UNIP – UNIVERSIDADE PAULISTA

Curso de Ciência da Computação

ATIVIDADES PRÁTICAS SUPERVISONADAS - APS

FUNDAMENTOS DA COMUNICAÇÃO DE DADOS EM REDE

FERRAMENTA PARA COMUNICAÇÃO EM REDE (CHAT)

GABRIEL CURSINO MEDEIROS DE ARAÚJO - RA – C7055E0

LEONARDO ARAUJO DAS NEVES - RA – T1877G5

OLÍVIO RODRIGUES DA SILVA NETO - RA – C7554F9

RAFAEL FELIPE MORAES - RA – C4548J0

São José dos Campos, 01 de abril de 2017.

UNIP – UNIVERSIDADE PAULISTA

Curso de Ciência da Computação

ATIVIDADES PRÁTICAS SUPERVISONADAS - APS

FUNDAMENTOS DA COMUNICAÇÃO DE DADOS EM REDE

FERRAMENTA PARA COMUNICAÇÃO EM REDE (CHAT)

Atividades Práticas Supervisionadas do 4º e 5º Semestres do Curso de Ciência da Computação da **Universidade Paulista – UNIP**.

Coordenador: Prof. Fernando A. **Gotti**

São José dos Campos, 01 de abril 2017.

UNIP – UNIVERSIDADE PAULISTA

Curso de Ciência da Computação

FICHA DE APROVAÇÃO

Tema: FUNDAMENTOS DA COMUNICAÇÃO DE DADOS EM REDE

Este Trabalho foi aprovado como avaliação semestral da disciplina Atividades Práticas Supervisionadas - APS

Os alunos receberam as seguintes notas:

Aluno	**Trabalho Impresso**	**Apresentação**	**Média (Total)**

Professor Orientador

_____

André Y. **Kusumoto**

São José dos Campos, ____ de _____de 2017.

RESUMO

O presente trabalho tem como objetivo apresentar uma ferramenta de comunicação que expõe os princípios fundamentais da comunicação de dados em rede. Além disso, esta obra está composta teoricamente dos pilares da computação em redes, como: Objetivo da comunicação de dados, aplicações principais deste tipo comunicação, topologia e terminologia de rede e outros tópicos relacionados ao funcionamento dessa complexa rede. Desta forma, a implementação de uma aplicação baseada nestes fundamentos torna-se um tema indiscutivelmente atual e pertinente aos estudos da Ciência da Computação por abordar não somente os conceitos de redes, mas também os conhecimentos de lógica, programação e mais uma vasta gama de áreas da computação. Não obstante, o trabalho tem o intuito de enriquecer os conhecimentos da comunidade de informática, os próprios atuantes na obra e a Ciência da Computação como um todo.

Palavras-chave: redes, comunicação, dados, TCP, IP, chat.

ABSTRACT

The objective of this work presents a communication tool witch exposes the fundamental principles of network communication. Furthermore, this job is composed by the pillars of network computing, as: Network communication objective, main network applications, topology, termnology and other related topics. In this way, the implementation of an application based on these fundamentals becomes an indisputably current topic and pertinent to Computer Science studies by addressing not only the concepts of networks, but also the knowledge of logic, programming and a wide range of areas Of computing. Further, the work aims enrich the computing community knowledge, the authors of the work and the Computing Science as a whole.

Key words: network, communication, data, TCP, IP, chat.

LISTA DE FIGURAS

SUMÁRIO

1 OBJETIVO E MOTIVAÇÃO DO TRABALHO

O objetivo geral do trabalho é desenvolver uma aplicação que utilize da comunicação de redes para trocar dados.

A fim de atingir o objetivo geral foram definidos alguns objetivos específicos pertinentes a ele.

a) Indicar os conceitos gerais da comunicação em rede;
b) Analisar as principais aplicações que utilizam a comunicação de dados em rede;
c) Analisar os tipos de conexões em redes, diferentes terminologias e diferentes topologias.
d) Reunir a maior quantidade de áreas da Ciência da Computação na composição do trabalho teórico e prático;
e) Definir e diagramar o funcionamento da aplicação prático de acordo com as normas pré-definidas pela comunidade da computação, a exemplo dos diagramas de classe e de uso.

Desta maneira, a motivação do trabalho está baseada nos conhecimentos dos integrantes e autores do presente trabalho e a possibilidade de transmitir estes conhecimentos adquiridos de forma prática e clara. De forma que todo o estudo da obra possa servir como inspiração e fonte para próximos trabalhos realizados.

2 INTRODUÇÃO

Após o advento dos primeiros computadores a necessidade de compartilhar dados entre eles forçou a comunidade de pesquisadores a projetar redes de dados onde fosse possível fazer transmissões em diversas máquinas diferentes.

Foi na década de 60 que os estudos e estruturas de redes começaram a se desenvolver, após a segunda guerra mundial os estudiosos que estavam concentrados na intercepção e decodificação de mensagens que viajavam em redes primitivas continuaram o seu trabalho, mas foram deslocados para outros objetivos de estudos. O meio acadêmico foi um dos que mais atraiu esses pesquisadores.

Até esse ponto as redes de comunicação utilizadas se limitavam a telefônica, esse tipo de rede funcionava com comutação de circuitos em taxa constante entre origem e destino, foi nessa mesma época que a instalação de microcomputadores aumentou consideravelmente, junto a isso surgiu a multiprogramação, devido a isso começou a haver uma necessidade de conexão entre os computadores, de início as redes de comutação por circuito foram utilizadas, mas elas não eram ideias, pois o usuário que necessitava da utilização do computador remoto precisava esperar o processamento do computador e neste tempo a rede estaria ocupada.

Visando desenvolver uma rede por comutação de pacotes estudiosos do MIT criaram a ARPANET que seria a percussora da Internet atual. Após esse acontecimento outras redes se desenvolveram em outras localidades, cada uma com sua estrutura e peculiaridades. Com o intuito de interconectar essas redes foram definidos protocolos de comunicação sendo eles OSI e TCP/IP e UDP.

Esses protocolos perduram até os dias de hoje, mas foram melhorados e adaptados para atender diversos tipos de utilizadores. Diversos tipos de redes foram criadas a partir destes padrões na atualidade é possível encontrar redes locais, regionais e a grande rede mundial de computadores.

Segundo (TANENBAUM, 2011), as redes de computadores têm diversas aplicações e tipos de utilizadores. Para cada fim, a uma estrutura de rede específica. A aplicação comercial por exemplo visa compartilhar recursos e acima disso informações críticas, além disso, visam realizar negócios com empresas parceiras e atualmente fazer contato com os consumidores.

Existe também as aplicações domésticas, que tem como objetivo o acesso a informações remotas, a comunicação entre pessoas, o entretenimento interativo e o comércio eletrônico. No geral, as aplicações domésticas estão centradas no acesso à Internet. Os meios móveis também caracterizam outro tipo de utilização da rede de dados, neste caso uma rede móvel visa realizar todas as atividades de uma rede cabeada, mas com a facilidade e flexibilidade de uma rede sem fio. A utilização desta rede está fortemente ligada a utilização de dispositivos como celulares, notebooks e veículos por exemplo.

Desta forma, é possível perceber que o assunto de redes de computadores é extensamente abordado e pode ser conectado a várias áreas de conhecimento. Além de fazer referência a área de Ciência da Computação, os estudos estão fortemente interligados a Engenharia da computação, por meio da estruturação dos hardwares e softwares que são necessários para a formação de qualquer rede de computador.

Este tema, apesar de ser relativamente novo, apresentou um crescimento muito grande nos últimos anos e atualmente é um assunto de extrema importância. Por isso, há uma grande motivação na contribuição desta obra para toda uma comunidade envolvida na evolução dos meios computacionais.

Fica claro que a delimitação do trabalho está nos estudos básicos da estrutura física e lógica de redes de computadores e também em aspectos históricos importantes, bem como nas implicações atuais que esse conjunto de tecnologias trazem para a sociedade.

Por isso, a obra é de grande pertinência e importância para os estudos de Ciência da computação, pois além de possibilitar o aprendizado teórico é possível demonstrar na prática como uma rede de computadores funciona, no caso, a implementação de um chat, mostra na essência a comunicação de dados por meio de uma rede da forma mais clara e ilustradora possível. Por isso é justificável a escolha do tema, porque se trata de um estudo atual, socialmente importante e que pode ser aplicável e demonstrado na prática.

Logo, se torna claro que é de extrema importância a comunicação de dados em rede e que existem diversos utilizadores e aplicações para uma rede de dados. O presente trabalho visa a utilização de uma rede local com troca de dados através de um chat, e por isso tem-se como questão norteadora da obra: “Como desenvolver uma aplicação de comunicação de dados em rede através do protocolo de comunicação TCP/IP?”.

3 FUNDAMENTOS DA COMUNICAÇÃO DE DADOS EM REDE (CONCEITOS BÁSICOS)

3.1 Aplicações de redes de computadores

No geral, as redes de computadores são utilizadas para compartilhar dados entre computadores, compartilhar recursos ou transmitir informações, nos tópicos do capítulo 3 estão descritas algumas áreas de utilização da comunicação em rede e suas principais utilizações.

3.1.1 Aplicações comerciais

Segundo (TANENBAUM, 2011), o grande objetivo da utilização de redes de computadores no meio comercial é o compartilhamento de recursos, no mais primitivo dos exemplos, pode-se dizer que o compartilhamento de uma impressora entre vários computadores de uma rede de uma empresa é uma aplicação comercial de redes.

Mais importante do que compartilhar recursos é o fluxo de informações críticas que são distribuídas para os utilizadores da rede da empresa, qualquer média ou grande empresa não duraria muito tempo sem a circulação dessas informações pela rede, entre os dados importantes que são propagados para os utilizadores estão os dados estoque, fluxo financeiro, informações de impostos e muitas outras informações online.

É interessante salientar que, apesar de que nos casos de empresas pequenas os computadores estejam quase sempre no mesmo escritório ou área geográfica, nas grandes empresas esses computadores servidores podem estar dispostos em outras nações os estados de uma nação, em outras palavras, as redes de computadores também têm o objetivo de quebrar essa barreira geográfica para diversas empresas.

De forma prática é possível definir como aplicações comerciais das redes de computadores o uso de e-mails, vídeo conferencias, compartilhamento de

documentos e outras interações online como uso de comércio eletrônico e centro de atendimento a fornecedores, por exemplo.

3.1.2 Aplicações domésticas

De acordo com (TANENBAUM, 2011), pode-se definir como algum dos usos mais populares de redes de computadores para usuários domésticos são: Acesso a informações remotas, comunicação entre pessoas, entretenimento interativo, comércio eletrônico.

O acesso a informações remotas, muitas vezes está relacionado a requisição de informações na Internet, o que implica no uso de grandes bancos de dados que guardam dados sobre todo o tipo de coisas.

A comunicação entre pessoas pode acontecer de diversas formas em uma rede de computadores, popularmente o uso de mensagens instantâneas é uma das formas mais amplamente utilizadas pelos usuários domésticos, esse tipo de comunicação deriva do programa talk do unix, que era uma aplicação de mensagens em tempo real. Além disso, a troca de e-mails e a transmissão de conferências também são muito utilizados.

O entretenimento interativo está relacionado principalmente a utilização de mídias online e conteúdo de entretenimento, hoje em dia é muito comum o uso de vídeos por demanda, ou músicas no mesmo modelo de disposição online.

Por fim, existe o comércio eletrônico, que na mais básica das explicações é a venda de produtos pelos meios online, as grandes empresas físicas, pequenas empresas e até mesmo empresas especificamente online anunciam e vendem seus produtos por meio da Internet.

3.1.3 Usuários móveis

Para (TANENBAUM, 2011), esse é um dos segmentos com mais rápido crescimento na indústria da informática. Esse tipo de uso de redes de computadores

se caracteriza pela ausência de ligações físicas na rede e sim pela conexão sem fio a uma rede de computadores, por isso é uma área de extrema importância principalmente em ambientes móveis, como: Carros, aviões, trens etc.

De certa forma, o meio comercial também se aproveita muito desse tipo de utilização da rede de computadores, hoje em dia é comum ver em vários lugares máquinas que vendem produtos sem a necessidade de uma pessoa, essas máquinas muitas vezes processam os pedidos por meio de uma rede sem fio que está conectada a máquina. Além disso, existe também o m-commerce, um tipo de comércio online que está cada vez mais difundido e utilizado no mercado, esse tipo de comércio é feito exclusivamente por celulares e smartphones, desde o pedido ao processamento do pagamento tudo é criado exclusivamente para atender os utilizadores móveis.

Em resumo, os utilizadores móveis buscam estar sempre conectados mesmo longe de casa, no trânsito, em lugares onde não é possível ter uma infraestrutura cabeada. Existe uma grande necessidade de se fazer tudo ou quase tudo que em uma rede normal, como: trocar mensagens, fazer vídeo conferências, acessar sites etc.

3.2 Terminologia

De acordo com (TEIXEIRA; HORTA, 2003), a terminologia de redes se refere aos termos mais técnicos nas ciências de redes de computadores. A ciência de redes de computadores é muito abrangente quanto aos seus termos, por isso estão apresentados apenas termos de grande popularidade:

ASCII (American Standart Code for Information Interchange): Trata-se de um esquema que atribui valores numéricos a letras, números, sinais de pontuação e outros símbolos especiais para ser usado em computadores e outros meios eletrônicos.

Bridge: Um dispositivo que conecta duas ou mais redes de computadores transferindo, seletivamente, dados entre ambas.

Domínio: É uma parte da hierarquia de nomes na Internet que permite identificar uma instituição ou parte dela na rede. Ex.: www.google.com.br.

DNS: O Domain Name System é um serviço e protocolo da família TCP/IP para o armazenamento e consulta a informações sobre recurso de rede. A implementação

é distribuída entre diferentes servidores e trata principalmente na conversão de nomes em seus IP’s correspondentes.

Ethernet: Um padrão muito usado para conexão física de redes locais, originalmente surgido pelo Palo Alto Research Center (PARC) da Xerox nos EUA. Descreve protocolo, cabeamento, topologia e mecanismos de transmissão.

Firewall: Um sistema de segurança de rede, cujo principal objetivo é filtrar o acesso a uma rede.

FTP: File Transfer Protocol, Protocolo padrão da Internet, usado para transferência de arquivos entre computadores.

Host: Computador ligado a Internet.

HTTP: O protocolo HTTP (HyperText Transfer Protocol) permite que os autores de hipertextos incluam comandos que possibilitam saltos para recursos e outros documentos disponíveis em sistemas remotos, de forma transparente para o usuário.

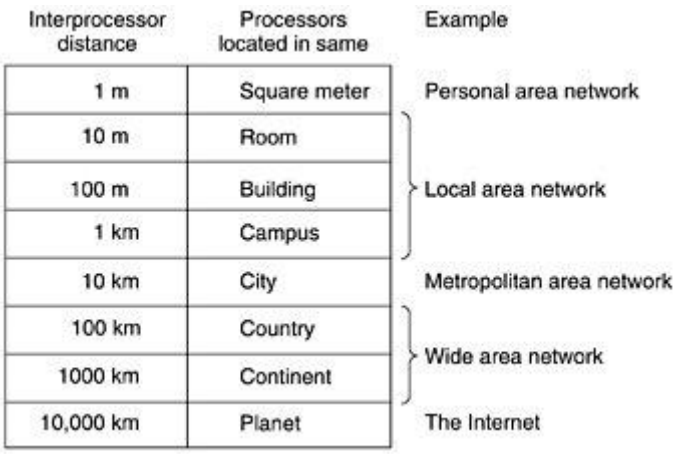
IP: O Internet Protocol, é o protocolo responsável pelo roteamento de pacotes entre dois sistemas que utilizam a família de protocolos TCP/IP, desenvolvida e usada na Internet. É considerado o mais importante dos protocolos em que a Internet é baseada.

Pacotes: Dado encapsulado para transmissão na rede. Um conjunto de bits compreendendo informações de controle, endereço fonte e destino dos nós envolvidos na transmissão.

WWW: World Wide Web, ou Web. Meta-rede, baseada em hipertextos, que intergra diversos serviços Internet, através de uma Interface que possibilita o acesso a informações multimídia.

3.3 Taxonomia de redes

Para (TANENBAUM, 2011), as redes de computadores podem ser classificadas pela sua escala organizadas pelo seu tamanho físico. Existem redes pessoais ou locais, metropolitanas e geograficamente distribuídas. Conforme mostrado na figura 1:



Interprocessor distance	Processors located in same	Example
1 m	Square meter	Personal area network
10 m	Room	Local area network
100 m	Building	
1 km	Campus	
10 km	City	Metropolitan area network
100 km	Country	Wide area network
1000 km	Continent	
10,000 km	Planet	The Internet

Figura 1 - Taxonomia de redes por escala

Fonte: TANENBAUM, 2011.

As redes locais (LAN) podem ser classificadas como redes que podem ter até alguns quilômetros, podem ser usadas em residências, edifícios ou um campus por exemplo.

As redes metropolitanas (MAN), são aquelas que abrangem uma cidade. Enquanto, as redes geograficamente distribuídas (WAN), ocupam uma grande área geográfica, com frequência um pais ou um continente.

Existem ainda as redes pessoais (PAN), que tem como característica o uso da tecnologia wireless. Redes privadas globais (GAN), muito utilizadas por multinacionais para manter uma conexão privada em grandes áreas geográficas e, finalmente, as redes de armazenamento de dados (SAN), que tem utilização em servidores de backup e servidores datacenter.

3.4 Topologia de redes

Segundo (MARTINEZ, 2010), a topologia de redes visa demonstrar como os dispositivos estão conectados em uma rede, tanto do ponto de vista físico quanto do lógico.

As topologias físicas descrevem os nós e as estações de uma rede de computadores, como as descritas a seguir:

BARRAMENTO: Todos os nós estão conectados por uma barra única, é utilizado o cabo coaxial como meio de transmissão. Conforme a figura 2:

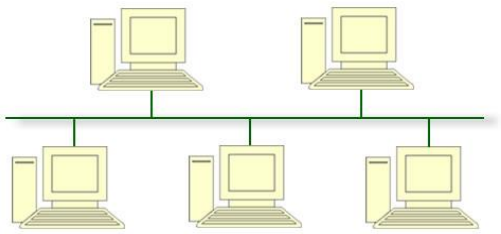


Figura 2 - Topologia barramento

Fonte: MARTINEZ, 2010.

ANEL ou RING: Funciona por várias conexões ponto a ponto que operam em apenas um sentido. Há pouca tolerância a falhas nessa topologia. Como na figura 3:

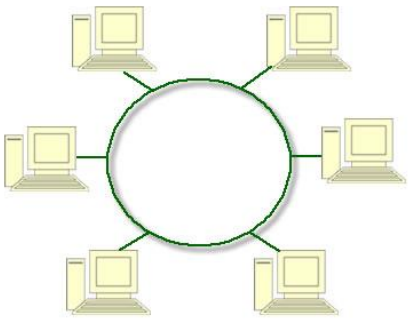


Figura 3 - Topologia anel

Fonte: MARTINEZ, 2010.

ESTRELA: Existe um nó central gerenciador (Switch ou comutador), todas as máquinas conectadas ao nó central simulam um ponto a ponto. Como na figura 4:

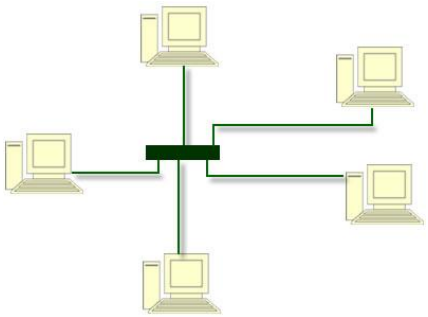


Figura 4 - Topologia estrela

Fonte: MARTINEZ, 2010.

Existem ainda outros tipos de topologias. A de árvore, por exemplo, funciona com várias barras conectas como se fossem várias redes estrelas conectadas pelos seus nós centrais. A mista, é um conjunto das topologias como barramento, estrela e anel. E ainda a grafo, que é uma mistura das topologias que funciona com várias rotas de transmissão em caso de falhas.

A topologia lógica está relacionado ao modelo de transmissão em rede, geralmente existem o broadcast, que é o envio a todos na rede, e o token ring que é o uso de token para controlar os envios na rede.

3.5 Hardware de redes

De acordo com (TANENBAUM, 2011), pode-se dividir as tecnologias de transmissão: Por difusão e ponto a ponto. A primeira corresponde a disseminar a informação para todas as máquinas, mas só a correspondente aceitará a mensagem, já a segunda, por meio do roteamento envia a mensagem diretamente ao destino. A transmissão por difusão geralmente é aplicável em redes menores e a ponto a ponto em redes de grande escala.

As redes LAN, por exemplo, admitem a topologia de redes anel e barramento conforme o tópico 3.4. Redes MAN e WAN são misturas das topologias, podem ser por grafos ou mista.

Conforme informado no tópico 3.4 a topologia anel funciona com o cabo coaxial ou par-trançado, a topologia estrela utiliza de um hub ou switch que dissemina os dados.

3.6 Software de redes

Conforme descrito por (TANENBAUM, 2011), inicialmente os projetos de redes de computadores se preocupavam mais com o hardware do que o software, mas isso foi deixado para trás. Logo após isso, seguindo a maioria das arquiteturas da ciência da computação, adotou-se um modelo de pilha de protocolos, onde um protocolo só

se comunicava com seus vizinhos e abstraia os dados necessários, informando apenas o importante à camada seguinte.

Na maioria das camadas existem algumas preocupações que devem ser levadas em considerações pelas camadas vizinhas, como: Sentido da transferência, quem são os transmissores e receptores, se existem erros na mensagem e a sequência das mensagens. No caso da última, ainda existe a preocupação se um canal de dados é muito rápido e o receptor é lento, para isso é preciso aplicar técnicas de controle de fluxo.

Além disso, existem os serviços orientados a conexão que tem um fluxo sequencial de dados e por isso são muito aplicáveis a streaming de voz, vídeo etc. Já o não orientado a conexão transmite pacotes individuais que podem chegar ordenados ao destino, mas que, em alguns casos, podem chegar fora de ordem por retardo e devem ser ordenados quando chegarem.

Em resumo, a arquitetura de redes trabalha em camadas que são um ou vários protocolos reunidos. Eles estão conectados por interfaces que também podem ser chamados de serviços.

3.7 Modelo OSI

(TORRES, 2007) explica que inicialmente, no surgimento das redes de computadores, a maioria delas era de tecnologia proprietária e por isso era inviável conectar duas redes de tecnologias diferentes. Dessa forma, A ISO (Internation Standarts Organization) desenvolveu um modelo de referência chamado OSI (Open Systems Interconection), por meio deles as empresas deveriam criar os protocolos de maneira organizada e padronizada.

Esse modelo é simplificado em 7 camadas, (TANENBAUM, 2011), explica que para o surgimento dessas camadas foram aplicados alguns princípios:

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 es colhidos 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 pequeno o suficiente para que a arquitetura não se torne difícil de controlar.

Por meio desses princípios foram criadas as camadas física, de enlace de dados, rede, transporte, sessão, apresentação e aplicação. Essas camadas devem se comunicar apenas com as camadas vizinhas, exemplo: a camada 2 só se comunica com a 1 e a 3. Por isso, suas funções devem ser bem estabelecidas.

Na camada física são transmitidos os bits brutos pelo canal de comunicação. Além disso, a voltagem a ser usada para representar os bits, a forma inicial de estabelecimento da conexão, a quantidade de tempo que um bit pode durar e o sentido da transmissão são algumas das funções dessa camada.

A camada de enlace deve organizar os bits em quadros, tentando evitar erros no envio, assim como regular o fluxo de dados enviados para que não haja sobrecarga.

A camada de rede tem o objetivo de rotear dados até o destino correto, evitando gargalos e congestionamento de dados.

A camada de transporte deve conectar origem e destino da transmissão, deve dividir as informações em divisões menores e checar se estas chegaram corretamente.

A camada de sessão cria uma sessão entre os usuários, controlando quem deve transmitir em cada momento, quais operações estão disponíveis, e sincronizando as transmissões.

A camada de apresentação se preocupa com a semântica dos dados enviados, verificando se eles foram transmitidos na representação correta.

Por fim, a camada de aplicação possuí os protocolos comumente utilizados pelos usuários como o HTTP, base da web, SMTP, de e-mails, entre outros.

3.8 TCP/IP

Segundo (OLIVER, 1999), o TCP/IP foi desenvolvido em 1969 nos EUA com a necessidade de comunicação entre sistemas daquela época. Hoje, o TCP/IP é um conjunto de protocolos de extrema importância que realiza a comunicação entre máquinas em uma rede. TCP – Transmission Control Protocol e IP – Internet Protocol.

Os protocolos podem ser divididos em 5, descrito nos itens do tópico 3.8:

3.8.1 Camada física

De acordo com o modelo OSI, a camada física define as características mecânicas, elétricas, funcionais e os procedimentos para manusear conexões para a transmissão dos bits. As características mecânicas se referem ao tamanho e forma dos conectores, as elétricas se referem aos níveis de tensão e corrente e as funcionais definem se os sinais foram enviados ou recebidos.

3.8.2 Camada de enlace de dados

De acordo com o modelo OSI, a camada de enlace é responsável pela correção de erros que possam acontecer na camada Física. Ela também estabelece um protocolo de comunicação entre sistemas diretamente conectados.

3.8.3 Camada de Internet

A camada de Rede ou Internet é a responsável por adicionar o cabeçalho no pacote de dado recebido da camada de Transporte onde, onde além de outros dados de controle, será adicionado o endereço IP fonte e o endereço IP de destino, ou

melhor, o endereço IP do computador que estão enviando o dado e o endereço IP do computador que vai receber o dado. Existem diversos protocolos na camada de internet e podemos citar os seguintes: ARP, IP, RARP, ICMP, IGMP, etc.

3.8.4 Camada de transporte

A camada de transporte, tanto no modelo TCP/IP ou no modelo OSI é responsável pela transferência dos dados com eficiência entre a máquina de origem e a máquina destino. Ela garante que os dados cheguem sem erros e na sequência correta em que foram enviados. Essa camada reúne protocolos que realizam as funções de transporte fim a fim, considerando a origem do dado e seu destino, ignorando elementos intermediários. A camada de transporte possui 2 protocolos, sendo eles:

UDP: realiza apenas a multiplexação para que várias aplicações possam acessar o sistema de comunicação de forma coerente.

TCP: realiza, além da multiplexação, uma série de funções para tornar a comunicação entre origem e destino mais confiável. São responsabilidades do protocolo TCP: o controle de fluxo, o controle de erro, a sequência a multiplexação de mensagens.

3.8.5 Camada de aplicação

A camada de aplicação é responsável por prover serviços e aplicações de modo a separar a existência em rede entre processos de diferentes computadores.

No modelo OSI a camada de aplicação tem como função fazer a comunicação entre a rede e os aplicativos na máquina. A camada de aplicação possui alguns protocolos como: TELNET, FTP, SMTP, DNS, HTTP, RTP, etc.

4 PLANO DE DESENVOLVIMENTO

Neste capítulo estão descritos os elementos e ferramentas utilizados na composição da parte prática de uma aplicação de comunicação em rede. Nesse caso, o aplicativo desenvolvido se refere a um chat que foi intitulado “JChat”, ele é uma ferramenta de comunicação por mensagens instantâneas que funcionam com sockets.

As tecnologias empregadas, assim como as aplicações que auxiliaram na diagramação do projeto e no desenvolvimento gráfico e de programação do projeto estão descritos nos tópicos a seguir:

4.1 Arquitetura de software

Para garantir maior padronização e organização no desenvolvimento da aplicação e permitir que todo o grupo pudesse desenvolver e entender tudo o que já havia sido desenvolvido por outra pessoa foi utilizado um padrão de arquitetura de software. Nesse caso, o padrão utilizado foi o MVC que funciona separando o código em três camadas principais o Model, o View e o Controller. Resumidamente, o model representa os dados da aplicação e o meio de obtê-los, o View reúne todo o esquema visual da aplicação neste contexto o JavaFX foi a principal ferramenta utilizada na interface visual, já o Controller manipula todas as requisições da View ao Model e devolve os dados obtidos a interface gráfica.

4.2 Wireframe

Antes mesmo de desenvolver o programa em si, ocorreu a etapa de planejamento e estruturação da aplicação. Nesse sentido, uma das técnicas importantes utilizadas foi a criação dos wireframes, estes são como esboços bem

simples, mas que agregam todas as funcionalidades que devem ter no aplicativo. Desta forma, tudo o que haveria de ser criado no sistema de chat já estava pensado e por isso houve uma economia de tempo e replanejamento, pois não foi preciso adicionar novas funcionalidades com o desenvolvimento em andamento, por exemplo.

Um exemplo de wireframe, da tela principal no caso, está presente na figura 5:

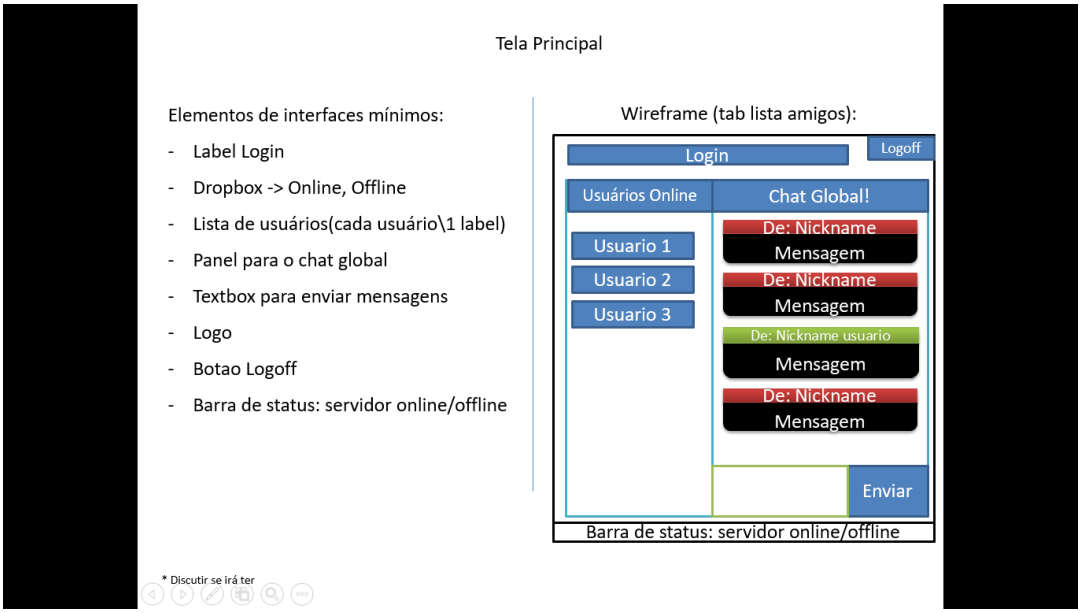


Figura 5 - Wireframe

Fonte: Autoria própria.

4.3 Diagramação

Outra ferramenta utilizada na etapa de planejamento do projeto foi os diagramas de classe. Este tipo de diagrama auxilia na exposição, entendimento e manipulação dos dados da aplicação. No caso do trabalho em questão ele foi essencial para estabelecer as relações necessárias entre o Model, View e Controller e definir o que cada um desses mebros do programa conteriam.

A parte do diagrama que se refere ao login pode ser observada na figura 6:

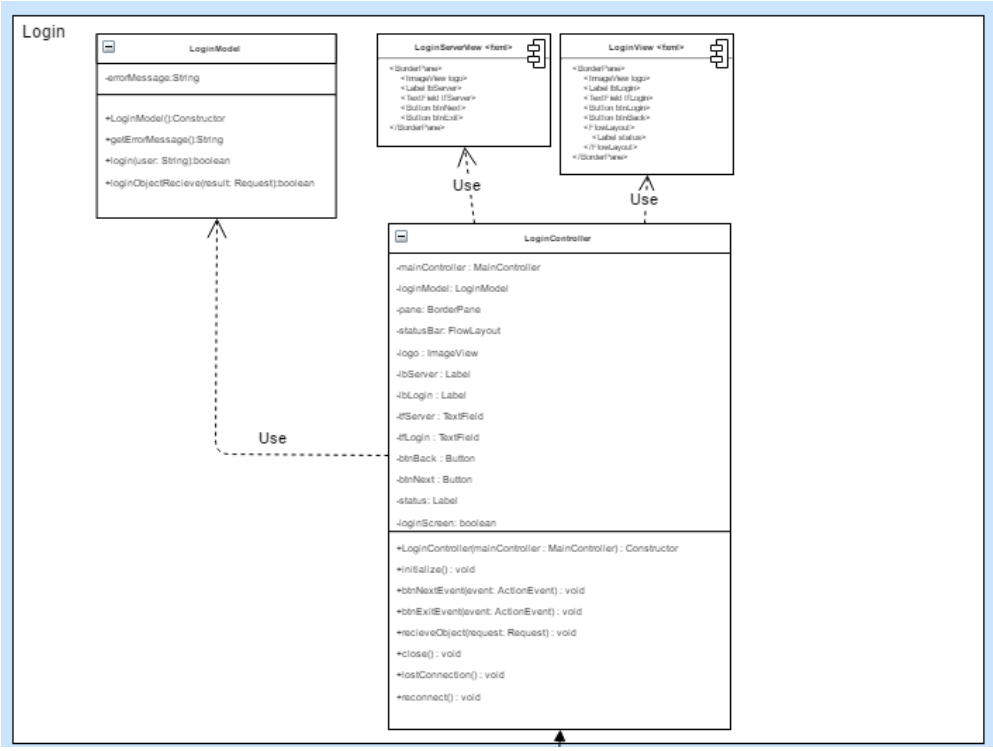


Figura 6 - Diagrama de classe

Fonte: Autoria própria.

4.4 Tecnologias da linguagem

As principais tecnologias usadas no desenvolvimento do chat na parte do servidor foram relacionadas ao SocketServer, que basicamente é um servidor que escuta requisições socket e as intercepta. Além disso, para serialização dos dados que teriam que ser enviados ao servidor foi utilizado o ObjectInputStream.

Para manter essa conexão entre servidor e cliente aberta foi utilizado o conceito de Threads.

No lado do cliente, é interessante salientar a utilização do JavaFx e a estruturação de interfaces por FXML, que é um esquema em que cada elemento da interface é representado por uma tag, que pode ser facilmente estilizado com CSS, assim como o HTML.

4.5 Ambiente de desenvolvimento

Como a linguagem de programação adota foi o JAVA optou-se por usar o eclipse como IDE de desenvolvimento. O Java utilizado corresponde a versão 8, JRE 1.8.0_121 e a API de desenvolvimento de interface foi o JavaFX 2.1.0, para auxiliar na criação da interface foi utilizado o Gluon Scene Builder 8.3.0 que é uma ferramenta de estruturação de interfaces por meio de drag and drop de elementos, o Scene Builder utilizado será melhor detalhado no tópico 4.7.

4.6 Interface do usuário

Como mencionado no tópico 4.4 e 4.5 o projeto foi desenvolvido com base na linguagem de programação JAVA, essa linguagem oferece diversos conjuntos de ferramentas e API’s para o desenvolvimento de uma interface gráfica.

Os conjuntos de ferramentas mais populares para o desenvolvimento da parte gráfica do programa são: AWT, Swing e JavaFX. No caso do projeto em questão a tecnologia implementada dentre estas foi o JavaFX. A justificativa para a utilização desta ferramenta foi a facilidade de criar interfaces complexas rapidamente em relação às outras. Além disso, o JavaFX é ferramenta que oferece os recursos mais atuais para o desenvolvimento e também tende a ser a que entrega o resultado mais bonito e agradável.

Uma comparação entre três telas semelhantes criadas a partir do AWT, Swing e JavaFX respectivamente pode ser vista na figura 7:

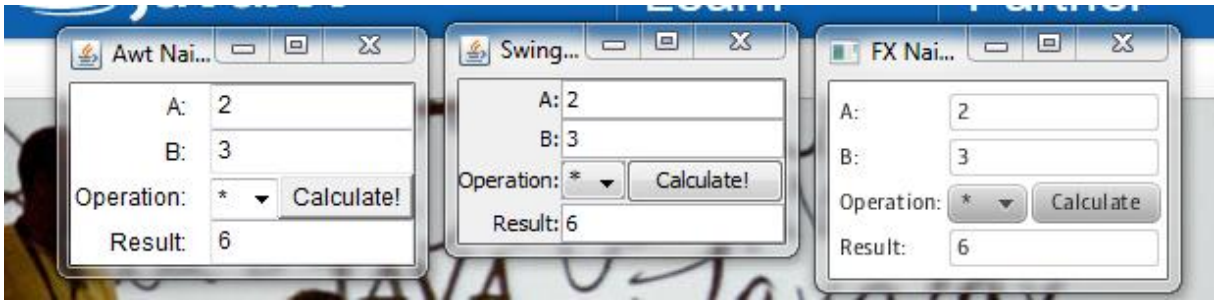


Figura 7 - Diferença entre AWT, Swing e JavaFX

Fonte: Etf_DevLab[1].

4.7 Criação de interfaces gráficas

A partir da escolha do JavaFX como API de desenvolvimento para interfaces gráficas, como exposto no tópico 4.6, foi padrozinado que as telas do programa seriam feitas a partir do fxml e css. O fxml é um método de se escrever estruturas da interface adotado pelo JavaFX que funciona como o xml, cada tag representa um elemento da interface e pode-se colocar tags dentro de tags gerando um esquema hierárquico e organizado. Já o css é uma linguagem de estilos muito utilizada nos programas Web, mas que também se aplica ao JavaFx.

Com o objetivo de agilizar a criação de interfaces gráficas, o grupo optou pela utilização de uma ferramenta de criação de interfaces baseadas no fxml do JavaFX. A ferramenta que é chamada de Gluon SceneBuilder foi criada para gerar interfaces por meio do drag and drop, ou seja, no menu de elemenos é possível escolher uma gama de componentes gráficos do JavaFX e arrastar direto para a janela a ser criada, dessa forma, há uma grande abstração de qualquer tipo de código que poderia levar um tempo desnessário para ser programado.

A janela de desenvolvimento do SceneBuilder e uma interface criada a partir dele podem ser vistas na figura 8:

[1] Disponível em: http://etfdevlab.blogspot.com.br/2011/06/javafx-vs-java-swing-vs-awt.html

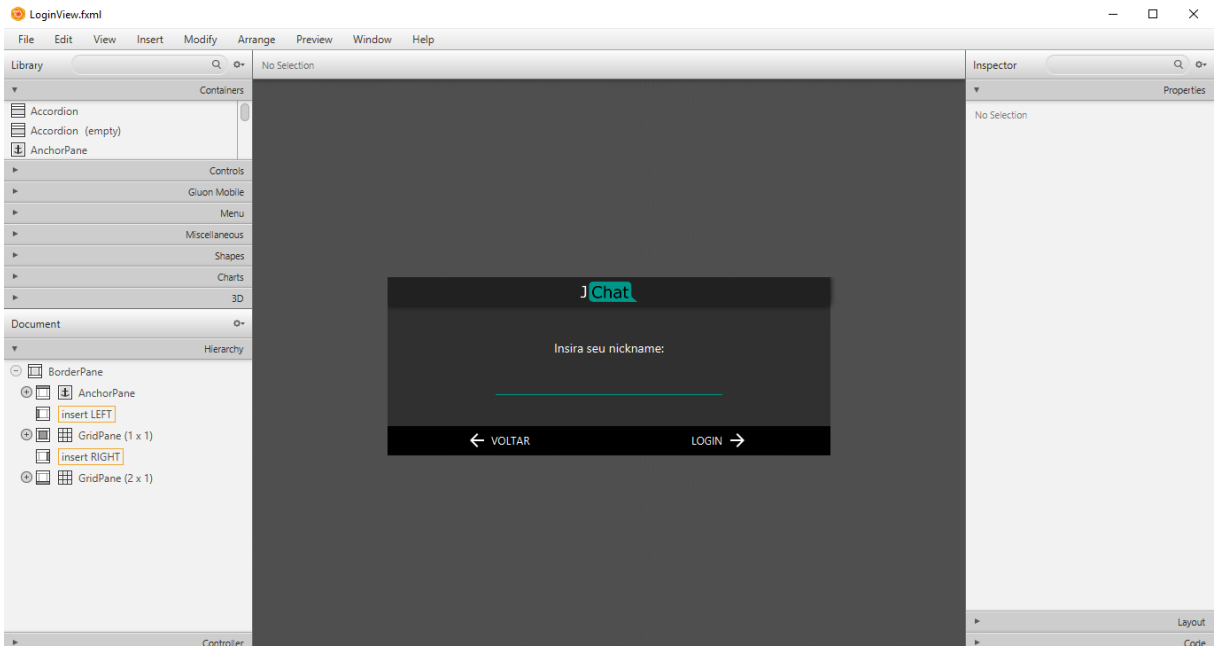


Figura 8 - Gluon SceneBuilder

Fonte: Autoria própria.

4.8 Compartilhador de arquivos e controlador de versões

Como o desenvolvimento do projeto foi separado em blocos onde cada um dos integrantes participava de diferentes etapas da aplicação foi importante encontrar uma ferramenta que pudesse auxiliar no controle de compartilhamento dos arquivos do projeto, bem como verificar as mudanças nos arquivos e auxiliar na resolução de conflitos (partes de arquivos contenham mudanças distintas em computadores diferentes).

Para sanar o problema descrito os integrantes do grupo optaram por utilizar o sistema GIT em conjunto com o Github. Dessa forma, toda vez que algum arquivo fosse alterado era possível verificar se haviam sido feitas alterações no mesmo arquivo por outros participantes e facilmente gerenciar essas alterações.

5 PROJETO

Para demonstrar de forma prática um exemplo de comunicação de dados em redes foi produzido um chat de mensagens instantâneas intitulado “JChat”, assim como descrito nos capítulos anteriores.

Este chat foi projetado nos módulos de cliente e servidor, onde os dois mantém uma conexão através da tecnologia de sockets.

A estrutura e os módulos deste chat estão descritos nos tópicos deste capítulo:

5.1 Cliente

O lado do cliente é composto por três áreas principais de interação, a tela de configuração, o chat global e a janela de chat individual privado. Todas estas telas foram criadas de forma a parecer simples ao usuário, mas com funções bem definidas e devidamente organizadas na interface. Os detalhes de cada um destes módulos estão a seguir:

5.1.1 Tela de configuração

Após ter rodado o servidor de sockets em qualquer IP disponível a tela de configuração é utilizada pelo usuário para a escolha deste servidor por meio do IP, após ter escolhido o servidor de sockets ele é testado, caso não ocorra erros o utilizador é encaminhado a tela de login, senão é lançada uma mensagem de erro contendo os detalhes do ocorrido. Além disso, é possível encontrar na tela um botão para sair do aplicativo, conforme a figura 9:

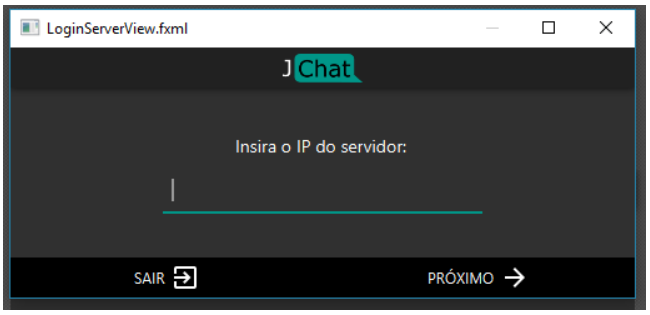


Figura 9 - Tela de configurações

Fonte: Autoria própria.

5.1.2 Tela de login

A tela de login é destinada a inserção de um nickname único disponível para que o usuário possa ser indentificado por todos os outros utilizadores que estejam online no momento. Outras funções desta tela são: Voltar a tela de configuração do servidor e testar o nickname e aceder a interface principal, caso haja um erro uma mensagem será mostrada ao usuário e o acesso ao chat será impedido. A estrutura desta tela está disponível na figura 10:

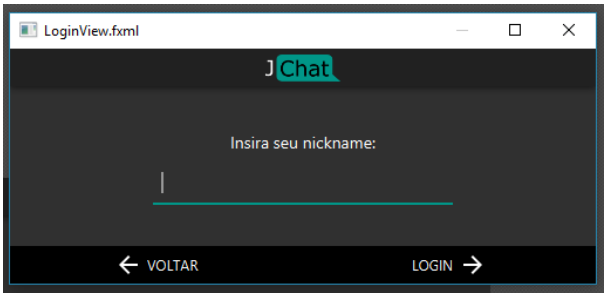


Figura 10 - Tela de login

Fonte: Autoria própria.

5.1.3 Chat global

Esta tela corresponde a principal do sistema, por meio dela é possível conversar com todos os usuários por meio de um chat global. O usuário também tem a opção de fazer logoff. O chat global pode ser visto na figura 11:

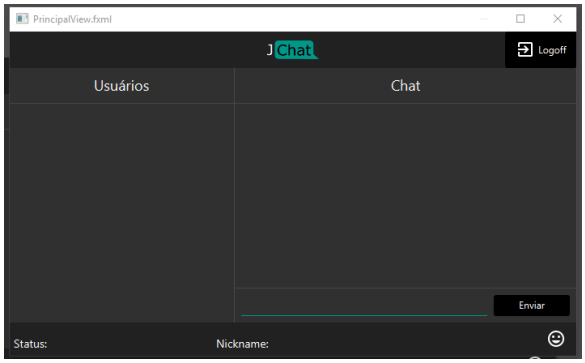


Figura 11 - Chat global

Fonte: Autoria própria.

5.1.4 Usuários disponíveis

No menu lateral presente na tela principal está presente a lista com todos os usuários disponíveis no momento, eles são apresentados por meio do nickname que eles escolheram na entrada da aplicação, a partir do clique em qualquer um dos itens da lista é aberta uma janela de chat privado com o utilizador escolhido. Como apresentado na figura 12:

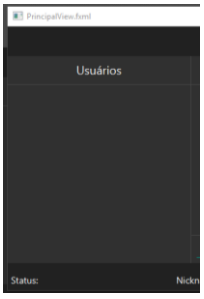


Figura 12 - Usuários disponíveis

Fonte: Autoria própria.

5.1.2 Chat privado

A partir da escolha do usuário na lista de utilizadores disponíveis é apresentada uma janela com a possibilidade de chat privado. Trata-se de uma troca de mensagens parecida com a que ocorre no chat global, mas que tem destino individual. Esta janela

pode ser fechada em qualquer momento através do botão destinado. A estrutura da tela de chat privado está disponível na figura 13:

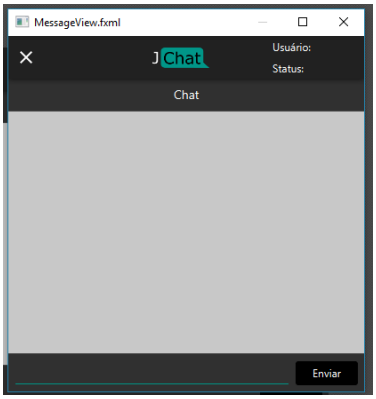


Figura 13 - Chat privado

Fonte: Autoria própria.

5.2 Servidor

A módulo do servidor é responsável por interceptar e processar as requisições do cliente e mandar uma resposta adequada para ele ou para outros utilizadores pertinentes, esse canal de comunicação entre servidor e usuários e possível por meio da utilização dos sockets.

O servidor funciona basicamente em um loop continuo em que tenta aceitar novas conexões socket. Ao aceitar uma conexão é aberta uma thread que ficará responsável por manter o canal entre cliente e servidor aberto e reconhecendo as interações do usuário, no projeto esse fluxo é denominado ClientSession.

5.2.1 Sessão do cliente

A principal função da sessão do cliente é receber um objeto do lado do cliente serializado e identificá-lo, a partir disso, o servidor tenta devolver um objeto de resposta que corresponda com a solicitação. Ocasionalmente, esta resposta pode ser

interpretada como um erro ou como uma mensagem de sucesso significativa para o lado do cliente.

5.2.2 Tipos de requisições

A fim de identificar as diversas interações que o usuário pode ter com o servidor e enviar uma resposta correta a ele foi estabelecido um conjunto de operações que manipulam cada uma das requisições do utilizador. As operações estabelecidas foram: SEND_OR_RECEIVE_MSG, LOGIN, LOGOFF, INFO, ERROR_MSG, SUCCESS_MSG.

SEND_OR_RECEIVE_MSG: Este tipo de requisição é processada pelo servidor como o envio ou recebimento de mensagens, caso o servidor consiga enviar uma resposta ao utilizador correto ele envia uma SUCCESS_MSG com as informações corretas, senão envia uma ERROR_MSG.

LOGIN: No momento da entrada do usuário do sistema é enviada uma requisição deste tipo ao servidor, caso o login ocorra de forma bem-sucedida é enviada uma mensagem que é interpretada pelo cliente como sucesso, caso haja erro é enviada uma mensagem que o cliente interpreta como erro.

LOGOOFF: Basicamente fecha uma conexão do cliente com o servidor, sempre que der certo o log apresentará uma mensagem de sucesso, caso algo de errado uma mensagem de detalhes será mostrada no log.

INFO: Esta interação é processada pelo servidor como um pedido do cliente para obter certas informações do sistema, geralmente a informação requisitada é a de usuários online no chat. Caso uma resposta possa ser enviada, ela é devolvida com sucesso, senão uma mensagem de inválida é enviada.

ERROR_MSG e SUCESS_MSG: Estes dois identificadores estão relacionados ao SEND_OR_RECEIVE_MSG, pois quando há uma mensagem válida é enviada do servidor a mensagem de sucesso, senão é enviada a mensagem de erro.

6 REFERÊNCIA BIBLIOGRÁFICA

FAZENDA, Alessandro. Graduado em Ciência da Computação pela Freevale. Disponível em: <https://pt.slideshare.net/AlessandroFazenda/historia-das-redes-de-computadores>. 2016. Acesso em: 8 de abril de 2017.

LUPPI, Iria. Desenvolvedora de sistemas Web. Disponível em: <https://www.oficinadanet.com.br/post/10123-historia-das-redes-de-computadores>. 2014. Acesso em: 8 de abril de 2017.

MARTINEZ, Marina. Escritora do InfoEscola. Disponível em: <http://www.infoescola.com/informatica/topologias-de-redes/>. 2010. Acesso em: 23 de abril de 2017.

OLIVEIRA, Felipe Soares. Professor no centro universitário Unipê. Disponível em: <http://blog.unipe.br/graduacao/conheca-tudo-sobre-a-historia-das-redes-de-computadores>. 2017. Acesso em: 8 de abril de 2017.

OLIVEIR, Mike. IT professional. Disponível em: <http://www.itprc.com/tcpipfaq/ >. 1999. Acesso em: 22 de abril de 2017.

TANENBAUM, Andrew S. **Computer Networks, Fourth edition**. Vrije Universiteit. Amsterdã, Holanda. Tradução: Vandenberg D. de Souza. Campus, 2011.

TEIXEIRA, Nuno; HORTA, Tiago. Formação tecnológica na área de servidores. Disponível em: <https://sites.google.com/site/wsxixa/terminologia-de-redes>. 2003. Acesso em: 22 de abril de 2017.

TORRES, Gabriel. Editor executivo do Clube do Hardware. Disponível em: <http://www.clubedohardware.com.br/artigos/redes/o-modelo-de-refer%C3%AAncia-osi-para-protocolos-de-rede-r34766/>. 2007. Acesso em: 15 de abril de 2017.

APÊNDICE

APÊNDICE A – Relatório com algumas linhas de código do programa

```
//Parte principal do servidor
package src;

import java.io.IOException;
import java.net.ServerSocket;
import java.util.ArrayList;
import java.util.List;

import javafx.collections.FXCollections;
import javafx.collections.ListChangeListener;
import javafx.collections.ObservableList;
import model.requests.InfoRequest;
import model.requests.InfoUserModel;
import model.requests.Request;
import util.DateUtil;

public class ServerInstance {

	private static ObservableList<ClientSession> loggedClients =
FXCollections.observableArrayList();
	private static List<ClientSession> clientsThread = new ArrayList<>();

	public static void main(String[] args) {
		loggedClients.addListener(new LoggedClientEvent());

		try(ServerSocket serverInstance = new ServerSocket(9876)) {
			System.out.println("<" + DateUtil.dateTimeNow() + "> Server started!
Logs below:\n\n");

			while(true) {
				ClientSession clientConnection = new
ClientSession(serverInstance.accept());
				clientsThread.add(clientConnection);
				clientConnection.start();
				System.out.println("<" + DateUtil.dateTimeNow() + ">" + " Client
connection. IP: " + clientConnection.getSession().getInetAddress().getHostAddress() + "\n");
			}

		} catch (IOException e) {
			System.out.println("<" + DateUtil.dateTimeNow() + ">" + " IOExeption
error.\nDetails: " + e.getMessage() + "\n");
		}

	}
```

```
	public static void logoffClient(ClientSession client) {
		loggedClients.remove(client);
		int index;
		if((index = clientsThread.indexOf(client)) > -1) {
			clientsThread.get(index).interrupt();
			clientsThread.remove(client);
		}
	}

	public static void loginClient(ClientSession client) {
		loggedClients.add(client);
	}

	public static ClientSession getClientSession(String user) {
		for(ClientSession client : loggedClients) {
			if(client.getUser().equals(user))
				return client;
		}
		return null;
	}

	public static boolean checkExistClient(String user) {
		for(ClientSession client : loggedClients) {
			if(client.getUser().equals(user))
				return true;
		}
		return false;
	}

	public static InfoUserModel[] getOnlineUsers() {
		List<InfoUserModel> users = new ArrayList<>();
		for(ClientSession client : loggedClients) {
			InfoUserModel user = new InfoUserModel();
			user.setLogin(client.getUser());
			user.setStatus(true);
			users.add(user);
		}
		InfoUserModel[] usersArray = new InfoUserModel[users.size()];
		return users.toArray(usersArray);
	}

	public static ClientSession[] getLoggedClients() {
		ClientSession[] loggedClientsArray = new ClientSession[loggedClients.size()];
		return loggedClients.toArray(loggedClientsArray);
	}

	private static class LoggedClientEvent implements
ListChangeListener<ClientSession> {

		@Override
		public void onChanged(Change<? extends ClientSession> c) {

			while(c.next()) {
```

```
                                if(c.wasAdded()) {
                                        ArrayList<InfoUserModel> users = new ArrayList<>();

                                        for(ClientSession client : c.getAddedSubList()) {
                                                InfoUserModel user = new InfoUserModel();
                                                user.setLogin(client.getUser());
                                                user.setStatus(true);
                                                users.add(user);
                                        }


                                        InfoUserModel[] infoUser = new
InfoUserModel[users.size()];
                                        ClientSession[] clients = new
ClientSession[loggedClients.size()];
                                        ClientSession[] exceptions = new
ClientSession[c.getRemoved().size()];
                                        InfoRequest infoRequest = new InfoRequest("Server");
                                        infoRequest.setUsers(users.toArray(infoUser));
                                        Request request = (Request) infoRequest;
                                        ServerTasks.broadcast(request,
loggedClients.toArray(clients), c.getAddedSubList().toArray(exceptions));

                                } else {
                                        List<InfoUserModel> users = new ArrayList<>();

                                        for(ClientSession client : c.getRemoved()) {
                                                InfoUserModel user = new InfoUserModel();
                                                user.setLogin(client.getUser());
                                                user.setStatus(false);
                                                users.add(user);
                                        }

                                        InfoUserModel[] infoUser = new
InfoUserModel[users.size()];
                                        ClientSession[] clients = new
ClientSession[loggedClients.size()];
                                        ClientSession[] exceptions = new
ClientSession[c.getRemoved().size()];
                                        InfoRequest infoRequest = new InfoRequest("Server");
                                        infoRequest.setUsers(users.toArray(infoUser));
                                        Request request = (Request) infoRequest;
                                        ServerTasks.broadcast(request,
loggedClients.toArray(clients), c.getRemoved().toArray(exceptions));
                                }

                        }

                }

        }

}

//Parte principal do cliente
```

```
package controller;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.InetSocketAddress;
import java.net.Socket;
import java.util.ArrayList;

import controller.controllers.LoginController;
import controller.controllers.MessageController;
import controller.controllers.PrincipalController;
import javafx.application.Application;
import javafx.application.Platform;
import javafx.beans.property.BooleanProperty;
import javafx.beans.property.SimpleBooleanProperty;
import javafx.beans.value.ChangeListener;
import javafx.collections.FXCollections;
import javafx.collections.ObservableSet;
import javafx.stage.Stage;
import javafx.stage.StageStyle;
import model.requests.OperationType;
import model.requests.Request;

public class MainController extends Application {

	private static Socket connection = new Socket();
	private static ObjectInputStream ois;
	private static ObjectOutputStream oos;
	private static RecieveObjectThread recieveObject;
	private static Thread recieveObjectT = new Thread(recieveObject);
	private LoginController loginController;
	private PrincipalController principalController;
	private ArrayList<MessageController> messageWindows = new ArrayList<>();
	private ObservableSet<String> chatWindowsUsers = FXCollections.observableSet();
	private BooleanProperty connectionStatus = new SimpleBooleanProperty(true);
	private boolean userLogged;
	private boolean logoffRequested = false;
	private String nickname;
	private Stage rootStage;
	private String host;


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

	public static Socket getConnection() {
		return connection;
	}

	public static ObjectInputStream getOis() {
		return ois;
	}
```

```
public static ObjectOutputStream getOos() {
	return oos;
}

public String getNickname() {
	return nickname;
}

public synchronized boolean setConnection(String host) {
	this.host = host;
	connection = new Socket();
	try {
		connection.connect(new InetSocketAddress(host, 9876), 1200);
		connection.setKeepAlive(true);
		oos = new ObjectOutputStream(connection.getOutputStream());
		ois = new ObjectInputStream(connection.getInputStream());
		return true;
	} catch (IOException e) {
		return false;
	}
}

@Override
public void start(Stage stage) {
	rootStage = stage;
	userLogged = false;
	recieveObject = new RecieveObjectThread();
	connectionStatusEvent(this.connectionStatus);
	rootStage.setOnCloseRequest((event) -> closeApp());
	openLogonScreen();
}

public Stage getStage() {
	return rootStage;
}

public void closeApp() {
	logoff();
	rootStage.close();
	Platform.exit();
	System.exit(0);
}

public void recieveObject(Request request) {
	if(loginController != null && !userLogged) {
		new Thread(() -> {
			Platform.runLater(() -> {
				loginController.recieveObject(request);
			});
		}).start();
	} else if(principalController != null && userLogged) {
		new Thread(() -> {
			Platform.runLater(() -> {
				if(!isMessageWindowRequest(request));
```

```
                                principalController.recieveObject(request);
                        });
                }).start();
        }
}

public void openLogonScreen() {
        if(principalController == null) {
                loginController = new LoginController(this);

        } else {
                principalController.close();
                principalController = null;
                loginController = new LoginController(this);
        }
        rootStage.setResizable(false);
        rootStage.initStyle(StageStyle.UNDECORATED);
        rootStage.show();
}

public void openPrincipalScreen(String nickname) {
        if(loginController == null) {
                principalController = new PrincipalController(this, nickname);
        } else {
                loginController = null;
                principalController = new PrincipalController(this, nickname);
        }
        this.nickname = nickname;
        userLogged = true;
}

public void openMessageScreen(String loginRecipient) {
        if(chatWindowsUsers.add(loginRecipient)) {
                messageWindows.add(new MessageController(loginRecipient, this));
        } else {
                for(MessageController msgWindow : messageWindows) {
                        if(msgWindow.getRecipient().equals(loginRecipient)) {
                                msgWindow.showWindow();
                                break;
                        }
                }
        }
}

public void closeChatWindow(String loginRecipient) {
        if(chatWindowsUsers.remove(loginRecipient)) {
                messageWindows.removeIf((t) -> {
                        if(t.getRecipient().equals(loginRecipient)) {
                                t.getStage().close();
                                return true;
                        } else {
                                return false;
                        }
                });
        }
```

```
	}

	public void offlineUser(String loginRecipient) {
		for(MessageController window : messageWindows) {
			if(window.getRecipient().equals(loginRecipient))
				window.recipientDisconnected();
		}
	}

	public void onlineUser(String loginRecipient) {
		for(MessageController window : messageWindows) {
			if(window.getRecipient().equals(loginRecipient))
				window.recipientReconnected();
		}
	}

	public boolean isMessageWindowRequest(Request msg) {
		if(msg.getOperation() == OperationType.SEND_OR_RECIEVE_MSG &&
!msg.getUserFrom().equals("Server") && msg.getUserTo() != null) {
			boolean found = false;
			for(MessageController msgWindow : messageWindows) {
				if(msgWindow.getRecipient().equals(msg.getUserFrom())) {
					msgWindow.recieveObject(msg);
					found = true;
					break;
				}
			}
			if(!found) {
				openMessageScreen(msg.getUserFrom());
				isMessageWindowRequest(msg);
			}
			return true;
		} else if(msg.getOperation() == OperationType.SUCCESS_MSG ||
msg.getOperation() == OperationType.ERROR_MSG &&
!msg.getUserFrom().equals("Server")){
			for(MessageController msgWindow : messageWindows) {
				if(msgWindow.getRecipient().equals(msg.getUserFrom())) {
					msgWindow.recieveObject(msg);
					break;
				}
			}
			return true;
		} else {
			return false;
		}
	}

	public void initializeThreads() {
		recieveObjectT = new Thread(recieveObject);
		recieveObjectT.start();
	}

	public void finalizeConnection() {
		if(connection != null) {
			if(!connection.isClosed())
```

```
                        try {
                                connection.close();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }
                connection = null;
        }

        public boolean getConnectionStatus() {
                return connectionStatus.get();
        }

        public void connectionStatusEvent(BooleanProperty connectionStatus) {
                connectionStatus.addListener((ChangeListener<Boolean>) (observable,
oldValue, newValue) -> {
                        if(newValue && oldValue != newValue) {
                                reconnectionAction();
                        }
                        else if(!newValue && oldValue != newValue) {
                                lostConnectionAction();
                        }
                });
        }

        public void lostConnectionMessages() {
                for(MessageController window : messageWindows) {
                        window.lostConnection();
                }
        }

        public void reconnectConnectionMessages() {
                for(MessageController window : messageWindows) {
                        window.reconnected();
                }
        }

        public void lostConnectionAction() {
                if(loginController != null && !userLogged) {
                        finalizeConnection();
                        if(!logoffRequested)
                                Platform.runLater(() -> loginController.lostConnection());
                } else {
                        finalizeConnection();
                        if(!logoffRequested)
                                Platform.runLater(() -> principalController.lostConnection());
                        userLogged = false;
                }
        }

        public boolean reconnect(String nickname) {
                Request loginRequest = new Request(OperationType.LOGIN);
                loginRequest.setUserFrom(nickname);
                loginRequest.setUserTo("Server");
                try {
```

```
                    if(connection != null && oos != null) {
                        if(!connection.isClosed() && connection.isConnected()) {
                            oos.writeObject(loginRequest);
                            return true;
                        } else {
                            return false;
                        }
                    } else {
                        return false;
                    }
                } catch(Exception e) {
                    return false;
                }
            }

            public void reconnectionAction() {
                if(loginController != null) {
                    Platform.runLater(() -> loginController.reconnect());
                } else {
                    Platform.runLater(() -> principalController.reconnect());
                    userLogged = true;
                }
            }

            public void logoff() {
                try {
                    logoffRequested = true;
                    if(connection != null) {
                        if(!connection.isClosed()) {

                            if(connection.isConnected()) {
                                Request requestLogoff = new
Request(OperationType.LOGOFF);
                                requestLogoff.setUserTo("Server");

                                if(loginController != null) {
                                    requestLogoff.setUserFrom(null);
                                } else {

            requestLogoff.setUserFrom(principalController.getNickname());
                                }

                                oos.writeObject(requestLogoff);
                                Thread.sleep(100);
                            }
                            if(connection != null) connection.close();
                            recieveObjectT.interrupt();
                        }
                    }
                } catch (Throwable e) {
                    e.printStackTrace();
                }
            }

            /*---------- Threads ----------*/
```

```
	private class RecieveObjectThread implements Runnable {

		@Override
		public void run() {
			new Thread(() -> {

				while(true) {
					if(connection != null) {

						try {
							if((!connection.isConnected() ||
!connection.getInetAddress().isReachable(1500)) && !logoffRequested) {

								if(!(connectionStatus.getValue() ==
false))

	connectionStatus.set(false);

							}
						} catch (IOException e) {
							// TODO Auto-generated catch block
							e.printStackTrace();
						}

					}

					try {
						Thread.sleep(1000);
					} catch(InterruptedException e) {
						e.printStackTrace();
					}

				}

			}).start();

			boolean down = false;
			while(true) {

				if(!down) {

					try {
						Object obj = ois.readObject();

						if(obj != null) {

							if(obj instanceof Request) {
								Request request = (Request) obj;

								if(loginController != null ||
!userLogged) {

									try {
										Thread.sleep(500);
```

```
                                                                        } catch
(InterruptedException e) {
                                                                                e.printStackTrace();
                                                                        }

                                                                }

                                                                recieveObject(request);
                                                        }

                                                } else {
                                                        throw new IOException("Lost
connection");
                                                }

                                        } catch(IOException e) {
                                                if(!(connectionStatus.get() == false))
                                                        connectionStatus.setValue(false);
                                                down = true;

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

                                } else {
                                        if(!logoffRequested) {
                                                try {
                                                        Thread.sleep(500);
                                                } catch (InterruptedException e) {
                                                }

                                                if(setConnection(host)) {
                                                        if(nickname != null) {
                                                                if(reconnect(nickname)) {

        connectionStatus.setValue(true);
                                                                        down = false;
                                                                }
                                                        }
                                                        down = false;

                                                } else {
                                                        if(!(connectionStatus.getValue() == false))
                                                                connectionStatus.setValue(false);
                                                }
                                        } else {
                                                break;
                                        }
                                }
                        }
                }
        }
}
```