

UNIVERSIDADE PAULISTA – UNIP

**GABRIEL DE ALMEIDA BATISTA
FELIPE DA SILVA BORGES NEVES
IGOR FAGGION SILVEIRA SANTOS
ANDERSON ALVES SCHINAID**

**DESENVOLVIMENTO DE UMA FERRAMENTA PARA COMUNICAÇÃO EM REDE
CHAT E CHAMADA DE VOZ**

**SÃO PAULO
2017**

**GABRIEL DE ALMEIDA BATISTA
FELIPE DA SILVA BORGES NEVES
IGOR FAGGION SILVEIRA SANTOS
ANDERSON ALVES SCHINAID**

**DESENVOLVIMENTO DE UMA FERRAMENTA PARA COMUNICAÇÃO EM REDE
CHAT E CHAMADA DE VOZ**

Atividade prática supervisionada e apresentada ao curso Ciência da Computação, para fins de conhecimento na área.

Orientador: Arthur Battaglia.

**SÃO PAULO
2017**

DEDICATÓRIA

Dedicamos este trabalho primeiramente a Deus por ter nos dados vida até este presente momento e aos nossos pais que nos educaram com força e amor incondicional.

AGRADECIMENTOS

Agradecemos em primeiro lugar a Deus por ser à base das nossas conquistas.

Aos nossos pais, por acreditarem e terem interesse em nossas escolhas, apoiando-nos e esforçando-se junto a nós para que supríssemos todas elas.

Ao professor Arthur Battaglia, pela dedicação em suas orientações prestadas na elaboração deste trabalho, nos incentivando e colaborando no desenvolvimento de nossas ideias.

*“O mundo é um enigma inofensivo, tornado
terrível pela nossa insana tentativa de
interpreta-lo como se existisse alguma
verdade secreta.”*

(Umberto Eco)

RESUMO

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

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

ABSTRACT

Computer Networks is a set of devices (commonly known as nodes) connected by communication links. A node can be a computer, a printer or another device for sending and / or receiving data that are connected to other nodes in the network. A computer network allows the user to share a huge amount of information and send messages to each other, including texts, images, audios and videos. The operation of a computer network has as basic objectives to provide reliable communication between the various information systems, to improve the flow and access to information, as well as to make administrative decisions easier, facilitating the communication between its users. The objective of this work is to clearly evaluate the concept of networks, their use and their applicability, through a network communication tool developed in the Java language to meet the needs of the National Environment Council (CONAMA), which wants to know what industrial activities are generating pollution from the Tietê River from its source in Salesópolis (SP) until its passage through the region of Greater São Paulo. In order to do this, it needs to exchange information from the teams of trained and trained inspectors who will be taking turns within each industry, controlling the processes and passing information online to the Secretariat. The work carried out was qualitative, based on works published by expert scholars, who have already pointed out the concepts, uses, applications, performance and failures of certain types of networks.

Keywords: information network; Berkeley; TCP/IP; sockets.

LISTA DE ILUSTRAÇÕES

Imagem 01 – Classificação de processadores por escala.	Pag. 40.
Imagem 02 – Posição da interface socket.	Pag. 49.
Imagem 03 – Sockets usados como entrada de dados.	Pag. 49.
Imagem 04 – Uso de sockets na comunicação.	Pag. 50.
Imagem 05 – Endereço de socket.	Pag. 51.
Imagem 06 – Diagrama de fluxo para uma comunicação TCP.	Pag. 52.
Imagem 07 – Diagrama de fluxo para a comunicação UDP.	Pag. 53.
Imagem 08 – DER do banco de dados.	Pag. 63.
Imagem 09 – MER do banco de dados.	Pag. 63.
Imagem 10 – Diagramas da UML 2.0.	Pag. 67.
Imagem 11 – Caso de uso login.	Pag. 68.
Imagem 12 – Caso de uso cadastro.	Pag. 70.
Imagem 13 – Caso de uso chat.	Pag. 71.
Imagem 14 – Caso de uso informação.	Pag. 72.
Imagem 15 – Diagrama do menu.	Pag. 74.
Imagem 16 – Diagrama de cadastro.	Pag. 75.
Imagem 17 – Diagrama do chat.	Pag. 76.
Imagem 18 – Diagrama da rede.	Pag. 77.
Imagem 19 – Protótipo de Login.	Pag. 78.
Imagem 20 – Protótipo de Chat.	Pag. 78.
Imagem 21 – Protótipo de Informação.	Pag. 79.
Imagem 22 – Protótipo de Cadastro.	Pag. 79.
Imagem 23 – Login.	Pag. 87.
Imagem 24 – Informação.	Pag. 88.
Imagem 25 – Cadastro.	Pag. 88.
Imagem 26 – Chat 01.	Pag. 89.
Imagem 27 – Chat 02.	Pag. 89.
Imagem 28 – Gabriel de Almeida Batista.	Pag. 97.
Imagem 29 – Felipe da Silva Borges Neves.	Pag. 98.
Imagem 30 – Igor Faggion Silveira Santos.	Pag. 99.
Imagem 31 – Anderson Alves Schinaid.	Pag. 100.

LISTA DE TABELAS E GRÁFICOS

Tabela 01 – DD + Trigramação e-mail.	Pag. 64.
Tabela 02 – DD + Trigramação usuário.	Pag. 64.
Tabela 03 – Tabela cadastro.	Pag. 66.
Tabela 04 – Tabela chaves candidatas.	Pag. 66.
Tabela 05 – Fluxo principal login.	Pag. 69.
Tabela 06 – Fluxo alternativo 01 login.	Pag. 69.
Tabela 07 – Fluxo alternativo 02 login.	Pag. 69.
Tabela 08 – Fluxo exceção login.	Pag. 69.
Tabela 09 – Fluxo principal cadastro.	Pag. 70.
Tabela 10 – Fluxo alternativo cadastro.	Pag. 70.
Tabela 11 – Fluxo exceção cadastro.	Pag. 71.
Tabela 12 – Fluxo principal chat.	Pag. 72.
Tabela 13 – Fluxo exceção chat.	Pag. 72.
Tabela 14 – Fluxo principal informação.	Pag. 73.

LISTA DE ABREVIATURA E SIGLAS

ACM – *Association for Computing Machinery.*
ANS – *Advanced Network & Service.*
ANSNET – *Advanced Network Services Network.*
API – *Application Programming Interface.*
ARPA – *Advanced Research Project Agency.*
ARPANET – *Advanced Research Project Agency Network.*
ATA – *Analog Telephone Adapter.*
AT&T – *American Telephone and Telegraph.*
AWT – *Abstract Window Toolkit.*
CERN – *Conseil Européen pour la Recherche Nucléaire.*
CONAMA – *Conselho Nacional do Meio Ambiente.*
CSNET – *Computer Science Network.*
CRTP – *Compressed Real-Time Protocol.*
DBA – *Administrador de Banco de Dados.*
DD – *Dicionário de Dados.*
DDL – *Data Definition Language.*
DER – *Diagrama de Entidade e Relacionamento.*
DNS – *Domain Name System.*
DoD – *Departamento de Defesa dos Estados Unidos.*
ER – *Entidade e Relacionamento.*
FN – *Forma Normal.*
GB – *Giga Byte.*
GUI – *Graphic User Interface.*
GVT – *Global Village Telecom.*
HD – *Hard Disk Drive.*
HTTP – *Hyper Text Transfer Protocol.*
IDE – *Integrated Development Environment.*
IETF – *Internet Engineering Task Force.*
IMPs – *Interface Message Processors.*
Inc – *Incorporation.*
IO – *Input e Output.*
IP – *Internetworking Protocol.*

IRC - *Internet Relay Chat*.

ISP – *Internet Service Provider*.

ISO – *International Standards Organization*.

ITU-T – *International Telecommunications Union - Telecommunications standardization sector*.

JCP – *Java Community Process*.

JDBC – *Java Database Connectivity*.

JDK – *Java Developer's Kit*.

JRE – *Java Runtime Environment*.

JSE 8 – *Java Standard Edition 8*.

LAN – *Local Area Network*.

MAN – *Metropolitan Area Networks*.

MER – *Modelo de Entidade e Relacionamento*.

NCP – *Network Control Protocol*.

NSF – *National Science Foundation*.

NSFNET – *National Science Foundation Network*.

OSI – *Open Systems Interconnection*.

PL/SQL – *Procedural Language/Structured Query Language*.

Pro – *Professional*.

QoS – *Quality of Service*.

RAM – *Random Access Memory*.

RIA – *Rich Internet Application*.

RSVP – *Resource Reservation Protocol*.

RTP – *Real Time Protocol*.

RTCP – *Real Time Control Protocol*.

SGBD – *Sistema de Gerenciamento de Banco de Dados*.

SIP – *Session Initiation Protocol*.

SQL – *Structured Query Language*.

SRI – *Stanford Research Institute*.

TB – *Tera Byte*.

TCP – *Transmission Control Protocol*.

UCLA – *University of California at Los Angeles*.

UCSB – *University of California at Santa Bárbara*.

UDP – *User Datagram Protocol*.

UML – *Unified Modeling Language.*

URL – *Uniform Resource Locator.*

UU – *University of Utah.*

VDL – *Vision Definition Language.*

VoIP – *Voice Over Internet Protocol.*

WAN – *Wide Area Network.*

WWW – *World Wide Web.*

LISTA DE SÍMBOLOS

@ – Arroba.

® – Marca registrada.

™ – Trade Mark.

SUMÁRIO

1	INTRODUÇÃO.....	25
2	OBJETIVOS	31
2.1	Objetivos gerais.....	31
2.2	Objetivos específicos.....	31
3	REDES DE COMPUTADORES.....	33
3.1	O que é uma rede de computadores?.....	36
3.2	Tipos de redes	39
3.2.1	Redes locais	40
3.2.2	Redes metropolitanas.....	41
3.2.3	Redes geograficamente distribuídas	41
3.2.4	Redes sem fios.....	41
3.2.5	Inter-redes	42
3.3	Modelos de referência.....	42
3.3.1	Referência OSI	42
3.3.2	Referência TCP/IP	45
3.4	Paradigmas da camada de aplicação	46
3.4.1	Paradigma cliente-servidor.....	47
3.4.2	Paradigma par a par	47
3.5	Protocolos da camada de transporte.....	47
3.6	Protocolo de datagramas de usuário.....	48
3.7	Protocolo de controle de transmissão	48
3.8	Sockets.....	48
3.8.1	Endereços de socket	50
3.8.2	Sockets usados no TCP	51
3.8.3	Sockets usados no UDP	52
3.9	VoIP.....	53
4	PLANO DE DESENVOLVIMENTO	61
4.1	Conceitos básicos de banco de dados relacionais	61
4.1.1	DER	63
4.1.2	MER.....	63
4.1.3	DD + trigramação	64
4.1.4	Classificação de dados.....	64
4.1.5	Primeira forma normal	65
4.1.6	Segunda forma normal	65
4.1.7	Terceira forma normal	65
4.1.8	Forma normal de boyce-codd.....	66
4.2	Modelagem do sistema	66
4.2.1	Diagrama de casos de uso	67
4.2.1.1	Caso de uso login	68
4.2.1.2	Caso de uso cadastro	70
4.2.1.3	Caso de uso chat	71
4.2.1.4	Caso de uso informação	72
4.3	Linguagem de programação.....	73
4.3.1	Java Standard Edition.....	73
4.4	Diagrama de bloco.....	73

4.4.1 Diagrama do menu	74
4.4.2 Diagrama de cadastro	75
4.4.3 Diagrama do chat	76
4.4.4 Diagrama da rede	77
4.5 Conceito geral do programa	77
4.5.1 Protótipos	77
4.5.2 Protótipo de login.....	78
4.5.3 Protótipo de chat.....	78
4.5.4 Protótipo de informação.....	79
4.5.5 Protótipo de cadastro.....	79
5 DESENVOLVIMENTO	81
5.1 Interface gráfica	81
5.2 Banco de dados	82
5.3 Servidor de mensagem	82
5.4 Cliente.....	84
5.5 Servidor de áudio	85
5.6 Captura de áudio	86
6 RESULTADOS	87
7 CONCLUSÃO	91
REFERÊNCIA BIBLIOGRÁFICA	95
8 FICHAS DE ATIVIDADES PRÁTICAS SUPERVISIONADAS	97
ANEXOS.....	101

1 INTRODUÇÃO

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

Sempre procuramos maneiras mais rápidas de comunicarmos informações importantes, seja por um e-mail entre funcionários de uma empresa ou um chat entre dois funcionários em indústrias diferentes.

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

Uma das primeiras ferramentas para chat, que foi documentada, foi a internet relay chat (IRC) desde então sempre houve melhorias em ferramentas de chat em tempo real, sempre buscando melhorar a velocidade e a segurança do mesmo.

Atualmente chats em tempo real estão presentes mundialmente desde ferramentas em computador até aplicativos em celular. Diferente dos primeiros chats esses são mais rápidos e possuem criptografia, sempre buscando a segurança de seus usuários.

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

Inicialmente, os computadores eram máquinas caríssimas que centralizaram em um único ponto o processamento das aplicações de vários usuários e, muitas vezes, de toda uma organização. Com a redução de custos do hardware e a introdução dos microcomputadores no cenário da informática, a estrutura

centralizada cedeu lugar a uma estrutura totalmente distribuída, na qual, diversos equipamentos dos mais variados portes, processam informações de formas isoladas, o que acarreta uma série de problemas. Dentre eles destaca-se a duplicação desnecessária de recursos de hardware (impressoras, discos, etc.) e de software (programas, arquivos de dados, etc.).

A fusão dos computadores e das comunicações teve uma profunda influência na forma como os sistemas computacionais eram organizados. Está totalmente ultrapassado o conceito de um “centro de computadores”, como uma sala para onde os usuários levam os programas a serem processados. O velho modelo de um computador atendendo a todas às necessidades computacionais de organização foi substituído pelas chamadas redes de computadores, nas quais os trabalhos são realizados por uma série de computadores interconectados.

Uma rede é um grupo de pelo menos dois computadores que são ligados entre si de forma que possam se comunicar uns com os outros, compartilhando recursos e informações com maior velocidade e praticidade.

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

Olifer & Olifer (2008, p. 04) comprovam que as redes de computadores também podem ser consideradas um meio de transmissão de informações a longa distância. Para isso as redes de computadores implementam diversos métodos de codificação de dados e multiplexação amplamente adotados nos sistemas de telecomunicações.

Segundo Forouzan (2008, p. 07), redes de computadores é um conjunto de dispositivos (normalmente conhecido como nós) conectados por links de comunicação. Um nó pode ser um computador, uma impressora ou outro dispositivo de envio e/ou recepção de dados, que estejam a outros nós da rede.

Scrimger, LaSalle, Parihar e Gupta (2002, p. 03) titubeiam a rede como dois ou mais computadores que compartilham informações. Contudo, as redes podem ser bastante diversificadas.

Uma rede de computadores permite que o usuário compartilhe uma enorme quantidade de informações e enviem mensagens uns aos outros, incluindo textos, imagens, áudios e vídeos.

Olifer & Olifer (2008, p. 04) provem a influência das redes de computadores sobre outros tipos de redes de telecomunicações, onde resultou em uma convergência de redes, um processo que começou muito antes da internet.

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

Apesar de a indústria de informática ser jovem quando comparada a outros setores (como, por exemplo, o de automóveis e o de transportes aéreos), foi simplesmente espetacular o progresso que os computadores conheceram em um curto período de tempo. Nas duas primeiras décadas de sua existência, os sistemas computacionais eram acondicionados, geralmente, em uma grande sala com paredes de vidro, através das quais os visitantes podiam contemplar embevecido o cérebro daquela maravilha eletrônica. Uma empresa de médio porte ou uma universidade conta com apenas um ou dois computadores, enquanto as grandes instituições tinham, no máximo, algumas dezenas. Era pura ficção científica a ideia de que, em apenas 20 anos, haveria milhões de computadores igualmente avançados do tamanho de um selo postal.

A partir destas considerações, visa-se responder a seguinte pergunta: como que se utiliza esse conjunto de dispositivos conectados?

Partindo-se da hipótese, que assimilando o tempo e o conteúdo de estudo com as informações obtidas através de pesquisas é possível, de forma estratégica, desenvolver um mecanismo de comunicação em rede na linguagem Java, com a utilização das primitivas dos *sockets* de Berkeley, ou componentes derivados destes, e voz sobre IP.

Com pesquisas, avaliações e benefícios, o trabalho busca relatar o processo de desenvolvimento de uma ferramenta de comunicação em rede, desde a sua concepção até a sua finalização.

A ferramenta tem como objetivo atender as necessidades do Conselho Nacional do Meio Ambiente (CONAMA) que deseja saber quais atividades industriais estão gerando poluição do Rio Tietê desde sua nascente em Salesópolis (SP) até a sua passagem pela região da grande São Paulo. Para tal ela precisa trocar

informações das equipes de inspetores treinados e capacitados que estarão se revezando dentro de cada indústria, controlando os processos e passando informações online para a Secretaria.

O Conselho Nacional do Meio Ambiente (CONAMA) é o órgão consultivo e deliberativo do Sistema Nacional do Meio Ambiente - SISNAMA. Em outras palavras, o CONAMA existe para assessorar, estudar e propor ao Governo, as linhas de direção que devem tomar as políticas governamentais para a exploração e preservação do meio ambiente e dos recursos naturais. Além disso, também cabe ao órgão, dentro de sua competência, criar normas e determinar padrões compatíveis com o meio ambiente ecologicamente equilibrado e essencial à sadia qualidade de vida. Tendo como seu foco o desenvolvimento ambiental de uma cidade seja cuidando da poluição já existente na mesma ou criando novos projetos ambientais para a cidade.

O trabalho intenciona esclarecer conceitos, funções e aplicações da comunicação em rede. Para tanto, apresentando o desenvolvimento de uma ferramenta simples com a integração de um banco de dados, voz sobre IP e as primitivas dos *sockets* de Berkeley.

Fundamentado em obras publicadas por estudiosos especialistas que já apontaram os conceitos, usos, aplicações e falhas no processo de desenvolvimento de uma ferramenta para comunicação em rede. A presente pesquisa busca unificar e transmitir os principais conceitos de suas obras. A junção de oito obras, de ano e autores diferentes, resulta no presente trabalho.

No primeiro capítulo reunimos conceitos gerais de comunicação de dados em rede, história, mercado e tipos de redes. Nesta etapa consideramos as definições de sete autores dos que foram consultados. A saber, Comer (1999), Stevens (1999), Forouzan (2008), Mosharraf (2012), Olifer & Olifer (2008), Scrimger (2002), Lasalle (2002), Parihar (2002), Gupta (2002), Soares (1995), Lemos (1995), Colcher (1995), Tanenbaum (2011) e Wetherall (2011).

No segundo capítulo é descrito, parcialmente, o planejamento de desenvolvimento da aplicação. Nesta etapa consideramos as definições do autor Bezerra (2015) e usufruímos das ferramentas TFS – *Team Foundation Service*, *Pencil Project*, *StarUML*, *brModelo*, *Toad Data Modeler 6.1*, *Microsoft Visio 2016*, *Microsoft Excel 2016* e *Microsoft Word 2016*.

No terceiro capítulo é descrito, parcialmente, o processo (estrutura) de desenvolvimento do programa. Nesta etapa consideramos as definições do autor Santos (2014), os principais softwares utilizados no processo foram a IDE Eclipse Java Mars x64 com acesso ao JDK 1.8.0_121 x64 e JRE 1.8.0_121 x64, últimas versões até o presente momento de dissertação, *Oracle Database 11g Express Edition*, *Sublime Text* três, *Visual StudioCode*, *SQL Developer*, TFS - *Team Foundation Service*, *Notepad* e o *Prompt* de Comando. A linguagem de programação utilizada foi *Java Standard Edition 8* (JSE 8).

O quarto capítulo apresenta algumas imagens da aplicação em funcionamento.

2 OBJETIVOS

Esclarecer conceitos, funções e aplicações da comunicação em rede. Para tanto, apresentando o desenvolvimento de uma ferramenta simples com a integração de um banco de dados, as primitivas dos *sockets* de Berkeley e utilização de voz sobre protocolo de internet (VoIP).

2.1 Objetivos gerais

Contribuir para a sociedade acadêmica com explicações para o processo de utilização das primitivas dos *sockets* de Berkeley. Tendo como resultado uma ferramenta de comunicação em rede desenvolvida na linguagem Java com conexão ao banco de dados Oracle e suporte a voz sobre protocolo de internet.

Contribuir para o melhor entendimento de questões ambientais envolvendo o departamento de poluições e como esta ferramenta iria colaborar nesse contexto.

2.2 Objetivos específicos

Visando atingir o objetivo principal, alguns objetivos específicos são requeridos, entre eles:

- Pesquisar e entender o conceito, utilização e aplicação das redes de computadores.
- Avaliar as principais técnicas usadas e selecionar uma para a elaboração da dissertação.
- Buscar os benefícios que a técnica selecionada proporciona.
- Divulgar aplicações que fazem ou fizeram uso desta técnica.
- Desenvolver um software em cima da técnica selecionada.
- Realizar uma discussão dos resultados oriundos do processo de descoberta de conhecimento.
- Apresentar o funcionamento do software e suas instruções.

3 REDES DE COMPUTADORES

No auge da Guerra Fria, o Departamento de Defesa dos Estados Unidos queria uma rede de controle e comando capaz de sobreviver a uma guerra nuclear. Nessa época, todas as comunicações militares passavam pela rede de telefonia pública, considerada vulnerável. Por sua vez, essas centrais de comutação estavam conectadas as centrais de comutação de nível mais alto (centrais interurbanas), formando uma hierarquia nacional com apenas uma pequena redundância. A vulnerabilidade do sistema era o fato de que a destruição de algumas centrais interurbanas importantes poderia fragmentar o sistema em muitas ilhas isoladas.

Por volta de 1960, o Departamento de Defesa dos Estados Unidos firmou um contrato com *RAND Corporation* para encontrar uma solução. Um de seus funcionários, Paul Baran, apresentou o projeto altamente distribuído e tolerante a falhas. Tendo em vista que os caminhos entre duas centrais de comutação quaisquer eram muito mais longos do que a distância que os sinais analógicos podiam percorrer sem distorção, Baran propôs o uso da tecnologia digital de comutação de pacotes. Ele enviou diversos relatórios para o Departamento de Defesa descrevendo suas ideias em detalhes. Os funcionários do Pentágono gostaram do conceito e pediram à AT&T, na época a empresa que detinha o monopólio nacional da telefonia nos Estados Unidos, que construísse um protótipo. A AT&T descartou as ideias de Baran e informou que a rede de Baran não podia ser construída e a ideia foi abandonada.

Durante os primeiros anos, a ARPA tentou compreender qual deveria ser sua missão. Porém, em 1967, a atenção do então diretor de programas da ARPA, Larry Roberts, que estava tentando descobrir como oferecer acesso remoto aos computadores, se voltou para as redes. Ele entrou em contato com diversos especialistas para decidir o que fazer. Um deles, Wesley Clark, sugeriu a criação de uma sub-rede comutada por pacotes, dando a cada host seu próprio roteador.

A sub-rede consistia em minicomputadores chamados processadores de mensagens de interface, ou IMPs (*Interface Message Processors*), conectados por linhas de transmissão de 56 Kbps. Para garantir sua alta confiabilidade, cada IMP seria conectado a pelo menos dois outros IMPs. A sub-rede tinha de ser uma sub-rede de datagrama, de modo que, se algumas linhas e alguns IMPs fossem

destruídos, as mensagens poderiam ser roteadas automaticamente para caminhos alternativos.

A ARPA abriu uma concorrência para a construção da sub-rede. Doze empresas apresentaram propostas. Depois de avaliar todas as propostas, a ARPA selecionou a BBN, uma empresa de consultoria de Cambridge. A BBN resolveu utilizar, como IMPs, minicomputadores *Honeywell* DDP-316 especialmente modificados, com 12 palavras de 16 bits de memória principal. Os IMPs não tinham unidades de discos, pois os componentes móveis eram considerados pouco confiáveis. Os IMPs eram interconectados por linhas privadas das companhias telefônicas, de 56 Kbps.

Em meados dos anos 1960, os mainframes em organizações de pesquisa eram dispositivos isolados. Computadores de diferentes fabricantes eram incapazes de se comunicar entre si. A *Advanced Research Project Agency* (ARPA) do Departamento de Defesa dos Estados Unidos (DoD) estava interessada em descobrir uma maneira de conectar computadores, de forma que os pesquisadores, os quais eram por eles subsidiados, pudessem compartilhar suas descobertas, reduzindo, portanto, custos e eliminando duplicação de esforços.

Em 1967, em uma reunião da *Association for Computing Machinery* (ACM), a ARPA apresentou suas ideias para a ARPANET, uma pequena rede de computadores conectados. A ideia era a de que cada host (não necessariamente do mesmo fabricante) estaria conectado a um computador especializado chamado processador de mensagens de interface (IMP). Os IMPs, por sua vez, estariam conectados entre si. Cada IMP tinha de ser capaz de se comunicar com outros IMPs, assim como com seu próprio host conectado.

Por volta de 1969, a ARPANET era uma realidade. Quatro nós, na *University of California at Los Angeles* (Ucla), na *University of California at Santa Bárbara* (UCSB), no *Stanford Research Institute* (SRI) e na *University of Utah* (UU), foram conectados através de IMPs para formarem uma rede. O software chamado Protocolo de Controle de Redes (NCP) fornecia a comunicação entre os hosts.

No final da década de 1970, a NSF (*National Science Foundation*) percebeu o enorme impacto que a ARPANET estava causando nas pesquisas universitárias nos Estados Unidos, permitindo que cientistas de todo o país compartilhassem dados e trabalhassem juntos em projetos de pesquisa. No entanto, para entrar na ARPANET, uma universidade precisava ter um contrato de pesquisa com o Departamento de

Defesa dos Estados Unidos, e muitas não o tinham. A resposta inicial da NSF foi patrocinar a *Computer Science Network* (CSNET) em 1981. Ela conectava departamentos de ciência da computação e laboratórios de pesquisa industrial à ARPANET por meio de linhas discadas e privadas. No final da década de 1980, a NSF foi ainda mais longe e decidiu desenvolver uma sucessora para a ARPANET, que seria aberta a todos os grupos de pesquisa universitários.

Em 1972, Vint Cerf e Bob Kahn, ambos participantes do grupo principal da ARPANET, trabalharam juntos no que eles denominaram *Internetting Project*. O feito histórico de Cerf e Kahn, registrado em um artigo de 1973, descrevia os protocolos para conseguir a entrega de pacotes de um extremo ao outro. Esse artigo sobre o TCP (*Transmission Control Protocol*) incluía conceitos como encapsulamento, datagrama e as funções de um gateway. Logo depois, autoridades decidiram dividir o TCP em dois protocolos: o TCP (*Transmission Control Protocol*) e o IP (*Internetworking Protocol*). O IP trataria do roteamento de datagramas, ao passo que o TCP seria responsável por funções de mais alto nível, como segmentação, remontagem e detecção de erros. O protocolo de interligação de redes (*Internetworking Protocol*) passou a ser conhecido como TCP/IP.

Durante a década de 1980, novas redes, em particular as LANs, foram conectadas à ARPANET. À medida que a escala aumentou, tornou-se cada vez mais dispendioso localizar os hosts, e assim foi criado o sistema de nomes de domínio, ou DNS (*Domain Name System*), para organizar máquinas em domínios e relacionar nomes de hosts com endereços IP. Desde então, o DNS se transformou em um sistema generalizado de bancos de dados distribuídos, capaz de armazenar uma série de informações referentes à atribuição de nomes.

Em 1983, autoridades aboliram os protocolos originais da ARPANET, e o TCP/IP tornou-se o protocolo oficial da ARPANET. Aqueles que desejassem usar a Internet para acessar um computador em uma rede diferente tinham que estar utilizando o TCP/IP.

Durante a década de 1990, muitos outros países e regiões também construíram redes nacionais de pesquisa, geralmente moldadas de acordo com a ARPANET e a NSFNET. Na Europa, essas redes incluíram EuropaNET e EBONE, que começam com linhas de 2 Mbps e depois foram atualizadas para linhas de 34 Mbps. Mais tarde, a infraestrutura de rede na Europa também foi entregue à indústria.

Atualmente, a Internet não é uma estrutura hierárquica única. Ela é composta por várias redes locais e remotas, reunidas por meio de dispositivos de conexão e estações comutadoras. É difícil termos uma representação precisa da Internet, pois ela está em contínua mutação – novas redes são acrescentadas, redes existentes adicionaram continuamente novos endereços e redes de empresas extintas são eliminadas. Na atualidade, a maioria dos usuários que deseja conexão com a Internet utiliza os serviços de Provedores de Acesso à Internet (ISPs). Há provedores internacionais, nacionais, regionais e locais. Hoje em dia, a Internet é operada por empresas privadas, não pelo governo. (FOROUZAN, 2008, p. 17)

Em 1990, a ARPANET foi oficialmente aposentada e substituída pela NSFNET. Em 1995, a NSFNET foi revertida de volta ao seu conceito original de uma rede de pesquisa.

Em 1991, o governo dos Estados Unidos decidiu que a NSFNET não era capaz de suportar o tráfego da Internet que vinha aumentando rapidamente. Três empresas, IBM, Merit e Verizon, preencheram esta lacuna ao formar uma organização sem fins lucrativa chamada Serviços & Redes Avançados (ANS – *Advanced Network & Service*) para construir um novo *backbone* de Internet de alta velocidade chamado *Advanced Network Services Network* (ANSNET).

A década de 1990 testemunhou a explosão de aplicações de Internet devido ao surgimento da *World Wide Web* (WWW). A Web foi inventada no CERN por Tim Berners-Lee e foi adicionada às aplicações comerciais desenvolvidas para a Internet.

3.1 O que é uma rede de computadores?

A evolução tecnológica e a consequente diminuição dos custos dos computadores tornaram cada vez mais atraentes a distribuição do poder computacional em módulos processadores localizados em diversos pontos de uma organização. A necessidade de interconexão desses módulos processadores, para permitir o compartilhamento de recursos de hardware e software e a troca de informação entre seus usuários, criou o ambiente propício para o desenvolvimento das redes de computadores.

Rede, do latim rete, é uma estrutura que dispõe de um padrão característico. Um computador, por sua vez, é uma máquina eletrônica que processam dados e que possibilita a execução de diversas sequências ou rotinas indicadas pelo utilizador.

Redes de computadores refere-se a interconexão por meio de um sistema de comunicação baseado em transmissões e protocolos de vários computadores com o objetivo de trocar informações, além de outros recursos. Essa conexão é chamada de estações de trabalho (nós, pontos ou dispositivos de rede).

Forouzan (2008, p. 07) reconhece que uma rede é um conjunto de dispositivos (normalmente conhecido como nós) conectados por links de comunicação. Um nó pode ser um computador, uma impressora ou outro dispositivo de envio e/ou recepção de dados, que estejam conectados a outros nós da rede.

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

Olifer & Olifer (2008, p. 04) provam que as redes de computadores também podem ser consideradas um meio de transmissão de informações a longa distância. Para isso as redes de computadores implementam diversos métodos de codificação de dados e multiplexação amplamente adotados nos sistemas de telecomunicações.

Scrimger, LaSalle, Parihar e Gupta (2002, p. 03) determinam a rede como dois ou mais computadores que compartilham informações. Contudo, as redes podem ser bastante diversificadas.

Uma rede de computadores permite que o usuário compartilhe uma enorme quantidade de informações e enviem mensagens uns aos outros, incluindo textos, imagens, áudios e vídeos.

Olifer & Olifer (2008, p. 04) provem a influência das redes de computadores sobre outros tipos de redes de telecomunicações, onde resultou em uma convergência de redes, um processo que começou muito antes da internet.

As redes são divididas em duas categorias principais: redes locais (LANs) e redes remotas (WANs). Esses dois tipos têm diferentes características e diferentes funcionalidades. A Internet é um conjunto de redes LANs e WANs interconectadas por dispositivos de ligação entre elas.

Sua composição é formada por cinco componentes essenciais, que são:

- **Mensagem:** As mensagens são as informações a serem transmitidas. Entre as formas populares de informação, temos: texto, números, figuras, áudio e vídeo.
- **Emissor:** O emissor é o dispositivo que envia a mensagem de dados. Pode ser um computador, estação de trabalho, aparelho telefônico, televisão e assim por diante.
- **Receptor:** O receptor é o dispositivo que recebe a mensagem. Pode ser um computador, estação de trabalho, aparelho telefônico, televisão e assim por diante.
- **Meio de transmissão:** O meio de transmissão é o caminho físico pelo qual uma mensagem trafega do emissor ao receptor. Alguns exemplos de meio de transmissão são os seguintes: cabo par traçado, cabo coaxial, cabo de fibra ótica e ondas de rádio.
- **Protocolo:** O protocolo é um conjunto de regras que controla a comunicação de dados. Representa um acordo entre os dispositivos de comunicação. Sem um protocolo, dois dispositivos podem estar conectados, mas, sem se comunicar. De modo semelhante, uma pessoa que fala francês não consegue entender outra que fala apenas o idioma japonês.

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

Segundo Comer e Stevens (1999, p. 38), algumas das primeiras redes de computadores foram construídas para expandir equipamentos de computação já existente. Por exemplo, foram projetadas redes que permitiam a múltiplos computadores acessar um dispositivo periférico compartilhado, como uma impressora ou um disco. Isto é, ao invés de conectar um dispositivo periférico a um único computador, o dispositivo era conectado a uma rede, o que permitia acesso a partir de qualquer computador conectado à rede. As motivações iniciais para redes

de dados de larga escala não surgiam do desejo de compartilhar dispositivos periféricos ou mesmo de fornecer outra forma das pessoas se comunicarem. Em vez disso, as primeiras redes foram projetadas para compartilhar poder computacional em grande escala.

3.2 Tipos de redes

Não existe nenhuma taxonomia de aceitação geral na qual todas as redes de computadores possam ser classificadas, mas duas dimensões se destacam das demais: a tecnologia de transmissão e a escala.

Em termos gerais, há dois tipos de tecnologias de transmissão em uso disseminado nos dias de hoje:

1. Links de difusão.
2. Links ponto a ponto.

Tanenbaum e Wetherall (2011, p. 28) definem que as redes de difusão têm apenas um canal de comunicação, compartilhadas por todas as máquinas da rede. Mensagens curtas, que em determinados contextos são chamadas pacotes, enviadas por qualquer máquina, são recebidas por todas as outras. Um campo de endereço dentro do pacote especifica o destinatário pretendido. Quando recebe um pacote, uma máquina verifica o campo de endereço. Se o pacote se destinar à máquina receptora, ela o processará; se for destinado a alguma outra máquina, o pacote será simplesmente ignorado.

Em contraste, as redes ponto a ponto consistem em muitas conexões entre pares de máquinas individuais. Para ir da origem ao destino, um pacote nesse tipo de rede talvez tenha de visitar primeiro uma ou mais máquinas intermediárias. Como normalmente é possível haver várias rotas com diferentes tamanhos, encontrar boas rotas é algo importante em redes ponto a ponto. Como regra geral (embora existam muitas exceções), redes menores geograficamente localizadas tendem a usar difusão, enquanto redes maiores em geral são redes ponto a ponto. (TANENBAUM E WETHERALL, 2011, p. 08)

Um critério alternativo para classificar as redes é sua escala. A imagem abaixo mostra uma classificação. Na parte superior encontram-se as redes pessoais, redes destinadas a uma única pessoa. Por exemplo, uma rede sem fios conectando um computador com o mouse, o teclado e a impressora é uma rede pessoal. Além disso, um PDA que controla o aparelho de audição ou o marca-passo de um usuário se enquadra nessa categoria. Além das redes pessoais, encontramos redes de maior abrangência. Essas redes podem ser divididas em redes locais, metropolitanas e geograficamente distribuídas (ou remotas). Finalmente, a conexão de duas ou mais redes é chamada inter-rede. A Internet mundial é um exemplo bastante conhecido de inter-rede. A distância é importante como uma métrica de classificação, porque são empregadas diferentes técnicas em escalas distintas.

Imagem 01 – Classificação de processadores por escala.

Distância de Interconexão	Área de Conexão		
1m	Metro Quadrado	→	Rede de Área Pessoal
10m	Sala	}	Rede Local
100m	Prédio		
1km	Campus		
10km	Cidade	→	Rede Metropolitana
100km	País	}	Rede <u>Wide</u>
1.000km	Continente		
10.000km	Mundo	→	Internet

Fonte: sites.google.com/site/estudandoredes/capitulo-01---introducao/1-2-hardware-de-rede, 2017.

3.2.1 Redes locais

Forouzan e Mosharraf (2013, p. 02) explicam que uma rede local (LAN – *Local Area Network*) geralmente é uma propriedade privada e conecta alguns hosts em um único escritório, prédios, campus. Dependendo da necessidade de uma organização, uma LAN pode ser simples com apenas dois computadores e uma impressora no escritório da casa de alguém ou pode se estender por toda a empresa e incluir dispositivos de áudio e vídeo. Cada host em uma LAN possui um identificador, ou seja, um endereço que o define de forma unívoca na LAN. Um

pacote enviado de um host para outro carrega tanto o endereço do host de origem como o de destino.

3.2.2 *Redes metropolitanas*

Uma rede metropolitana, ou MAN, abrange uma cidade. O exemplo mais conhecido de uma MAN é a rede de televisão a cabo disponível em muitas cidades. Esse sistema cresceu a partir de antigos sistemas de antenas comunitárias usadas em áreas com fraca recepção do sinal de televisão pelo ar. Nesses primeiros sistemas, uma grande antena era colocada no alto de colina próxima e o sinal era então conduzido até a casa dos assinantes.

Olifer & Olifer (2008, p. 11) evidenciam que as MANs oferecem um eficiente meio de interligar LANs bem como de conectar LANs a WANs. Essas redes inicialmente foram desenvolvidas somente para transmissão de dados; hoje em dia, a amplitude de seus serviços foi expandida.

3.2.3 *Redes geograficamente distribuídas*

Forouzan e Mosharraf (2013, p. 02) determinam uma rede de longa distância (WAN – *Wide Area Network*) como uma interligação de dispositivos capazes de se comunicar. No entanto existem, algumas diferenças entre uma LAN e uma WAN. A LAN normalmente tem tamanho limitado, estendendo-se por um escritório, edifícios ou campus, enquanto uma WAN tem uma extensão geográfica maior, abrangendo uma cidade, um estado, um país ou até mesmo o mundo.

Soares, Lemos e Colher (1995, p.24) demonstra que características geográficas das redes locais e metropolitana levam a considerá-la bastante diferentes das redes de longa distância.

3.2.4 *Redes sem fios*

Em uma primeira aproximação, redes sem fios podem ser divididas em três categorias principais:

1. **Interconexão de sistemas:** a interconexão de sistemas significa interconectar os componentes de um computador usando rádio de alcance limitado.
2. **LANs sem fios:** são sistemas em que todo computador tem um modem de rádio e uma antena por meio dos quais pode se comunicar com outros sistemas.
3. **WANs sem fios:** é usada em sistemas geograficamente distribuídos.

3.2.5 Inter-redes

Uma inter-rede é formada quando diferentes redes estão interconectadas. No nosso ponto de vista, a conexão de uma LAN e uma WAN ou a conexão de duas LANs forma uma inter-rede, mas ainda não existe um consenso na indústria quanto à terminologia a ser usada nessa área. Uma regra prática é que, se diferentes organizações pagam pela construção de partes distintas da rede e cada mantém sua parte, temos uma inter-rede, e não uma única rede. Além disso, se a tecnologia subjacente é diferente em partes distintas (por exemplo, difusão versus ponto a ponto), provavelmente temos duas redes.

3.3 Modelos de referência

Examinaremos duas importantes arquiteturas de rede, o modelo de referência OSI e o modelo de referência TCP/IP. Embora os protocolos associados ao modelo OSI raramente sejam usados nos dias de hoje, o modelo em si é de fato bastante geral e ainda válido, e as características descritas em cada camada ainda são muito importantes. O modelo TCP/IP tem características opostas: o modelo propriamente dito não é muito utilizado, mas os protocolos têm uso geral.

3.3.1 Referência OSI

“Esse modelo se baseia em uma proposta desenvolvida pela ISO (*International Standards Organization*) como um primeiro passo em direção à padronização internacional dos protocolos empregados nas diversas camadas” (TANENBAUM e WETHERALL, 2011, p. 28). Ele foi revisto em 1995. O modelo é

chamado Modelo de Referência ISO OSI (*Open Systems Interconnection*), pois ele trata da interconexão de sistemas abertos. O modelo OSI tem sete camadas. Veja a seguir um resumo dos princípios aplicados para se chegar às sete camadas.

1. Uma camada deve ser criada onde houver necessidade de outro grau de abstração.
 2. Cada camada deve executar uma função bem definida.
 3. A função de cada camada deve ser escolhida tendo em vista a definição de protocolos padronizados internacionalmente.
 4. Os limites de camadas devem ser escolhidos para minimizar o fluxo de informações pelas interfaces.
 5. O número de camadas deve ser grande o bastante para que funções distintas não precisem ser desnecessariamente colocadas na mesma camada e pequenas o suficiente para que a arquitetura não se torne difícil de controlar.
- **Camada Física:** a camada física trata da transmissão de bits brutos por um canal de comunicação. O projeto da rede deve garantir que, quando um lado enviar um bit um, o outro lado o receberá como um bit um, não como um bit zero. Nesse caso, as questões mais comuns são a voltagem a ser usada para representar um bit um e um bit zero, a quantidade de nano segundos que um bit deve durar, o fato de a transmissão poder ser ou não realizada nos dois sentidos simultaneamente, a forma como a conexão inicial será estabelecida e de que maneira ela será encerrada quando ambos os lados tiverem terminado, e ainda quantos pinos o conector de rede terá e qual será a finalidade de cada pino. Nessa situação, as questões de projeto lidam em grande parte com interfaces mecânicas, elétricas e de sincronização, e com o meio físico de transmissão que se situa abaixo da camada física.
 - **Camada de Enlace de Dados:** a principal tarefa da camada de enlace de dados é transformar um canal de transmissão bruta em uma linha que pareça livre de erros de transmissão não detectados para a camada de rede. Para executar essa tarefa, a camada de enlace de

dados faz com que o transmissor divida os dados de entrada em quadros de dados (que, em geral, têm algumas centenas ou alguns milhares de bytes), e transmita os quadros sequencialmente. Se o serviço for confiável, o receptor confirmará a recepção correta de cada quadro, enviando de volta um quadro de confirmação.

- **Cama de Rede:** controla a operação da sub-rede. Uma questão fundamental de projeto é determinar a maneira como os pacotes são roteados da origem até o destino. As rotas podem se basear em tabelas estáticas, "amarradas" à rede e raramente alteradas. Elas também podem ser determinadas no início de cada conversação; por exemplo, uma sessão de terminal (como um login em uma máquina remota). Por fim, elas podem ser altamente dinâmicas, sendo determinadas para cada pacote, com o objetivo de refletir a carga atual da rede.
- **Cama de Transporte:** a função básica da camada de transporte é aceitar dados da camada acima dela, dividi-los em unidades menores, caso necessário, repassar essas unidades à camada de rede e assegurar que todos os fragmentos chegarão corretamente à outra extremidade. Além do mais, tudo isso deve ser feito com eficiência e de forma que as camadas superiores fiquem isoladas das inevitáveis mudanças na tecnologia de hardware.
- **Camada de Sessão:** permite que os usuários de diferentes máquinas estabeleçam sessões entre eles. Uma sessão oferece diversos serviços, inclusive o controle de diálogo (mantendo o controle de quem deve transmitir em cada momento), o gerenciamento de símbolos (impedindo que duas partes tentem executar a mesma operação crítica ao mesmo tempo) e a sincronização (realizando a verificação periódica de transmissões longas para permitir que elas continuem a partir do ponto em que estavam ao ocorrer uma falha).
- **Camada de Apresentação:** diferente das camadas mais baixas, que se preocupam principalmente com a movimentação de bits, a camada de apresentação está relacionada à sintaxe e à semântica das informações transmitidas. Para tornar possível a comunicação entre computadores com diferentes representações de dados, as estruturas

de dados a serem intercambiadas podem ser definidas de maneira abstrata, juntamente com uma codificação padrão que será usada durante a conexão. A camada de apresentação gerencia essas estruturas de dados abstratas e permite a definição e o intercâmbio de estruturas de dados de nível mais alto (por exemplo, registros bancários).

- **Camada de Aplicação:** contém uma série de protocolos comumente necessários para os usuários. Um protocolo de aplicação amplamente utilizado é o HTTP (*Hyper Text Transfer Protocol*), que constitui a base para a *World Wide Web*. Quando um navegador deseja uma página da *Web*, ele envia o nome da página desejada ao servidor, utilizando o HTTP. Então, o servidor transmite a página de volta. Outros protocolos de aplicação são usados para transferências de arquivos, correio eletrônico e transmissão de notícias pela rede.

3.3.2 Referência TCP/IP

Quando foram criadas as redes de rádio e satélite, começaram a surgir problemas com os protocolos existentes, o que forçou a criação de uma nova arquitetura de referência. Desse modo, a habilidade para conectar várias redes de maneira uniforme foi um dos principais objetivos de projeto, desde o início. Mais tarde, essa arquitetura ficou conhecida como Modelo de Referência TCP/IP, graças a seus dois principais protocolos.

Segundo Comer e Stevens (1999, p. 01) o conjunto de protocolos da internet TCP/IP é o padrão mundial para interconexão de sistemas abertos. Nenhum outro conjunto de protocolos proporciona tanta interoperabilidade ou abrange sistemas de tantos fornecedores.

- **Camada Inter-rede:** sua tarefa é permitir que os hosts injetassem pacotes em qualquer rede e garantir que eles trafegarão independentemente e até o destino (talvez em uma rede diferente). Eles podem chegar até mesmo em uma ordem diferente daquela em que foram enviados, obrigando as camadas superiores a reorganizá-los, caso a entrega em ordem seja desejável. Observe que, nesse

caso, a expressão "inter-rede" é usada em sentido genérico, muito embora essa camada esteja presente na Internet.

- **Camada de Transporte:** a finalidade dessa camada é permitir que as entidades pares dos hosts de origem e de destino mantenham uma conversação, exatamente como acontece na camada de transporte OSI. Dois protocolos fim a fim foram definidos aqui. O primeiro deles, o TCP, é um protocolo orientado a conexões confiável que permite a entrega sem erros de um fluxo de bytes originário de uma determinada máquina em qualquer computador da inter-rede. Esse protocolo fragmenta o fluxo de bytes de entrada em mensagens discretas e passa cada uma delas para a camada inter-redes. No destino, o processo TCP receptor volta a montar as mensagens recebidas no fluxo de saída. O TCP também cuida do controle de fluxo, impedindo que um transmissor rápido sobrecarregue um receptor lento com um volume de mensagens maior do que ele pode manipular.
- **Camada de Aplicação:** acima da camada de transporte, encontramos a camada de aplicação. Ela contém todos os protocolos de nível mais alto. Dentre eles estão o protocolo de terminal virtual (TELNET), o protocolo de transferência de arquivos (FTP) e o protocolo de correio eletrônico (SMTP).
- **Camada Host/Rede:** abaixo da camada inter-redes, encontra-se um grande vácuo. O modelo de referência TCP/IP não especifica muito bem o que acontece ali, exceto o fato de que o host tem de se conectar à rede utilizando algum protocolo para que seja possível enviar pacotes IP. Esse protocolo não é definido e varia de host para host e de rede para rede. Os livros e a documentação que tratam do modelo TCP/IP raramente descrevem esse protocolo.

3.4 Paradigmas da camada de aplicação

Dois paradigmas foram desenvolvidos desde o surgimento da Internet: o paradigma cliente-servidor e o paradigma *peer-to-peer*.

3.4.1 *Paradigma cliente-servidor*

No paradigma cliente-servidor o provedor de serviços é um aplicativo chamado processo-servidor, é executado continuamente à espera de outro aplicativo, chamado processo-cliente, que cria uma conexão através da rede e solicita o serviço. Normalmente, existem alguns processos-servidores que podem fornecer um tipo específico de serviço, mas há muitos clientes que solicitam o serviço de qualquer um desses processos-servidores. O processo-servidor deve ser executado o tempo todo, o processo-cliente é iniciado quando o cliente precisa receber o serviço.

Em um paradigma cliente-servidor, a comunicação na camada de aplicação se dá entre dois aplicativos em execução denominados processos: um cliente e um servidor. Um cliente é um programa em execução que inicia a comunicação enviando um pedido (ou solicitação); um servidor é outro programa que aguarda pedidos de clientes. O servidor trata o pedido recebido, prepara um resultado e envia o resultado de volta ao cliente. Nessa definição, um servidor deve estar sendo executado quando o pedido de um cliente chegar, mas o cliente pode ser executado somente quando necessário.

3.4.2 *Paradigma par a par*

O paradigma par a par surgiu para responder às necessidades de novas aplicações. Nesse paradigma, não há necessidade de um processo-servidor que seja executado o tempo todo e espere pela conexão dos processos clientes. A responsabilidade é compartilhada entre os pares (os nós da rede). Um computador conectado à Internet pode prover serviços em dado momento e receber serviços em outro; pode até mesmo prover e receber serviços ao mesmo tempo.

3.5 **Protocolos da camada de transporte**

A camada de transporte está localizada entre a camada de aplicação e a camada de rede na pilha de protocolos TCP/IP. Ela fornece serviços à camada de aplicação e recebe serviços da camada de rede. A camada de transporte atua como uma ligação entre um programa-cliente e um programa-servidor, como uma conexão

de processo para processo. A camada de transporte é o coração da pilha de protocolos TCP/IP; ela é o meio lógico fim a fim para transferir dados de um ponto a outro na Internet.

3.6 Protocolo de datagramas de usuário

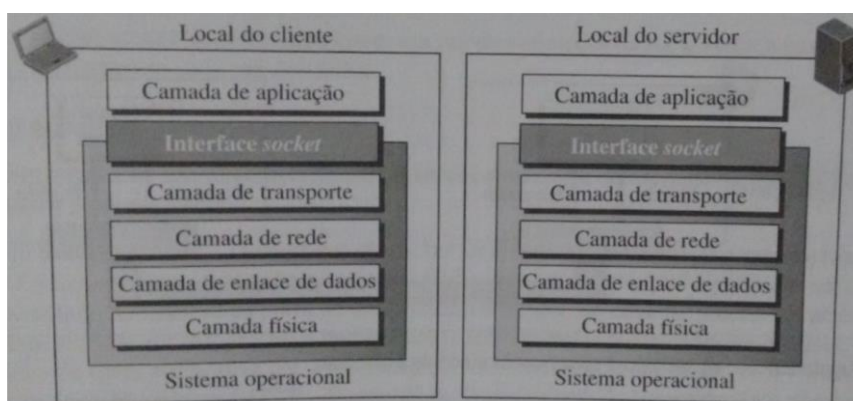
O protocolo de datagramas de usuário (UDP – *User Datagram Protocol*) é um protocolo de transporte não confiável e não orientado à conexão. Ele não acrescenta nada aos serviços do IP, exceto pelo fornecimento de comunicação processo a processo em vez de comunicação *host a host*. O UDP é um protocolo muito simples, que tem um uso mínimo de recursos. Se um processo quiser enviar uma mensagem pequena e não se importar muito com a confiabilidade da transmissão, ele pode usar o UDP. Enviar uma mensagem pequena usando o UDP exige muito menos interação entre o emissor e o receptor do que se o TCP for usado.

3.7 Protocolo de controle de transmissão

O protocolo de controle de transmissão (TCP – *Transmission Control Protocol*) é um protocolo orientado à conexão e confiável. O TCP define explicitamente o estabelecimento da conexão, a transferência de dados e o fechamento para oferecer um serviço orientado à conexão. O TCP usa uma combinação de protocolos do tipo GBN e RS para fornecer confiabilidade. Para atingir esse objetivo, faz uso de mecanismos de soma de verificação (*checksum*, para detecção de erros), retransmissão de pacotes perdidos ou corrompidos, confirmações cumulativos e seletivos e temporizadores.

3.8 Sockets

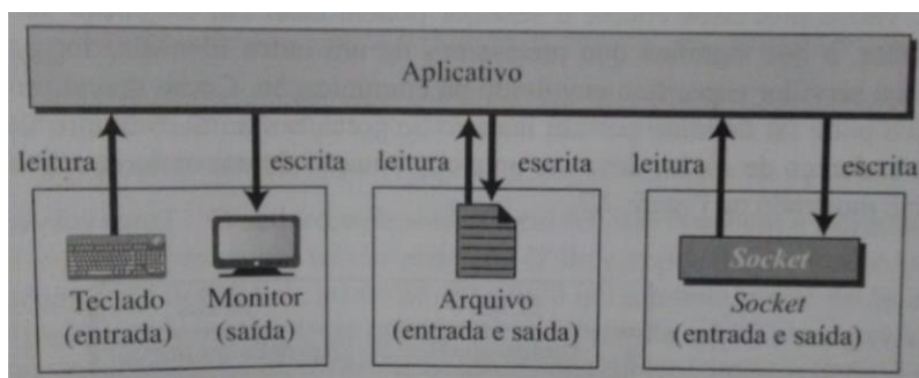
A interface *socket* surgiu no início de 1980 na Universidade de Berkeley como parte de um ambiente UNIX; um conjunto de instruções que fornecem comunicação entre a camada de aplicação e o sistema operacional. É um conjunto de instruções que podem ser usadas por um processo para se comunicar com outro.

Imagem 02 – Posição da interface *socket*.

Fonte: Própria, 2017.

A ideia de *sockets* nos permite usar o conjunto de todas as instruções já projetadas em uma linguagem de programação para outros dispositivos de entrada e saída. Por exemplo, na maioria das linguagens de programação existem várias instruções para ler e gravar dados em dispositivos de entrada e saída. Podemos usar as mesmas instruções para ler ou escrever em *sockets*, ou seja, estamos adicionando apenas novas entidades de entrada e saída à linguagem de programação, sem alterar a forma de enviar ou receber dados.

Apesar de um *socket* ter que se comportar como um terminal ou um arquivo, ele não é uma entidade física como eles, mas sim, uma abstração; é uma estrutura de dados criada e utilizada pelo aplicativo.

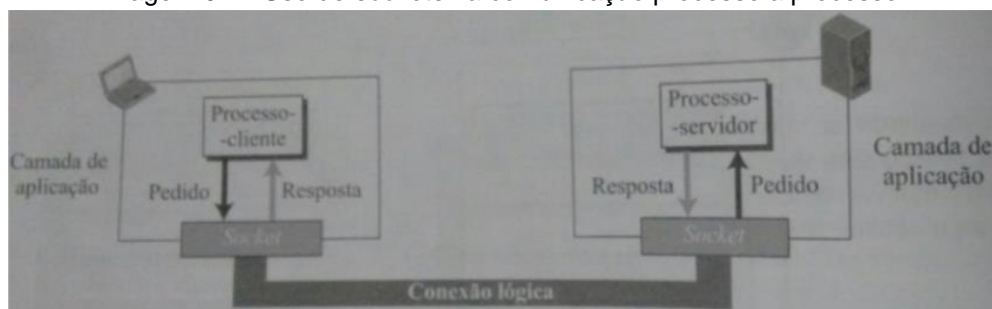
Imagem 03 – *Sockets* usados da mesma forma que outras entidades de entrada e saída.

Fonte: Própria, 2017.

Pode-se dizer que, no que diz respeito à camada de aplicação, a comunicação entre um processo-cliente e um processo-servidor se limita à comunicação entre dois *sockets* como a entidade que recebe o pedido e envia a

resposta: o servidor percebe o *socket* como aquele que tem um pedido e precisa da resposta. Se criarmos dois *sockets*, um em cada sistema final, e definirmos os endereços de origem e destino corretamente, podem usar as instruções disponíveis para enviar e receber dados. O resto é responsabilidade do sistema operacional e do protocolo TCP/IP a ele incorporado.

Imagem 04 – Uso de *sockets* na comunicação processo a processo.

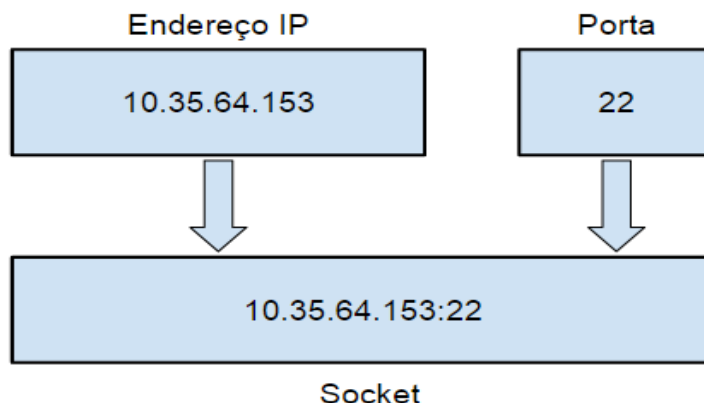


Fonte: Própria, 2017.

3.8.1 Endereços de *socket*

A interação entre um cliente e um servidor é uma comunicação bidirecional, na qual precisamos de um par de endereços: um local (remetente) e um remoto (receptor). O endereço local em uma direção é o endereço remoto na outra direção e vice-versa. Como a comunicação no paradigma cliente-servidor é entre dois *sockets*, precisamos de um par de endereços de *socket* para a comunicação: um endereço de *socket* local e outro de *socket* remoto. No entanto, devemos definir um endereço de *socket* em termos de identificadores utilizados na pilha de protocolos TCP/IP.

Um endereço de *socket* deve primeiro definir o computador no qual um cliente ou um servidor está sendo executado. No entanto, vários processos cliente e servidor podem estar em execução ao mesmo tempo em um computador, o que significa que precisamos de um outro identificador para definir o processo-cliente ou servidor específico envolvido na comunicação.

Imagem 05 – Endereço de *socket*.

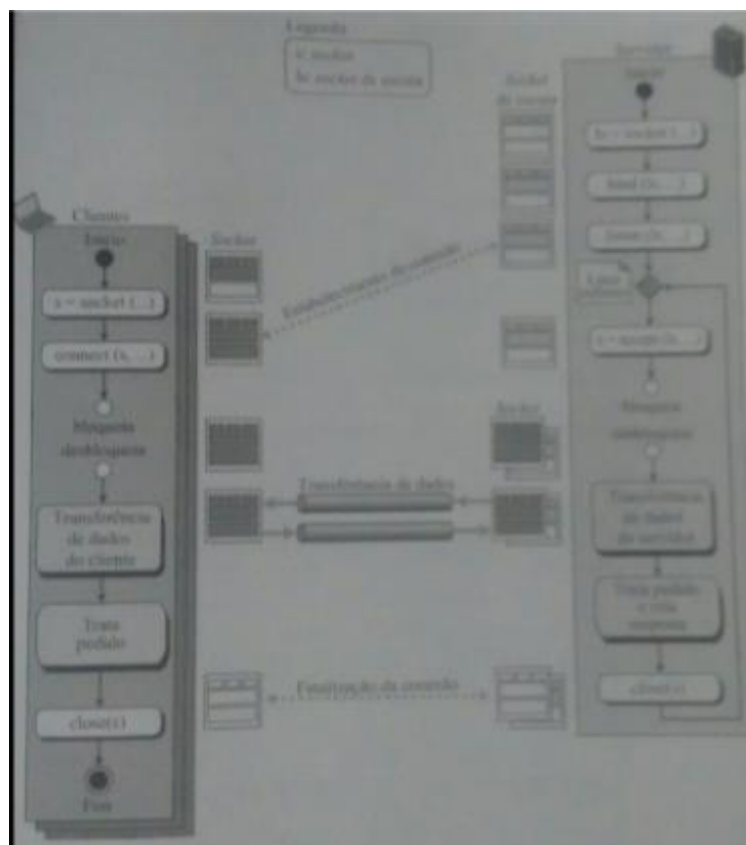
Fonte: bosontreinamentos.com.br/redes-computadores/curso-de-redes-camada-de-transporte-da-pilha-tcpip, 2017.

Como um *socket* define o ponto final da comunicação, podemos dizer que um *socket* é identificado por um par de endereços de *socket*, um local e um remoto.

3.8.2 Sockets usados no TCP

O servidor TCP usa dois *sockets* distintos, um para o estabelecimento da conexão e outro para a transferência de dados. Chamamos o primeiro de *socket* de escuta (ou *listen socket*) e o segundo simplesmente de *socket*. O objetivo de se ter dois tipos de *socket* é separar a fase de conexão da fase de troca de dados. Um servidor usa um *socket* de escuta para ficar escutando novos clientes tentando estabelecer conexões. Quando uma conexão for estabelecida, o servidor cria um *socket* para trocar dados com o cliente e, finalmente, encerra a conexão.

Imagem 06 – Diagrama de fluxo para uma comunicação TCP iterativa.

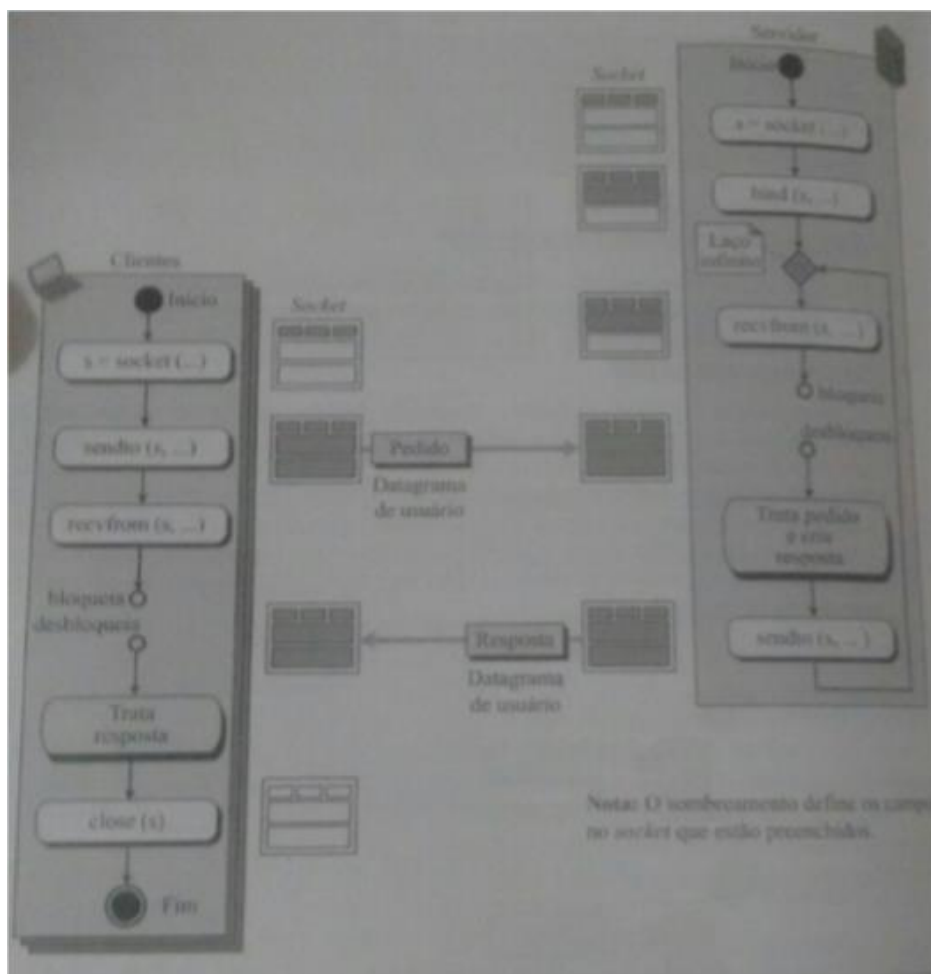


Fonte: Própria, 2017.

3.8.3 Sockets usados no UDP

Na comunicação via UDP, o cliente e o servidor utilizam apenas um *socket* cada. O *socket* criado no lado do servidor nunca é fechado; o criado no lado do cliente é fechado (destruído) quando o processo-cliente é encerrado. Em outras palavras, diferentes clientes usam *sockets* distintos, mas o servidor cria um único *socket* e só muda o endereço de *socket* remoto quando um novo cliente o acessa. Isto faz sentido, já que o servidor sabe seu próprio endereço de *socket*, mas não sabe o dos clientes que precisam acessá-lo; ele precisa esperar pelo acesso do cliente para preencher esse campo do *socket*.

Imagem 07 – Diagrama de fluxo para a comunicação iterativa via UDP.



Fonte: Própria, 2017.

3.9 VoIP

A telefonia via Internet é uma tecnologia que permite a transmissão de voz por IP, possibilitando a realização de chamadas telefônicas pela internet.

Imagina-se a tecnologia de Voz sobre IP como algo muito recente, porém ela surgiu em Israel em 1995, quando um grupo interessado no assunto desenvolveu um sistema que permitisse utilizar os recursos multimídia de um computador doméstico para iniciar conversas de voz através da Internet. A qualidade do sistema era sofrível, mas este era o primeiro passo para que outros pesquisadores se interessassem pelo assunto. Tanto é que, ainda no mesmo ano, uma empresa chamada *Vocaltec Inc*, lançava o primeiro software dedicado à comunicação por Voz sobre IP, batizado de *Internet Phone Software*. Este software foi designado para rodar em um PC 486/33 MHz com placa de som, alto-falantes, microfone e modem.

E, embora a qualidade de som estivesse muito abaixo da telefonia convencional, este esforço representou o primeiro telefone por IP. Nos primeiros estágios da implementação do VoIP a qualidade da comunicação ainda era muito baixa (o famoso "picote" e *delay* na conversação, desconexões e incompatibilidade). Por volta de 1998 o VoIP obteve um progresso considerável. Com o desenvolvimento de *Gateways* (equipamentos capazes de interligar aparelhos telefônicos convencionais ou centrais telefônicas de empresas, os *PABX's*, à rede de dados para comunicação entre estes sistemas com sistemas VoIP), foi permitido a conexão PC para telefone e mais tarde telefone para telefone. Posteriormente surgiram *Gateways* especializados e dispositivos denominados ATA (*Analog Telephone Adapter*, ou Adaptador para Telefone Analógico), para interligar dois sistemas convencionais e/ou *PABX's* utilizando como meio de transmissão redes IP. Mas o grande ponto da história do VoIP ocorreu quando fabricantes de hardware como Cisco e Nortel começaram a produzir equipamentos VoIP capazes de *switching*. Assim, as funções antes tratadas pela CPU da máquina (como mudar um pacote de dados de voz para algo que possa ser lido pela rede de telefonia convencional, e vice-versa), pôde ser tratado por outro dispositivo. Dessa forma, o hardware VoIP fica menos dependente da máquina e mais acessível, e, conseqüentemente, as grandes empresas puderam implementar VoIP em suas redes internas.

O VoIP é uma tecnologia que viabiliza a comunicação telefônica utilizando a internet como meio de transmissão da voz. Em português também é conhecida por Voz sobre IP. Basicamente, a tecnologia VoIP converte sinal de voz analógico através da quebra da conversação em pacotes de dados no formato digital e os transmite através da internet ou redes digitais privadas. E esses mesmos pacotes são utilizados com vantagens enormes sobre a telefonia convencional (como, por exemplo, possibilitar que diversas chamadas telefônicas ocupem o espaço antes ocupado por somente uma chamada da rede convencional). Inicialmente o VoIP foi implementado para efetuar a comunicação de voz entre dois computadores ligados à Internet. Para isso, se fazem necessários uma saída de áudio do sistema multimídia do microcomputador (caixa de som ou *headfone*) e um microfone conectado à entrada do sistema (placa de som do PC). Também é utilizado um *softphone*, que é um software responsável pelo o gerenciamento desta comunicação, iniciando ou recebendo uma ligação, mostrando os contatos on-line, compactando e descompactando o áudio. Existem diversos *softphones* e programas de troca de

mensagens instantâneas que permitem conversas com áudio pela Internet, disponíveis gratuitamente para download, como por exemplo: *MSN Messenger*, *Net2phone*, *X-ten*, *Skype* (que é o mais utilizado atualmente), etc. Além deles, existem programas fornecidos por operadoras de telefonia, como o *WebFone Virtual* da GVT (*Global Village Telecom*), que possui outros serviços agregados a custos atrativos. Para efetuar a comunicação através de voz sobre IP também podem ser utilizados os *softphones*, que são aparelhos telefônicos reais que são diretamente ligados à internet ou rede Lan e que processam a comunicação e tradução dos pacotes em voz sem utilizar o computador. Existem também adaptadores (ATAs) que permitem o uso de seu aparelho telefônico convencional transformando o mesmo em um telefone IP. E, por fim, *Gateways*, que são aparelhos especiais geralmente utilizados para conectar os sistemas telefônicos (*PABX*) das empresas a internet, transformando os ramais e extensões em Telefones IP.

A rede no qual o VoIP trafega é uma WAN, que geralmente se trata diretamente a Internet em si, ou ao menos está inserida nela ou a utiliza como meio de conexão. A rede não sabe o que é voz, ela apenas foi projetada para levar pacotes de dados de uma ponta à outra com a maior agilidade. Porém o conteúdo dos pacotes só é conhecido pelas aplicações que os geraram ou vai tratá-los. As interfaces de voz sobre IP recebem a voz a partir do sistema telefônico, digitalizam, comprimem e acomodam-na em pacotes idênticos aos que trafegam normalmente pela rede. Cabe aos elementos da rede (*switches*, *hubs*, *interface* de VoIP, roteadores, etc.) conduzir os pacotes ao destino, para que possam ser “reconvertidos” para voz. Todos esses procedimentos são transparentes aos usuários do sistema de telefonia. Segundo Soares, e Freire (2002. p, 134), “As técnicas empregadas para enviar informações de um ponto VoIP a outro podem ser as mais variadas (Frame Relay 2-3 0SI, ADSL, PPP, etc.) e meios físicos (par metálico, rádio, fibra ótica, etc.),” porém sempre utilizando o protocolo IP.

Para haver a comunicação entre os aparelhos de telefonia IP eles devem falar a mesma língua, ou seja, utilizar o mesmo protocolo de comunicação, além do protocolo IP. Eles servem para lidar especificamente com a fragmentação e remontagem dos pacotes de voz. Existem hoje dois protocolos mais utilizados pelos aparelhos de telefonia IP, eles são H.323 e SIP.

- H.323: o primeiro protocolo utilizado é que define especificações para comunicação em tempo real de dados para vídeo, dados e voz. Segundo Tanenbaum (2011, p. 530), “A recomendação H.323 é mais uma avaliação da arquitetura de telefonia da Internet que um protocolo específico”. Isso porque ela referencia muitos protocolos específicos para codificação de voz, configuração de chamadas, sinalização, transporte de dados e outras áreas, mas não especifica propriamente cada um desses elementos. Nele a qualidade de serviço é relativa, uma vez que ela não está no âmbito do H.323. Se a rede subjacente for capaz de produzir uma conexão estável e livre de flutuação entre o PC chamador e o *gateway*, então a QoS da chamada será boa; caso contrário, ela não será boa.
- *Session Initiation Protocol* (SIP): desenvolvido especificamente para telefonia IP é muito mais simples e eficiente e por isto está substituindo gradativamente o anterior (H.323). Nas palavras de Tanenbaum (2011, p. 531), “Esse protocolo descreve como instalar chamadas telefônicas da Internet, videoconferências e outras conexões de multimídia”. O SIP é um único módulo, diferente do H.323, que é um conjunto de protocolos completo, mas foi projetado para inter-operar bem com aplicações da Internet. O SIP pode estabelecer sessões de duas partes (ligações telefônicas comuns), sessões de várias partes (onde todos podem ouvir e falar) e sessões de multidifusão (com um transmissor e muitos receptores), sendo que as sessões podem conter áudio, vídeo ou dados. O SIP cuida apenas da configuração, do gerenciamento e do encerramento de sessões, enquanto outros protocolos (como RTP e RTCP) são usados para transporte de dados. O SIP admite uma grande variedade de serviços, inclusive localização do chamado (que pode não estar em sua máquina local) e determinação dos recursos do chamado, bem como tratamento do mecanismo de configuração e encerramento de chamadas. Ele ainda possui uma grande variedade de outros recursos, como a espera de chamadas, triagem de chamadas, criptografia e autenticação. E também tem a habilidade de efetuar chamadas de um computador para um telefone comum, se houver um *gateway* apropriado disponível entre a Internet e o sistema

de telefonia. Embora tenham características semelhantes, os dois protocolos são muito diferentes.

Enquanto o H.323 é um padrão pesado, típico da indústria de telefonia, especificando a pilha de protocolos completa e definindo com precisão o que é permitido e o que é proibido, o SIP é um protocolo típico da Internet e funciona permutando pequenas linhas de texto ASCII. Essa abordagem leva a protocolos muito bem definidos em cada camada, facilitando a tarefa de interoperabilidade. O H.323 é um padrão grande, complexo e rígido e o SIP é um módulo leve que interopera bem com os protocolos da Internet, mas não muito bem com os protocolos de sinalização do sistema telefônico existente.

Para que ocorra a transmissão de voz, o VoIP captura a voz, ainda na forma analógica e a transforma em pacotes de dados (digitais), que podem ser enviados por qualquer rede TCP/IP, possibilitando que trafeguem normalmente pela internet. Assim que os pacotes chegam ao destino, são transformados em sinais analógicos e transmitidos a um meio no qual seja possível ouvir o som. O VoIP não é uma tecnologia nova, mas só ganhou destaque recentemente pelo fato da velocidade de transmissão de dados ser baixo na época em que foi criado (ele já era trabalhado antes mesmo da popularização da internet), impedindo-a de se tornar funcional na maioria das redes. Assim, foi necessário investir em QoS (*Quality of Service*, isto é, em qualidade de serviço) e uma das soluções seria o aumento da velocidade de transmissão e recepção de dados. E como o acesso à internet em banda larga é cada vez mais comum o VoIP passou pôde tirar proveito disso. Porém, apenas mais velocidade não é o bastante. Por isso, surgiram outras soluções, como o protocolo RTP (*Real Time Protocol*), que, basicamente, faz com que os pacotes sejam recebidos na mesma ordem de envio. O RTP "ordena" os pacotes de dados, possibilitando que a transmissão de dados em tempo real. No caso de algum pacote se atrasar, o RTP cria uma interpolação entre o "intervalo" deixado pelo pacote e este não é entregue. O atraso de pacotes pode ocorrer, porque eles podem seguir caminhos diferentes para chegar ao destino. Se você estiver transmitindo um arquivo isso não significa um problema, já que seus pacotes são "encaixados" no destinatário. Porém com voz e vídeo em tempo real, isso nem sempre acontece. Logo, fica claro que o RTP é um recurso muito útil em aplicações que de som e vídeo. Devido a esta característica, seu funcionamento é atrelado a outro protocolo,

o RTCP (*Real Time Control Protocol*), que é responsável pela compressão dos pacotes dos dados e também atua no monitoramento destes. Por ainda ser necessárias melhorias, a IETF (*Internet Engineering Task Force*), entidade responsável pelo RTP e pelo RTCP, sugeriu a aplicação do protocolo RSVP (*Resource Reservation Protocol*), que tem como principal função alocar parte da banda disponível para a transmissão de voz. Existem ainda os *codecs*, que são protocolos extras que somam mais funcionalidades e maior qualidade à comunicação. Entre eles estão o G.711, o G.722, o G.723, o G.727, entre outros. O que os diferencia são os algoritmos usados, a média de atraso e principalmente a qualidade da voz. Neste último aspecto, o G.711 é considerado excelente. Todos esses *codecs* são recomendados pela entidade ITU-T (*International Telecommunications Union - Telecommunications standardization sector*) e geralmente trabalham em conjunto com mais outro protocolo o CRTP.

O *Compressed Real-Time Protocol*, responsável por melhorar a compressão de pacotes e assim dar mais qualidade ao VoIP. Para possibilitar a interligação do VoIP com as redes telefônicas convencionais, geralmente usa-se um equipamento denominado *Gateway*, que é responsável converter o sinal analógico em digital e vice-versa, além de fazer a conversão para os sinais das chamadas telefônicas. Existe ainda o *Gateway Controller* (ou *Call Agent*), que é responsável por controlar as chamadas feitas pelo *Gateway*. Para as ligações em longa distância, são utilizados equipamentos conhecidos por *Gatekeeper*, que gerenciam uma série de outros equipamentos e podem autorizar chamadas, fazer controle da largura de banda utilizada, ou seja, ele pode ser descrito como uma central telefônica para VoIP.

O VoIP alcançou uma popularidade imensa devido a suas muitas vantagens. Primeiramente o custo: com os preços irrisórios da mensalidade do provedor, podem-se fazer quantas ligações quiser, e não importa, incluindo ligações locais e de longa distância nacional e internacional. Ou seja, o VoIP é muito mais barato de se manter do que as linhas telefônicas normais e, além disso, vêm ocorrendo uma redução de preços dos equipamentos que o torna ainda mais acessível. O avanço dessa tecnologia representa um novo conceito de custo se apresenta na área de telecomunicações. Somando-se a isso, tanto a instalação como a utilização do VoIP são simples e não é preciso nenhum especialista para instruir o usuário. O VoIP trouxe mais facilidade para muitas tarefas difíceis em redes tradicionais. Chamadas

entrantes podem ser automaticamente roteadas para o telefone VoIP, independentemente da localização na rede. Por exemplo, se a conexão for rápida e estável o suficiente, é possível levar um telefone VoIP para uma viagem e onde você conectá-lo à Internet pode-se receber ligações. Além disso, a tecnologia VoIP apresenta ainda mais vantagens no uso empresarial, fornecendo importantes ferramentas de comunicação interna ou interligando a matriz com suas filiais e/ou escritórios. Dentre elas, estão:

1. A necessidade de uma única infra-estrutura para prover serviços de link de dados e telefonia;
2. Custo zero para ligações dentro da empresa (entre matriz e filiais ou entre filiais) e de Telefone IP para qualquer outro Telefone IP;
3. Redução drástica de custos para ligações interurbanas e internacionais;
4. Eficiência em comunicação com custo acessível;
5. Integração com o PABX da empresa;
6. Correio de Voz com integração de correio eletrônico;
7. Centralização do fluxo telefônico gerando melhor controle;
8. Otimização;
9. Redução do número de troncos convencionais;
10. Fácil Implantação dos equipamentos VoIP;
11. Suporte e assistência permanentes;
12. Qualidade de voz equivalente à telefonia convencional;
13. Mobilidade e flexibilidade dos ramais, já que os ramais dotados de infra-estrutura VoIP (*IPFone*, *Softfone* ou dispositivo ATA) podem conectar-se com a estrutura VoIP da empresa de qualquer ponto do mundo, bastando para isso um link Web e um *IPFone*.

O VoIP faz com que as redes de telefonia se "misturem" às redes de dados. Isso possibilita que, usando um microfone, caixas ou fones de som e um software apropriado, o usuário faça uma ligação para telefones convencionais por meio de seu computador para qualquer telefone fixo e celular de qualquer parte do Brasil e ou do mundo, pagando valores irrisórios, se comparado às tarifas das operadoras

convencionais. Por tudo isso o VoIP está cada vez mais popular e surge cada vez mais empresas que lidam com essa tecnologia.

4 PLANO DE DESENVOLVIMENTO

No plano de desenvolvimento do software foi utilizado dois laptops da fabricante Samsung, um com o processador *Intel® Core™ i5 – 3230M CPU @ 2.60 GHz*, RAM de 8.00 GB, sistema operacional *Windows 10 Pro x64* e HD de 1.00 TB, outro com o processador *Intel® Core™ i7 – 5500U CPU @ 2.40 GHz*, RAM de 8.00 GB, sistema operacional *Windows 10 Home Single Language x64* e HD de 1.00 TB, um computador da fabricante Acer com o processador *Intel® Core™ i5 – 525D CPU @ 1.80 GHz*, RAM de 2.00 GB, sistema operacional *Windows 8.1 Pro x32* e HD de 500 GB, e um laptop da fabricante Dell com o processador *Intel® Core™ i7 – 5500U CPU @ 2.40 GHz*, RAM de 8.00 GB, sistema operacional *Windows 10 Home Single Language x64* e HD de 1.00 TB. Os principais softwares utilizados no processo foram TFS – *Team Foundation Service*, *Pencil Project*, *StarUML*, *brModelo*, *Toad Data Modeler 6.1*, *Microsoft Visio 2016*, *Microsoft Excel 2016* e *Microsoft Word 2016*.

O principal livro utilizado foi “Princípios de Análise e Projeto de Sistemas com UML” de Bezerra. Ele se encontra na referência bibliográfica.

4.1 Conceitos básicos de banco de dados relacionais

O elemento fundamental de um banco de dados relacional é a tabela. Uma tabela é um elemento do banco de dados que consiste num conjunto de dados dispostos em forma de linhas com conjuntos idênticos de propriedades (registros). Os valores associados aos registros da tabela aparecem em colunas (campos).

Um banco de dados é um conjunto de tabelas relacionadas.

Uma tabela é um conjunto de informações sobre uma entidade dispostas em formas de linhas e colunas.

Um registro (tupla) é uma linha da tabela, ou seja, representa todas as informações de uma entidade em particular.

Um campo é uma coluna da tabela, ou seja, representa uma das informações do registro (funcionário).

A interseção entre a linha e a coluna de uma tabela é chamada atributo e representa o valor de um campo.

É desejável, mas não essencial que cada registro de uma tabela tenha um conjunto de atributos segundo os quais seja possível identificar inequivocamente o

registro dentro da tabela. Esse conjunto é chamado identificador, sendo boa prática de projeto de banco de dados a sua existência. O campo que possui o atributo identificador é chamado chave primária (ID).

Por definição, o conjunto de valores dos atributos constituintes da chave primária deve ser único para cada registro.

Um Sistema de Gerenciamento de Banco de Dados (SGBD) é o conjunto de programas de computador responsáveis pelo gerenciamento de uma base de dados. Seu principal objetivo é retirar da aplicação cliente a responsabilidade de gerenciar o acesso, a manipulação e a organização dos dados. O SGBD disponibiliza uma interface para que seus clientes possam incluir, alterar ou consultar dados previamente armazenados. Em bancos de dados relacionais a interface é constituída pelas APIs (*Application Programming Interface*) ou drivers do SGBD, que executam comandos na linguagem SQL.

Foi utilizado o sistema gerenciador de banco de dados Oracle *Database* 11g *Release* dois para a construção de nosso banco de dados, do qual armazenará os registros de cada jogador e ordenará em ordem decrescente com base na pontuação individual.

O Oracle *Database* 11g *Release* oferece a base para que a TI forneça com sucesso mais informações com melhor qualidade de serviço, reduza o risco de mudança dentro de suas funções e faça um uso mais eficiente dos orçamentos de TI. Ao implantar o Oracle *Database* 11g *Release* dois como base para a gestão de dados, as organizações podem usar a capacidade total do banco de dados líder mundial para:

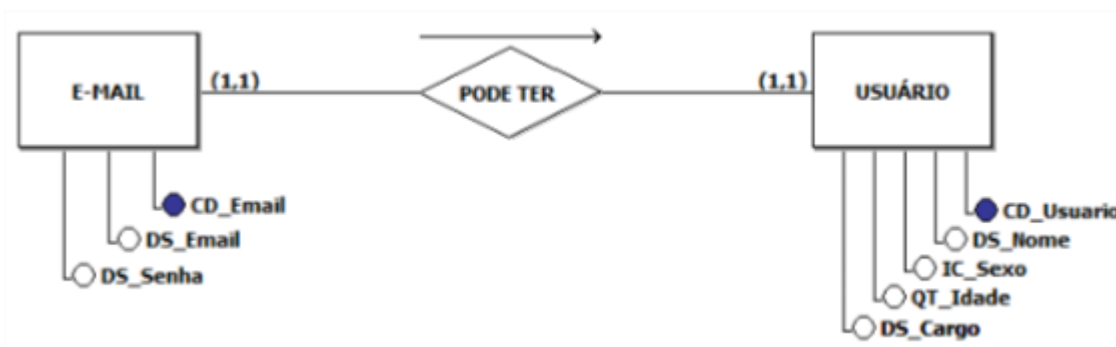
- Reduzir os custos com servidor por um fator de cinco;
- Reduzir os custos com armazenamento por um fator de 12;
- Melhorar o desempenho dos sistemas de missão crítica por um fator de 10;
- Aumentar a produtividade dos DBAs (administradores de banco de dados) por um fator de dois;
- Eliminar a redundância ociosa no centro de dados;
- Simplificar o portfólio geral de software para TI.

Para o planejamento do banco de dados, estivermos levantando a seguinte questão: “um e-mail pode ter quantas contas? Uma conta pode ter quantos e-mail?”. Para replicar este ponto foram empregados o DER, MER, DD + trigramação e as cinco normalizações.

4.1.1 DER

Para a construção do diagrama de entidade e relacionamento foi utilizado o programa brModelo. Segue o resultado:

Imagem 08 – DER do banco de dados.

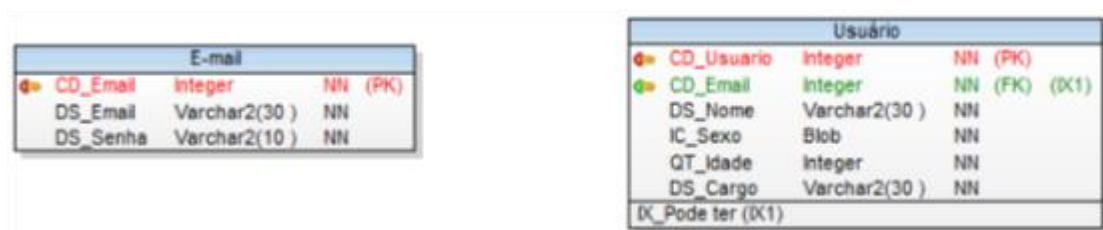


Fonte: Própria, 2017.

4.1.2 MER

Para a construção do modelo de entidade e relacionamento foi utilizado o programa *Toad Data Modeler*. Segue o resultado:

Imagem 09 – MER do banco de dados.



Fonte: Própria, 2017.

4.1.3 DD + trigramação

Para a construção do dicionário de dados mais trigramação (é uma técnica de banco de dados que se utiliza de prefixos para a identificação de campos conforme suas tabelas) foi utilizado o programa Microsoft Excel 2016. Segue o resultado:

Tabela 01 – DD + Trigramação e-mail.

E-mail (EMA)			
Campo	TD	TM	CH
EMA_Codigo	+	-	PK
EMA_Email	A	30	-
EMA_Senha	A	10	-

Fonte: Própria, 2017.

Tabela 02 – DD + Trigramação usuário.

Usuário (USU)			
Campo	TD	TM	CH
USU_Codigo	+	-	PK
EMA_Codigo	N	-	FK
USU_Nome	A	30	-
USU_Sexo	B	-	-
USU_Idade	N	-	-
USU_Cargo	A	30	-

Fonte: Própria, 2017.

4.1.4 Classificação de dados

TABELA CADASTRO

CD_TABELA	+
{ DS_EMAIL }	+
DT_REALIZACAO	+
DS_SENHA	+
DS_NOME	+
IC_SEXO	+
QT_IDADE	+
DS_CARGO	+

4.1.5 *Primeira forma normal***CADASTRO**CD_TABELA

DT_REALIZACAO

DS_SENHA

DS_NOME

IC_SEXO

QT_IDADE

DS_CARGO

E-MAILCD_EMAIL

DS_EMAIL

4.1.6 *Segunda forma normal***CADASTRO**CD_TABELA

DT_REALIZACAO

DS_NOME

IC_SEXO

QT_IDADE

DS_CARGO

E-MAILCD_EMAIL

DS_EMAIL

DS_SENHA

4.1.7 *Terceira forma normal***CADASTRO**CD_TABELACD_EMAILCD_USUARIO

DT_REALIZACAO

E-MAILCD_EMAIL

DS_EMAIL

DS_SENHA

USUARIOCD_USUARIO

QT_IDADE

CD_EMAIL

DS_CARGO

DS_NOME

IC_SEXO

4.1.8 Forma normal de boyce-codd

Tabela 03 – Tabela cadastro.

TABELA CADASTRO
CD_TABELA
DS_EMAIL
DT_REALIZACAO
DS_SENHA
DS_NOME
IC_SEXO
QT_IDADE
DS_CARGO

Fonte: Própria, 2017.

Chaves candidatas: DS_EMAIL + DS_SENHA. DS_NOME + DS_EMAIL. DT_REALIZACAO + DS_EMAIL + DS_NOME.

Tabela 04 – Tabela chaves candidatas.

Determinante	Dependentes Funcionais
DS_EMAIL + DS_SENHA	-
DS_NOME + DS_EMAIL	-
DT_REALIZACAO + DS_EMAIL + DS_NOME	CD_TABELA

Fonte: Própria, 2017.

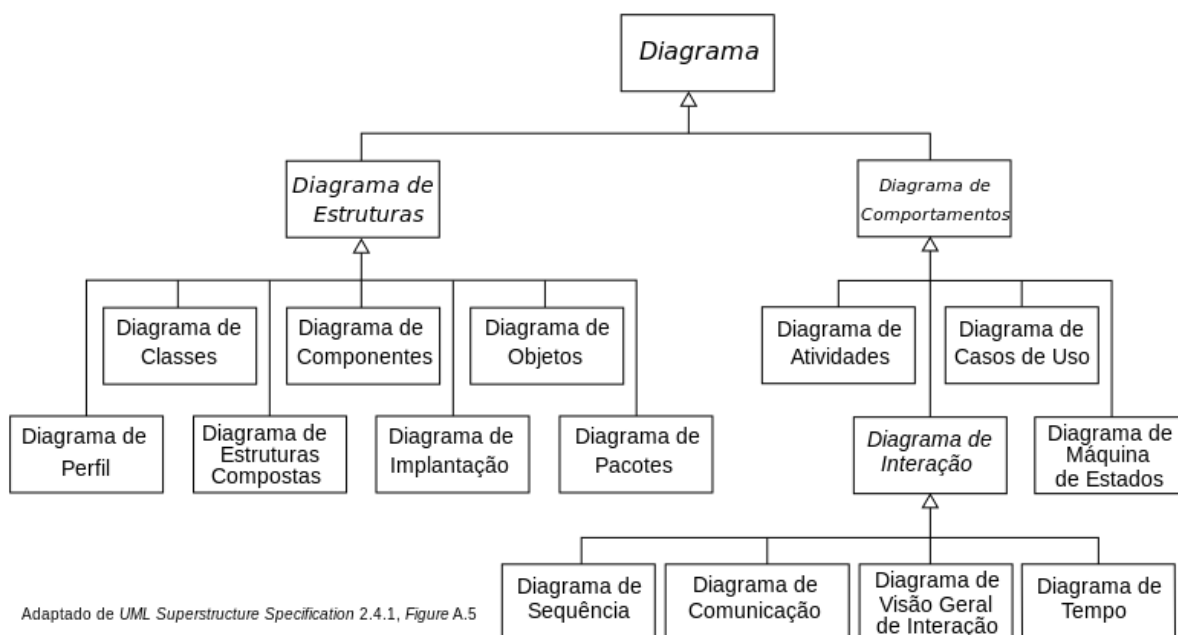
4.2 Modelagem do sistema

A UML (*Unified Modeling Language*) é uma linguagem visual para modelar sistemas orientados a objetos. Isso quer dizer que a UML é uma linguagem constituída de elementos gráficos (visuais) utilizados na modelagem que permitem representar os conceitos do paradigma da orientação a objetos. Através dos elementos gráficos definidos nesta linguagem pode-se construir diagramas que apresentam diversas perspectivas de um sistema.

Cada elemento gráfico possui uma sintaxe (isto é, uma forma predeterminada de desenhar o elemento) e uma semântica que definem o que significa o elemento e para que ele deve ser utilizado. Além disso tanto a sintaxe quanto a semântica da UML são extensíveis. Essa extensibilidade permite que a UML seja adaptada às características específicas de cada projeto de desenvolvimento.

Bezerra (2015, p. 14) destaca que a UML é independente tanto de linguagens de programação quanto de processos de desenvolvimento. Isso quer dizer que a UML pode ser utilizada para a modelagem de sistemas, não importa qual a linguagem de programação será utilizada na implementação do sistema, ou qual a forma (processo) de desenvolvimento adotada.

Imagem 10 – Diagramas da UML 2.0.



Fonte: <https://pt.wikipedia.org/wiki/UML>, 2017.

Foi utilizado o software StarUML para a construções dos diagramas da UML. Não foram aplicados todos os quatorze diagramas, porque não foram vistas necessidades.

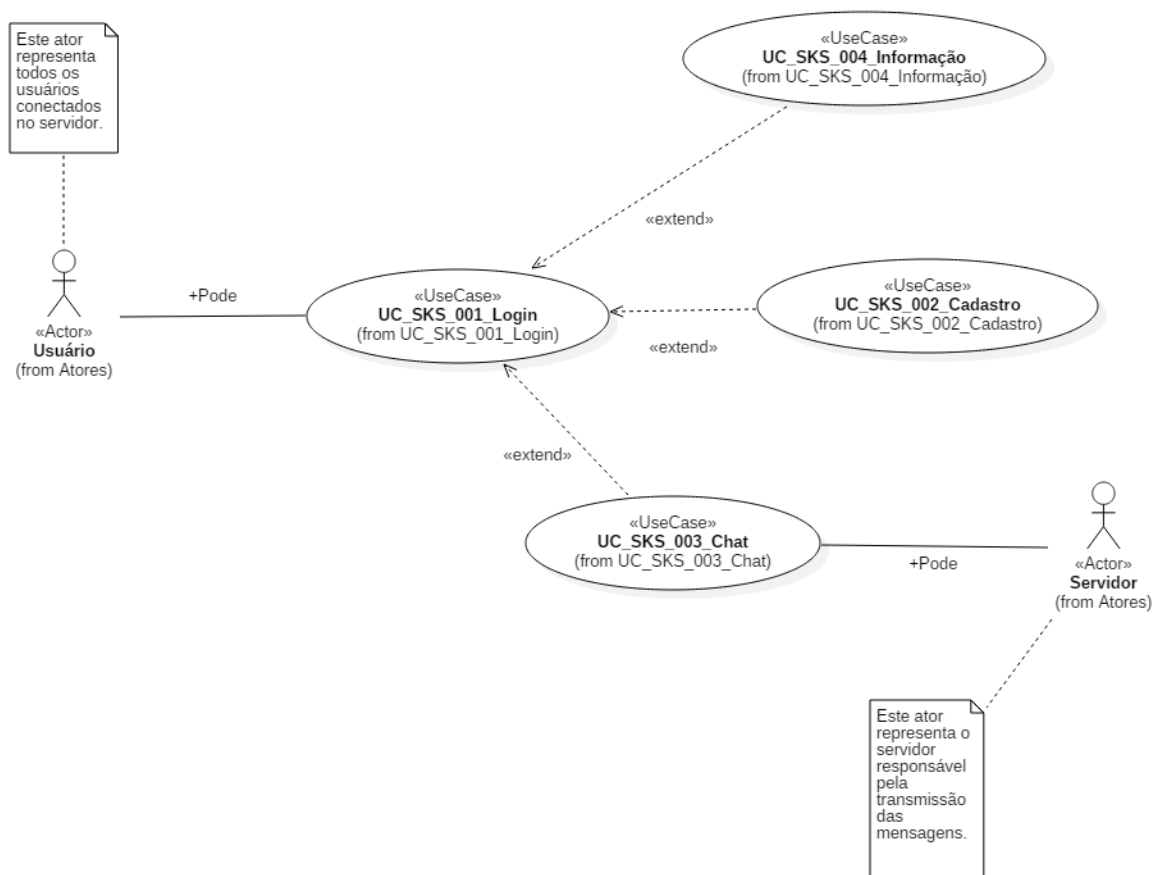
4.2.1 Diagrama de casos de uso

O modelo de casos de uso é uma representação das funcionalidades externamente observáveis do sistema e dos elementos externos ao sistema que

interagem com ele. Este modelo é parte integrante da especificação de requisitos. Na verdade, o modelo de casos de uso molda os requisitos funcionais do sistema.

4.2.1.1 Caso de uso login

Imagem 11 – Caso de uso login.



Fonte: Própria, 2017.

Nome: login.

Descrição: acesso ao chat e ao cadastro.

Atores: usuário.

Condições: software instalado e iniciado.

Dados Input: mouse e teclado.

Dados Output: janela conforme o evento que o usuário causar.

Tabela 05 – Fluxo principal login.

Fluxo Principal	
Ator	Sistema
01 – O usuário digita o nome e a senha.	-----
02 – O usuário seleciona o botão “entrar”.	01 – O sistema verifica se o usuário e a senha estão no banco de dados.
-----	02 – O sistema leva o usuário a tela do “chat”.

Fonte: Própria, 2017.

Tabela 06 – Fluxo alternativo 01 login.

Fluxo Alternativo	
Ator	Sistema
01 – O usuário seleciona o botão “cadastrar”.	01 – O sistema abre a janela “cadastro”.

Fonte: Própria, 2017.

Tabela 07 – Fluxo alternativo 02 login.

Fluxo Alternativo	
Ator	Sistema
01 – O usuário seleciona o botão “informação”.	01 – O sistema abre a janela “informação”.

Fonte: Própria, 2017.

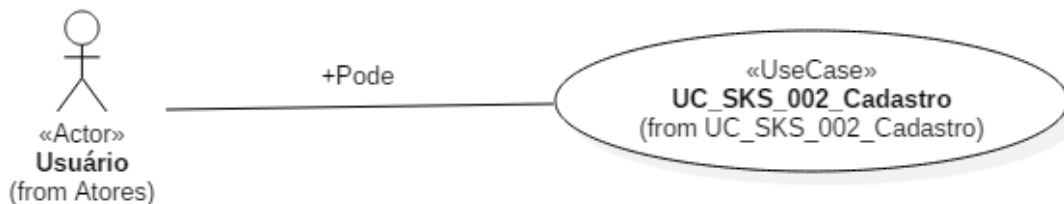
Tabela 08 – Fluxo exceção login.

Fluxo Exceção	
Ator	Sistema
01 – O usuário digita o nome ou a senha errada.	
02 – O usuário seleciona o botão “entrar”.	01 – O sistema mostra um aviso de “nome ou senha inválido”.

Fonte: Própria, 2017.

4.2.1.2 Caso de uso cadastro

Imagem 12 – Caso de uso cadastro.



Fonte: Própria, 2017.

Nome: cadastro.

Descrição: realização de um cadastro.

Atores: usuário.

Condições: software instalado e iniciado.

Dados Input: mouse e teclado.

Dados Output: janela conforme o evento que o usuário causar.

Tabela 09 – Fluxo principal cadastro.

Fluxo Principal	
Ator	Sistema
01 – O usuário preenche todos os campos.	-----
02 – O usuário seleciona o botão “cadastrar”.	01 – O sistema verifica se as informações são válidas.
-----	02 – O sistema armazena as informações no banco de dados.

Fonte: Própria, 2017.

Tabela 10 – Fluxo alternativo cadastro.

Fluxo Alternativo	
Ator	Sistema
01 – O usuário seleciona o botão “voltar”.	01 – O sistema volta para a tela de “login”.

Fonte: Própria, 2017.

Tabela 11 – Fluxo exceção cadastro.

Fluxo Exceção	
Ator	Sistema
01 – O usuário preenche todos os campos.	-----
02 – O usuário seleciona o botão “cadastrar”.	01 – O sistema verifica se as informações são válidas.
-----	02 – O sistema identifica que o e-mail já consta na base de dados.
-----	03 – O sistema mostra uma mensagem e aviso.

Fonte: Própria, 2017.

4.2.1.3 Caso de uso chat

Imagem 13 – Caso de uso chat.



Fonte: Própria, 2017.

Nome: chat.

Descrição: realização da comunicação.

Atores: usuário e servidor.

Condições: software instalado e iniciado.

Dados Input: mouse e teclado.

Dados Output: mensagem enviada por outro usuário.

Tabela 12 – Fluxo principal chat.

Fluxo Principal	
Ator	Sistema
01 – O usuário digita uma mensagem.	-----
02 – O usuário seleciona o botão “enviar”.	01 – O sistema envia a mensagem para o servidor.
03 – O servidor encaminha a mensagem para os demais usuários na rede.	02 – O sistema recebe a mensagem do servidor e mostra para o usuário.

Fonte: Própria, 2017.

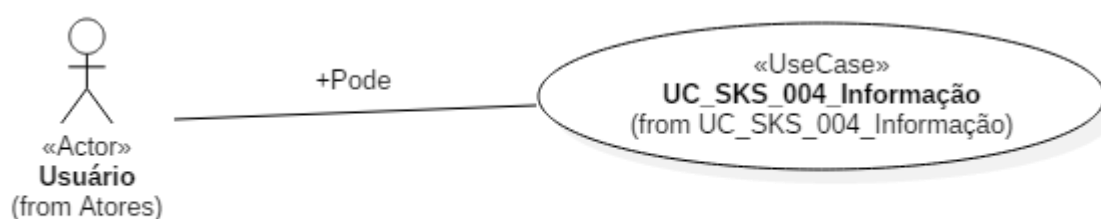
Tabela 13 – Fluxo exceção chat.

Fluxo Exceção	
Ator	Sistema
01 – O usuário digita uma mensagem.	-----
02 – O usuário seleciona o botão “enviar”.	01 – O sistema envia a mensagem para o servidor.
-----	02 – O sistema identifica que o servidor não está ligado.
-----	03 – O sistema mostra uma mensagem de erro.

Fonte: Própria, 2017.

4.2.1.4 Caso de uso informação

Imagem 14 – Caso de uso informação.



Fonte: Própria, 2017.

Nome: informação.

Descrição: mostra os nomes dos desenvolvedores.

Atores: usuário.

Condições: software instalado e iniciado.

Dados Input: mouse e teclado.

Dados Output: nomes dos desenvolvedores.

Tabela 14 – Fluxo principal informação.

Fluxo Principal	
Ator	Sistema
01 – O usuário seleciona o botão “voltar”.	01 – O sistema volta para a tela de “login”.

Fonte: Própria, 2017.

4.3 Linguagem de programação

A linguagem de programação definida é o Java *Standard Edition* 8 (JSE 8).

4.3.1 Java Standard Edition

É uma ferramenta de desenvolvimento para a plataforma Java. Ela contém todo o ambiente necessário para a criação e execução de aplicações Java, incluindo a máquina virtual Java, o compilador Java, as APIs do Java e outras ferramentas utilitárias para uma melhor funcionalidade.

4.4 Diagrama de bloco

Para a construção do diagrama de bloco foi utilizado o programa Microsoft Visio 2016.

4.4.1 Diagrama do menu

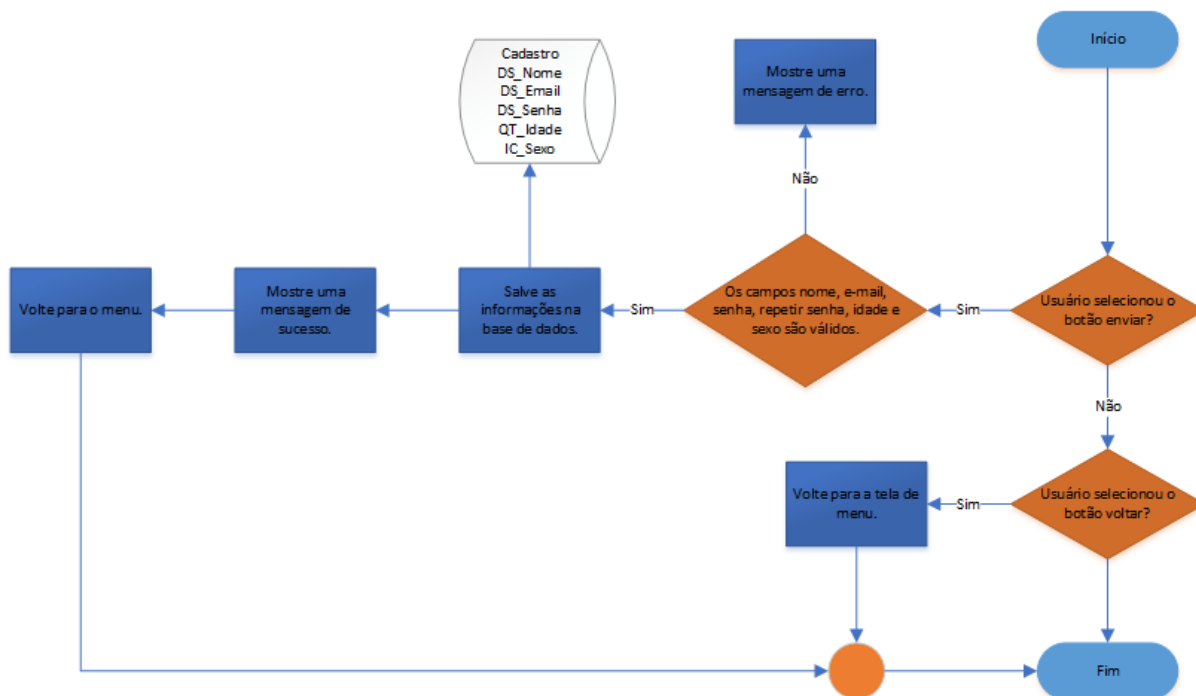
Imagem 15 – Diagrama do menu.



Fonte: Própria, 2017.

4.4.2 Diagrama de cadastro

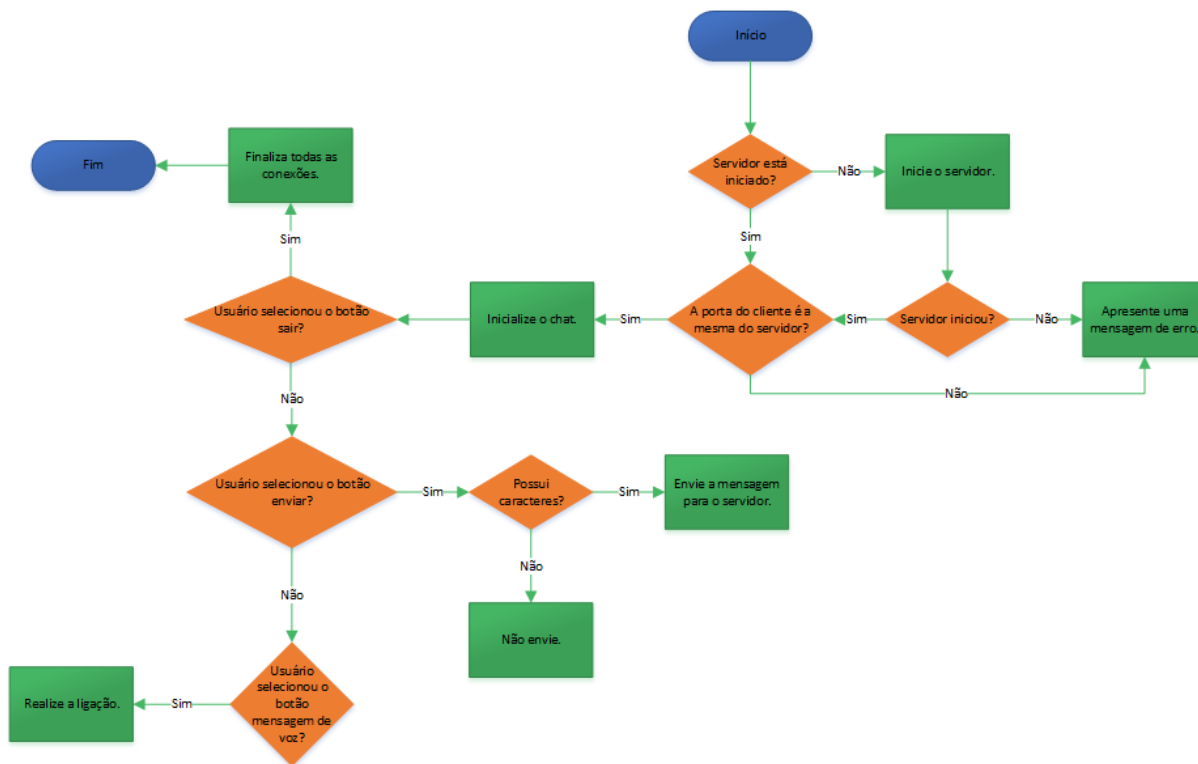
Imagem 16 – Diagrama de cadastro.



Fonte: Própria, 2017.

4.4.3 Diagrama do chat

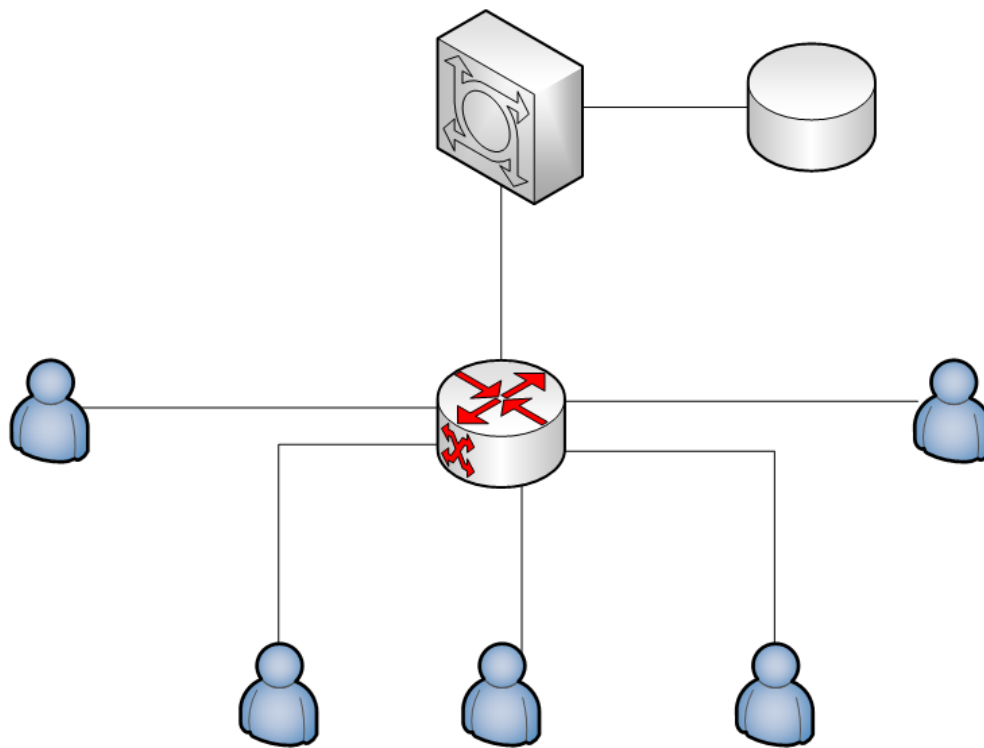
Imagem 17 – Diagrama do chat.



Fonte: Própria, 2017.

4.4.4 Diagrama da rede

Imagem 18 – Diagrama da rede.



Fonte: Própria, 2017.

4.5 Conceito geral do programa

O programa *SmokeSignal* (Sinal de Fumaça) foi desenvolvido com o intuito de aproximar pessoas de lugares diferentes. O nome foi inspirado na antiga forma de comunicação à distância.

4.5.1 Protótipos

Para a construção dos protótipos de telas foi utilizado o programa *Pencil Project*.

4.5.2 Protótipo de login

Imagem 19 – Protótipo de Login.

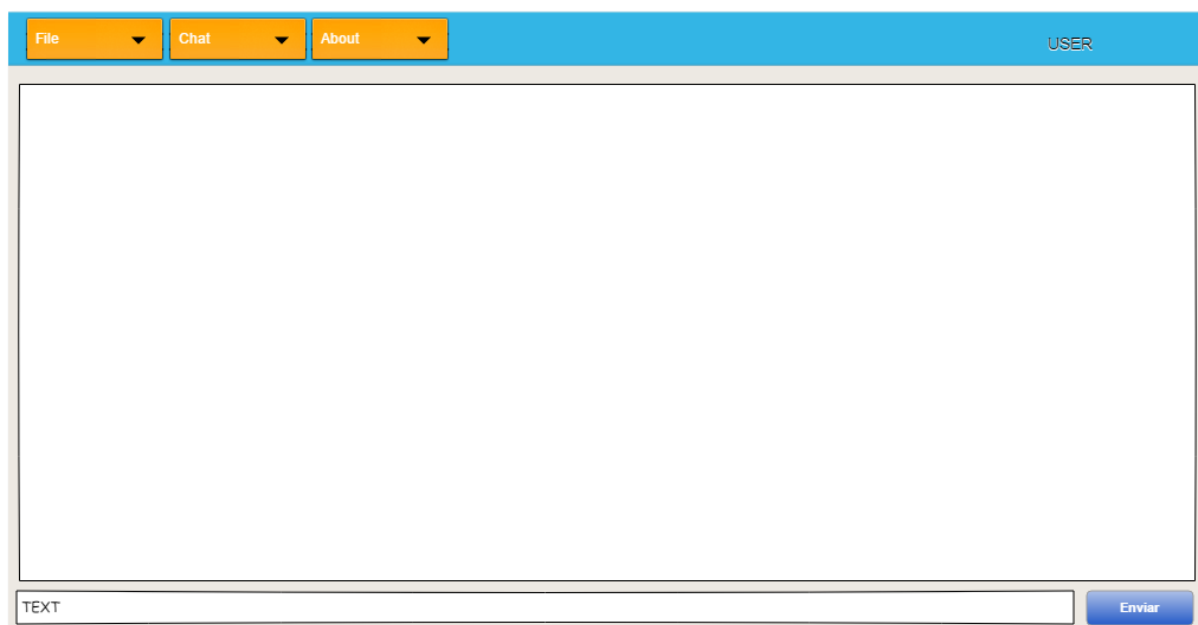


The image shows a login form titled "SmokeSignal" centered on a light gray background. The form has a white background and a blue header bar with the text "SmokeSignal". Below the header, there are two input fields: "Nome" with a placeholder "TEXT" and "Senha" with a placeholder "*****". Below the input fields, there are two blue buttons: "Login" and "Esqueceu a senha ?".

Fonte: Própria, 2017.

4.5.3 Protótipo de chat

Imagem 20 – Protótipo de Chat.



The image shows a chat application interface. At the top, there is a blue header bar with three orange buttons labeled "File", "Chat", and "About", each with a downward arrow. To the right of these buttons, the text "USER" is displayed. Below the header bar is a large white rectangular area for chat messages. At the bottom, there is a text input field with the placeholder "TEXT" and a blue button labeled "Enviar".

Fonte: Própria, 2017.

4.5.4 Protótipo de informação

Imagem 21 – Protótipo de Informação.

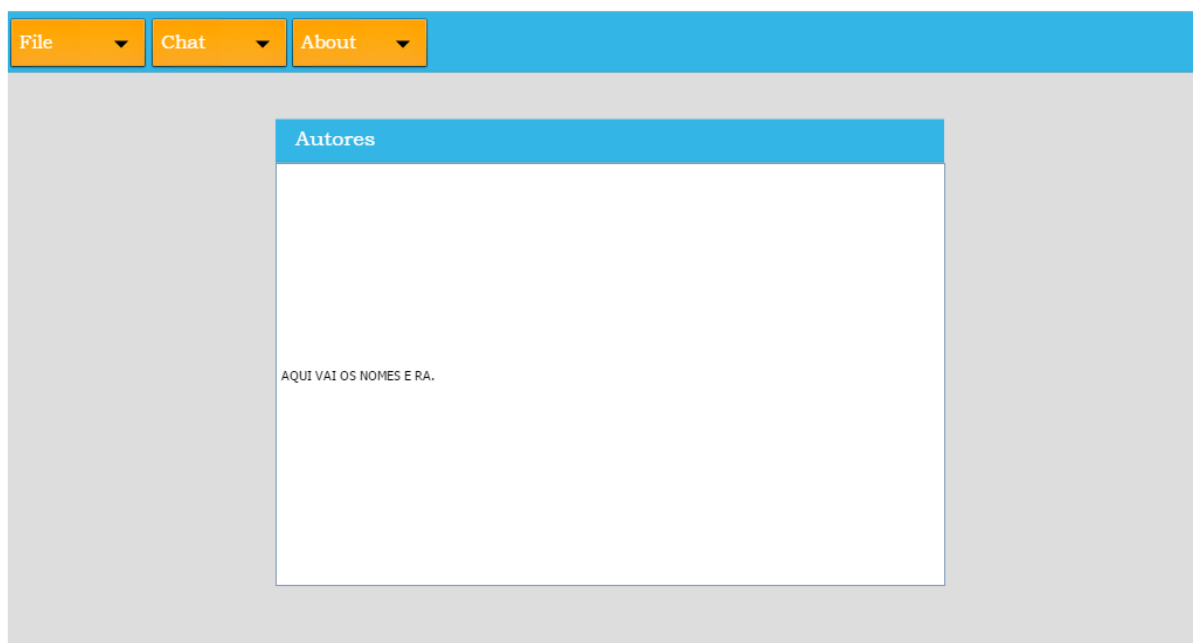


Imagem 21 – Protótipo de Informação. A interface apresenta uma barra superior azul com três botões de menu: 'File', 'Chat' e 'About', todos em fundo amarelo com setas para baixo. Abaixo, há uma caixa centralizada com o título 'Autores' em uma barra azul. O corpo da caixa é branco e contém o texto 'AQUI VAI OS NOMES E RA.'.

Fonte: Própria, 2017.

4.5.5 Protótipo de cadastro

Imagem 22 – Protótipo de Cadastro.

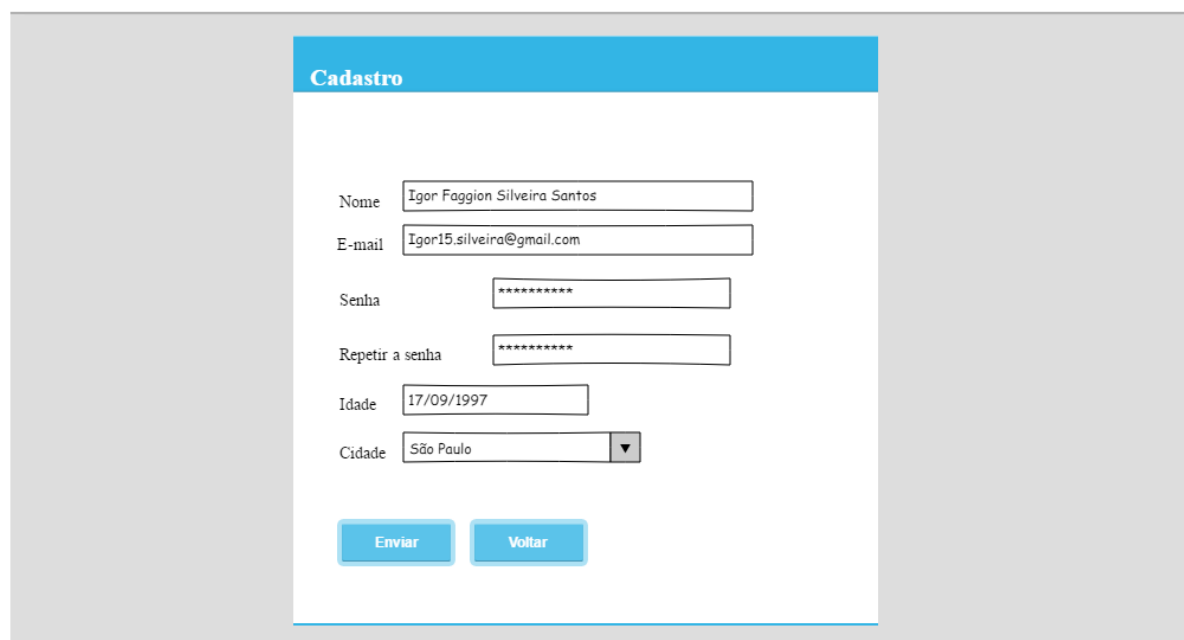


Imagem 22 – Protótipo de Cadastro. A interface apresenta uma caixa centralizada com o título 'Cadastro' em uma barra azul. Abaixo, há campos de formulário para: Nome (Igor Faggion Silveira Santos), E-mail (Igor15.silveira@gmail.com), Senha (*****), Repetir a senha (*****), Idade (17/09/1997) e Cidade (São Paulo). Abaixo dos campos, há dois botões: 'Enviar' e 'Voltar'.

Fonte: Própria, 2017.

5 DESENVOLVIMENTO

No desenvolvimento do software foi utilizado dois laptops da fabricante Samsung, um com o processador *Intel® Core™ i5 – 3230M CPU @ 2.60 GHz*, RAM de 8.00 GB, sistema operacional *Windows 10 Pro x64* e HD de 1.00 TB, outro com o processador *Intel® Core™ i7 – 5500U CPU @ 2.40 GHz*, RAM de 8.00 GB, sistema operacional *Windows 10 Home x64* e HD de 1.00 TB e um laptop da fabricante Dell com o processador *Intel® Core™ i7 – 5500U CPU @ 2.40 GHz*, RAM de 8.00 GB, sistema operacional *Windows 10 Home Single Language x64* e HD de 1.00 TB. Os principais softwares utilizados no processo foram a IDE Eclipse Java Mars x64 com acesso ao JDK 1.8.0_121 x64 e JRE 1.8.0_121 x64, últimas versões até o presente momento de dissertação, *Oracle Database 11g Express Edition*, *Sublime Text* três, *Visual StudioCode*, *SQL Developer*, *TFS - Team Foundation Service*, *SceneBuilder*, *Notepad* e o *Prompt de Comando*.

Os principais livros utilizados foram “Programação de computadores em Java” de Rui Rossi dos Santos e “JavaFX: interfaces com qualidade para aplicações desktop” de Bruno Oliveira. Eles se encontram na referência bibliográfica

A linguagem de programação utilizada foi *Java Standard Edition 8 (JSE 8)*.

5.1 Interface gráfica

No desenvolvimento da interface gráfica foi utilizado a ferramenta *JavaFX Scene Builder*. Esta ferramenta permite desenhar interfaces de usuário de aplicativo *JavaFX* sem codificação. Os usuários podem arrastar e soltar componentes de UI para uma área de trabalho, modificar suas propriedades, aplicar folhas de estilo e o código *FXML* para o layout que eles estão criando é gerado automaticamente em segundo plano. O resultado é um arquivo *FXML* que pode ser combinado com um projeto *Java* vinculando a interface ao aplicativo.

A primeira versão do *JavaFX Script* saiu em maio de 2007, em uma conferência da *JavaOne*. Os planos eram audaciosos: em pouco tempo, elevar o *JavaFX* para *Desktop* e *Browser*, e futuramente para dispositivos móveis. (OLIVEIRA, 2015, p. 01).

JavaFX utiliza o conceito RIA (*Rich Internet Application*), tornando aplicações Desktop com qualidade gráfica altíssima e conceitos de programação eficazes, o que o fez ser uma saída para as aplicações Swing, do Java, cujo gráfico deixava a desejar.

5.2 Banco de dados

Para o desenvolvimento do banco de dados foi utilizado o SGBD *Oracle Database 11g Express Edition*. A linguagem utilizada no *Oracle Database 11g Express Edition* é a *PL/SQL*. Segue um dos códigos utilizados no desenvolvimento do banco.

```
CREATE TABLE USUARIO(
    "USU_CODIGO" NUMBER(4,0) NOT NULL ENABLE,
    "EMA_CODIGO1" NUMBER(4,0) NOT NULL ENABLE,
    "USU_NOME" VARCHAR(30) NOT NULL ENABLE,
    "USU_SEXO" VARCHAR(2) NOT NULL ENABLE,
    "USU_IDADE" NUMBER(8) NOT NULL ENABLE,
    "USU_CARGO" VARCHAR(30) NOT NULL ENABLE,
    PRIMARY KEY("USU_CODIGO") ENABLE,
    FOREIGN KEY("EMA_CODIGO1") REFERENCES EMAIL("EMA_CODIGO")
);
```

Após a criação da entidade “EMAIL” foi criada a entidade “USUARIO” com cinco atributos, do qual o primeiro, USU_CODIGO é chave primário e os demais são atributos “comuns”. Ela possui uma referência para a chave primária “EMA_CODIGO” da entidade “EMAIL” e aceita apenas os caracteres “F” e “M” no atributo “USU_SEXO”.

5.3 Servidor de mensagem

A classe *ChatServer* representa um modelo de servidor de mensagens, o serviço do computador que funciona como base deve, primeiro, abrir uma porta e ficar ouvindo até alguém tentar se conectar.

```
try {
    server = new ServerSocket(5000);
} catch (IOException e1) {
    e1.printStackTrace(); }
```

Se o objeto for realmente criado, significa que a porta 5000 estava fechada e foi aberta. Se outro programa possui o controle desta porta neste instante, é normal que a nossa classe não funcione, pois ele não consegue utilizar uma porta que já está em uso.

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 *bloquante*, segura a *thread* até que algo notifique.

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*.

A *thread* criada será responsável pelo tratamento dessa conexão, enquanto o laço do servidor disponibilizará a porta para uma nova conexão.

```
while(true) {
    try {
        socket = server.accept();
        new Thread(new EscutaCliente(socket)).start();
        p = new PrintWriter(socket.getOutputStream());
        escritores.add(p);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

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

```
public EscutaCliente(Socket socket) {
    try {
        leitor = new Scanner(socket.getInputStream());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```

        }
    }
    public void run() {
        String texto = null;
        while ((texto = leitor.nextLine()) != null)
            encaminharParaTodos(texto);}
    }
}

```

5.4 Cliente

A classe *JFChat*, além de cuidar da parte GUI do chat, possui a tarefa de conectar o cliente no servidor de mensagens e áudios. O cliente é ainda mais simples do que o servidor.

O código a seguir é a parte principal e tenta se conectar a um servidor na máquina local e porta 5000.

```
socket = new Socket(InetAddress.getLocalHost().getHostAddress(), 5000);
```

Para ler os dados que o cliente digita no campo *JTextField* foi criado uma classe interna a classe *JFChat*. Segue o código:

```

private class falaServidor implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        escritor.println(nome + " : " + textEnvia.getText());
        escritor.flush();
        textEnvia.setText("");
        textEnvia.requestFocus();
    }
}

```

Quando o cliente seleciona o botão “enviar” o sistema captura essa mensagem manda para o servidor e escrever no objeto *JTextArea*.

O trecho do código responsável por pegar e enviar as mensagens se encontra logo abaixo:

```

try{
    escritor = new PrintWriter(socket.getOutputStream());
    leitor = new Scanner(socket.getInputStream());
}

```

```

        new Thread(new escutaServidor()).start();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

5.5 Servidor de áudio

A classe *AudioServer* representa um modelo de servidor de áudio. Ela funciona basicamente igual ao servidor de mensagens, as diferenças é que em vez dela receber caracteres, ela recebe bytes que são transformados em voz.

Primeiro devermos buscar as informações do equipamento de entrada e saída de voz. Segue o segmento:

```

mixer = AudioSystem.getMixer(mixerInfo[1]);
audioFormat = getAudioFormat();
dataLineInfo = new DataLine.Info(SourceDataLine.class, audioFormat);
sourceDataLine = (SourceDataLine) AudioSystem.getLine(dataLineInfo);
sourceDataLine.open(audioFormat);
sourceDataLine.start();

```

Segue o fragmento do código que é responsável por capturar o áudio.

```

private void captureAudio() {
    try {
        audioFormat = getAudioFormat();
        dataLineInfo1 = new DataLine.Info(TargetDataLine.class, audioFormat);
        for (int cnt = 0; cnt < mixerInfo.length; cnt++) {
            mixer1 = AudioSystem.getMixer(mixerInfo[3]);
            if (mixer1.isLineSupported(dataLineInfo))
                targetDataLine = (TargetDataLine) mixer.getLine(dataLineInfo);
            targetDataLine.open(audioFormat);
            targetDataLine.start();
            Thread captureThread = new CaptureThread();
            captureThread.start();
        } catch (Exception e) {
            e.printStackTrace();}}

```

Após a captura do áudio, basta enviá-lo para o cliente.

```
input = new BufferedInputStream(ClientSocket.getInputStream());
out = new BufferedOutputStream(ClientSocket.getOutputStream());
while (input.read(tempBuffer) != -1)
    sourceDataLine.write(tempBuffer, 0, size);
```

5.6 Captura de áudio

A classe VoIP é responsável por capturar o áudio do cliente, transformá-lo em bytes e enviá-lo para o servidor de áudio.

As configurações de hardware são semelhantes ao do servidor a única diferença é que a VoIP possui o método responsável por encaminhar o áudio para o servidor. Segue um fragmento do mesmo:

```
public VoIP() {
    try {
        sock = new Socket(InetAddress.getLocalHost().getHostAddress(), 500);
        out = new BufferedOutputStream(sock.getOutputStream());
        in = new BufferedInputStream(sock.getInputStream());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

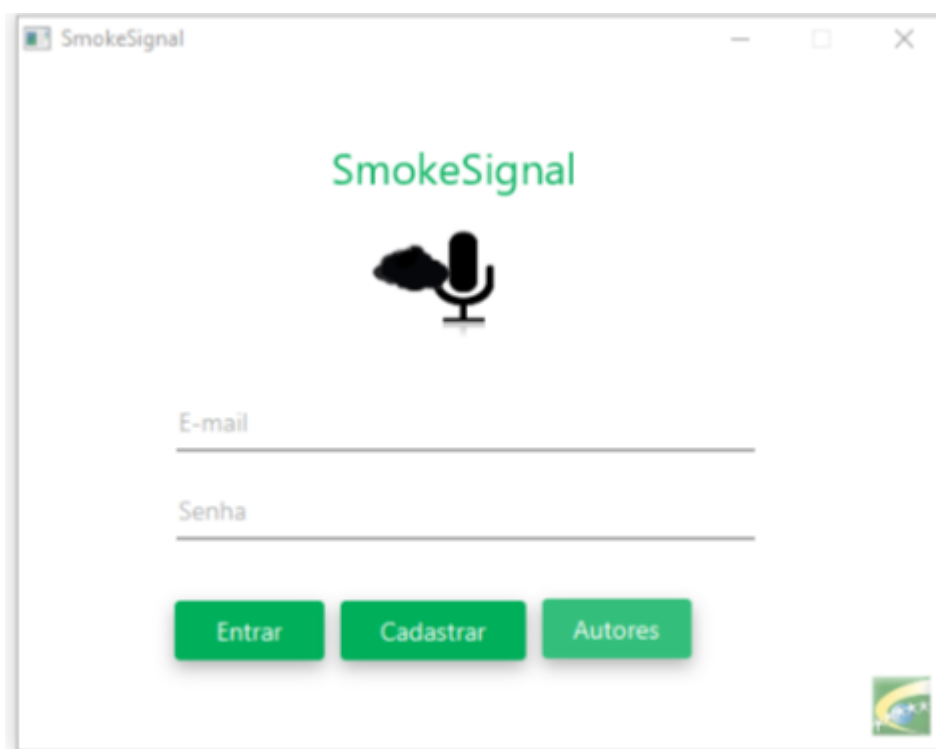
Assim que o objeto VoIP é criado, ele já inicia a captura e o encaminhamento do áudio para o servidor.

6 RESULTADOS

Nos testes do *SmokeSignal* foram utilizados dois laptops da fabricante Samsung, um com o processador *Intel® Core™ i5 – 3230M CPU @ 2.60 GHz*, RAM de 8.00 GB, sistema operacional *Windows 10 Pro x64* e HD de 1.00 TB, outro com o processador *Intel® Core™ i7 – 5500U CPU @ 2.40 GHz*, RAM de 8.00 GB, sistema operacional *Windows 10 Home x64* e HD de 1.00 TB, um laptop da fabricante Dell com o processador *Intel® Core™ i7 – 5500U CPU @ 2.40 GHz*, RAM de 8.00 GB, sistema operacional *Windows 10 Home Single Language x64* e HD de 1.00 TB, um computador da fabricante QBEX com o processador *Intel® Core™ i5 – 3330 CPU @ 3.00 GHz*, RAM de 6.00 GB, sistema operacional *Windows 7 Home Premium x64* e HD de 750.00 GB, dois roteadores um da fabricante *D-Link* modelo *DI – 524 Wireless 150* e outro da Cisco modelo *DPC 3925 DOCIS 3.0 Wireless*.

Segue uma lista de Printscreen do *SmokeSignal* em execução.

Imagem 23 – Login.



Fonte: Própria, 2017.

Imagem 24 – Informação.



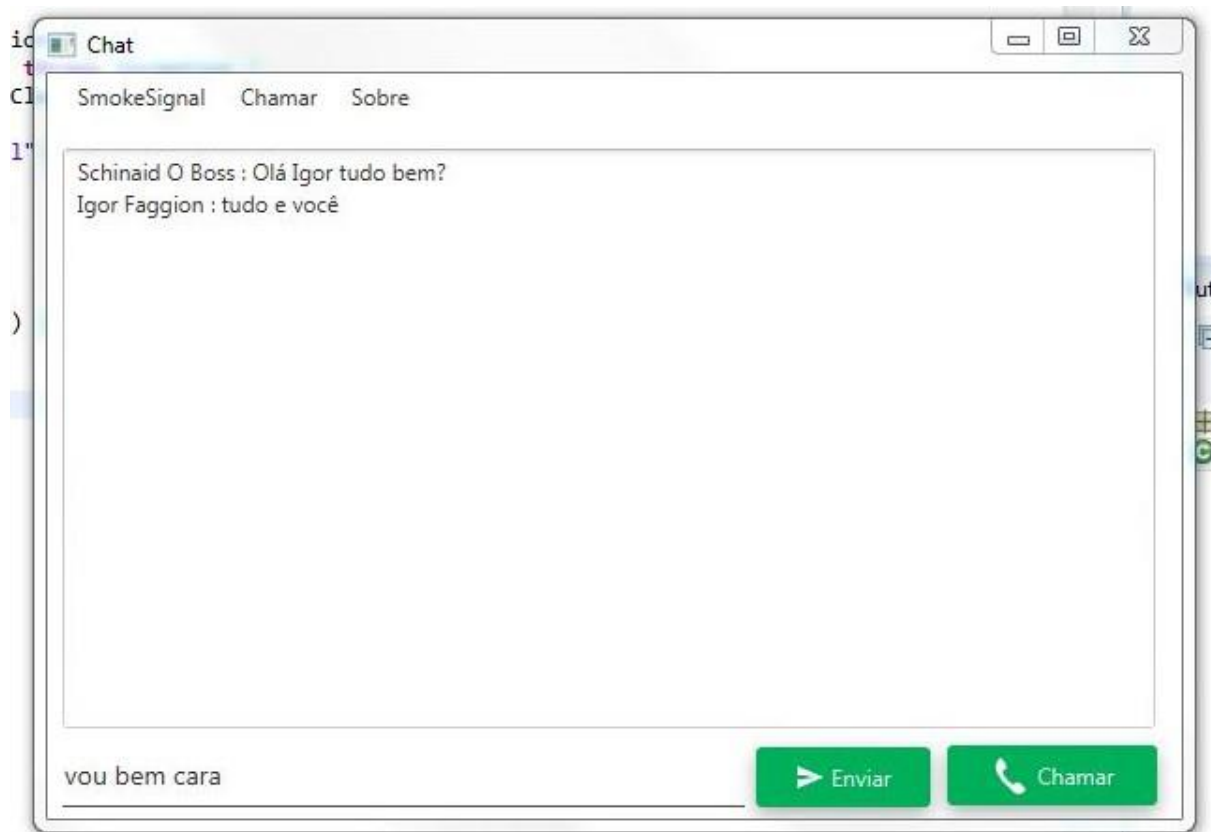
Fonte: Própria, 2017.

Imagem 25 – Cadastro.

A screenshot of a software window titled 'Cadastro'. The window has a standard Windows-style title bar with minimize, maximize, and close buttons. Below the title bar, there are two menu items: 'SmokeSignal' and 'Sobre'. The main content area displays the word 'Cadastro' in a large, green, sans-serif font. Below it, there is a registration form with several fields: an email field containing 'smk@smokesignal.com.br', a name field containing 'Smoke Signal', two password fields represented by dots, a gender dropdown menu currently showing 'Masculino', a role dropdown menu currently showing 'Gerente', and a date field containing '01/02/1979' with a calendar icon. At the bottom of the form, there are two green buttons: 'Cadastrar' and 'Voltar'.

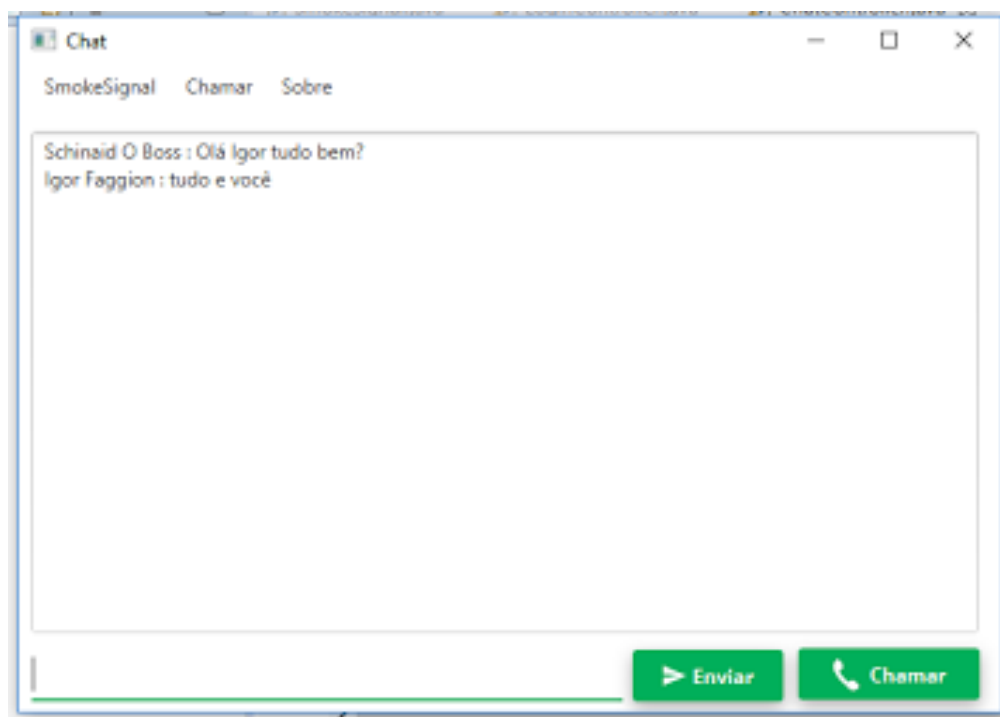
Fonte: Própria, 2017.

Imagem 26 – Chat 01.



Fonte: Própria, 2017.

Imagem 27 – Chat 02.



Fonte: Própria, 2017.

7 CONCLUSÃO

A necessidade por comunicação a longa distância sempre esteve presente desde uma simples carta até uma mensagem transmitida via internet por algum tipo de chat.

Sempre procuramos maneiras mais rápidas de comunicarmos informações importantes, seja por um e-mail entre funcionários de uma empresa ou um chat entre dois funcionários em indústrias diferentes.

Uma das primeiras ferramentas para chat, que foi documentada, foi a internet *relay chat* (IRC) desde então sempre houve melhorias em ferramentas de chat em tempo real, sempre buscando melhorar a velocidade e a segurança do mesmo.

Atualmente chats em tempo real estão presentes mundialmente desde ferramentas em computador até aplicativos em celular. Diferente dos primeiros chats esses são mais rápidos e possuem criptografia, sempre buscando a segurança de seus usuários.

A maioria dos chats utilizam o protocolo de rede TCP/IP que é algo indispensável para qualquer rede tanto que o mesmo é conhecido como o mestre das redes, pois o mesmo é padronizado logo é sempre o preferido para a montagem de redes e o mesmo sendo mais preciso que a maioria dos protocolos.

Em redes também é utilizado o protocolo UDP sendo o protocolo menos confiável pois não possui a precisão e garantia de envio que o protocolo TCP, logo não possui necessidade de conexão longa entre cliente e servidor.

A base para qualquer rede seria a utilização do IP. Ele conserva toda a estrutura da rede permitindo diversos protocolos como TCP e UDP, seria a "identidade" de um cliente ou servidor sendo o que permitiria a conexão. Para um melhor uso dos endereços de equipamentos em rede pelas pessoas, utiliza-se a forma de endereços de domínio, tal como "www.unip.br". Cada endereço de domínio é convertido em um endereço IP pelo DNS (*Domain Name System*). Este processo de conversão é conhecido como "resolução de nomes", logo permitindo comunicação por distâncias.

Um exemplo seria se Secretaria de Estado do Meio Ambiente precisasse saber quais atividades industriais estão gerando poluição no Rio Tietê desde sua nascente em até a sua passagem por São Paulo. Para conseguir tais ela precisa trocar informações das equipes de inspetores dentro de cada indústria, controlando

os processos e passando informações online para a Secretaria. Logo esta ferramenta possibilitaria a troca de informações entre os próprios inspetores quanto para a secretaria sendo em mensagens escritas ou por voz.

O desenvolvimento do presente estudo possibilitou uma análise das redes de computadores, que trouxe consigo a busca do conhecimento para a construção do software de telecomunicação com VoIP, e também o conhecimento da história das redes de computadores até os dias atuais. Perante todas essas dificuldades que os especialistas de redes de computadores tiveram ao criar as redes modernas que hoje em dia conhecemos. Podemos ver claramente até onde a evolução das redes de computadores nos levou, com aplicativos como Skype, MSN e até mesmo o *SmokeSignal* que hoje passamos a conhecer.

De modo geral, o aplicativo de telecomunicação com VoIP obteve bons resultados de comunicação em voz/escrita, que desde o início na criação das tabelas e seus relacionamentos, até a comunicação feita por voz (um dos principais objetivos do presente projeto) que esse projeto conseguiu alcançar o seu objetivo que é a comunicação entre dois computadores semelhantes. Apesar de ter apresentado algumas falhas e dificuldades até obter o resultado desejado do trabalho, o aplicativo *SmokeSignal* conseguiu bater todos os objetivos apresentado no trabalho.

Com a utilização da ferramenta apresentada neste trabalho, a CONAMA poderia se comunicar rapidamente com seus inspetores em campo podendo recolher informações de maneira rápida e precisa logo a ferramenta seria eficiente no uso para a comunicação ponta a ponta onde qualquer problema que fosse necessário ser relatado poderia ser informado da maneira rápida sendo a mesma por áudio ou texto.

O desenvolvimento do aplicativo *SmokeSignal* teve seu surgimento através de ferramentas e ideias que ajudaram bastante para a criação do software em questão. Uma dessas ideias foi o nome do aplicativo *SmokeSignal* que foi retirado dos povos antigos, que utilizavam a fumaça como meio de comunicação a distância, já o desenvolvimento do aplicativo foi através de uma IDE chamada Eclipse, e para guardar os dados dos usuários foi utilizado o Banco de Dados Relacional Oracle 11g.

O desenho das telas gráficas foi feito através de um framework chamado JavaFX. Com esse framework as telas foram desenhadas com mais rapidez e

precisão, isso tudo sem precisar ficar executando a todo momento o programa para visualizar os resultados. Já o motor do aplicativo foi feito em cima de duas classes que utiliza sockets, uma delas é o servidor que fica ouvindo o cliente conectar e solicitar as mensagens, e a outra é o cliente que manda as mensagens para o servidor responder (o mesmo plano se aplica no VoIP).

O Banco de Dados Relacional foi algo mais complexo e não simplesmente a criação de tabelas que se relaciona um com a outra, mas sim, com um planejamento que possuí toda uma estrutura importante para a criação das tabelas, que começou pela normalização separando os atributos que compõe a tabela para que não haja duplicação de dados, e foi indo até a trigonometria. Toda essa sequência foi precisamente esculpida para que não tivesse erro para ser construído no Banco de Dado Oracle 11g.

O maior problema ao desenvolver o código, foi a dificuldade de encontrar exemplos claros e que consumam poucos recursos de memória. Ao desenvolver o programa, percebemos que apesar de sua dificuldade ao encontrar exemplos, sempre há possibilidade de encontrar bibliotecas e a comunidade de programadores que sempre nos dão apoio. Percebemos que para fazer uma análise sobre o programa não poderíamos apenas entender sobre o conceito de VoIP, mas também seríamos obrigados a entender um pouco sobre servidores de áudio, hardware, tipos de redes e sobre programação.

O maior benefício da tecnologia VoIP, se comparada a outras tecnologias de comunicação é pouco custo ao se comunicar. Para fazer a comunicação é utilizado apenas uma rede de computadores, lógica para fazer o algoritmo e hardware que sustente a capacidade de gravar, capturar e reproduzir áudio. O servidor de áudio é usado para distribuir o áudio para todas as pessoas que estão conectadas a mesma rede.

A conexão VoIP é usada na captura de áudio e é necessário a utilização dos drivers de captura de áudio, além de sua placa de som e placa de captura, aonde é conectado o seu microfone em caso de desktop ou seu microfone interno no caso de notebook.

Uma desvantagem do uso do VoIP é a conexão com a sua rede, pois se a mesma não estiver conectada à internet será obrigada a usar a rede local ou no caso de perda de conexões ou pacotes de dados pode deixar a sua conexão não tão eficaz quanto a utilização de um recurso.

Dada a importância do assunto, torna-se necessário o desenvolvimento de formas de melhorar o entendimento das redes de computadores para que os próximos software possam ter uma qualidade cada vez maior, com ferramentas que facilite o desenvolvimento dos próximos aplicativos de telecomunicações.

Neste sentido, como foi demonstrado nesse trabalho, as redes de computadores só têm a evoluir cada vez mais, mostrando os passos percorrido que esse projeto traçou para a criação desse aplicativo, que por sua vez permite a comunicação entre computadores semelhantes, e que um dia se espera que esse projeto ajude outras pessoas a criar seus próprios aplicativos de comunicação.

REFERÊNCIA BIBLIOGRÁFICA

COMER, D. E; STEVENS, D. L. **Interligação de Redes com TCP/IP**. 02. ed. Rio de Janeiro: Campus, 1999. 592 p.

BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML**. 03. ed. São Paulo: Campus, 2015. 320p.

FOROUZAN, B. A. **Comunicação de Dados e Redes de Computadores**. 04. ed. São Paulo: Amgh Editora, 2008. 1168 p.

FOROUZAN, B. A; MOSHARRAF, F. **Redes de Computadores: uma abordagem top-down**. 01. ed. Rio Grande do Sul: Bookman, 2012. 89 p.

OLIFER, N; OLIFER, V. **Redes de Computadores – Princípios, Tecnologias e Protocolos para o Projeto de Redes**. 01. ed. São Paulo: LTC, 2008. 576 p.

OLIVEIRA, B. **JavaFX: interfaces com qualidade para aplicações desktop**. 01. ed. São Paulo: Casa do Código, 2015. 137 p.

SANTOS, R. R. **Programação de computadores em Java**. 02. ed. Rio de Janeiro: Novaterra Editora, 2014. 1437 p.

SCRIMGER, R; LASALLE, P; PARIHAR, M; GUPTA, M. **TCP/IP a Bíblia**. 01. ed. Rio de Janeiro: Campus, 2002. 664 p.

SOARES, L. F. G; LEMOS, G; COLCHER, S. **Redes de Computadores das LANs, MANs e WANs às Redes ATM**. 15. ed. Rio de Janeiro: Elsevier, 1995. 704 p.

SOARES, L. C; FREIRE, V. A. **Redes Convergentes: estratégias para transmissão de voz sobre Frame Relay, ATM e IP**. 01. ed. Rio de Janeiro: Alta Books, 2002. 368 p.

TANENBAUM, A. S; WETHERALL, D. **Redes de Computadores**. 05. ed. São Paulo: Pearson Education, 2011. 582 p.

8 FICHAS DE ATIVIDADES PRÁTICAS SUPERVISIONADAS

Imagem 28 – Gabriel de Almeida Batista.

[illegible]

Fonte: Própria, 2017.

ANEXOS

ANEXO A – Código da tabela E-mail

```
CREATE TABLE EMAIL(  
  "EMA_CODIGO" NUMBER(4,0) NOT NULL ENABLE,  
  "EMA_EMAIL" VARCHAR(30) NOT NULL ENABLE,  
  "EMA_SENHA" VARCHAR(10) NOT NULL ENABLE,  
  PRIMARY KEY("EMA_CODIGO") ENABLE  
);
```

ANEXO B – Código da tabela Usuário

```
CREATE TABLE USUARIO(  
  "USU_CODIGO" NUMBER(4,0) NOT NULL ENABLE,  
  "EMA_CODIGO1" NUMBER(4,0) NOT NULL ENABLE,  
  "USU_NOME" VARCHAR(30) NOT NULL ENABLE,  
  "USU_SEXO" VARCHAR(2) NOT NULL ENABLE,  
  "USU_IDADE" NUMBER(8) NOT NULL ENABLE,  
  "USU_CARGO" VARCHAR(30) NOT NULL ENABLE,  
  PRIMARY KEY("USU_CODIGO") ENABLE,  
  FOREIGN KEY("EMA_CODIGO1") REFERENCES EMAIL("EMA_CODIGO")  
ENABLE  
  );
```

ANEXO C – Código da Sequence AUTO_INCREMENTAL_U

```
CREATE SEQUENCE AUTO_INCREMENTAL_U  
START WITH 1  
INCREMENT BY 1  
MINVALUE 1  
MAXVALUE 1000;
```

ANEXO D – Código da Sequence AUTO_INCREMENTAL_E

```
CREATE SEQUENCE AUTO_INCREMENTAL_E  
START WITH 1  
INCREMENT BY 1  
MINVALUE 1  
MAXVALUE 1000;
```

ANEXO E – Código da Sequence AUTO_INCREMENTAL_EU

```
CREATE SEQUENCE AUTO_INCREMENTAL_EU  
START WITH 1  
INCREMENT BY 1  
MINVALUE 1  
MAXVALUE 1000;
```


ANEXO F – Código da classe CadastroController

```

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;

import javax.swing.JOptionPane;

import com.jfoenix.controls.JFXButton;
import com.jfoenix.controls.JFXComboBox;
import com.jfoenix.controls.JFXDatePicker;
import com.jfoenix.controls.JFXPasswordField;
import com.jfoenix.controls.JFXTextField;

import br.com.unip.cc.dao.EmailDAO;
import br.com.unip.cc.dao.UsuarioDAO;
import br.com.unip.cc.model.Email;
import br.com.unip.cc.model.Usuario;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.MenuItem;
import javafx.stage.Stage;

public class CadastroController implements Initializable {
    ObservableList<String>                                tipoSexo
    FXCollections.observableArrayList("Masculino","Feminino");

```

```

ObservableList<String>                                tipoHierarquia            =
FXCollections.observableArrayList("Gerente","Analista","Programador");

```

```

@FXML private JFXButton btnCadastrar;
@FXML private JFXButton btnVoltar;
@FXML private JFXTextField txtEmail;
@FXML private JFXTextField txtNome;
@FXML private JFXPasswordField pswSenha1;
@FXML private JFXPasswordField pswSenha2;
@SuppressWarnings("rawtypes")
@FXML private JFXComboBox cbSexoCadastro;
@FXML private JFXDatePicker dtNascimentoCadastro;
@SuppressWarnings("rawtypes")
@FXML private JFXComboBox cbHierarquiaCadastro;
@FXML private MenuItem itemSair;
@FXML private MenuItem itemAutores;

```

```

private Stage stage = null;
private Parent root = null;
private Scene scene = null;
private Email email = null;
private Usuario usuario = null;
private EmailDAO obEmail = null;
private UsuarioDAO obUsuario = null;
private int ok = 0;

```

```

public void initialize(URL url, ResourceBundle rb) {
    stage = new Stage();
    usuario = new Usuario();
    email = new Email();
    obUsuario = new UsuarioDAO();
    obEmail = new EmailDAO();

```

```

    btnCadastrar.setOnAction(new EventHandler<ActionEvent>() {

```

```

        public void handle(ActionEvent event) {
            usuario.setUsu_nome(txtNome.getText());

            if(cbSexoCadastro.getValue().equals("Masculino"))
                usuario.setUsu_sexo("M");
            else
                usuario.setUsu_sexo("F");

            usuario.setUsu_idade(2015
Integer.parseInt((dtNascimentoCadastro.getValue().toString().substring(0, 4))));

            usuario.setUsu_cargo(cbHierarquiaCadastro.getValue().toString());
            email.setEmail_email(txtEmail.getText());

            if(pswSenha1.getText().equals(pswSenha2.getText())) {

                email.setEmail_senha(pswSenha1.getText());

                ok =
Integer.parseInt(obEmail.inserirEmail(email));

                ok +=
Integer.parseInt(obUsuario.inserirUsuario(usuario));

                if(ok == 2) {

                    JOptionPane.showMessageDialog(null, "Usuário cadastrado com sucesso",
"Succeso", JOptionPane.PLAIN_MESSAGE);

                    try {
                        root =
FXMLLoader.load(getClass().getResource("/br/com/unip/cc/windows/FXMLLogin.fxml
l"));

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

```

```

    }

    scene = new Scene(root);
    stage.setTitle("Login");
    stage.setScene(scene);
    stage.setResizable(false);
    stage.show();

    btnCadastrar.getScene().getWindow().hide();
    }
    } else
        JOptionPane.showMessageDialog(null, "As
senhas não são iguais.", "Senhas divergentes", JOptionPane.ERROR_MESSAGE);
    }
});

    btnVoltar.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent event) {
            try {
                root =
FXMLLoader.load(getClass().getResource("/br/com/unip/cc/windows/FXMLLogin.fxml
"));
            } catch (IOException e) {
                e.printStackTrace();
            }

            scene = new Scene(root);
            stage.setTitle("Login");
            stage.setScene(scene);
            stage.setResizable(false);
            stage.show();

            btnVoltar.getScene().getWindow().hide();

```

```

        }
    });

    itemSair.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent event) {
            btnVoltar.getScene().getWindow().hide();
        }
    });

    itemAutores.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent event) {
            try {
                root =
FXMLLoader.load(getClass().getResource("/br/com/unip/cc/windows/FXMLInfo.fxml")
);
            } catch (IOException e) {
                e.printStackTrace();
            }

            scene = new Scene(root);
            stage.setTitle("Autores");
            stage.setScene(scene);
            stage.setResizable(false);
            stage.show();
        }
    });
}

@SuppressWarnings("unchecked")
@FXML public void addSexo(){
    cbSexoCadastro.setItems(tipoSexo);
}

@SuppressWarnings("unchecked")

```

```
@FXML public void addHierarquia(){  
    cbHierarquiaCadastro.setItems(tipoHierarquia);  
}  
}
```

ANEXO G – Código da classe ChatController

```
import java.awt.event.ActionListener;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.Socket;
import java.net.URL;
import java.util.ResourceBundle;
import java.util.Scanner;

import com.jfoenix.controls.JFXButton;
import com.jfoenix.controls.JFXTextField;

import br.com.unip.cc.hardware.VoIP;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Menu;
import javafx.scene.control.MenuItem;
import javafx.scene.control.TextArea;
import javafx.stage.Stage;

public class ChatController implements Initializable {
    @FXML private TextArea txtRecebe = null;
    @FXML private Menu itemSmoke = null;
    @FXML private MenuItem itemDesconectar = null;
    @FXML private MenuItem itemSair = null;
    @FXML private MenuItem itemLigar = null;
    @FXML private MenuItem itemAutores = null;
    @FXML private JFXButton btnEnviar = null;
```

```
@FXML private JFXButton btnChamar = null;
@FXML private JFXTextField textEnvia = null;
```

```
private Stage stage = null;
private Parent root = null;
private Scene scene = null;
private PrintWriter escritor = null;
private Scanner leitor = null;
private Socket socket = null;
private String texto = null;
private String nome = null;
```

```
public void initialize(URL url, ResourceBundle rb) {
    stage = new Stage();
    configurarRede();
    nome = new LoginController().getNome();

    btnEnviar.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent event) {
            new FalaServidor().actionPerformed(null);
        }
    });

    btnChamar.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent event) {
            new VoIP().captureAudio();
        }
    });

    itemSair.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent event) {
            System.exit(0);
        }
    });
}
```



```

        itemAutores.setOnAction(new EventHandler<ActionEvent>() {
            public void handle(ActionEvent event) {
                try {
                    root =
FXMLLoader.load(getClass().getResource("/br/com/unip/cc/windows/FXMLInfo.fxml")
);
                } catch (IOException e) {
                    e.printStackTrace();
                }

                scene = new Scene(root);
                stage.setTitle("Autores");
                stage.setScene(scene);
                stage.setResizable(false);
                stage.show();
            }
        });

        itemDesconectar.setOnAction(new EventHandler<ActionEvent>()
{
    public void handle(ActionEvent event) {
        try {
            root =
FXMLLoader.load(getClass().getResource("/br/com/unip/cc/windows/FXMLLogin.fxm
l"));
        } catch (IOException e) {
            e.printStackTrace();
        }

        scene = new Scene(root);
        stage.setTitle("Login");
        stage.setScene(scene);
        stage.setResizable(false);
    }
}

```

```

        stage.show();

        btnEnviar.getScene().getWindow().hide();
    }
});

itemLigar.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent event) {
        new VoIP().captureAudio();
    }
});
}

private void configurarRede() {
    try {
        socket = new Socket("25.2.143.59", 1350);
        escritor = new PrintWriter(socket.getOutputStream());
        leitor = new Scanner(socket.getInputStream());
        new Thread(new EscutaServidor()).start();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private class EscutaServidor implements Runnable {
    public void run() {
        while((texto = leitor.nextLine()) != null)
            txtrRecebe.appendText(texto + "\n");
    }
}

private class FalaServidor implements ActionListener {
    public void actionPerformed(java.awt.event.ActionEvent e) {
        escritor.println(nome + " : " + textEnvia.getText());
    }
}

```

```
        escritor.flush();

        textEnvia.setText("");
        textEnvia.requestFocus();
    }
}
}
```

ANEXO H – Código da classe LoginController

```
import java.io.IOException;
import java.net.URL;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ResourceBundle;

import javax.swing.JOptionPane;

import com.jfoenix.controls.JFXButton;
import com.jfoenix.controls.JFXPasswordField;
import com.jfoenix.controls.JFXTextField;

import br.com.unip.cc.dao.EmailDAO;
import br.com.unip.cc.dao.UsuarioDAO;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;

public class LoginController implements Initializable {
    @FXML private JFXTextField txtEmail = null;
    @FXML private JFXPasswordField pwdSenha = null;
    @FXML private JFXButton btnEntrar = null;
    @FXML private JFXButton btnCadastre = null;
    @FXML private JFXButton btnInfo = null;

    private Stage stage = null;
```

```

private Scene scene = null;
private Parent root = null;
private ResultSet rs = null;
private EmailDAO email = null;
private UsuarioDAO usuario = null;
private String nome = null;

public void initialize(URL location, ResourceBundle resources) {
    btnEntrar.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent event) {
            email = new EmailDAO();
            usuario = new UsuarioDAO();
            rs = email.pesquisarEmail(txtEmail.getText(),
pwdSenha.getText());

            try{
                if(rs.next()) {
                    stage = new Stage();

                    if(rs.next())
                        rs =
usuario.pesquisarUsuario(txtEmail.getText(), pwdSenha.getText());

                    nome = rs.getString(1);
                    root =
FXMLLoader.load(getClass().getResource("/br/com/unip/cc/windows/FXMLChat.fxml
"));

                    scene = new Scene(root);
                    stage.setTitle("Chat");
                    stage.getIcons().add(new
Image("icon.png"));

                    stage.setScene(scene);

```

```

        btnEntrar.getScene().getWindow().hide();

        stage.show();
    }

    else

        JOptionPane.showMessageDialog(null, "E-mail ou senha inválido", "Uuário
        Inválido", JOptionPane.ERROR_MESSAGE);
        } catch (SQLException e1) {
            e1.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
});

    btnCadastre.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent event) {
            try {

                root =
FXMMLoader.load(getClass().getResource("/br/com/unip/cc/windows/FXMLCadastro.
fxml"));

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

                scene = new Scene(root);
                stage.setTitle("Cadastro");
                stage.getIcons().add(new Image("icon.png"));
                stage.setScene(scene);
                stage.show();

                btnCadastre.getScene().getWindow().hide();

```

```

        }
    });

    btnInfo.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent event) {
            try {
                root =
FXMLLoader.load(getClass().getResource("/br/com/unip/cc/windows/FXMLInfo.fxml")
);
            } catch (IOException e) {
                e.printStackTrace();
            }

            scene = new Scene(root);
            stage.setTitle("Autores");
            stage.getIcons().add(new Image("icon.png"));
            stage.setScene(scene);
            stage.setResizable(false);
            stage.show();
        }
    });
}

public String getNome() {
    return nome;
}
}

```

ANEXO I – Código da classe Conexao

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

import javax.swing.JOptionPane;

public class Conexao {
    private static String url = "jdbc:oracle:thin:@127.0.0.1:1521:XE";
    private static String login = "Gabriel";
    private static String senha = "gabriel2b21";
    private static Connection con = null;

    public static Connection getConexao() {
        if(con == null) {
            try{
                Class.forName("oracle.jdbc.driver.OracleDriver");
                con = DriverManager.getConnection(url, login,
senha);
            } catch(ClassNotFoundException e) {
                JOptionPane.showMessageDialog(null, "Classe do
Driver de conexão com Oracle não encontrada!", e.getMessage(),
JOptionPane.ERROR_MESSAGE);
            } catch(SQLException e) {
                JOptionPane.showMessageDialog(null, "Problemas
com os parâmetros de conexão!", e.getMessage(),
JOptionPane.ERROR_MESSAGE);
            }
        }

        return con;
    }
}

```



```
public static void closeConexao() {  
    try {  
        con.close();  
    } catch(SQLException e) {  
        JOptionPane.showMessageDialog(null, "Não foi possível  
fechar a conexão com BD!", e.getMessage(), JOptionPane.ERROR_MESSAGE);  
    }  
}  
}
```

ANEXO J – Código da classe EmailDAO

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import br.com.unip.cc.model.Email;

public class EmailDAO {
    private Connection con = null;
    private PreparedStatement stmt = null;
    private String sql = null;

    public String inserirEmail(Email email) {
        con = Conexao.getConexao();
        sql = "INSERT INTO EMAIL
VALUES(AUTO_INCREMENTAL_E.NEXTVAL, ?, ?)";

        try {
            stmt = con.prepareStatement(sql);
            stmt.setString(1, email.getEmail_email());
            stmt.setString(2, email.getEmail_senha());
            stmt.execute();

            return "1";
        } catch (SQLException e) {
            return e.getMessage();
        }
    }

    public ResultSet pesquisarEmail(String email, String senha) {
        ResultSet rs = null;
        con = Conexao.getConexao();
```

```
sql = "SELECT * FROM EMAIL WHERE EMA_EMAIL = ? AND  
EMA_SENHA = ?";
```

```
    try {  
        stmt = con.prepareStatement(sql);  
        stmt.setString(1, email);  
        stmt.setString(2, senha);  
        rs = stmt.executeQuery();  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
  
    return rs;  
}  
}
```

ANEXO K – Código da classe UsuarioDAO

```

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import br.com.unip.cc.model.Usuario;

public class UsuarioDAO {
    private Connection con = null;
    private PreparedStatement stmt = null;
    private String sql = null;

    public String inserirUsuario(Usuario usuario) {
        con = Conexao.getConexao();
        sql = "INSERT INTO USUARIO
VALUES(AUTO_INCREMENTAL_U.NEXTVAL,
AUTO_INCREMENTAL_EU.NEXTVAL, ?, ?, ?, ?)";

        try {
            stmt = con.prepareStatement(sql);
            stmt.setString(1, usuario.getUsu_nome());
            stmt.setString(2, usuario.getUsu_sexo());
            stmt.setInt(3, usuario.getUsu_idade());
            stmt.setString(4, usuario.getUsu_cargo());
            stmt.execute();

            return "1";
        } catch (SQLException e) {
            return e.getMessage();
        }
    }
}

```

```
public ResultSet pesquisarUsuario(String email, String senha) {  
    ResultSet rs = null;  
    con = Conexao.getConexao();  
    sql = "SELECT  USU_NOME  FROM  USUARIO  WHERE  
USU_CODIGO = (SELECT EMA_CODIGO FROM EMAIL WHERE EMA_EMAIL = ?  
AND EMA_SENHA = ?)";  
  
    try {  
        stmt = con.prepareStatement(sql);  
        stmt.setString(1, email);  
        stmt.setString(2, senha);  
        rs = stmt.executeQuery();  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
  
    return rs;  
}  
}
```

ANEXO L – Código da classe HostingAudio

```
import br.com.unip.cc.hosting.AudioServer;
```

```
public class HostingAudio {  
    public static void main(String[] args) {  
        new AudioServer();  
    }  
}
```

ANEXO M – Código da classe HostingChat

```
import br.com.unip.cc.hosting.ChatServer;

public class HostingChat {
    public static void main(String[] args) {
        new ChatServer();
    }
}
```

ANEXO N – Código da classe SmokeSignal

```

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;

public class SmokeSignal extends Application {
    public void start(Stage primaryStage) throws Exception {
        Parent root =
FXMLLoader.load(getClass().getResource("/br/com/unip/cc/windows/FXMLLogin.fxml
l"));

        Scene scene = new Scene(root);
        primaryStage.setTitle("SmokeSignal");
        primaryStage.getIcons().add(new Image("icon.png"));
        primaryStage.setScene(scene);
        primaryStage.setResizable(false);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}

```


ANEXO O – Código da classe VoIP

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.net.Socket;

import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.Mixer;
import javax.sound.sampled.SourceDataLine;
import javax.sound.sampled.TargetDataLine;

public class VoIP {
    private ByteArrayOutputStream byteArrayOutputStream = null;
    private AudioFormat audioFormat = null;
    private TargetDataLine targetDataLine = null;
    private BufferedOutputStream out = null;
    private BufferedInputStream in = null;
    private Socket sock = null;
    private SourceDataLine sourceDataLine = null;
    private Mixer.Info[] mixerInfo = null;
    private Mixer mixer = null;
    private DataLine.Info dataLineInfo = null;
    private DataLine.Info dataLineInfo1 = null;
    private Thread captureThread = null;
    private Thread playThread = null;
    private boolean stopCapture = false;

    public VoIP() {
        try {
```

```

        sock = new Socket("25.2.100.155", 500);
        out = new
BufferedOutputStream(sock.getOutputStream());
        in = new BufferedInputStream(sock.getInputStream());
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public void captureAudio() {
    mixerInfo = AudioSystem.getMixerInfo();
    mixer = AudioSystem.getMixer(mixerInfo[3]);

    audioFormat = getAudioFormat();
    dataLineInfo = new DataLine.Info(TargetDataLine.class,
audioFormat);

    try {
        targetDataLine = (TargetDataLine)
mixer.getLine(dataLineInfo);
        targetDataLine.open(audioFormat);
        targetDataLine.start();

        captureThread = new CaptureThread();
        captureThread.start();

        dataLineInfo1 = new DataLine.Info(SourceDataLine.class,
audioFormat);
        sourceDataLine = (SourceDataLine)
AudioSystem.getLine(dataLineInfo1);
        sourceDataLine.open(audioFormat);
        sourceDataLine.start();
    } catch (LineUnavailableException e) {

```

```

        e.printStackTrace();
    }

    playThread = new PlayThread();
    playThread.start();
}

private AudioFormat getAudioFormat() {
    float sampleRate = 8000.0F;
    int sampleSizeInBits = 16;
    int channels = 2;
    boolean signed = true;
    boolean bigEndian = false;

    return new AudioFormat(sampleRate, sampleSizeInBits,
channels, signed, bigEndian);
}

private class CaptureThread extends Thread {
    byte tempBuffer[] = new byte[10000];

    public void run() {
        byteArrayOutputStream = new ByteArrayOutputStream();
        stopCapture = false;

        try {
            while (!stopCapture) {
                int cnt = targetDataLine.read(tempBuffer, 0,
tempBuffer.length);

                out.write(tempBuffer);

                if (cnt > 0)

```

```

byteOutputStream.write(tempBuffer, 0, cnt);
    }

    byteOutputStream.close();
} catch (Exception e) {
    System.out.println(e);
}
}
}

private class PlayThread extends Thread {
    byte tempBuffer[] = new byte[10000];

    public void run() {
        try {
            while (in.read(tempBuffer) != -1)
                sourceDataLine.write(tempBuffer, 0, 10000);

            sourceDataLine.drain();
            sourceDataLine.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
}

```

ANEXO P – Código da classe AudioServer

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;

import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.Mixer;
import javax.sound.sampled.SourceDataLine;
import javax.sound.sampled.TargetDataLine;

public class AudioServer {
    private ServerSocket MyService = null;
    private Socket ClientSocket = null;
    private InputStream input = null;
    private TargetDataLine targetDataLine = null;
    private OutputStream out = null;
    private AudioFormat audioFormat = null;
    private Mixer mixer = null;
    private Mixer mixer1 = null;
    private Mixer.Info[] mixerInfo = null;
    private DataLine.Info dataLineInfo = null;
    private SourceDataLine sourceDataLine = null;
    private int size = 0;
    private byte tempBuffer[] = null;

    public AudioServer() {
```

```

        mixerInfo = AudioSystem.getMixerInfo();
        size = 10000;
        tempBuffer = new byte [size];
        try {
            mixer = AudioSystem.getMixer(mixerInfo[1]);
            audioFormat = getAudioFormat();
            dataLineInfo = new DataLine.Info(SourceDataLine.class,
audioFormat);

            sourceDataLine = (SourceDataLine)
AudioSystem.getLine(dataLineInfo);
            sourceDataLine.open(audioFormat);
            sourceDataLine.start();

            MyService = new ServerSocket(500);
            ClientSocket = MyService.accept();

            captureAudio();

            input = new
BufferedInputStream(ClientSocket.getInputStream());
            out = new
BufferedOutputStream(ClientSocket.getOutputStream());

            while (input.read(tempBuffer) != -1)
                sourceDataLine.write(tempBuffer, 0, size);
        } catch (LineUnavailableException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private AudioFormat getAudioFormat() {
        float sampleRate = 8000.0F;

```

```

        int sampleSizeInBits = 16;
        int channels = 2;
        boolean signed = true;
        boolean bigEndian = false;

        return new AudioFormat(sampleRate, sampleSizeInBits,
            channels, signed, bigEndian);
    }

    private void captureAudio() {
        try {
            audioFormat = getAudioFormat();
            new DataLine.Info(TargetDataLine.class, audioFormat);

            for (int cnt = 0; cnt < mixerInfo.length; cnt++) {
                mixer1 = AudioSystem.getMixer(mixerInfo[cnt]);
                if (mixer1.isLineSupported(dataLineInfo))
                    targetDataLine = (TargetDataLine)
mixer.getLine(dataLineInfo);
            }

            targetDataLine.open(audioFormat);
            targetDataLine.start();

            Thread captureThread = new CaptureThread();
            captureThread.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    class CaptureThread extends Thread {
        byte tempBuffer[] = new byte[size];
    }

```

```
public void run() {  
    try {  
        while (true) {  
            targetDataLine.read(tempBuffer, 0,  
tempBuffer.length);  
  
            out.write(tempBuffer);  
            out.flush();  
        }  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}  
}
```


ANEXO Q – Código da classe ChatServer

```
import java.io.IOException;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class ChatServer {
    private List<PrintWriter> escritores = null;
    private ServerSocket server = null;
    private Socket socket = null;
    private PrintWriter p = null;
    private String texto = null;

    public ChatServer(){
        escritores = new ArrayList<>();

        try {
            server = new ServerSocket(5000);
        } catch (IOException e1) {
            e1.printStackTrace();
        }

        while(true) {
            try {
                socket = server.accept();
                new Thread(new EscutaCliente(socket)).start();
                p = new PrintWriter(socket.getOutputStream());
                escritores.add(p);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```

        }
    }
}

private void encaminharParaTodos(String texto){
    for(PrintWriter w : escritores){
        w.println(texto);
        w.flush();
    }
}

private class EscutaCliente implements Runnable {
    Scanner leitor = null;

    public EscutaCliente(Socket socket) {
        try {
            leitor = new Scanner(socket.getInputStream());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void run() {
        try{
            while ((texto = leitor.nextLine()) != null)
                encaminharParaTodos(texto);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
}

```

ANEXO R – Código da classe Email

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "EMAIL")
public class Email {
    private int email_codigo = 0;
    private String email_email = null;
    private String email_senha = null;

    @Id
    @GeneratedValue
    public int getEmail_codigo() {
        return email_codigo;
    }

    public void setEmail_codigo(int email_codigo) {
        this.email_codigo = email_codigo;
    }

    @Column(name = "EMA_EMAIL", length = 30, nullable = false)
    public String getEmail_email() {
        return email_email;
    }

    public void setEmail_email(String email_email) {
        this.email_email = email_email;
    }
}
```

```
@Column(name = "EMA_SENHA", length = 10, nullable = false)
public String getEmail_senha() {
    return email_senha;
}

public void setEmail_senha(String email_senha) {
    this.email_senha = email_senha;
}

@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;

    result = prime * result + email_codigo;

    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;

    if (obj == null)
        return false;

    if (getClass() != obj.getClass())
        return false;

    Email other = (Email) obj;

    if (email_codigo != other.email_codigo)
        return false;
```

```
        return true;
    }
}
```

ANEXO S – Código da classe Usuario

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "USUARIO")
public class Usuario {
    private int usu_codigo = 0;
    private int ema_codigo1 = 0;
    private String usu_nome = null;
    private String usu_sexo = null;
    private int usu_idade = 0;
    private String usu_cargo = null;

    @Id
    @GeneratedValue
    public int getUsu_codigo() {
        return usu_codigo;
    }

    public void setUsu_codigo(int usu_codigo) {
        this.usu_codigo = usu_codigo;
    }

    public int getEma_codigo1() {
        return ema_codigo1;
    }

    public void setEma_codigo1(int ema_codigo1) {
        this.ema_codigo1 = ema_codigo1;
    }
}
```

```
}
```

```
@Column(name = "USU_NOME", length = 30, nullable = false)
```

```
public String getUsu_nome() {
```

```
    return usu_nome;
```

```
}
```

```
public void setUsu_nome(String usu_nome) {
```

```
    this.usu_nome = usu_nome;
```

```
}
```

```
@Column(name = "USU_SEXO", length = 2, nullable = false)
```

```
public String getUsu_sexo() {
```

```
    return usu_sexo;
```

```
}
```

```
public void setUsu_sexo(String usu_sexo) {
```

```
    this.usu_sexo = usu_sexo;
```

```
}
```

```
@Column(name = "USU_IDADE", length = 8, nullable = false)
```

```
public int getUsu_idade() {
```

```
    return usu_idade;
```

```
}
```

```
public void setUsu_idade(int usu_idade) {
```

```
    this.usu_idade = usu_idade;
```

```
}
```

```
@Column(name = "USU_IDADE", length = 30, nullable = false)
```

```
public String getUsu_cargo() {
```

```
    return usu_cargo;
```

```
}
```

```
public void setUsu_cargo(String usu_cargo) {  
    this.usu_cargo = usu_cargo;  
}
```

@Override

```
public int hashCode() {  
    final int prime = 31;  
    int result = 1;  
  
    result = prime * result + usu_codigo;  
  
    return result;  
}
```

@Override

```
public boolean equals(Object obj) {  
    if (this == obj)  
        return true;  
  
    if (obj == null)  
        return false;  
  
    if (getClass() != obj.getClass())  
        return false;  
  
    Usuario other = (Usuario) obj;  
  
    if (usu_codigo != other.usu_codigo)  
        return false;  
  
    return true;  
}  
}
```


ANEXO T – Código da classe EntityManagerProvider

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class EntityManagerProvider {
    private static EntityManagerFactory factory = null;

    static {
        factory = Persistence.createEntityManagerFactory("conexao");
    }

    public static EntityManager getEntityManager() {
        return factory.createEntityManager();
    }

    public static void close() {
        factory.close();
    }
}
```

ANEXO U – Código do arquivo FXMLCadastro

```

<?xml version="1.0" encoding="UTF-8"?>

<?import com.jfoenix.controls.JFXButton?>
<?import com.jfoenix.controls.JFXComboBox?>
<?import com.jfoenix.controls.JFXDatePicker?>
<?import com.jfoenix.controls.JFXPasswordField?>
<?import com.jfoenix.controls.JFXTextField?>
<?import javafx.scene.control.Menu?>
<?import javafx.scene.control.MenuBar?>
<?import javafx.scene.control.MenuItem?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<AnchorPane prefHeight="534.0" prefWidth="532.0" style="-fx-background-
color: #ffffff;" xmlns="http://javafx.com/javafx/8.0.111"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="br.com.unip.cc.control.CadastroController">
    <children>
        <JFXButton id="btnCadastrar" fx:id="btnCadastrar" buttonType="RAISED"
layoutX="113.0" layoutY="459.0" prefHeight="34.0" prefWidth="110.0" style="-fx-
background-color: #00af5a;" text="Cadastrar" textFill="WHITE">
            <font>
                <Font name="Open Sans Bold" size="16.0" />
            </font>
        </JFXButton>
        <JFXTextField id="txtEmail" fx:id="txtEmail" focusColor="#00af5a"
layoutX="113.0" layoutY="135.0" prefHeight="26.0" prefWidth="306.0"
promptText="E-mail">
            <font>

```

```

        <Font name="Open Sans" size="14.0" />
    </font>
</JFXTextField>
    <JFXTextField id="txtNome" fx:id="txtNome" focusColor="#00af5a"
layoutX="113.0" layoutY="176.0" prefHeight="26.0" prefWidth="306.0"
promptText="Nome">
    <font>
        <Font name="Open Sans" size="14.0" />
    </font>
</JFXTextField>
    <JFXPasswordField id="pswSenha1" fx:id="pswSenha1"
focusColor="#00af5a" layoutX="113.0" layoutY="222.0" prefHeight="25.0"
prefWidth="306.0" promptText="Senha">
    <font>
        <Font name="Open Sans" size="14.0" />
    </font>
</JFXPasswordField>
    <JFXPasswordField id="pswSenha2" fx:id="pswSenha2"
focusColor="#00af5a" layoutX="113.0" layoutY="267.0" prefHeight="25.0"
prefWidth="306.0" promptText="Repitir a Senha">
    <font>
        <Font name="Open Sans" size="14.0" />
    </font>
</JFXPasswordField>
    <JFXComboBox id="cbSexoCadastro" fx:id="cbSexoCadastro"
focusColor="#00af5a" layoutX="113.0" layoutY="310.0" onShown="#addSexo"
prefHeight="25.0" prefWidth="210.0" promptText="Sexo" />
    <JFXDatePicker id="dtNascimentoCadastro"
fx:id="dtNascimentoCadastro" defaultColor="#00af5a" layoutX="113.0"
layoutY="394.0" prefHeight="28.0" prefWidth="210.0" promptText="Nascimento" />
    <JFXButton id="btnVoltar" fx:id="btnVoltar" buttonType="RAISED"
layoutX="242.0" layoutY="459.0" prefHeight="34.0" prefWidth="87.0" style="-fx-
background-color: #00af5a;" text="Voltar" textFill="WHITE">
    <font>

```

```

        <Font name="Open Sans Bold" size="16.0" />
    </font>
</JFXButton>
    <Text          fill="#00af5a"          layoutX="203.0"          layoutY="85.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Cadastro">
    <font>
        <Font name="Open Sans" size="30.0" />
    </font>
</Text>
    <MenuBar layoutX="14.0" layoutY="14.0" style="-fx-background-color:
#ffffff;" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0"
AnchorPane.topAnchor="0.0">
    <menus>
    <Menu mnemonicParsing="false" text="SmokeSignal">
    <items>
    <MenuItem id="itemSair" fx:id="itemSair" mnemonicParsing="false"
text="Sair">
        <graphic>
            <ImageView          fitHeight="10.0"          fitWidth="10.0"
pickOnBounds="true" preserveRatio="true">
                <image>
                    <Image url="@res/forbidden-mark.png" />
                </image>
            </ImageView>
        </graphic></MenuItem>
    </items>
</Menu>
    <Menu mnemonicParsing="false" text="Sobre">
    <items>
    <MenuItem          id="itemAutores"          fx:id="itemAutores"
mnemonicParsing="false" text="Autores">
        <graphic>
            <ImageView          fitHeight="16.0"          fitWidth="17.0"
pickOnBounds="true" preserveRatio="true">

```

```

        <image>
            <Image url="@res/book-and-pen.png" />
        </image>
    </ImageView>
</graphic></MenuItem>
</items>
</Menu>
</menus>
</MenuBar>
    <JFXComboBox id="cbHierarquiaCadastro" fx:id="cbHierarquiaCadastro"
focusColor="#00af5a" layoutX="113.0" layoutY="353.0" onShown="#addHierarquia"
prefHeight="25.0" prefWidth="210.0" promptText="Hierarquia" />
</children>
</AnchorPane>

```

ANEXO V – Código do arquivo FXMLChat

```

<?xml version="1.0" encoding="UTF-8"?>

<?import com.jfoenix.controls.JFXButton?>
<?import com.jfoenix.controls.JFXTextField?>
<?import javafx.scene.control.Menu?>
<?import javafx.scene.control.MenuBar?>
<?import javafx.scene.control.MenuItem?>
<?import javafx.scene.control.TextArea?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>

<AnchorPane prefHeight="397.0" prefWidth="605.0" style="-fx-background-
color:          #ffffff;"          xmlns="http://javafx.com/javafx/8.0.111"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="br.com.unip.cc.control.ChatController">
    <children>
        <JFXTextField id="textEnvia" fx:id="textEnvia" focusColor="#00af5a"
layoutX="14.0" layoutY="291.0" prefHeight="30.0" prefWidth="364.0"
promptText="Digite uma mensagem aqui" AnchorPane.bottomAnchor="8.0"
AnchorPane.leftAnchor="8.0" AnchorPane.rightAnchor="233.0">
            <font>
                <Font name="Open Sans" size="14.0" />
            </font>
        </JFXTextField>
        <TextArea id="txtrRecebe" fx:id="txtrRecebe" editable="false"
layoutX="11.0" layoutY="46.0" prefHeight="286.0" prefWidth="557.0"
AnchorPane.bottomAnchor="48.0" AnchorPane.leftAnchor="8.0"
AnchorPane.rightAnchor="14.0" AnchorPane.topAnchor="40.0" />
    </children>

```

```

        <MenuBar layoutX="6.0" layoutY="14.0" style="-fx-background-color:
#ffffff;" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0"
AnchorPane.topAnchor="0.0">
            <menus>
                <Menu id="itemSmoke" fx:id="itemSmoke" mnemonicParsing="false"
text="SmokeSignal">
                    <items>
                        <MenuItem id="itemDesconectar" fx:id="itemDesconectar"
mnemonicParsing="false" text="Desconectar">
                            <graphic>
                                <ImageView fitHeight="10.0" fitWidth="13.0"
pickOnBounds="true" preserveRatio="true">
                                    <image>
                                        <Image url="@../res/car-seat-belt.png" />
                                    </image>
                                </ImageView>
                            </graphic></MenuItem>
                        <MenuItem id="itemSair" fx:id="itemSair" mnemonicParsing="false"
text="Sair">
                            <graphic>
                                <ImageView fitHeight="10.0" fitWidth="10.0"
pickOnBounds="true" preserveRatio="true">
                                    <image>
                                        <Image url="@../res/forbidden-mark.png" />
                                    </image>
                                </ImageView>
                            </graphic></MenuItem>
                    </items>
                </Menu>
                <Menu mnemonicParsing="false" text="Chamar">
                    <items>
                        <MenuItem id="itemLigar" fx:id="itemLigar" mnemonicParsing="false"
text="Ligar">
                            <graphic>

```

```

        <ImageView          fitHeight="12.0"          fitWidth="12.0"
pickOnBounds="true" preserveRatio="true">
        <image>
            <Image url="@../../../../res/Telephone-icon.png" />
        </image>
        </ImageView>
    </graphic></MenuItem>
</items>
</Menu>
<Menu mnemonicParsing="false" text="Sobre">
    <items>
        <MenuItem          id="itemAutores"          fx:id="itemAutores"
mnemonicParsing="false" text="Autores">
            <graphic>
                <ImageView          fitHeight="15.0"          fitWidth="17.0"
pickOnBounds="true" preserveRatio="true">
                    <image>
                        <Image url="@../../../../res/book-and-pen.png" />
                    </image>
                </ImageView>
            </graphic></MenuItem>
        </items>
    </Menu>
</menus>
</MenuBar>
<JFXButton  id="btnEnviar"  fx:id="btnEnviar"  buttonType="RAISED"
layoutX="379.0" layoutY="360.0" prefHeight="30.0" prefWidth="91.0" style="-fx-
background-color: #00af5a;" text="Enviar" textFill="WHITE"
AnchorPane.bottomAnchor="7.0" AnchorPane.rightAnchor="135.0">
    <font>
        <Font name="Open Sans Bold" size="12.0" />
    </font>
    <graphic>

```



```

        <ImageView fitHeight="14.0" fitWidth="16.0" pickOnBounds="true"
preserveRatio="true">
            <image>
                <Image url="@../../../../../../res/send-button.png" />
            </image>
        </ImageView>
    </graphic>
</JFXButton>

    <JFXButton id="btnChamar" fx:id="btnChamar" buttonType="RAISED"
layoutX="455.0" layoutY="336.0" prefHeight="30.0" prefWidth="110.0" style="-fx-
background-color: #00af5a;" text="Chamar" textFill="WHITE"
AnchorPane.bottomAnchor="8.0" AnchorPane.rightAnchor="14.0">
        <graphic>
            <ImageView fitHeight="20.0" fitWidth="20.0" pickOnBounds="true"
preserveRatio="true">
                <image>
                    <Image url="@../../../../../../res/Telephone-icon-bra.png" />
                </image>
            </ImageView>
        </graphic>
        <font>
            <Font name="Open Sans Bold" size="12.0" />
        </font>
    </JFXButton>
</children>
</AnchorPane>

```

ANEXO W – Código do arquivo FXMLInfo

```

<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<AnchorPane prefHeight="305.0" prefWidth="401.0" style="-fx-background-
color:          #ffffff;"          xmlns="http://javafx.com/javafx/8.0.111"
xmlns:fx="http://javafx.com/fxml/1">
    <children>
        <Text          fill="#00af5a"          layoutX="144.0"          layoutY="54.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Autores">
            <font>
                <Font name="Lucida Sans Regular" size="30.0" />
            </font>
        </Text>
        <Text          layoutX="80.0"          layoutY="119.0"          strokeType="OUTSIDE"
strokeWidth="0.0" text="Gabriel de Almeida Batista">
            <font>
                <Font name="Lucida Sans Regular" size="18.0" />
            </font>
        </Text>
        <Text          layoutX="80.0"          layoutY="159.0"          strokeType="OUTSIDE"
strokeWidth="0.0" text="Felipe da Silva Borges Neves">
            <font>
                <Font name="Lucida Sans Regular" size="18.0" />
            </font>
        </Text>
        <Text          layoutX="78.0"          layoutY="199.0"          strokeType="OUTSIDE"
strokeWidth="0.0" text="Igor Faggion Silveira Santos">
            <font>
                <Font name="Lucida Sans Regular" size="18.0" />
            </font>
        </Text>
    </children>
</AnchorPane>

```

```
        </font>
    </Text>
    <Text    layoutX="78.0"    layoutY="236.0"    strokeType="OUTSIDE"
strokeWidth="0.0" text="Anderson Alves Schinaid">
        <font>
            <Font name="Lucida Sans Regular" size="18.0" />
        </font>
    </Text>
</children>
</AnchorPane>
```

ANEXO X – Código do arquivo FXMLLogin

```

<?xml version="1.0" encoding="UTF-8"?>

<?import com.jfoenix.controls.JFXButton?>
<?import com.jfoenix.controls.JFXPasswordField?>
<?import com.jfoenix.controls.JFXTextField?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<AnchorPane prefHeight="372.0" prefWidth="505.0" style="-fx-background-
color: #ffffff;" xmlns="http://javafx.com/javafx/8.0.111"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="br.com.unip.cc.control.LoginController">
    <children>
        <JFXButton id="btnEntrar" fx:id="btnEntrar" buttonType="RAISED"
layoutX="88.0" layoutY="300.0" prefHeight="31.0" prefWidth="81.0" style="-fx-
background-color: #00af5a;" text="Entrar" textFill="WHITE">
            <font>
                <Font name="Open Sans Bold" size="14.0" />
            </font>
        </JFXButton>
        <JFXButton id="btnCadastre" fx:id="btnCadastre" buttonType="RAISED"
layoutX="180.0" layoutY="300.0" prefHeight="31.0" prefWidth="101.0" style="-fx-
background-color: #00af5a;" text="Cadastrar" textFill="WHITE">
            <font>
                <Font name="Open Sans Bold" size="14.0" />
            </font>
        </JFXButton>
    </children>

```

```

        <JFXTextField    id="txtEmail"    fx:id="txtEmail"    focusColor="#00af5a"
layoutX="88.0" layoutY="186.0" prefHeight="28.0" prefWidth="321.0" promptText="E-
mail">
    <font>
        <Font name="Open Sans" size="14.0" />
    </font>
</JFXTextField>
<Text            fill="#00af5a"            layoutX="174.0"            layoutY="68.0"
strokeType="OUTSIDE"            strokeWidth="0.0"            text="SmokeSignal"
wrappingWidth="156.38671875">
    <font>
        <Font name="Open Sans" size="24.0" />
    </font>
</Text>
<JFXPasswordField            id="pwdSenha"            fx:id="pwdSenha"
focusColor="#00af5a"    layoutX="88.0"    layoutY="235.0"    prefHeight="28.0"
prefWidth="321.0" promptText="Senha">
    <font>
        <Font name="Open Sans" size="14.0" />
    </font>
</JFXPasswordField>
<ImageView    id="microfone"    fx:id="microfone"    fitHeight="65.0"
fitWidth="101.0"    layoutX="215.0"    layoutY="91.0"    pickOnBounds="true"
preserveRatio="true">
    <image>
        <Image url="@../../../../res/microphone.png" />
    </image></ImageView>
<ImageView    fitHeight="33.0"    fitWidth="65.0"    layoutX="194.0"
layoutY="100.0" pickOnBounds="true" preserveRatio="true">
    <image>
        <Image url="@../../../../res/nuvem-escura-Nova.png" />
    </image>
</ImageView>

```

```

        <ImageView      fitHeight="33.0"      fitWidth="33.0"      layoutX="424.0"
layoutY="433.0"      opacity="0.5"      pickOnBounds="true"      preserveRatio="true"
AnchorPane.bottomAnchor="8.0" AnchorPane.rightAnchor="8.0">
    <image>
        <Image url="@../../../../res/conama.PNG" />
    </image>
</ImageView>
    <JFXButton      id="btnInfo"      fx:id="btnInfo"      buttonType="RAISED"
layoutX="292.0" layoutY="299.0" prefHeight="31.0" prefWidth="81.0" style="-fx-
background-color: #00af5a;" text="Autores" textFill="WHITE">
    <font>
        <Font name="Open Sans Bold" size="14.0" />
    </font>
</JFXButton>
</children>
</AnchorPane>

```

ANEXO Y – Código do arquivo persistence

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
    xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
        http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="conexao"> <!-- Configura o nome do contexto
que será utilizado -->
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <properties>
            <property                name="javax.persistence.jdbc.url"
value="oracle.jdbc.OracleDriver" /> <!-- nome completo da classe do driver JDBC -->
            <property                name="javax.persistence.jdbc.user"
value="Gabriel" /> <!-- Usuário do banco de dados -->
            <property                name="javax.persistence.jdbc.password"
value="gabriel2b21" /> <!-- Senha do banco de dados -->
            <property                name="javax.persistence.jdbc.driver"
value="jdbc:oracle:thin:@127.0.0.1:1521:XE" /> <!-- Endereço do banco de dados --
>
            <property                name="hibernate.dialect"
value="org.hibernate.dialect.Oracle10gDialect" /> <!-- Configura o dialeto que o
Hibernate utilizará para a montagem dos comandos SQL -->
            <property name="hibernate.show_sql" value="true" /> <!--
Informa se os comandos SQL devem ser exibidos no console -->
            <property name="hibernate.format_sql" value="true" /> <!--
- Indica se os comandos SQL exibidos no console devem ser formatados -->
            <property name="hibernate.hbm2ddl.auto" value="update"
/> <!-- Cria ou atualiza automaticamente a estrutura das tabelas no banco de dados -
->
        </properties>
    </persistence-unit>
</persistence>

```