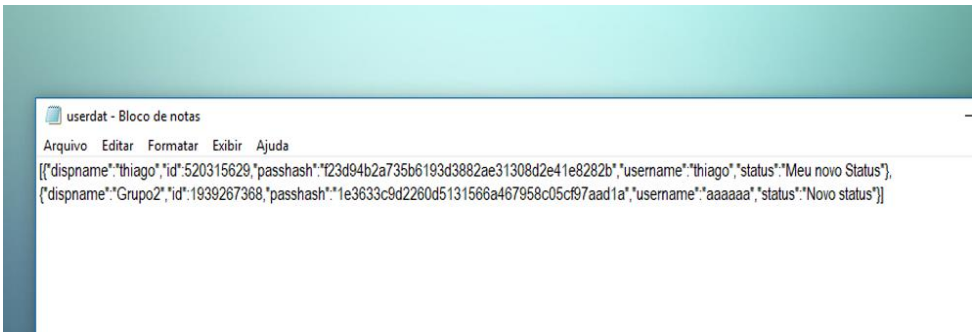
Página de boas-vindas:



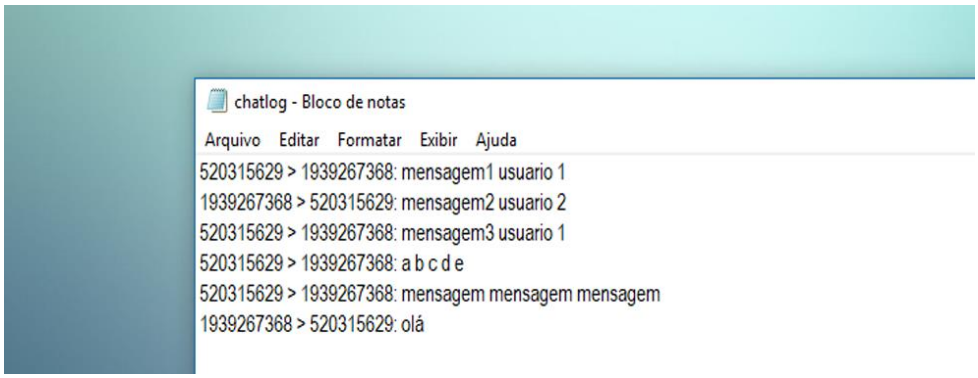
Página Principal do Servidor:



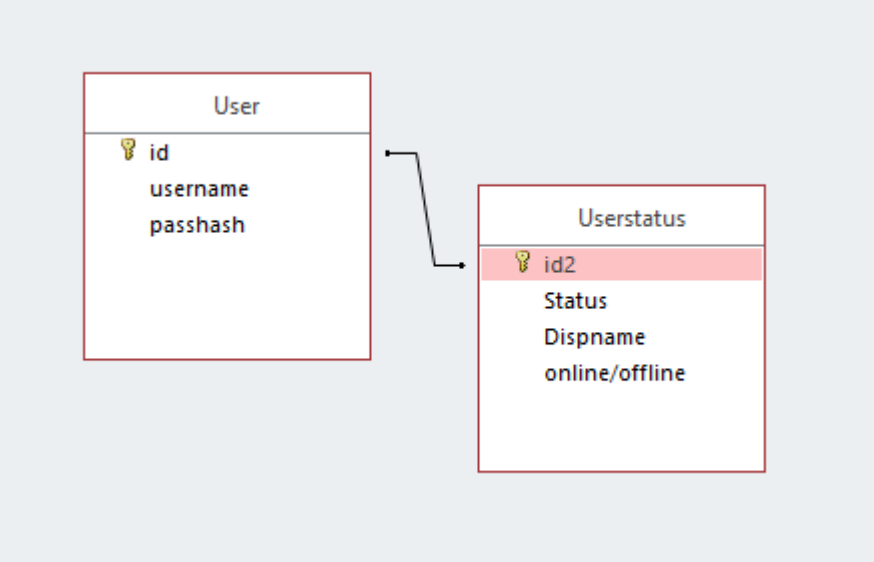
Demonstração do arquivo que armazena dados dos usuários:



Arquivo que armazena as mensagens digitadas no chat:



5.6 Diagrama DER do banco de dados



Conclusão

Neste trabalho abordamos o assunto de comunicação em rede entre duas ou mais máquinas, através da arquitetura Cliente-servidor, utilizando protocolos TCP/IP e utilizando a API de Sockets de Berkeley. Concluímos que os protocolos TCP/IP mantem intactas as conexões entre transmissor e receptor mesmo que algumas máquinas, ou linhas de transmissão intermediárias, deixem de operar, tornando-se um dos principais protocolos utilizados mundialmente.

Cumprimos todos os objetivos que nos foram propostos de acordo com os critérios e regras pré-estabelecidos pelo orientador em questão.

Este trabalho foi muito de bastante importância para a compreensão e conhecimento do grupo, pois compreender melhor nos permitiu desenvolver e aperfeiçoar competências em organização, comunicação da informação, programação, redes de computadores, entre outros.

Bibliografia

Caelum (2015). “Java e Orientação a Objetos”. Apostila do Curso FJ-11. Disponível em: <https://www.caelum.com.br/apostila-java-orientacao-objetos/apendice-sockets/>.

Stackoverflow (2015). “O que é uma Thread? Como ela funciona?”. Disponível em: <https://pt.stackoverflow.com/questions/95233/o-que-%C3%A9-uma-thread-como-ela-funciona>.

Ronaldo Lanhellas (2017). “Trabalhando com Threads em Java”. Disponível em: <https://www.devmedia.com.br/trabalhando-com-threads-em-java/28780>

Pereira, S. L. (2010). “Algoritmos e Lógica de Programação em Java: uma abordagem didática”1ª edição. São Paulo. Érica.

Natarajan Meghanathan (2015). “A Tutorial on Socket Programming in Java”. Jackson State University Jackson, MS 39217, USA. Disponível em: <http://www.jsums.edu/nmeghanathan/files/2015/05/CSC437-Fall2013-Java-Socket-Programming-Manual.pdf?x61976>

Nikhil Shetty (2006). “Socket Programming”. Disponível em: <https://inst.eecs.berkeley.edu/~ee122/sp06/LectureNotes/Socket%20Programming.pdf>

Jasmine J. Ahuja (1997). “Client-Server Applications in Java”. PACE UNIVERSITY, NY USA. Disponível em: <http://support.csis.pace.edu/csisweb/docs/msthesis/ahujajasmine.pdf>

Código fonte

Como o programa possui muitas linhas de código, colocaremos abaixo apenas os métodos principais de cada projeto (Cliente, Servidor).

Classe Program (pacote chatsocket.cliente, projeto Client)

```
package chatsocket.cliente;



import chatsocket.cliente.ui.ConnectionWindow;

public class Program {
	public static void main(String[] args) {
		Application.setSystemLookAndFeel();
		Application.run();


	}
}
```

Classe Program (pacote chatsocket.servidor, projeto Servidor)

```
package chatsocket.servidor;

import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;

import chatsocket.servidor.ui.serverUI;
import voicechat.networking.server.ServerControllerV2;



public final class Program {
	private static void setSystemLookAndFeel() {
		try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
		} catch (ClassNotFoundException | InstantiationException | IllegalAccessException
				| UnsupportedLookAndFeelException e) {
		}
	}

	public static void main(String[] args) {
		setSystemLookAndFeel();
		new serverUI().setVisible(true);
```

```
            }
        }
```

Todo o código fonte estará disponível no CD que será entregue junto com o trabalho.