**UNIVERSIDADE PAULISTA**

**MARCOS PAULO FRANCISCO VAZ F09BEG0**

**MATHEUS DOS SANTOS SILVA N590785**

**MATHEUS OLIVEIRA DE MORAES N477GA2**

**SAMUEL ARAUJO DE SOUZA F30AJG4**

**DESENVOLVIMENTO DE UMA FERRAMENTA PARA COMUNICAÇÃO EM REDE**

**SÃO PAULO**

**2022**

**SUMÁRIO**

[1 OBJETIVO 2](#_Toc104344926)

[1.1 Objetivos específicos 2](#_Toc104344927)

[2 INTRODUÇÃO 3](#_Toc104344928)

[3 FUNDAMENTOS DA COMUNICAÇÃO DE DADOS EM REDE 5](#_Toc104344929)

[3.1 Comunicação e seus elementos 5](#_Toc104344930)

[3.2 Rede de computadores 5](#_Toc104344931)

[3.3 Rede de Acesso 6](#_Toc104344932)

[3.4 Representação de dados 7](#_Toc104344933)

[3.5 Fluxo de dados 7](#_Toc104344934)

[3.6 Links de conexão 8](#_Toc104344935)

[3.7 Topologia de redes 8](#_Toc104344936)

[3.8 Arquitetura em camadas 9](#_Toc104344937)

[3.8.1 Modelo OSI 9](#_Toc104344938)

[3.8.2 Arquitetura TCP/IP 10](#_Toc104344939)

[3.9 Encapsulamento e desencapsulamento 11](#_Toc104344940)

[4 PLANO DE DESENVOLVIMENTO DA APLICAÇÃO 12](#_Toc104344941)

[4.1 Metodologia de desenvolvimento 12](#_Toc104344942)

[4.2 Product Backlog 13](#_Toc104344943)

[4.3 Design Sprint 14](#_Toc104344944)

[4.3.1 Wireframe 15](#_Toc104344945)

[5 PROJETO DA APLICAÇÃO 17](#_Toc104344946)

[5.1 Socket 17](#_Toc104344947)

[5.2 Buffer 18](#_Toc104344948)

[5.3 Thread 18](#_Toc104344949)

[5.3.1 Multithread 18](#_Toc104344950)

[6 RELATÓRIO COM AS LINHAS DE CÓDIGO 20](#_Toc104344951)

[7 PROGRAMA EM FUNCIONAMENTO 27](#_Toc104344952)

[Bibliografia 30](#_Toc104344953)

# **OBJETIVO**

Desenvolver uma aplicação para a comunicação em redes, passível de ser utilizada no seguinte caso: a Secretaria de Estado do Meio Ambiente 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 se revezarão dentro de cada indústria, controlando os processos e passando informações online para a Secretaria.

Portanto, um *chat* em grupo entre os inspetores e a Secretaria, para que esses sejam capazes de comunicar os problemas em tempo real.

## Objetivos específicos

* Construir a aplicação usando *Java*;
* Comunicação realizada por intermédio do protocolo TCP/IP;
* Primitivas dos *sockets* de Berkeley para a comunicação.

# INTRODUÇÃO

O primeiro meio de comunicação criado foi a voz humana. Esse é um meio de comunicação caraterístico, exclusivo, dos seres humanos. Desenvolve-la foi uma conquista crucial, que nos tornou humanos e separados de outras espécies. Quando aprendemos a falar? Provavelmente em algum período do *Paleolítico* — no entanto, esse é tema de um longo e acalorado debate, uma vez que a fala não tem nenhuma evidência física para estudo e datação (Cambridge University Press, 2008).

Uma vez criada essa capacidade, nós evoluímos e nosso cérebro se adaptou a ela. Como consequência, nos tornamos animais cuja natureza tornou forte aqueles que socializavam.

A camada mais externa do cérebro humano é chamada neocórtex. Se você colocar um ser humano num ambiente social e usar um Imageador por Ressonância Magnética (IRM) para escanear o cérebro dele, você irá perceber que o neocórtex é a região mais ativa — onde todas as ações estão. Quando você compara o neocórtex de seres humanos com o de primatas (macacos, gorilas, etc.) dá para perceber que o nosso é bem maior que o de qualquer um deles. Em poucas palavras, nosso cérebro evoluiu para a capacidade social (Dam, 2017, tradução nossa).

Sem esse senso social, seria impossível para o ser humano **cooperar** na escala que cooperam. Foi nossa capacidade de cooperação que deu origem as civilizações. Pudemos desfrutar da experiência e conhecimento de outros indivíduos. A sociedade potencializou o indivíduo, nos colocando no topo da cadeia alimentar. Como disse Rollo May, um psicólogo americano: “A comunicação leva à comunidade, isto é, à compreensão e à valorização mútua” (Dam, 2017, tradução nossa).

A potencialização do índividuo nos levou ao próximo passo da capacidade de comunicação: a origem da escrita.

Uma escrita sistematizada aparece somente por volta de 3500 a.C., quando os sumérios desenvolveram a escrita cuneiforme na Mesopotâmia. Os registros cotidianos, econômicos e políticos da época eram feitos na argila, com símbolos formados por cones. Nesse mesmo momento, surgem os hieróglifos no Egito. Essa escrita era dominada apenas por pessoas poderosas da sociedade, como escribas e sacerdotes (UFMG, 2020).

Ainda no Egito, mais de 4 mil anos antes da Era Cristã, já existiam os sigmanacis, mensageiros que levavam recados escritos a pé ou montados em cavalos e camelos.

Com o avanço e desenvolvimento de diferentes impérios e civilizações, começam os conflitos. Foram nesses tempos que os seres humanos aprenderam a importância da informação e o valor que ela tem. Como disse Ibiki Morino, um personagem fictício de Naruto (Kishimoto, 2007), quando descrevendo as qualidades necessárias para se tornar um ninja:

Porque... às vezes, a informação é mais importante que a vida ... e em missões e no campo de batalha, as pessoas arriscam suas vidas para pôr suas mãos nela. Informações importantes em suas mãos... podem ser uma arma poderosa para os seus companheiros e a vila (pp. 6-7).

Com isso, técnicas de criptografia foram desenvolvidas — manter as mensagens secretas era o objetivo. A cítala foi a primeira solução tecnológica para esse fim.

Do século XVIII para frente, foram desenvolvidas algumas tecnologias de comunicação à distância em tempo real: a telegrafia visual com Claude Chappe em 1792, na França; o telégrafo elétrico com Samuel Morse em 1836; a fibra óptica, em 1965; ARPANET/Internet em 1969 nos Estados Unidos (Garfinkel & Grunspan, 2018).

Hoje, a comunicação humana é, praticamente, instantânea. É possível tirar muito proveito dessa velocidade a partir da **cooperação**. E é nesse ponto que esse trabalho busca se aprofundar — uma ferramenta para a comunicação em tempo real de problemas ambientais.

# FUNDAMENTOS DA COMUNICAÇÃO DE DADOS EM REDE

## Comunicação e seus elementos

Comunicação, do latim “communicare”, que significa “partilhar” é uma forma de trocar informações entre dois ou mais indivíduos por meio de sinais que podem variar de diversas formas sendo sonoros, visuais, etc.

Para a comunicação existir são necessários 4 elementos básicos, sendo eles: um emissor, um receptor, um meio de comunicação e um sinal. Podemos ver essa descrição na imagem 1:

Imagem 1 – Diagrama com os elementos da comunicação

Fonte: própria

O emissor é quem transmite as informações, o receptor quem recebe essas informações, o meio de transmissão é a forma ou o caminho que transporta o sinal entre ambos e o sinal é a mensagem que é composta pelos dados e pelas informações.

## Rede de computadores

Segundo Tanenbaum (2013, p. 1) uma rede de computadores é “um conjunto de computadores autônomos interconectados por uma única tecnologia”.

A comunicação de dados em rede se refere a troca de informações entre um conjunto de computadores ou dispositivos interligados por um sistema de comunicação através de meios físicos. “A conexão não precisa ser feita por um fio de cobre; também podem ser usadas fibras ópticas, micro-ondas, ondas de infravermelho e satélites de comunicações” (Tanenbaum, 2013, p. 1)

Para que essa comunicação ocorra de forma eficaz, é necessário que os dados a serem transportados não sofram danos ou se percam no caminho. São elementos básicos para um sistema de comunicação de dados: os nós, a mensagem, os meios de transmissão e os protocolos.

1. **Mensagem**: a mensagem é a informação (dado) a ser transmitida. Pode ser feita de textos alfanuméricos, imagens, áudio, vídeo, entre outros;
2. **Nós**: os nós são os emissores e receptores quem transmitem e recebem as mensagens com as informações. Pode ser um computador, celular, televisão, impressora, entre outros;
3. **Meio de Transmissão**: o meio de transmissão é o caminho que une os nós de uma rede. Pode ser um cabo coaxial, cabo de fibra óptica, entre outros.
4. **Meio de Transmissão**: o meio de transmissão é o caminho que une os nós de uma rede. Pode ser um cabo coaxial, cabo de fibra óptica, entre outros.

## Rede de Acesso

Rede de acesso é a conexão física que conecta o sistema final ao roteador mais próximo (roteador de borda). Para que o acesso a nuvem seja feito a partir de sua casa, é necessário um provedor de acesso à internet. Esses provedores, normalmente empresas de telecomunicações, se conectam à backbones para que os usuários tenham acesso a qualquer rede conectada a web.

Principais tipos de redes de acesso:

1. **Discada (dial-up)**: uma conexão “discada” que era conectada através um modem e por meio da infraestrutura de telefonia. O modem, conectado no computador, disca (dial) para o provedor que estabelece uma chamada telefônica. Modem significa modulador e demudulador, e é quem faz o trabalho de converter o sinal digital em analógico e enviar para o modem do provedor. Um problema de do acesso discado é a que durante a navegação, a linha telefônica fica ocupada;
2. **Asymmetric Digital Subscriber Line (ADSL)**: é um dos tipos de acesso mais populares do Brasil, ele também utiliza da rede de telefonia, porém dividindo o par de fios de cobre em três canais de bandas distintas, voz, upload e download. Com isso se pode acessar a internet e falar ao telefone simultaneamente;
3. **Cabo**: diferente do ADSL, que utiliza da infraestrutura da telefonia, esse tipo de acesso utiliza da rede de cabos de televisões por assinatura e também precisa que um modem que receba e envie os sinais;
4. **Fiber-to-the-Home (FTTH)**: tecnologia que é capaz de transmitir internet, TV digital e telefonia com alta velocidade;
5. **Ethernet (Rede Local)**: na ethernet, os usuários se conectam via cabos de cobre trançado a um comutador (switches);
6. **Wireless Fidelity (Wi-Fi)**: fidelidade sem fio ou WiFi é uma tecnologia que se refere ao acesso à internet por meios de dispositivos sem fio como notebooks, celulares, tablets, assim por diante.

## Representação de dados

As informações (dados) podem ser transmitidas de diversas formas como textos, números, imagens, vídeos e áudios. Essas informações precisam ser representadas em algum meio físico, geralmente sinais elétricos ou eletromagnéticos. Temos também dois tipos de sinais, os analógicos e os digitais. Em computadores a transmissão da informação é feita de modo digital, onde as variações então entre 0 e 1 bits. Para a conversão de sinais digitais em analógicos são usados moduladores que podem alterar a amplitude, a fase ou a frequência do sinal (Forouzan & Oliveira, 2007).

## Fluxo de dados

Uma comunicação entre dois dispositivos pode ocorrer de três diferentes formas, sendo elas a *simplex*, *half-duplex* ou *full-duplex*.

* **Simplex**: nesse modo, a comunicação possui somente uma direção, sendo unidirecional, onde apenas um dos dispositivos poderá transmitir e o outro, por sua vez, apenas receber. Um teclado pode ser um exemplo desse modo, onde ele só pode enviar informações;
* **Half-duplex**: nesse modo, a comunicação possui ambas as direções com os dispositivos podendo enviar e receber, porém, não ao mesmo tempo. Deste modo, toda a capacidade do canal pode ser aproveitada;
* **Full-duplex**: nesse modo, ambas estações podem transmitir e receber simultaneamente e compartilham da mesma capacidade do canal.

## Links de conexão

Em uma rede possuímos dois ou mais dispositivos conectados através links de comunicação, link é um caminho de comunicação que transfere os dados. Os dois tipos de conexões que temos são; ponto a ponto e multiponto.

Uma conexão ponto a ponto dispõe de um link dedicado entre dois dispositivos, ou seja, ela fica reservada exclusivamente para a transmissão entre esses dispositivos. Um exemplo eram as telefonias que utilizavam esse tipo de conexão e um link ficava ocupado ao retirar o telefone do gancho.

A conexão multiponto, como o próprio nome sugere, é uma conexão na qual mais de dois dispositivos podem compartilhar de um mesmo link.

## Topologia de redes

Como topologia de rede, temos a topologia física e a lógica. Podemos entender a topologia física como a forma que é organizada fisicamente os dispositivos (nós), sendo uma representação, assim como um mapa — usando a teoria dos grafos, cada nó é um dispositivo de rede e cada aresta, arco, é uma rede de acesso. Existem diversas topologias como Barramento, Anel, Estrela, Árvore, Híbrida, entre outras.

* **Topologia de barramento**: todos os dispositivos estão conectados e compartilham um mesmo barramento de dados. Esse tipo de topologia utiliza de uma conexão multiponto sendo utilizados cabos coaxiais que interliga os dispositivos da rede;
* **Topologia anel**: utiliza a conexão ponto a ponto (3.6), cada dispositivo se conecta a outros dois, formando um grande círculo. O sinal é percorre em uma única direção do anel, de nó em nó, onde cada nó possui um repetidor que regenera os bits;
* **Topologia estrela**: os dispositivos não se conectam entre si. Cada dispositivo se conecta em um controlador central, que pode ser um hub ou switch;
* **Topologia de malha**: cada nó está conectado a diversos outros. Todos os dispositivos possuem um link ponto a ponto com os outros dispositivos da rede;
* **Topologia híbrida**: podemos ter mais de uma topologia, como uma principal e outras conectando ramificações. Sua maior característica é a flexibilidade e a possibilidade de juntar diversas outros topologias.

## Arquitetura em camadas

Com a complexidade de um sistema como a internet, com inúmeras aplicações, protocolos, conexões diferentes, meios de transmissão, tornou-se necessário organizar a arquitetura de rede. Com isso surgiu o conceito de arquitetura em camadas, onde uma fica disposta sobre as outras oferecendo seus serviços as camadas superiores, havendo uma hierarquia. Desse modo as tarefas devem ser realizadas na ordem determinada por essa hierarquia, onde camada implementará suas próprias regras e protocolos (Baptista, 2012).

### Modelo OSI

O Modelo *Open Systems Interconnections* (OSI) criado pelo órgão *International Standards Organization* (ISO), em 1984, é referência de arquitetura em camadas para o processo de transmissão de dados. Foi desenvolvido como um sistema aberto para facilitar a interoperabilidade entre dois diferentes sistemas, garantindo a comunicação independente de seus fabricantes. Dessa forma, não se era necessário fazer mudanças nos hardwares dos dispositivos.

O Modelo OSI é formado por sete camadas distintas e ordenadas, sendo elas:

* **Camada física (camada 1)**: é responsável pela transmissão dos bits pelo meio físico, especificando os padrões mecânicos e elétricos. Define o tipo meio de transmissão a ser usado, a taxa de dados, a sincronização de bits, configuração da linha, e a topologia;
* **Camada de enlace (camada 2):** é quem transforma do meio de transmissão físico, um link confiável, que parece livre de qualquer erro. Tem também como outras funções o empacotamento, o endereçamento físico, o controle de fluxo, o controle de erros e o controle de acesso;
* **Camada de rede (camada 3)**: é a responsável pela conexão das redes e entrega de pacotes. A camada de enlace direciona a entre do pacote apenas entre os dispositivos de uma mesma rede, já a camada de rede garante que este pacote saia de seu pondo de origem até o seu ponto de destino. Também tem como outras funções o endereçamento lógico e o roteamento;
* **Camada de transporte (camada 4)**: é responsável peça transferência de dados, independente de topologias de redes ou sub-rede. Mesmo que a camada de rede saiba a origem e o destino de cada pacote, é necessário ter conhecimento da relação entre os pacotes. Por isso a camada de transporte garante que a mensagem chegue intacta e na ordem correta. Também é responsável pela segmentação e remontagem, controle de conexão, controle de fluxo, controle de erros;
* **Camada de sessão (camada 5)**: é quem estabelece e sincroniza a comunicação entre os receptores e os transmissores. Tendo como suas responsabilidades o controle de diálogo e a sincronização para que caso a transferência de pacotes seja interrompida, não seja necessário o reenvio de tudo novamente, com a sincronização o reenvio pode começar de onde parou;
* **Camada de apresentação (camada 6)**: é responsável pela representação dos dados, sintaxes e semânticas das informações trocadas entre o transmissor e receptor. É responsável pela tradução, criptografia e compreensão dos dados;
* **Camada de aplicação (camada 7):** é responsável por fornecer ao usuário o acesso à rede, a interface e serviços. É responsável pelo terminal de rede virtual, transferência, acesso e gerenciamento de arquivos.

### Arquitetura TCP/IP

A arquitetura TCP/IP (Transmission Control Protocol), criada em 1974, é uma arquitetura flexível com um conjunto de protocolos centrais e hierárquicos para garantir também a interoperabilidade entre diferentes sistemas de hardware. Diferentemente do modelo OSI, essa arquitetura possui quatro camadas:

* **Camada de aplicação (camada 4)**: essa parte contém todos os protocolos para um serviço específico de comunicação de dados em um nível de processo-a-processo (por exemplo: como um *web browser* deve se comunicar com um servidor da web). Ela corresponde as 5ª, 6ª e 7ª camada do Modelo OSI;
* **Camada de transporte (camada 3)**: essa parte controla a comunicação host-a-host. Ela corresponde a 4ª camada do Modelo OSI;
* **Camada de internet (camada 2)**: essa parte é responsável pelas conexões entre as redes locais, estabelecendo assim a interconexão;
* **Camada de enlace (camada 1)**: essa é a parte responsável por enviar o datagrama recebido pela camada de "Internet" em forma de um quadro através da rede.[12] Tecnologias usadas para as conexões: Ethernet rede com fio e Wi-Fi rede sem fio. No modelo OSI, essa camada também é física, porém, é dividido em duas partes: física e enlace de dados. A física é a parte do hardware (por exemplo os cabos das redes com fio) e a enlace de dados é a parte lógica do hardware: endereço MAC de origem e destino; controle de enlace lógico; controle de acesso ao meio.

## Encapsulamento e desencapsulamento

Denomina-se encapsulamento o processo de passagem dos dados de uma camada para outra, onde cada camada adiciona seu próprio cabeçalho com informações. Para a comunicação entre nós de uma rede, os dados transmitidos passam por todas as camadas recebem esses cabeçalhos que permitem a recuperação dos dados na outra ponta. O processo é feito de forma hierárquica, indo da camada de aplicação até a camada física, já no destino o processo é o inverso, passando primeiro pela camada física, até a camada de aplicação, sendo o processo de desencapsulamento.

# PLANO DE DESENVOLVIMENTO DA APLICAÇÃO

## Metodologia de desenvolvimento

Como forma de trabalho em grupo foi escolhido usar o Scrum, uma metodologia ágil de gestão de projetos criada por Jeff Sutherland (2014). Nas palavras dele:

O Scrum trata-se de uma mudança radical das metodologias prescritivas e de cima para baixo usadas na gerência de projetos no passado. A maior parte do desenvolvimento de software era feita usando o método em cascata, no qual um projeto era concluído em todos os estágios distintos e seguia, passo a passo, em direção ao lançamento para os consumidores, ou usuários. O processo era lento, imprevisível e, em geral, nunca resultava em um produto que as pessoas queriam ou estavam dispostas a pagar para obter. Atrasos de meses ou até mesmo de anos eram endêmicos ao processo. Os planos iniciais de passo a passo, expostos em detalhes reconfortantes em diagramas de Gantt, asseguravam aos gestores que tínhamos total controle do processo de desenvolvimento — no entanto, quase sempre, nós rapidamente ficávamos atrasados em relação ao cronograma, e desastrosamente acima do orçamento.

O Scrum propõe que o trabalho de uma equipe seja dividido em pequenas iterações; ciclos de atividades. Ao invés de planejar tudo de princípio, a ideia é planejar o suficiente para um curto período de trabalho, parar constantemente e observar os resultados, de forma a ajustar a direção do projeto.

O Scrum é divido em alguns elementos:

* **Sprints**: é o nome dado para os ciclos de cada projeto. Em geral são ciclos mensais e são determinados para que as tarefas sejam realizadas;
* **Product Backlog**: é o nome dado para o conjunto de objetivos de um projeto. No caso de um projeto de desenvolvimento de software (para o qual o Scrum foi pensado inicialmente), é o nome dado ao pacote de funcionalidades a serem desenvolvidas em um projeto;
* **Planning Meeting**: são reuniões periódicas que acontecem no início de cada sprint, ou ciclo, para planejar e priorizar os itens do Product Backlog que serão desenvolvidos naquele período;
* **Sprint Backlog**: é como se chamam as tarefas específicas que serão realizadas e desenvolvidas em cada ciclo, ou sprint;
* **Daily Scrum**: essa é uma reunião diária para acompanhamento do projeto. A ideia é que toda a equipe se reúna diariamente para discutir as atividades desenvolvidas, disseminar conhecimento, identificar impedimentos e priorizar o trabalho daquele dia. Um ponto interessante é que o Scrum propõe que estas reuniões sejam realizadas com os participantes em pé, exatamente para serem rápidas e objetivas;
* **Review Meeting**: essa é a reunião que acontece ao final de cada sprint para que a equipe apresente o que foi realizado e os resultados do trabalho daquele ciclo. A ideia é que depois dessa etapa, todos sigam para o próximo ciclo;
* **Retrospective Meeting**: essa é uma reunião que acontece logo após ao *review meeting*. Nela, é aplicado um dos princípios do *Agile Manifest*, chamado de inspeção. Os membros de uma equipe vão olhar para a Sprint, citar os pontos positivos e negativos, e pensar em como eles podem melhorara para o próximo incremento.

No Scrum, ainda, surgem alguns papéis para cada membro:

* **Product Owner**: é o ponto central com poderes de liderança sobre o produto (objeto final do projeto). Ele é o único responsável por decidir o que será feito e em qual a ordem de prioridade.;
* **Scrum Master**: o Scrum Master é responsável por ajudar a todos os envolvidos a entender e abraçar os valores, princípios e práticas do Scrum. É um líder de equipe;
* **Time Scrum**: em projetos tradicionais são criadas várias “castas” com cargos e funções bem delineadas. Na metodologia Scrum é definido o papel do Time de Scrum, que é simplesmente a junção de todas essas pessoas em uma equipe multidisciplinar e são responsáveis pela concepção, construção e testes do produto final do projeto.

## Product Backlog

Depois de analisar os requisitos passados para a execução desse trabalho, foi chegado no seguinte backlog:

**Tabela 1 – Backlog inicial do projeto**

|  |  |  |
| --- | --- | --- |
| Tarefa/User Story | Tamanho | Status |
| Java: Socket | 34 | Não iniciado |
| Java: Buffered Reader & Buffered Writer | 21 | Não iniciado |
| Java: Input Stream & Output Stream | 13 | Não iniciado |
| Java: Multiple Threads | 21 | Não iniciado |
| Java: Enviar documento feature | 54 | Não iniciado |
| Documento: Esqueleto do documento | 3 | Não iniciado |
| Documento: Objetivo do trabalho | 5 | Não iniciado |
| Documento: Introdução | 8 | Não iniciado |
| Documento: Fundamentos de rede | 13 | Não iniciado |
| Documento: Projeto do programa | 8 | Não iniciado |
| Documento: Relatório com as linhas de código | 3 | Não iniciado |
| Documento: Ficha de atividades | 5 | Não iniciado |

Fonte: Própria

Os requisitos funcionais, no caso do desenvolvimento de um trabalho, descrevem o os objetivos para a conclusão do mesmo, seus detalhes, ou seja, os requisitos descrevem que deve ser feito e o trabalho que o sistema deve realizar. Eles são os cérebros do projeto.

## Design Sprint

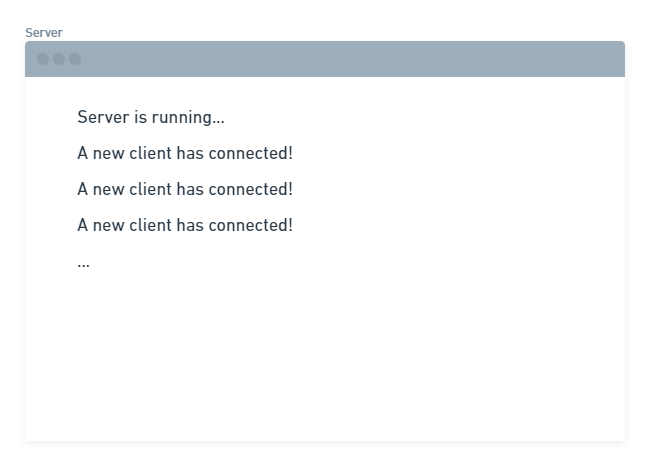
Os requisitos funcionais descrevem o comportamento do sistema, seus requisitos para o funcionamento de cada item, ou seja, os requisitos que descrevem o trabalho que o sistema deve realizar. Eles são os cérebros do projeto porque descrevem as funções que o sistema deve ter.

A metodologia foi idealizada por Jake Knapp. A primeira vez em que o Design Sprint foi colocado oficialmente em prática foi em 2012, quando Knapp apresentou o método para o Google Ventures (GV), firma de capital de risco da Google que investe em startups.

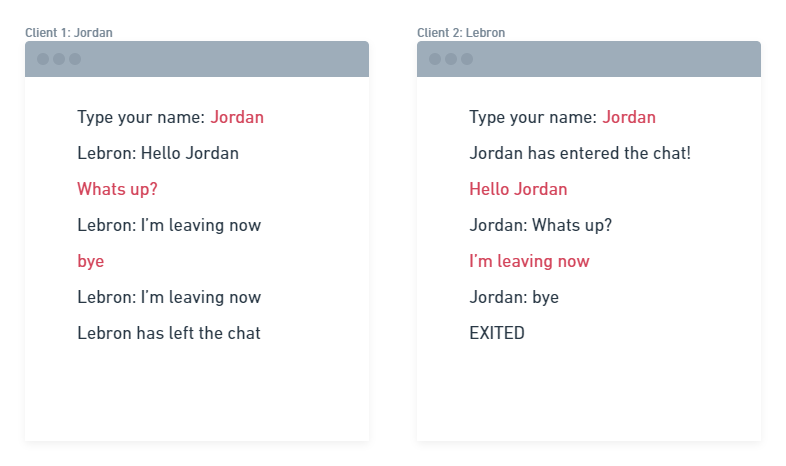
O princípio do Google Sprint é não esperar o lançamento de um Minimum Viable Product (MVP) para descobrir se uma solução é eficaz. O MVP é uma versão mais simples de um produto, feita com o mínimo de recursos possível para entregar a proposta de valor da ideia. Ele é desenvolvimento em 4 etapas padronizadas: ideia da solução; construção do produto; lançamento; aprendizado.

A execução destas etapas pode durar meses. O que o Google Sprint propõe é reduzir este processo para 40 horas de trabalho, concentrando o aprendizado na superfície e na aparência do produto. O aprendizado mais profundo viria depois da validação do protótipo criado no sprint, que pode ser aprimorado ou abandonado, dependendo dos resultados.

### Wireframe

Imagem 2 – Server wireframe

Fonte: Própria

Imagem 3 - Dois clientes conversando

Fonte: Própria

Como primeira etapa, a ideia é desenvolver o mecanismo por trás do chat em grupo, apenas com uma interface padrão não gráfica. Num segundo momento, um desenvolvimento utilizando as bibliotecas gráficas do Java.

Uma das vantagens do Scrum é que o desenvolvimento não precisa começar perfeito. A medida que os incrementos, Sprints, vão acontecendo, naturalmente a equipe vai melhorando, pegando mais velocidade e mais valor pode ser agregado ao projeto.

# PROJETO DA APLICAÇÃO

## Socket

Um dos principais conceitos da aplicação é o de *Socket*. Ele nada mais é do que uma forma de permitir que dois processos se comuniquem (Inter-process communication). Esses processos podem ou não estar na mesma máquina.

A linguagem Java nos oferece uma biblioteca para trabalhar com Sockets. Segundo a documentação deles:

Um socket é um ponto final de um link de comunicação bidirecional entre dois programas em execução na rede. As classes de socket são usadas para representar a conexão entre um programa cliente e um programa servidor. O pacote *java.net* fornece duas classes: Socket e ServerSocket — que implementam o lado cliente da conexão e o lado servidor da conexão, respectivamente (Java, s.d.).

Então criaremos uma classe *Server* que instanciará um objeto *ServerSocket*, que será usada pelo servidor, o qual precisará ficar esperando por requisições de clientes por conexão, e um objeto *Socket*, que será usado para receber e enviar dados para os clientes.

Já o cliente, para se comunicar com o servidor, usará apenas um *socket*, que é um objeto instanciado da classe *Socket*. Toda instância de cliente passará por uma classe controladora: *ClientHandler.* Dessa forma, é muito mais fácil fazer o *broadcast* das mensagens de todos os clientes, por exemplo.

Apesar de ser a instância da mesma classe *Socket* em todas as classes, o funcionamento é diferente para cada qual.

Seguindo essa estrutura, temos então a seguinte estrutura de pasta:

Imagem 4 – Estrutura de pasta

Fonte: Própria

## Buffer

Um buffer é uma pequena parte do armazenamento da memória do dispositivo usada para armazenar temporariamente alguma quantidade de dados. Normalmente, os buffers usam a RAM do dispositivo para armazenar os dados temporários e, portanto, acessar dados do buffer é muito mais rápido do que acessar a mesma quantidade de dados do disco rígido.

O Java oferece a classe *BufferedReader*. Ela é usada para ler dados de um determinado fluxo de caracteres. No nosso caso, o fluxo de caracteres estará vindo diretamente do input do nosso usuário. Então, também será usada a classe *InputStreamReader* no construtor do *BufferedReader* para pegar o input do usuário.

Também, existe a classe *BufferedWriter*, que escreve o input do usuário, junto da *ClientHandler*, para todo mundo — *broadcast*.

## Thread

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

Os diversos *threads* que existem em um programa podem trocar dados e informações entre si e compartilhar os mesmos recursos do sistema, incluindo o mesmo espaço de memória. Assim, um usuário pode utilizar uma funcionalidade do sistema enquanto outras linhas de execução estão trabalhando e realizando outros cálculos e operações. É como se um usuário virtual estivesse trabalhando de forma oculta no mesmo computador que você ao mesmo tempo.

### Multithread

*Multithreading* é a capacidade que o sistema operacional possui de executar vários threads simultaneamente sem que uma interfira na outra. Estes threads compartilham os recursos do processo, mas são capazes de ser executadas de forma independente. Para possuir processamento *multithread* “real”, os processadores precisam ser capazes de atender duas ou mais threads ao mesmo tempo e não simular este efeito, atendendo-as uma por vez em um curto período.

A maior vantagem trazida com a execução *multithread* é permitir que os computadores com múltiplos núcleos de processamento possam aproveitar todo o seu potencial e operar de forma mais rápida. Apenas para ilustrar, nos computadores sem este tipo de suporte, quando o processo principal toma muito tempo, a aplicação inteira parece travar.

Java oferece a classe *Thread* para que possamos trabalhar. Para criar um thread eu posso implementar a interface *Runnable*. Se a classe implementa a interface *Runnable*, o *thread* pode ser executado passando uma instância da classe para o construtor de um objeto *Thread* e, em seguida, chamando o método *start* do *thread*.

# RELATÓRIO COM AS LINHAS DE CÓDIGO

Classe do **Servidor**:

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

public class Server{

private ServerSocket serverSocket;

/\*\*

\* Receive the income communication

\* @param serverSocket socket object to communicate

\*/

public Server(ServerSocket serverSocket){

this.serverSocket = serverSocket;

}

/\*\*

\* Keeps the server running until the socket is closed

\*/

public void startServer(){

try{

// Isn't closed

// Wait a client to connect

while(!serverSocket.isClosed()){

Socket socket = serverSocket.accept();

System.out.println("A new inspector has connected!");

ClientHandler clientHandler = new ClientHandler(socket);

Thread thread = new Thread(clientHandler);

thread.start();

}

}catch(IOException error){

closeServerSocket();

}

}

/\*\*

\* Closes the server if an error occurs

\*/

public void closeServerSocket(){

try{

if(serverSocket != null){

serverSocket.close();

}

}catch(IOException error){

error.printStackTrace();

}

}

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

ServerSocket serverSocket = new ServerSocket(3000);

Server server = new Server(serverSocket);

server.startServer();

}

}

Classe ***ClientHandler***:

import java.io.\*;

import java.net.Socket;

import java.util.ArrayList;

public class ClientHandler implements Runnable{

// Group chat

public static ArrayList<ClientHandler> clientHandlers = new ArrayList<>();

// Establish a connection: Client -> Server

private Socket socket;

// Read and write messages

private BufferedReader bufferedReader;

private BufferedWriter bufferedWriter;

private String clientUserName;

public ClientHandler(Socket socket){

try{

this.socket = socket;

this.bufferedWriter = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));

this.bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

this.clientUserName = bufferedReader.readLine();

clientHandlers.add(this); // An instance of this class

broadcastMessage("SERVER: " + clientUserName + " has entered the chat!");

}catch(IOException error){

closeEverything(socket, bufferedReader, bufferedWriter);

}

}

@Override

public void run(){

String messageFromClient;

while(socket.isConnected()){

try{

// Each thread will hook here

messageFromClient = bufferedReader.readLine();

broadcastMessage(messageFromClient);

}catch(IOException error){

closeEverything(socket, bufferedReader, bufferedWriter);

break;

}

}

}

/\*\*

\* Broadcasts the message to every single client connected

\* @param message the message to send

\*/

public void broadcastMessage(String message){

for(ClientHandler clientHandler: clientHandlers){

try{

if(!clientHandler.clientUserName.equals(clientUserName)){

clientHandler.bufferedWriter.write(message);

clientHandler.bufferedWriter.newLine();

clientHandler.bufferedWriter.flush();

}

}catch(IOException error){

closeEverything(socket, bufferedReader, bufferedWriter);

}

}

}

/\*\*

\* Signal that the user has left the chat

\*/

public void removeClientHandler(){

clientHandlers.remove(this);

broadcastMessage("SERVER: " + clientUserName + " has left the chat!");

}

/\*\*

\* Closes down our connection and streams -- the user is leaving

\* @param socket connection to the server and client

\* @param bufferedReader output

\* @param bufferedWriter input

\*/

public void closeEverything(Socket socket, BufferedReader bufferedReader, BufferedWriter bufferedWriter){

removeClientHandler();

try{

if(bufferedReader != null){

bufferedReader.close();

}

if(bufferedWriter != null){

bufferedWriter.close();

}

if(socket != null){

socket.close();

}

}catch(IOException error){

error.printStackTrace();

}

}

}

Classe dos **clientes**:

import java.io.\*;

import java.net.Socket;

import java.util.Scanner;

public class Client{

private Socket socket;

private BufferedWriter bufferedWriter;

private BufferedReader bufferedReader;

private String username;

public Client(Socket socket, String username){

try{

this.socket = socket;

this.bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

this.bufferedWriter = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));

this.username = username;

}catch(IOException error){

closeEverything(socket, bufferedReader, bufferedWriter);

}

}

/\*\*

\* Sends messages to our ClientHandler

\*/

public void sendMessage(){

try{

bufferedWriter.write(username);

bufferedWriter.newLine();

bufferedWriter.flush();

Scanner scanner = new Scanner(System.in);

while(socket.isConnected()){

String message = scanner.nextLine();

bufferedWriter.write(username + ": " + message);

bufferedWriter.newLine();

bufferedWriter.flush();

}

}catch(IOException error){

closeEverything(socket, bufferedReader, bufferedWriter);

}

}

/\*\*

\* Listen to messages from the server

\*/

public void listenForMessage(){

new Thread(new Runnable(){

@Override

public void run(){

String messageFromGroupChat;

while(socket.isConnected()){

try{

messageFromGroupChat = bufferedReader.readLine();

System.out.println(messageFromGroupChat);

}catch(IOException error){

closeEverything(socket, bufferedReader, bufferedWriter);

}

}

}

}).start();

}

public void closeEverything(Socket socket, BufferedReader bufferedReader, BufferedWriter bufferedWriter){

try{

if(bufferedReader != null){

bufferedReader.close();

}

if(bufferedWriter != null){

bufferedWriter.close();

}

if(socket != null){

socket.close();

}

}catch(IOException error){

error.printStackTrace();

}

}

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

Scanner scanner = new Scanner(System.in);

System.out.println("Enter your name for the secretary group chat: ");

String username = scanner.nextLine();

Socket socket = new Socket("localhost", 3000);

Client client = new Client(socket, username);

// Separated thread. They'll be able to run at the same time

client.listenForMessage();

client.sendMessage();

}

}

# PROGRAMA EM FUNCIONAMENTO

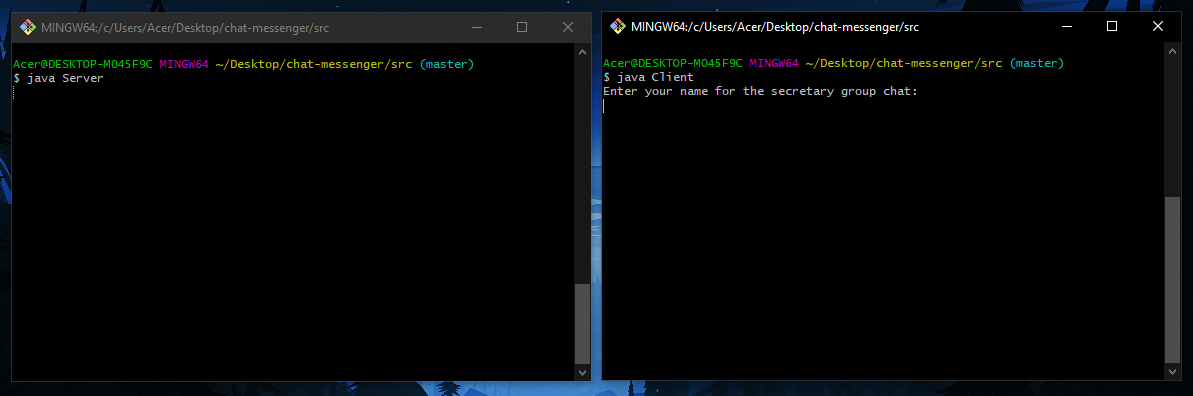


Imagem 5 – Servidor (esquerda) e cliente (direita) prestes a inserir o nome

Fonte: Própria

Imagem 6 – O cliente insere o nome e o servidor avisa que alguém se conectou

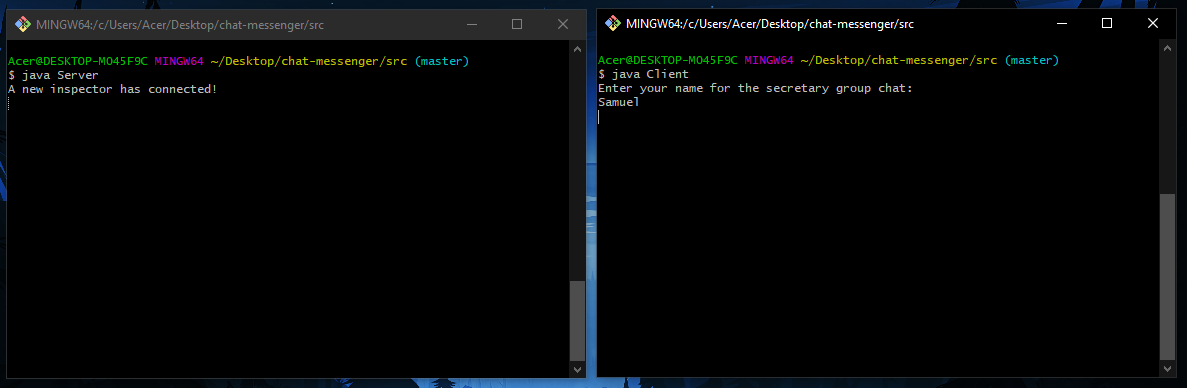
Fonte: Própria

Imagem 7 – Mais um cliente se conecta

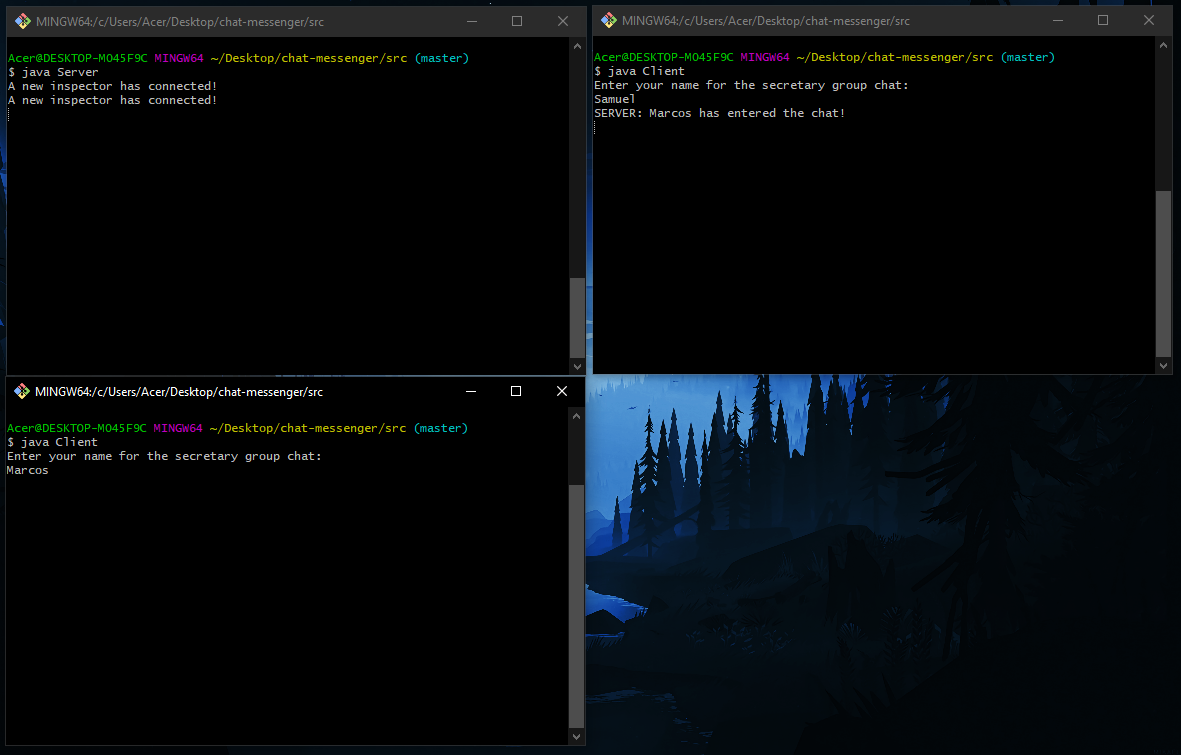
Fonte: Própria

Imagem 8 – Um cliente manda mensagem e ela é enviada para todos

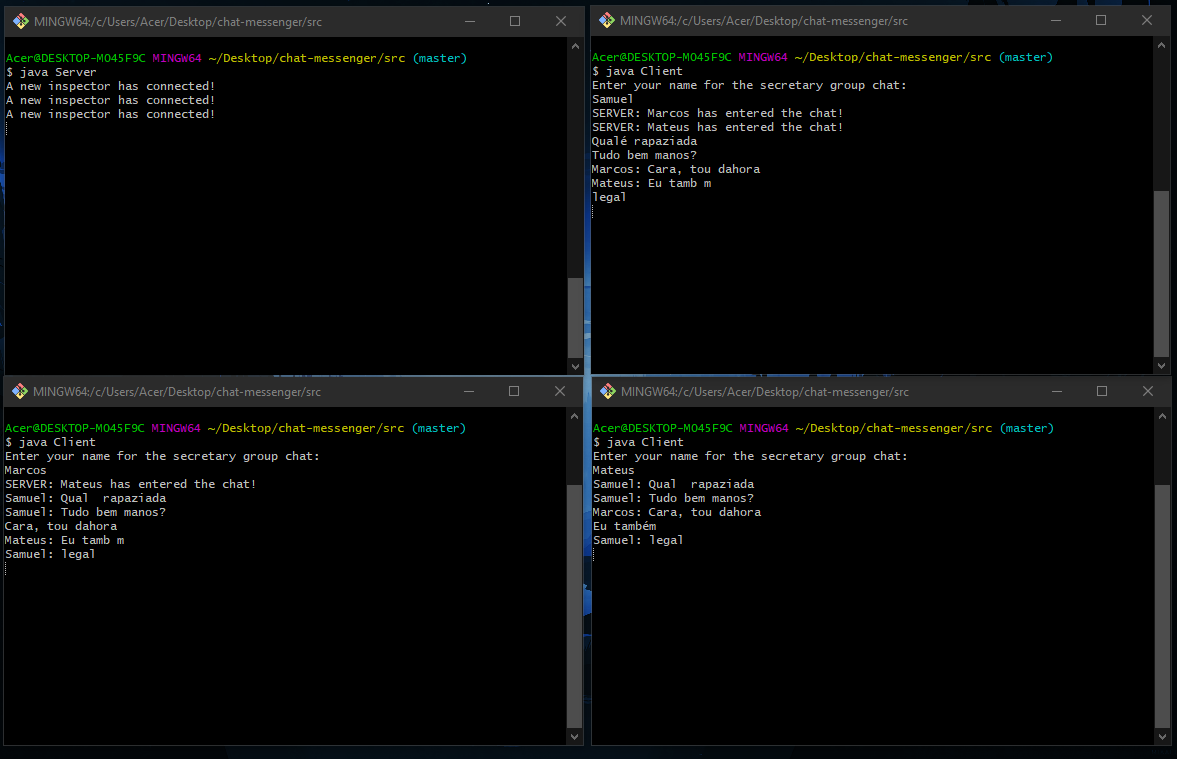
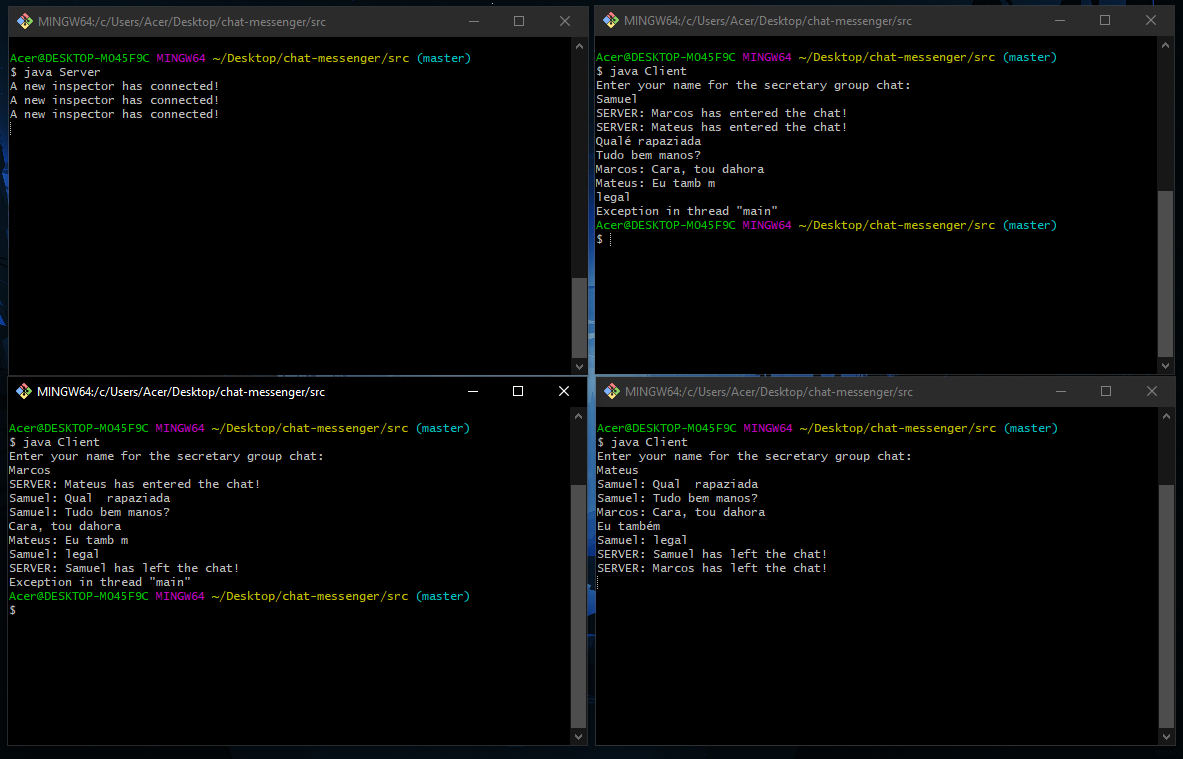
Fonte: Própria

Imagem 9 – Quando um usuário saí do chat, a aplicação avisa

Fonte: Própria

# Bibliografia

Baptista, C. M. (2012). *Rede de Computadores e Telecomunicações.* São Paulo: UNIP.

Cambridge University Press. (22 de Dezembro de 2008). *The Origins of Speech*. Fonte: Cambridge: https://www.cambridge.org/core/journals/cambridge-archaeological-journal/article/abs/origins-of-speech/A02A7642EFF9DC7B6BAFBC27D3CDE281

Dam, R. F. (03 de Abril de 2017). *Social Evolution and Why We Need to Communicate*. Fonte: Interaction Design Foundation: https://www.interaction-design.org/literature/article/social-evolution-and-why-we-need-to-communicate

Forouzan, B. A., & Oliveira, J. S. (2007). *Comunicação de Dados e Redes de Computadores* (4ª ed.). Porto Alegre: AMGH.

Garfinkel, S. L., & Grunspan, R. H. (2018). *The Computer Book.* New York: Sterling.

Java. (s.d.). *All About Sockets*. Fonte: Java Documentation: https://docs.oracle.com/javase/tutorial/networking/sockets/index.html

Kishimoto, M. (2007). *Naruto n° 5.* Barueri: Panini.

Sutherland, J. (2014). *Scrum: a arte de fazer o dobro do trabalho na metade do tempo.* São Paulo: Leya.

Tanenbaum, A. S. (2013). *Redes de Computadores* (5ª ed.). São Paulo: Pearson.

Universidade Federal de Minas Gerais (UFMG). (02 de Abril de 2020). *Uma breve história da escrita*. Fonte: Espaço do Conhecimento UFMG: https://www.ufmg.br/espacodoconhecimento/historia-escrita/#:~:text=Uma%20escrita%20sistematizada%20aparece%20somente,surgem%20os%20hier%C3%B3glifos%20no%20Egito.

**FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS (APS)**

**NOME:** Marcos Paulo Francisco Vaz— **RA:** F09BEG0 — **CURSO:** Ciência da Computação — **TURMA:** CC4P07

**CAMPUS:** Chácara Sto. Antônio II — **SEMESTRE:** 4º Semestre — **TURNO:** Noite — **TOTAL DE HORAS:** 105 horas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DATA** | **ATIVIDADE** | **TOTAL DE HORAS** | **ALUNO** | **PROFESSOR** |
| 25/abril | Reunião de Planejamento | 2 horas | Marcos |  |
| 25/abril | Semana 1: Estudar fundamentos de Java e OO | 8 horas | Marcos |  |
| 02/maio | Scrum meetings | 3 horas | Marcos |  |
| 02/maio | Semana 2: Estudar Threads em Java | 32 horas | Marcos |  |
| 09/maio | Scrum meetings | 3 horas | Marcos |  |
| 09/maio | Semana 3: Estudar Sockets e comunicação de aplicações | 32 horas | Marcos |  |
| 16/maio | Scrum meetings | 3 horas | Marcos |  |
| 16/maio | Semana 4: Desenvolver a aplicação e encaixar os módulos desenvolvidos nos estudos | 16 horas | Marcos |  |
| 23/maio | Semana 5: Escrever a documentação e a ficha de atividades | 6 horas | Marcos |  |

**FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS (APS)**

**NOME:** Matheus dos Santos Silva — **RA:** N590785 — **CURSO:** Ciência da Computação — **TURMA:** CC4P07

**CAMPUS:** Chácara Sto. Antônio II — **SEMESTRE:** 4º Semestre — **TURNO:** Noite — **TOTAL DE HORAS:** 105 horas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DATA** | **ATIVIDADE** | **TOTAL DE HORAS** | **ALUNO** | **PROFESSOR** |
| 25/abril | Reunião de Planejamento | 2 horas | Matheus dos Santos |  |
| 25/abril | Semana 1: Estudar fundamentos de Java e OO | 8 horas | Matheus dos Santos |  |
| 02/maio | Scrum meetings | 3 horas | Matheus dos Santos |  |
| 02/maio | Semana 2: Estudar Threads em Java | 32 horas | Matheus dos Santos |  |
| 09/maio | Scrum meetings | 3 horas | Matheus dos Santos |  |
| 09/maio | Semana 3: Estudar Sockets e comunicação de aplicações | 32 horas | Matheus dos Santos |  |
| 16/maio | Scrum meetings | 3 horas | Matheus dos Santos |  |
| 16/maio | Semana 4: Desenvolver a aplicação e encaixar os módulos desenvolvidos nos estudos | 16 horas | Matheus dos Santos |  |
| 23/maio | Semana 5: Escrever a documentação e a ficha de atividades | 6 horas | Matheus dos Santos |  |

**FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS (APS)**

**NOME:** Matheus Oliveira de Moraes— **RA:** N477GA2 — **CURSO:** Ciência da Computação — **TURMA:** CC5P07

**CAMPUS:** Chácara Sto. Antônio II — **SEMESTRE:** 5º Semestre — **TURNO:** Noite — **TOTAL DE HORAS:** 105 horas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DATA** | **ATIVIDADE** | **TOTAL DE HORAS** | **ALUNO** | **PROFESSOR** |
| 25/abril | Reunião de Planejamento | 2 horas | Matheus Oliveira |  |
| 25/abril | Semana 1: Estudar fundamentos de Java e OO | 8 horas | Matheus Oliveira |  |
| 02/maio | Scrum meetings | 3 horas | Matheus Oliveira |  |
| 02/maio | Semana 2: Estudar Threads em Java | 32 horas | Matheus Oliveira |  |
| 09/maio | Scrum meetings | 3 horas | Matheus Oliveira |  |
| 09/maio | Semana 3: Estudar Sockets e comunicação de aplicações | 32 horas | Matheus Oliveira |  |
| 16/maio | Scrum meetings | 3 horas | Matheus Oliveira |  |
| 16/maio | Semana 4: Desenvolver a aplicação e encaixar os módulos desenvolvidos nos estudos | 16 horas | Matheus Oliveira |  |
| 23/maio | Semana 5: Escrever a documentação e a ficha de atividades | 6 horas | Matheus Oliveira |  |

**FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS (APS)**

**NOME:** Samuel Araujo de Souza— **RA:** F30AJG4 — **CURSO:** Ciência da Computação — **TURMA:** CC5Q07

**CAMPUS:** Chácara Sto. Antônio II — **SEMESTRE: 5**º Semestre — **TURNO:** Noite — **TOTAL DE HORAS:** 105 horas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **DATA** | **ATIVIDADE** | **TOTAL DE HORAS** | **ALUNO** | **PROFESSOR** |
| 25/abril | Reunião de Planejamento | 2 horas | Samuel |  |
| 25/abril | Semana 1: Estudar fundamentos de Java e OO | 8 horas | Samuel |  |
| 02/maio | Scrum meetings | 3 horas | Samuel |  |
| 02/maio | Semana 2: Estudar Threads em Java | 32 horas | Samuel |  |
| 09/maio | Scrum meetings | 3 horas | Samuel |  |
| 09/maio | Semana 3: Estudar Sockets e comunicação de aplicações | 32 horas | Samuel |  |
| 16/maio | Scrum meetings | 3 horas | Samuel |  |
| 16/maio | Semana 4: Desenvolver a aplicação e encaixar os módulos desenvolvidos nos estudos | 16 horas | Samuel |  |
| 23/maio | Semana 5: Escrever a documentação e a ficha de atividades | 6 horas | Samuel |  |
| 25/abril | Reunião de Planejamento | 2 horas | Samuel |  |