

UNIVERSIDADE PAULISTA
BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

DIEGO FREIRE DE ALMEIDA, N8059D2

KAIKY DE LARA SALES, N9218H8

LEONARDO DE SOUZA RODRIGUES, F344HB2

NÍCOLAS PIMENTA DA SILVA, N863579

VITOR DOS SANTOS ROSA, G521CE3

APLICAÇÃO DE SOCKETS EM COMUNICAÇÃO EM TEMPO REAL

Jundiaí – SP

2024

DIEGO FREIRE DE ALMEIDA, N8059D2

KAIKY DE LARA SALES, N9218H8

LEONARDO DE SOUZA RODRIGUES, F344HB2

NÍCOLAS PIMENTA DA SILVA, N863579

VITOR DOS SANTOS ROSA, G521CE3

APLICAÇÃO DE SOCKETS EM COMUNICAÇÃO EM TEMPO REAL

Artigo das atividades práticas supervisionadas, apresentado ao Curso de Ciência da Computação para composição de nota.

Orientador: Prof. Rafael Gross

Jundiaí – SP

2024

ÍNDICE

1. OBJETIVO E MOTIVAÇÃO DO TRABALHO	4
2. INTRODUÇÃO	5
3. FUNDAMENTOS DA COMUNICAÇÃO DE DADOS EM REDE	6
3.1. Chats	7
3.1.1. A história.....	8
3.1.2. Chatbots	8
3.2. Fluxo de Dados em Rede	8
3.2.1. Transmission Control Protocol (TCP).....	9
3.2.2. User Datagram Protocol (UDP).....	10
3.2.3. Comunicação Via Sockets	11
4. PLANO DE DESENVOLVIMENTO DA APLICAÇÃO	13
4.1. A linguagem de programação C#	14
4.1.1. Java x C#.....	15
4.2. .NET MAUI	17
5. PROJETO DO PROGRAMA.....	19
6. RELATÓRIO COM AS LINHAS DE CÓDIGO.....	25
6.1. Estrutura do código.....	25
6.2. Servidor.....	26
6.3. Cliente	31
6.4. Interface	33
7. BIBLIOGRAFIA	35
8. FICHAS DE APS.....	37

1. OBJETIVO E MOTIVAÇÃO DO TRABALHO

Este artigo, visa propor e descrever uma solução, baseada na comunicação em rede, por meio de *sockets*, para uma situação, onde a Secretaria de Estado do Meio Ambiente quer identificar quais atividades industriais estão poluindo o Rio Tietê, desde sua nascente em Salesópolis (SP) até a sua passagem pela região da Grande São Paulo. Para isso, é necessário que as equipes de inspetores encarregados troquem informações, revezando-se em cada indústria, monitorando os processos e enviando dados online para a Secretaria.

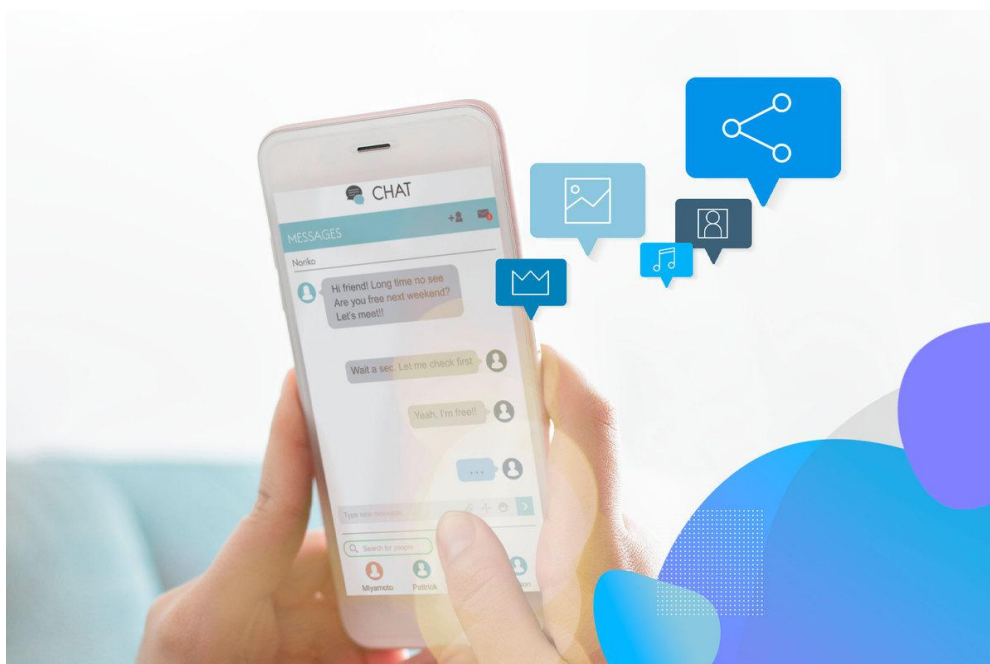
O projeto aqui explorado, terá como foco o desenlace para o problema de comunicação gerado pelo contexto, onde dois usuários realizarão a interação remota através de um *chat* de texto, podendo interagir com componentes gráficos, realizar envio de caracteres de ampla complexidade como emoticons, e efetuar transferência de arquivos. Ao final do artigo, espera-se que tenhamos contribuído para a construção de uma base sólida no que se refere a compreensão dos conceitos dos *sockets* de Berkeley e os principais elementos e primitivas que o cercam e como seu uso criativo pode ser utilizado para a solução de uma grande diversidade de problemas e demandas, como o apresentado neste trabalho, e muitos outros nas mais variadas esferas e naturezas.

2. INTRODUÇÃO

Quando pensamos sobre o assunto comunicação no ambiente virtual, nos vem a mente uma diversidade de conceitos imediatos e ferramentas das mais variadas utilidades das quais fazemos uso de maneira cotidiana: redes sociais, vídeo conferências, *webinars*, fóruns, *blogs* e muito mais.

No entanto, em todos esses ambientes, sem exceção, podemos notar a presença do mais versátil instrumento para estabelecimento de comunicação rápida entre usuários, o *chat*. Um recurso que permite o envio e de mensagens de texto de maneira direta de um usuário a outro.

Figura 1 - Chat mobile usual



Fonte: <<https://deskmanager.com.br/blog/chatbot-o-que-e/>>, 2024

Podendo ser *multicast* (interação em grupo de usuários) ou bilateral (entre dois usuários), o *chat* de texto permite interação simultânea e imediata dentro do contexto desejado, através da construção de frases e conjuntos de palavras sendo enviadas por parte de cada usuário dentro do *chat*, bem como o envio de arquivos das mais diversas extensões de arquivo e mensagens de áudio gravadas de imediato. É praticamente um emblema da geração atual que, de maneira corriqueira, mantém seus olhos focados no pequeno display do *smartphone* que possuem em mãos, seja utilizando o *Instagram*, o *Discord*, o *Facebook* ou até mesmo o *Microsoft Teams*, nota-se rapidamente o que todos possuem em comum: um versátil e icônico sistema de

chat, isso, quando marcadamente não são de forma objetiva um *chat* em si, como o *WhatsApp*, por exemplo.

Esse recurso, graças a sua usabilidade, pode ser muito bem utilizado para propósitos muito além do entretenimento, ele serve para sanar as demandas encontradas em nosso dia-a-dia, em um mundo cada vez mais globalizado e conectado, onde claramente notam-se demandas como: um analista de vendas de uma empresa chinesa precisando comunicar um potencial cliente aqui no Brasil sobre um novo produto que possui em mãos, um CEO que de um escritório de contabilidade necessitando comunicar seus funcionários sobre o aumento do PLR que acontecerá no mês seguinte, um professor realizando a devolutiva de uma prova na qual determinado aluno não desempenhou bem, e muito mais.

Fato é que a simples interação de usuários por intermédio de mensagens de texto em um *chat* imediatamente soluciona cada um desses casos, e para a proposta desse artigo, não seria diferente.

Aqui, teremos uma situação, onde a Secretaria de Estado do Meio Ambiente quer identificar quais atividades industriais estão poluindo o Rio Tietê, desde sua nascente em Salesópolis (SP) até a sua passagem pela região da Grande São Paulo. Para isso, os inspetores encarregados, trocarão informações, revezando-se em cada indústria, monitorando os processos e enviando dados online para a Secretaria, e o *chat* de texto adequado para isso, é sem dúvidas a ferramenta ideal.

Ao longo deste artigo, forneceremos uma visão abrangente e detalhada dos conceitos de chat, protocolo TCP e UDP e *websocket* equipando o leitor com o conhecimento necessário para entender os princípios subjacentes à comunicação de dados em rede e suas aplicações práticas no mundo real, particularmente, no contexto anteriormente apresentado.

3. FUNDAMENTO DE COMUNICAÇÃO EM REDE

A comunicação de dados em rede é um dos pilares fundamentais da era digital, permitindo a troca de informações entre dispositivos conectados. Neste momento, exploraremos os conceitos essenciais que sustentam essa área vital da tecnologia da informação.

Iniciaremos com uma definição clara do que são chats de texto, uma das formas mais populares de comunicação online que ilustram a aplicação prática dos conceitos de rede. Em seguida, abordaremos os protocolos de comunicação, destacando o UDP (*User Datagram Protocol*) e o TCP (*Transmission Control Protocol*). Esses protocolos são fundamentais para a transmissão de dados, cada um com suas características e aplicações específicas que influenciam o comportamento das redes. Finalmente, discutiremos os sockets de Berkeley, uma interface de programação de redes que permite a implementação de comunicação entre dispositivos.

3.1. Chats

O termo "*chat*" deriva do inglês, podendo ser traduzido como "bate-papo" ou "conversa". Embora seja um conceito de origem estrangeira, é amplamente utilizado em nosso idioma para se referir a uma ferramenta ou fórum que permite a comunicação escrita em tempo real pela Internet.

Exemplos: "Ontem à noite, estive até às três da madrugada no chat", "O futebolista participará de um chat e responderá a perguntas dos adeptos", "Conheci minha namorada em um chat", "Não aprecio o chat, prefiro as mensagens de correio eletrônico". Nos aplicativos de chat (conhecidos no Brasil também como "bate-papo"), é possível enviar mensagens de áudio, texto, imagens e vídeos. Além disso, alguns desses aplicativos permitem a realização de chamadas.

Geralmente, a noção de chat é utilizada para descrever a troca instantânea de mensagens escritas. Em outras palavras, quando um usuário escreve e envia uma mensagem, o destinatário a recebe imediatamente. O mesmo ocorre quando um usuário deixa sua mensagem em um fórum público.

É importante diferenciar entre o chat privado (realizado entre duas ou mais pessoas em um ambiente virtual com limites definidos pelos próprios participantes) e o chat público (onde todos que ingressam na sala ou fórum podem ler as mensagens).

Quando é possível ver e falar com a outra pessoa no chat, utiliza-se o termo *videochat* (ou videoconferência). Neste caso, os usuários podem trocar mensagens escritas, orais ou visuais, desde que disponham de um microfone e uma câmera (webcam).

A popularização dos chats ocorreu na década de 1990, permitindo que as pessoas se comunicassem em tempo real, eliminando a necessidade de enviar um e-mail e aguardar a resposta ou realizar chamadas telefônicas.

3.1.1. A história

O primeiro sistema similar a um chat foi um protocolo criado em 1988 pelo finlandês Jarkko Oikarinen, denominado IRC (Internet Relay Chat), que permitia a troca de mensagens em salas de bate-papo públicas ou privadas. O IRC foi, portanto, precursor dos chats modernos.

Na década de 1990, o sistema ganhou popularidade mundial, conectando pessoas de diversas partes do mundo em salas de chat com temas variados. Atualmente, com os avanços tecnológicos, existem chats automatizados, onde os usuários interagem com um robô que possui respostas prontas para diferentes tipos de perguntas. Esses são conhecidos como "*chatbots*".

3.1.2. Chatbots

O *chatbot*, trata-se de um programa de computador que utiliza inteligência artificial para desenvolver sua própria linguagem. Com o uso contínuo, as respostas automáticas geradas por este software tornam-se progressivamente mais sofisticadas, aproximando-se cada vez mais das conversas conduzidas por atendentes humanos, é amplamente utilizado por empresas que buscam oferecer atendimento instantâneo aos clientes. No caso de o *chatbot* não conseguir fornecer a resposta necessária, o atendimento é transferido para um humano.

3.2. Fluxo de Dados em Rede

A comunicação de dados envolve a transmissão de dados digitais entre dispositivos, facilitada por redes que podem variar de redes locais (LANs) de pequena escala a redes de longa distância (WANs). Essas redes empregam uma combinação de componentes de hardware - como roteadores, switches e mídias de transmissão (por

exemplo, cabos, fibras ópticas e sinais sem fio) - e protocolos de software para garantir a troca de dados precisa e eficiente.

Os protocolos são as regras que ditam como os dados são formatados, transmitidos e recebidos. Eles garantem que dispositivos, independentemente de suas diferenças, possam se comunicar efetivamente. Entre esses protocolos, o TCP e o UDP são essenciais devido aos seus papéis fundamentais no conjunto de protocolos da Internet (IP).

3.2.1 Transmission Control Protocol (TCP)

O TCP é um protocolo orientado à conexão, o que significa que requer uma conexão a ser estabelecida entre dispositivos de comunicação antes que a transmissão de dados comece. Essa conexão permanece ativa durante a sessão de comunicação e é encerrada quando a troca de dados é concluída. O TCP é projetado para fornecer entrega confiável, ordenada e verificada de uma sequência de *bytes* de um dispositivo para outro, garantindo que os dados cheguem íntegros e na sequência correta.

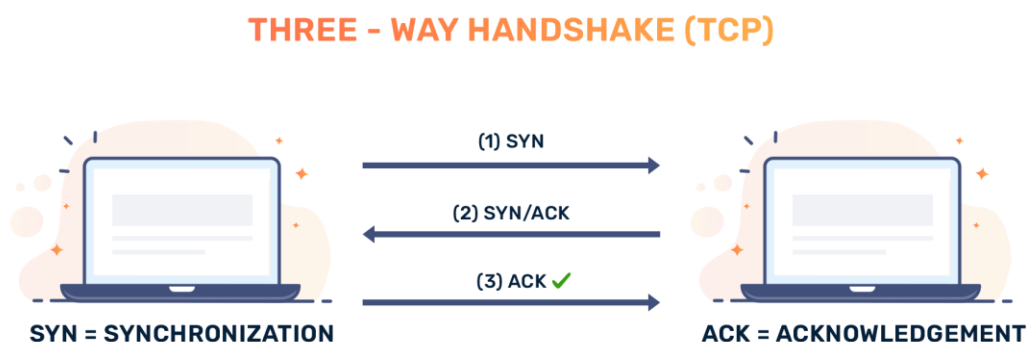
O TCP utiliza um *handshake* de três vias para estabelecer uma conexão. Esse processo envolve a troca de três pacotes: o remetente envia um pacote **SYN** (sincronização) ao receptor, o receptor responde com um pacote **SYN-ACK** (sincronização-reconhecimento) e o remetente então responde com um pacote **ACK** (reconhecimento). Esse *handshake* garante que ambas as partes estejam prontas para se comunicar e concordem com os números de sequência iniciais.

Durante a transmissão de dados, o TCP divide o fluxo de dados em segmentos menores, cada um com um número de sequência, permitindo que o receptor reúna os dados corretamente. Cada segmento é reconhecido pelo receptor ao chegar. Se um segmento não for reconhecido dentro de um determinado período, o TCP assume que foi perdido e o retransmite. O TCP inclui um *checksum* em cada cabeçalho de segmento para detectar erros durante a transmissão. Se o *checksum* não corresponder aos dados, o segmento é descartado e o remetente é notificado para retransmitir o segmento. A retransmissão é tratada por meio do mecanismo de reconhecimento, onde a falta de reconhecimento sinaliza a necessidade de retransmissão.

O TCP utiliza um mecanismo de controle de fluxo conhecido como protocolo de janela deslizante. Isso garante que o remetente não sobrecarregue o receptor com muitos dados de uma vez. O tamanho da janela, que pode ser ajustado dinamicamente, representa o número de bytes que o remetente pode transmitir sem receber um reconhecimento. O TCP incorpora algoritmos de controle de congestionamento (como *Slow Start*, *Congestion Avoidance*, *Fast Retransmit* e *Fast Recovery*) para gerenciar a congestão de rede e manter o fluxo de dados ideal. Esses algoritmos ajustam a taxa de transmissão com base nas condições da rede para evitar congestionamentos e perda de pacotes. O TCP encerra a conexão usando um processo de *handshake* de quatro vias: o remetente envia um pacote **FIN** (fim), o receptor reconhece com um **ACK**, o receptor envia seu próprio pacote **FIN** e o remetente responde com um **ACK**, completando o processo de terminação.

A confiabilidade e os robustos mecanismos de verificação de erros do TCP o tornam adequado para aplicativos onde a integridade dos dados é crítica. Estes incluem navegação na web, e-mail, transferência de arquivos e acesso remoto.

Figura 2 - Ilustração TCP



Fonte: <<https://bunny.net/academy/network/what-is-transmission-control-protocol-tcp/>>, 2024

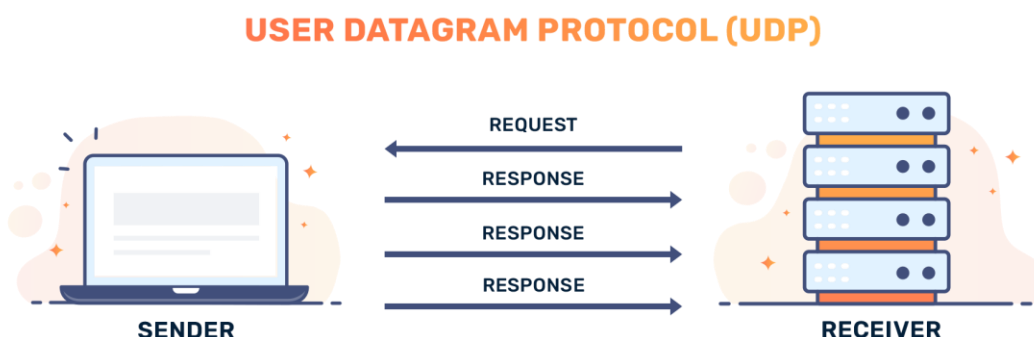
3.2.2. User Datagram Protocol (UDP)

O UDP é um protocolo que opera sem estabelecer uma conexão prévia entre os dispositivos de comunicação. Ele não garante a entrega confiável, a ordem ou a integridade dos dados, concentrando-se em vez disso na transmissão de baixa latência e no mínimo de *overhead*. Isso torna o UDP mais rápido, mas menos confiável em comparação com o TCP.

O UDP transmite dados na forma de datagramas, que são pacotes independentes que não requerem reconhecimento ou ordenação. Cada datagrama é autocontido, o que significa que a entrega de um datagrama não depende da entrega de outros. O UDP inclui um *checksum* para detecção básica de erros. Se ocorrer uma incompatibilidade de *checksum*, o datagrama corrompido é descartado sem nenhum mecanismo de retransmissão. O UDP não implementa mecanismos de controle de fluxo ou de controle de congestionamento, permitindo a transmissão potencialmente rápida de dados, mas também aumentando o risco de perda de pacotes em redes congestionadas.

A simplicidade e velocidade do UDP o tornam ideal para aplicativos onde a entrega oportuna é mais importante do que a confiabilidade. Casos de uso comuns incluem streaming de vídeo, jogos online, *Voice over IP* (VoIP) e comunicações de *broadcast* e *multicast*.

Figura 3 - Ilustração UDP



Fonte: <<https://bunny.net/academy/network/what-is-user-datagram-protocol-udp-and-how-does-it-work/>>, 2024

3.2.3. Comunicação Via Sockets

A comunicação via *socket* segue um modelo cliente-servidor, onde um socket age como servidor, aguardando conexões, e outro socket age como cliente, iniciando a conexão.

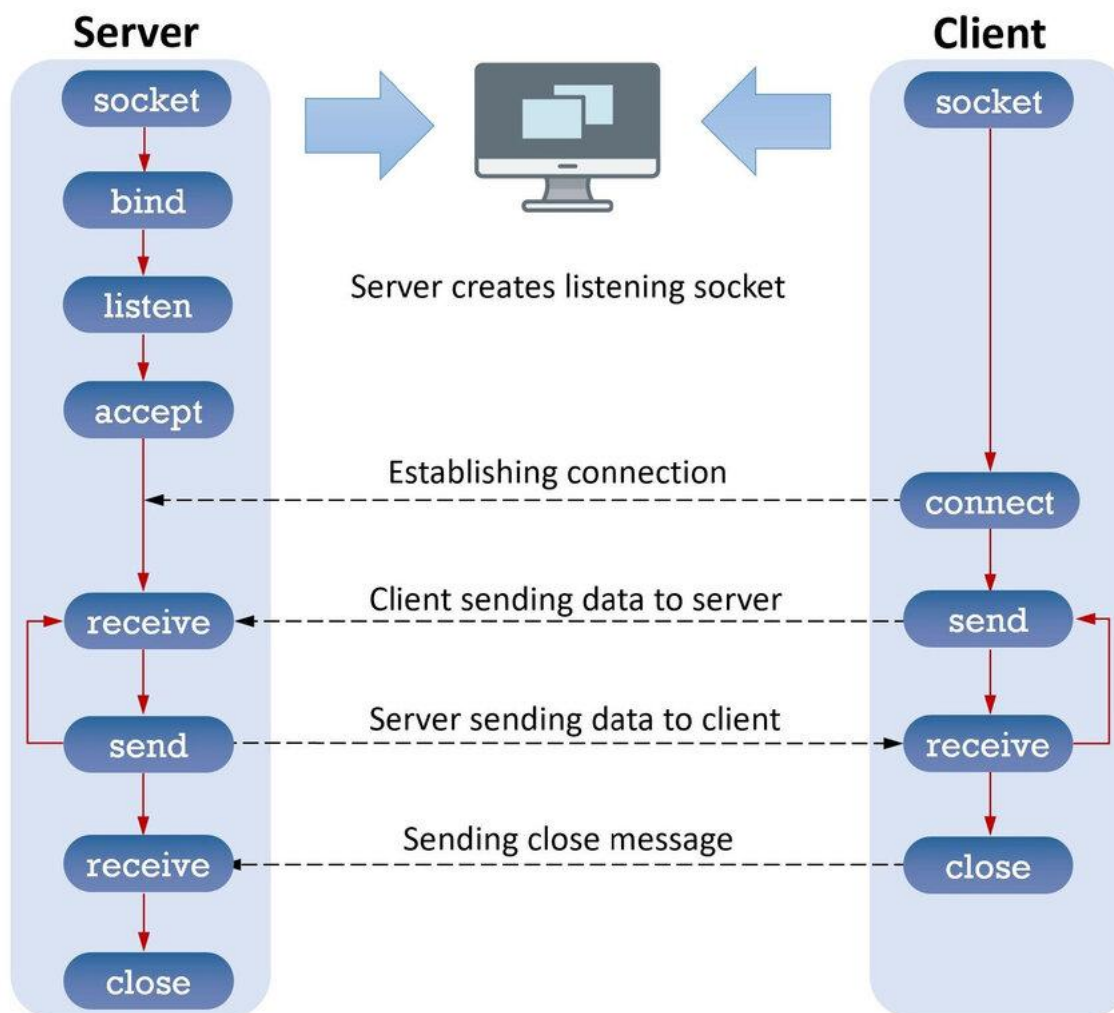
Um socket é uma extremidade de uma ligação de comunicação bidirecional entre dois programas em execução na rede. Ele está vinculado a um número de porta, o que permite que a camada TCP identifique a aplicação para a qual os dados estão sendo enviados.

Existem dois tipos principais de sockets usados na comunicação de rede: sockets de fluxo e sockets de datagrama.

Sockets de Fluxo (SOCK_STREAM) são usados para comunicação confiável e orientada à conexão, geralmente baseada em TCP, garantindo a entrega de dados na ordem correta.

Para estabelecer uma conexão, uma aplicação inicia uma conexão TCP de saída com outra aplicação que está ouvindo por conexões de entrada. As duas aplicações negociam um *handshake* de três vias.

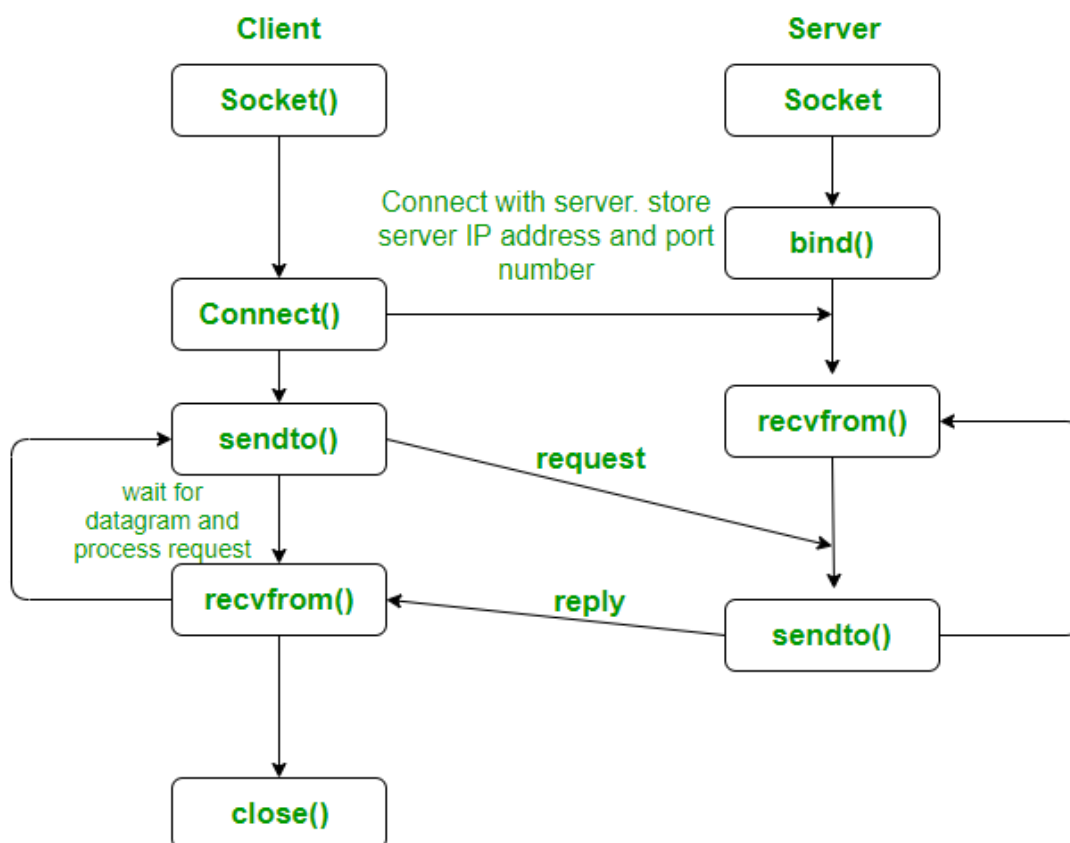
Figura 4 - Ilustração socket Server/Client



Fonte: <https://www.researchgate.net/figure/TCP-client-server-socket-flow_fig6_370038185>, 2024

Sockets de Datagrama (SOCK_DGRAM) são usados para comunicação não confiável e sem conexão, geralmente baseada em UDP, sem garantir entrega ou ordem.

Figura 5 - Diagrama socket



Fonte: <<https://www.geeksforgeeks.org/udp-client-server-using-connect-c-implementation/>>, 2024

4. PLANO DE DESENVOLVIMENTO DA APLICAÇÃO

Nesta seção, abordaremos o plano de desenvolvimento da aplicação no contexto prático, detalhando os elementos e ferramentas que foram utilizados ao longo do projeto. O desenvolvimento foi realizado utilizando a linguagem de programação C#, escolhida por sua robustez e ampla adoção na indústria, especialmente em sistemas de comunicação e aplicações empresariais. Além disso, empregaremos o framework .NET Multi-platform App UI (MAUI), que possibilita a criação de interfaces de usuário multiplataforma de maneira eficiente e coesa.

A escolha do C# se justifica pela sua sintaxe clara e recursos avançados, como a programação orientada a objetos, que facilitam a construção de aplicações complexas e escaláveis. O .NET MAUI, por sua vez, oferece um ambiente unificado para o desenvolvimento de aplicações que podem ser executadas em diferentes sistemas operacionais, como Windows, macOS, iOS e Android. Essa característica é particularmente vantajosa para o desenvolvimento de soluções de comunicação de dados, onde a interoperabilidade e a consistência da interface do usuário são cruciais.

Neste plano de desenvolvimento, descreveremos em detalhes as etapas e metodologias adotadas, incluindo a configuração do ambiente de desenvolvimento, bem como a integração de bibliotecas e ferramentas auxiliares. A seguir, apresentaremos uma visão abrangente dos componentes principais do projeto e as tecnologias que suportarão a implementação da solução proposta.

4.1. A linguagem de programação C#

A linguagem C# é a linguagem mais popular para a plataforma .NET, um ambiente de desenvolvimento gratuito, multiplataforma e de código aberto. Ela incorpora tipagem estática e forte, escopo lexical, além de paradigmas imperativo, declarativo, funcional, genérico, orientação a objetos baseada em classes e orientação a componentes.

Foi desenvolvida por Anders Hejlsberg na Microsoft em 2000 e padronizada pela Ecma e ISO/IEC em 2002 e 2003, respectivamente. Lançada com o .NET Framework e Visual Studio, inicialmente de código fechado, a linguagem ganhou suporte de código aberto com o projeto Mono em 2004. Em 2014, a Microsoft lançou o Visual Studio Code, Roslyn e a plataforma .NET unificada, todos gratuitos, de código aberto e multiplataforma.

Figura 6 - Trecho de código em C#

```
// Use a switch statement to do the math
switch (Console.ReadLine())
{
    case "a":
        Console.WriteLine($"Your result: {num1} + {num2} = " + (num1 + num2));
        break;
    case "s":
        Console.WriteLine($"Your result: {num1} - {num2} = " + (num1 - num2));
        break;
    case "m":
        Console.WriteLine($"Your result: {num1} * {num2} = " + (num1 * num2));
        break;
    case "d":
        // Ask the user to enter a non-zero divisor until they do so
        while (num2 == 0)
        {
            Console.WriteLine("Enter a non-zero divisor: ");
            num2 = Convert.ToInt32(Console.ReadLine());
        }
        Console.WriteLine($"Your result: {num1} / {num2} = " + (num1 / num2));
        break;
}
// Wait for the user to respond before closing
Console.Write("Press any key to close the Calculator console app...");
Console.ReadKey();
```

Fonte: <<https://learn.microsoft.com/pt-br/visualstudio/get-started/csharp/tutorial-console?view=vs-2022>>, 2024

O padrão Ecma formaliza o C# dentro de específicos parâmetros de Design, sendo estes: simplicidade, modernidade, orientação a objetos, internacionalidade, portabilidade e escalabilidade. Projetada para ser adequada para desenvolver aplicativos para sistemas hospedados e embarcados, abrangendo desde os muito grandes, que utilizam sistemas operacionais sofisticados, até os muito pequenos, com funções dedicadas, por exemplo, encontrou-se como a mais adequada opção de tecnologia de estruturação para o projeto em voga, principalmente se comparada a linguagem optativa levada em consideração no período de planejamento: o **Java**.

4.1.1. Java x C#

A principal distinção entre as linguagens de programação C# e Java reside no seu uso pretendido. Java é predominantemente utilizado no desenvolvimento de aplicativos móveis, especialmente no sistema Android. Em contraste, o C# é mais frequentemente empregado no desenvolvimento de aplicações web e embarcadas. Apesar dessas diferenças, é importante notar que ambas as linguagens apresentam

muitas similaridades e podem ser usadas no desenvolvimento web, embora cada uma tenha suas aplicações designadas específicas.

Além disso, é relevante destacar que, embora Java ofereça grande flexibilidade e suporte ao desenvolvimento multiplataforma, o C# não possui a mesma capacidade em termos de parâmetros para tal finalidade. Este é um aspecto crucial para programadores que valorizam a flexibilidade em seu trabalho diário.

Ao analisar as linguagens de programação Java e C# sob os aspectos de desempenho, escalabilidade e facilidade de uso, é possível identificar características específicas que influenciam sua adequação para diferentes tipos de projetos.

No que tange ao desempenho, tanto Java quanto C# oferecem ambientes de execução eficientes, embora com algumas diferenças. Java executa em cima da Java Virtual Machine (JVM), proporcionando uma grande portabilidade entre diferentes sistemas operacionais. A JVM é conhecida por sua capacidade de otimização e geração de código de máquina eficiente através do *Just-In-Time (JIT) compilation*. C#, por outro lado, é executado na *Common Language Runtime (CLR)* do .NET, que também utiliza técnicas de JIT para melhorar o desempenho. Em benchmarks comparativos, as diferenças de desempenho entre Java e C# são geralmente mínimas e dependem do tipo de aplicação e do contexto específico de uso. No entanto, é relevante notar que C# tende a se integrar de forma mais eficiente com o ecossistema Windows, enquanto Java possui uma vantagem em ambientes multiplataforma.

Em termos de escalabilidade, ambas as linguagens são robustas e capazes de suportar sistemas de grande escala. Java, com sua longa história no desenvolvimento de aplicações corporativas, oferece um conjunto abrangente de bibliotecas e frameworks (como Spring) que facilitam a criação de aplicativos escaláveis e distribuídos. C# também não fica atrás, especialmente com o .NET Core, que permite o desenvolvimento de aplicações *cross-platform* e escaláveis. Além disso, a arquitetura baseada em serviços e a integração com Azure e outros serviços de nuvem da Microsoft fortalecem a capacidade de C# para desenvolver soluções escaláveis.

A facilidade de uso é um fator crítico, especialmente para equipes que podem ter diferentes níveis de experiência. Java é amplamente reconhecida pela sua curva de aprendizado relativamente suave, além de uma vasta quantidade de documentação e uma comunidade ativa que oferece suporte. C# também é apreciada por sua sintaxe clara e recursos avançados que simplificam o desenvolvimento, como o *LINQ (Language Integrated Query)* e as capacidades de assíncronas e paralelismo.

Ambos os ambientes de desenvolvimento integrado (IDEs), como o IntelliJ IDEA para Java e o Visual Studio para C#, são bastante poderosos, mas o Visual Studio é muitas vezes elogiado por sua interface amigável e ferramentas de depuração avançadas.

Para o projeto específico de desenvolvimento de um chat de texto para a Secretaria de Estado do Meio Ambiente, onde é necessário identificar atividades industriais poluidoras no Rio Tietê e permitir a comunicação e troca de dados entre inspetores, ambas as linguagens poderiam ser adequadas. Entretanto, a escolha final pode depender de alguns fatores adicionais. Se a equipe de desenvolvimento já possui familiaridade com o ecossistema Java ou C#, isso pode influenciar a decisão. A interoperabilidade com sistemas legados e a infraestrutura existente também são aspectos cruciais a considerar.

Dada a necessidade de troca de informações em tempo real e envio de dados online, a integração com serviços de nuvem e a capacidade de criar aplicações robustas e escaláveis se tornam fundamentais. C# com .NET Core pode oferecer uma vantagem em termos de facilidade de desenvolvimento e integração com Azure, especialmente se a infraestrutura da Secretaria já utiliza soluções Microsoft. Contudo, se a portabilidade e a flexibilidade de rodar a aplicação em diferentes sistemas operacionais forem prioritárias, Java seria uma escolha sólida.

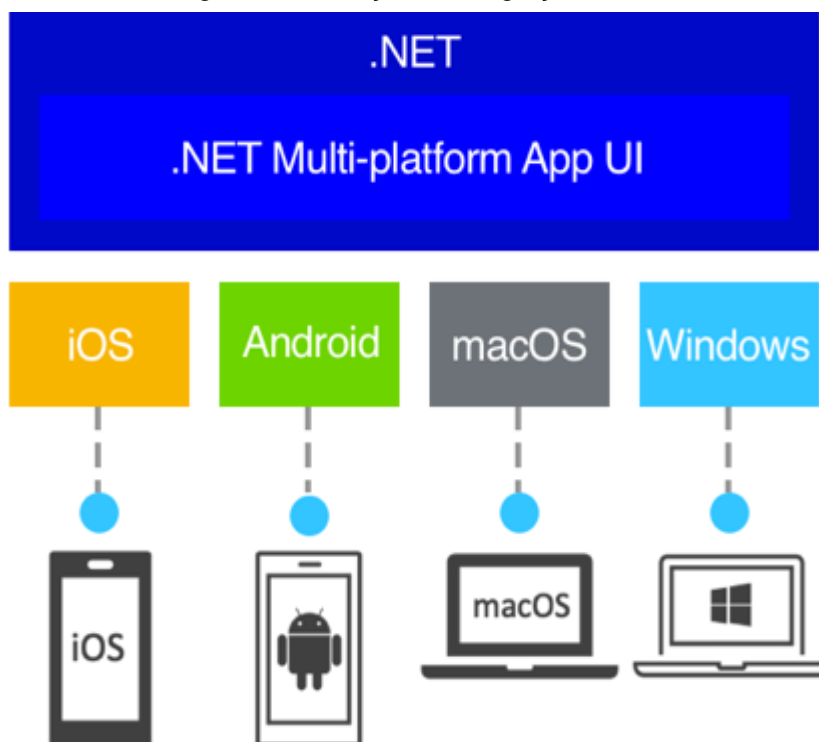
Concluindo, tanto Java quanto C# seriam opções viáveis para o desenvolvimento do chat de texto e utilizado pelos funcionários dentro do contexto. Se a portabilidade fosse uma necessidade, Java poderia ser preferível, no entanto, a integração com soluções Microsoft e uma experiência de desenvolvimento mais fluida foram cruciais para a escolha do C# com o .NET Core como nossa ferramenta para estruturação e escrita do código.

4.2. .NET MAUI, o framework

A interface do usuário da plataforma multiplataforma .NET (Microsoft UI para .NET, ou .NET MAUI) constitui um *framework* destinado ao desenvolvimento de aplicativos nativos para dispositivos móveis e computadores, utilizando as linguagens de programação C# e XAML.

Com a utilização do .NET MAUI, é possível criar aplicativos capazes de operar nos sistemas operacionais Android, iOS, macOS e Windows a partir de uma única base de código compartilhada.

Figura 7 - Ilustração da integração MAUI



Fonte: <<https://learn.microsoft.com/pt-br/dotnet/maui/what-is-maui?view=net-maui-8.0>>, 2024

Trata-se de uma evolução do *Xamarin.Forms*, incorporando diversas melhorias e funcionalidades adicionais, com o objetivo de simplificar e unificar o desenvolvimento de aplicações para Android, iOS, macOS, e Windows.

A principal finalidade do .NET MAUI é proporcionar aos desenvolvedores uma ferramenta robusta e eficiente para o desenvolvimento de aplicativos multiplataforma, eliminando a necessidade de escrever e manter código separado para cada sistema operacional. Através do uso de .NET MAUI, desenvolvedores podem criar interfaces de usuário, lógica de negócio, e funcionalidades específicas de plataforma em um único projeto, utilizando C# e *XAML (Extensible Application Markup Language)*.

Uma das características mais significativas do .NET MAUI é sua capacidade de renderizar interfaces de usuário nativas. Isso significa que os componentes visuais e controles utilizados no aplicativo são traduzidos para os equivalentes nativos de cada plataforma, garantindo uma experiência de usuário consistente e otimizada para cada sistema operacional. Além disso, o .NET MAUI oferece suporte a diversas

funcionalidades específicas de plataforma, permitindo o acesso a recursos de hardware e software nativos, como câmeras, sensores, e sistemas de notificação.

Em termos de arquitetura, o .NET MAUI adota uma abordagem modular, onde os desenvolvedores podem adicionar e configurar diferentes partes do aplicativo conforme necessário. Ele é construído sobre o .NET 6 e utiliza o projeto "Single Project", que unifica a estrutura de arquivos e configurações para todas as plataformas, facilitando o gerenciamento e a manutenção do código. A integração com o Visual Studio e outras ferramentas de desenvolvimento da Microsoft também é um ponto forte, oferecendo um ambiente de desenvolvimento integrado (IDE) completo com recursos como depuração, testes, e controle de versão.

O .NET MAUI pode ser utilizado em uma variedade de casos, incluindo, mas não se limitando a, desenvolvimento de aplicativos móveis, aplicações de desktop, e aplicações híbridas que requerem uma presença em múltiplas plataformas. Empresas e desenvolvedores independentes podem se beneficiar da eficiência de desenvolver uma única aplicação que funciona em diferentes sistemas operacionais, reduzindo significativamente o tempo e os recursos necessários para o desenvolvimento e manutenção.

Além disso, o .NET MAUI é particularmente útil em cenários onde a consistência da experiência do usuário e a manutenção de um design uniforme são cruciais. Por exemplo, empresas que desejam lançar um novo aplicativo ao mesmo tempo em várias plataformas podem utilizar o .NET MAUI para garantir que todas as versões do aplicativo tenham a mesma aparência e comportamento. Também é ideal para equipes de desenvolvimento que já estão familiarizadas com o ecossistema .NET e desejam expandir suas habilidades para o desenvolvimento multiplataforma sem a necessidade de aprender novas linguagens ou frameworks.

Neste projeto, foi essencial para a criação e modelação de todas as interfaces utilizadas para a interação do usuário com o chat de texto.

5. PROJETO DO PROGRAMA

Nesta seção, será detalhada a estrutura do programa em si, com ênfase na base de interface e nos módulos desenvolvidos. O objetivo é proporcionar uma compreensão clara e precisa das partes constitutivas do programa e de como elas interagem para cumprir os requisitos especificados.

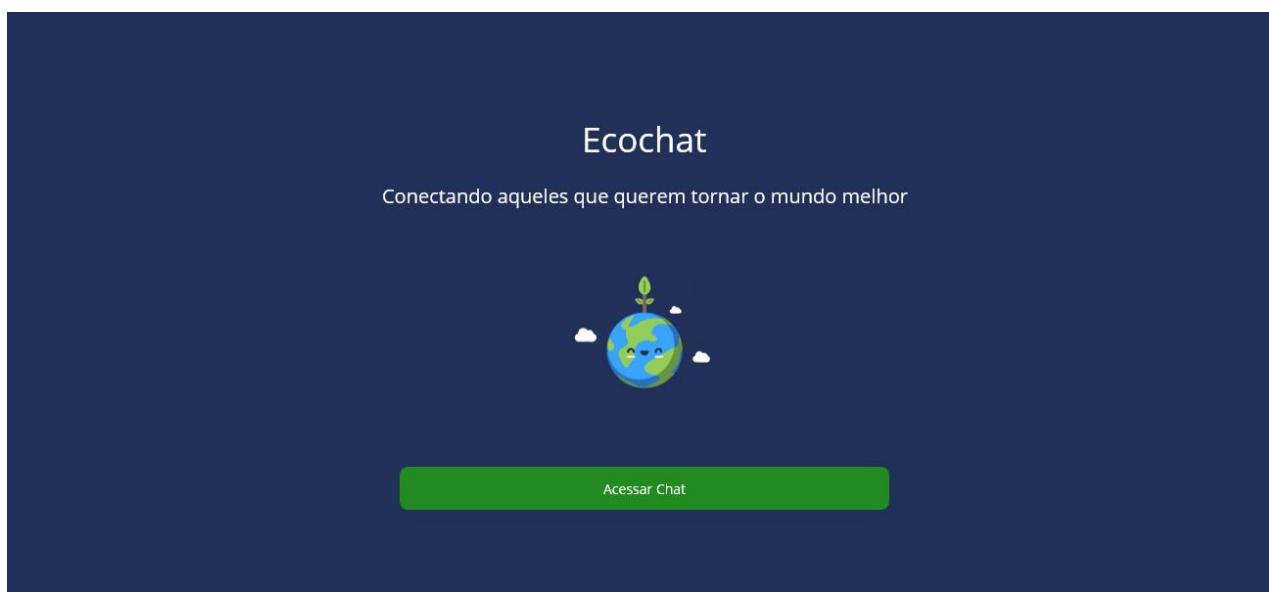
A apresentação começa com uma descrição da estrutura arquitetônica do programa, destacando a organização dos componentes e a lógica de interação entre eles. A discussão incluirá o modelo de arquitetura adotado, seja ele monolítico, em camadas ou baseado em micro serviços, juntamente com as justificativas para essa escolha.

Em seguida, será abordada a descrição detalhada dos módulos individuais que compõem o programa. Cada módulo será examinado quanto às suas responsabilidades específicas, interfaces e interdependências, destacando seu papel dentro do sistema, a lógica interna e os algoritmos implementados.

O fluxo de utilização do programa será então explicado, demonstrando como os usuários interagem com o sistema e como as funcionalidades principais são executadas. Serão apresentados cenários de uso típicos para ilustrar o funcionamento prático do programa, facilitando a compreensão das operações diárias e das interações dos usuários com o sistema. Esta visão geral integrará as informações discutidas anteriormente, reforçando a coesão e a lógica do projeto desenvolvido.

Ao abrir o programa, o usuário imediatamente se deparará com a tela de início (figura 8), com a apresentação do nome da aplicação: “Ecochat”.

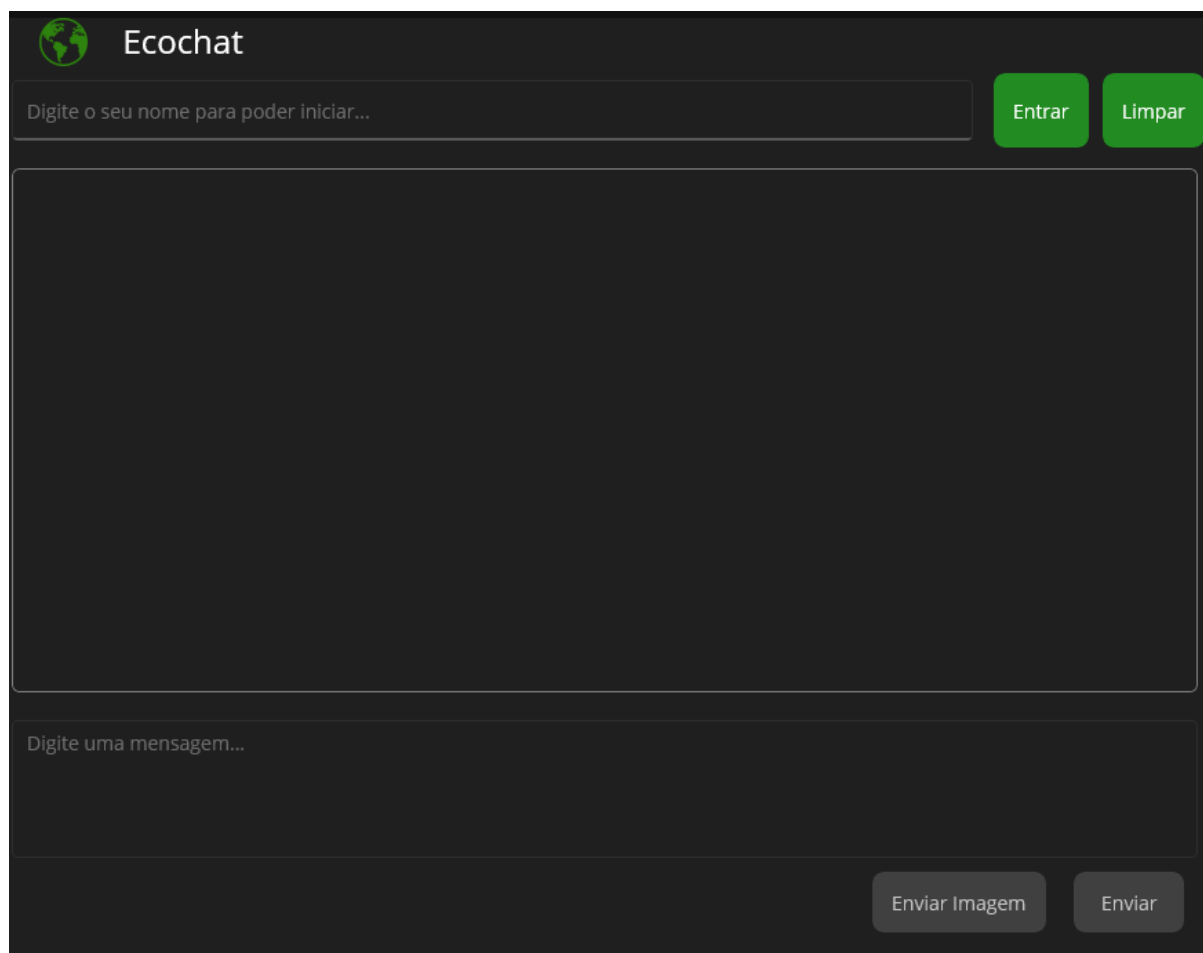
Figura 8 – Tela inicial do programa



Fonte: programa Ecochat, 2024

Ao pressionar o botão “Acessar Chat”, o usuário será então direcionado para a tela de login onde inserirá a descrição de qual equipe está realizando a utilização do chat na seção em questão (Equipe 1 e Equipe 2, por exemplo – Figuras 9, 10 e 11).

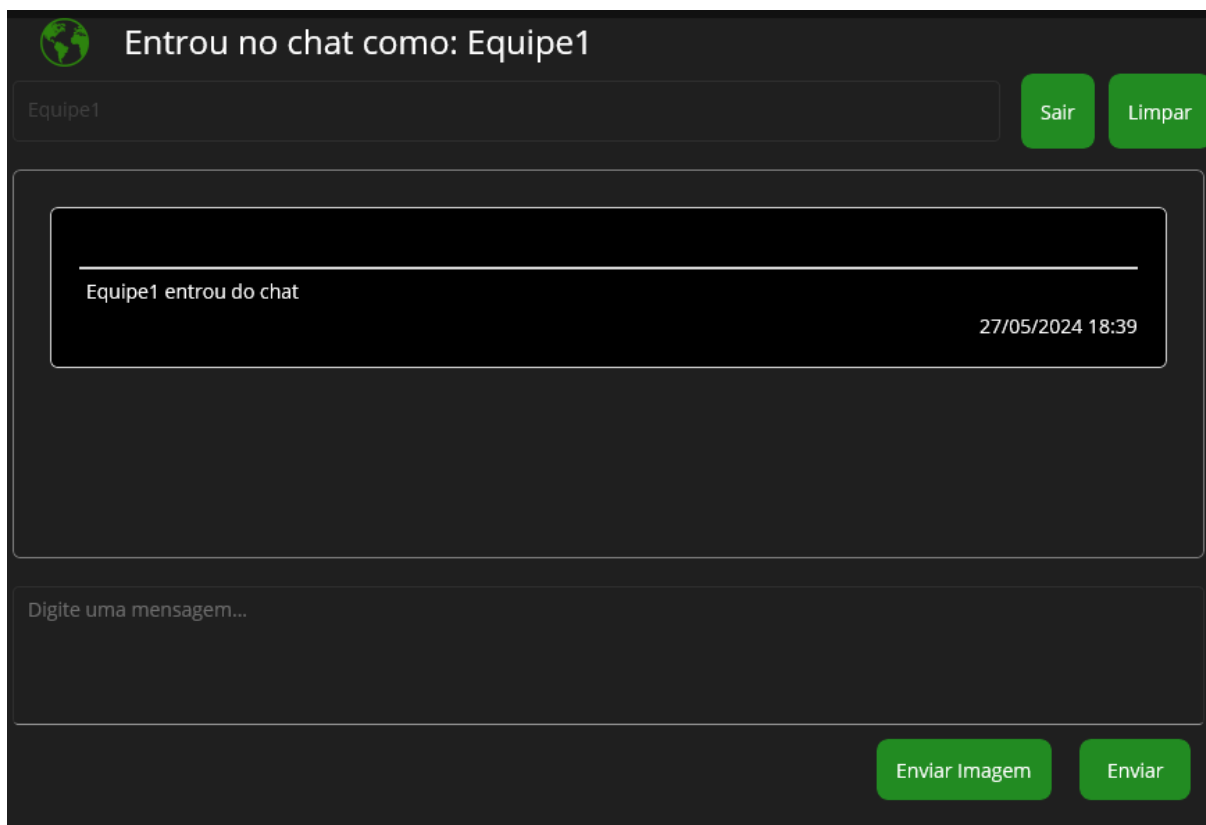
Figura 9 – Tela de login do Chat

A interface de login do aplicativo Ecochat, com fundo escuro. No topo, há um ícone de planeta verde e o nome "Ecochat". Abaixo, um campo de texto com o placeholder "Digite o seu nome para poder iniciar...". À direita deste campo estão dois botões verdes: "Entrar" e "Limpar". O corpo principal da tela é ocupado por uma grande área vazia, provavelmente para mensagens ou uma lista de equipes. Na base, há um campo de texto com o placeholder "Digite uma mensagem...". À direita deste campo estão dois botões cinza: "Enviar Imagem" e "Enviar".

Fonte: programa Ecochat, 2024

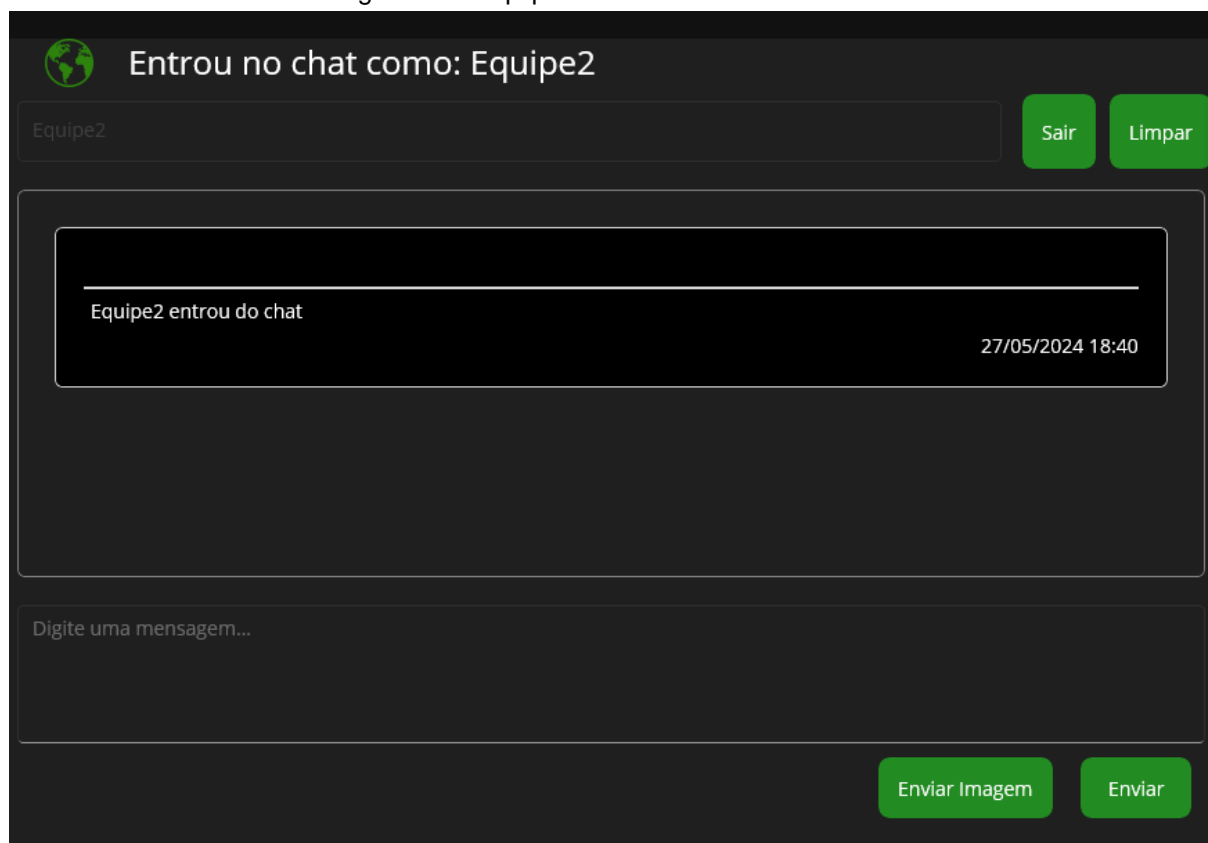
Nota-se que o chat apresenta também a data e horário exata do ingresso do usuário questão no chat, bem como de suas mensagens enviadas.

Figura 10 – Equipe 1 entrando no Chat



Fonte: programa Ecochat, 2024

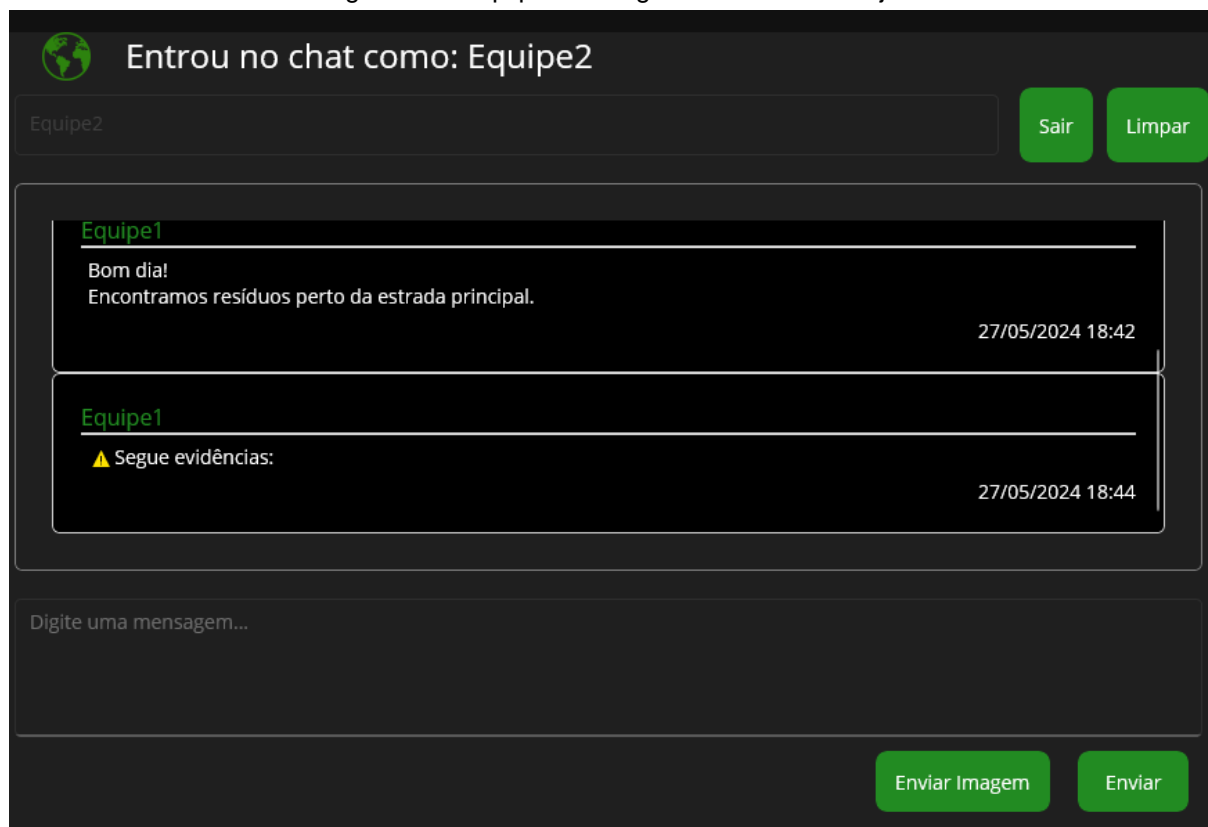
Figura 11 – Equipe 2 entrando no mesmo Chat



Fonte: programa Ecochat, 2024

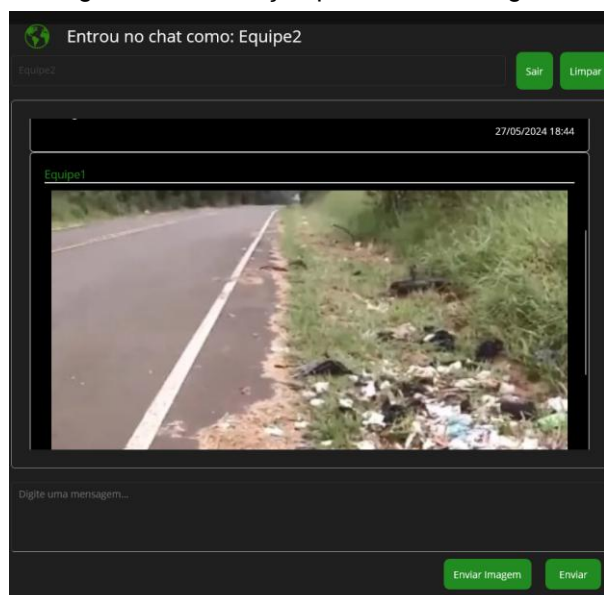
A partir daí, ambos os usuários estão aptos a realizar uma interação bilateral através do envio e recebimento de mensagens de texto e arquivos de imagem.

Figura 12 – Equipes interagindo utilizando emojis



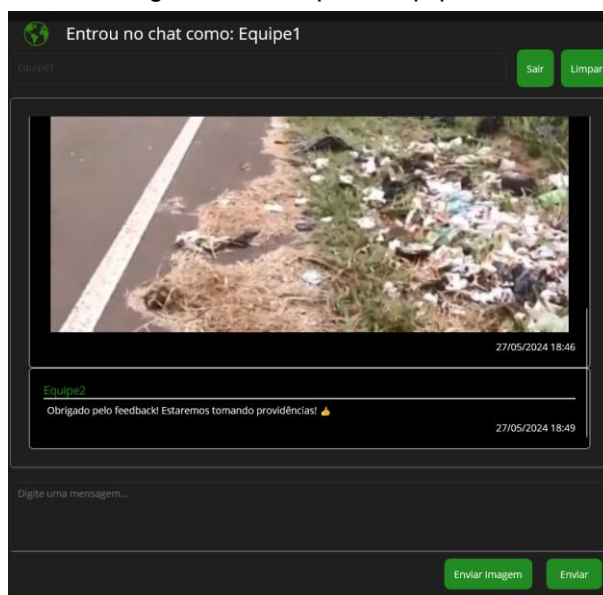
Fonte: programa Ecochat, 2024

Figura 13 – Interação por meio de imagens



Fonte: programa Ecochat, 2024

Figura 14 – Resposta equipe 2



Fonte: programa Ecochat, 2024

6. RELATÓRIO COM AS LINHAS DE CÓDIGO

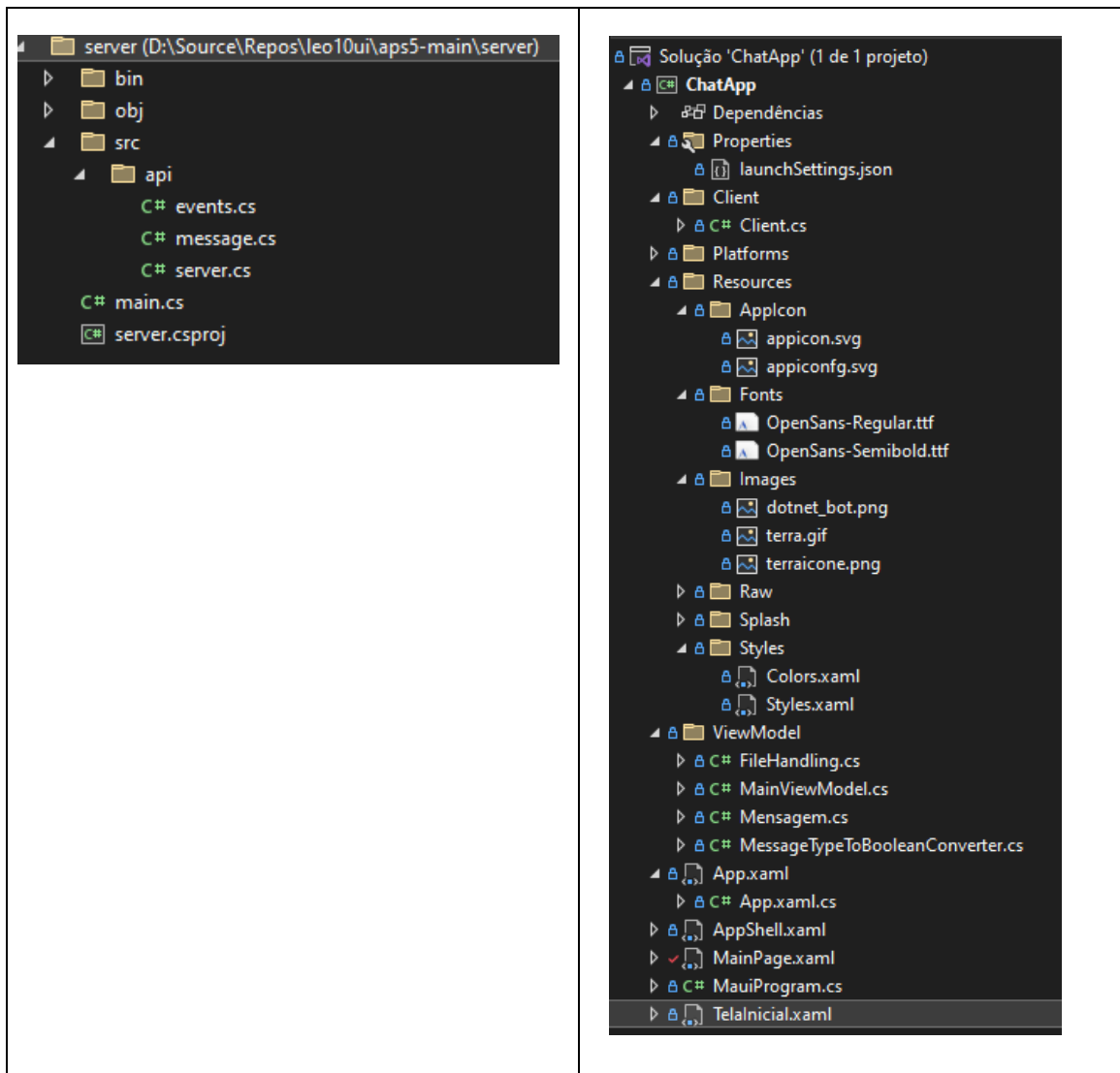
Nosso projeto é organizado em dois principais componentes:

- O servidor – Responsável pelo gerenciamento dos sockets e da propagação do conteúdo que está sendo enviado entre os nós.
- O cliente – Responsável pela interface do usuário, tornando a comunicação entre os usuários mais agradável.

Para o desenvolvimento destes componentes, foi utilizado a linguagem C#, usufruindo da plataforma .NET. Para o servidor, a plataforma já possui uma biblioteca própria para a implementação de comunicação via Sockets, e para o cliente utilizamos o framework .NET MAUI, que viabiliza a montagem de toda a interface gráfica utilizando arquivos XAML e arquivos .cs para o *Backend*.

6.1 Estrutura do código

Servidor	Cliente
----------	---------



6.2 Servidor

O servidor é a parte mais crucial do projeto, pois é aqui que o todo o fluxo de informações é orquestrado e enviado para cada nó dos sockets. A pasta src/api contém todos os principais componentes, sendo eles:

events.cs – instruções e rotinas para lidar com os eventos do servidor (Recebimento dos diferentes tipos de mensagens)

```
using System.Net.WebSockets;

public class EventsController
{
    private MessageController _messageController = new MessageController();

    public async Task HandleEvent(Websocket websocket, string message, List<Websocket>
sockets)
    {
```

```

var parts = message.Split(new char[] { '/' }, 2);
if (parts.Length != 2)
{
    Console.WriteLine($"Mensagem no formato errado: {message}");
    return;
}
var eventType = parts[0];
var eventData = parts[1];
switch (eventType)
{
    case "message":
        await _messageController.SendMessageEvent(webSocket, eventData, sockets);
        break;
    case "imagemessage":
        await _messageController.SendImageEvent(sockets, eventData, webSocket);
        break;
    case "documentmessage":
        await _messageController.SendDocumentEvent(sockets, eventData, webSocket);
        break;
    default:
        Console.WriteLine($"Evento desconhecido: {eventType}");
        break;
}
}
}

```

message.cs – Os eventos das diferentes mensagens que podemos receber dentro da aplicação.

```

using System.Net.WebSockets;
using System.Text;

public class MessageController
{
    public async Task SendMessageEvent(Websocket websocket, string message, List<Websocket> sockets)
    {
        //exemplo
        //"message/username$textodamensagem"
        var parts = message.Split(new char[] { '$' }, 2);
        if (parts.Length != 2)
        {
            Console.WriteLine($"Mensagem no formato errado: {message}");
            return;
        }
        var userName = parts[0];
    }
}

```

```

        var messageText = parts[1];

        messageText = $"message/{userName}${messageText}";
        await BroadcastMessageAsync(messageText, sockets, websocket);
    }

    public async Task SendImageEvent(List<WebSocket> sockets, string dataEvent, WebSocket
senderSocket)
    {
        // "documentmessage/nomedocara$base64"
        var partsEvent = dataEvent.Split(new char[] { '$' }, 2);
        var userName = partsEvent[0];
        var eventDataDoc = partsEvent[1];
        var messageText = $"imagemessage/{userName}${eventDataDoc}";
        await BroadcastMessageAsync(messageText, sockets, senderSocket);
    }

    public async Task SendDocumentEvent(List<WebSocket> sockets, string dataEvent,
WebSocket senderSocket)
    {
        //exemplo
        // "documentmessage/nomedocara$pdf!base64"
        var partsEvent = dataEvent.Split(new char[] { '$' }, 2);
        var userName = partsEvent[0];
        var eventDataDoc = partsEvent[1];
        var eventDataDocSplit = eventDataDoc.Split(new char[] { '!' }, 2);
        var documentType = eventDataDocSplit[0];
        var documentData = eventDataDocSplit[1];
        var messageText = $"documentmessage/{userName}${documentType}!{documentData}";
        await BroadcastMessageAsync(messageText, sockets, senderSocket);
    }

    private async Task BroadcastMessageAsync(string textMessage, List<WebSocket> sockets,
WebSocket senderSocket)
    {
        Console.WriteLine($"{{textMessage}} FINAL");
        byte[] messageBytes = Encoding.UTF8.GetBytes(textMessage);
        int bufferSize = 8192; // Tamanho do buffer para cada fragmento
        var tasks = new List<Task>();

        foreach (var socket in sockets)
        {
            if (socket != senderSocket && socket.State == WebSocketState.Open)
            {
                int offset = 0;

                while (offset < messageBytes.Length)
                {
                    int chunkSize = Math.Min(bufferSize, messageBytes.Length - offset);

```

```

        bool endOfMessage = (offset + chunkSize) == messageBytes.Length;

        var buffer = new ArraySegment<byte>(messageBytes, offset, chunkSize);
        tasks.Add(socket.SendAsync(buffer, WebSocketMessageType.Text, endOfMessage,
CancellationToken.None));

        offset += chunkSize;
    }
}
}
await Task.WhenAll(tasks);
}
}

```

server.cs – A principal classe, que implementa todo o gerenciamento dos sockets e as solicitações que poderão ser efetuadas pelos nós.

```

using System.Net;
using System.Net.WebSockets;
using System.Text;

public class WebSocketServer
{
    private readonly HttpListener _listener;
    public List<WebSocket> ConnectedSockets { get; } = new List<WebSocket>();
    private readonly EventsController eventsController = new EventsController();
    public WebSocketServer(string ipAddress, int port)
    {
        _listener = new HttpListener();
        _listener.Prefixes.Add($"http://{ipAddress}:{port}/");
    }
    public async Task Start()
    {
        _listener.Start();
        Console.WriteLine("Servidor Iniciado.");

        while (true)
        {
            var context = await _listener.GetContextAsync();
            if (context.Request.IsWebSocketRequest)
            {
                ProcessWebSocketRequest(context);
            }
            else
            {
                context.Response.StatusCode = 400;
                context.Response.Close();
            }
        }
    }
}

```

```

    }
}
private async void ProcessWebSocketRequest(HttpListenerContext context)
{
    var websocketContext = await context.AcceptWebSocketAsync(null);
    var websocket = websocketContext.WebSocket;

    try
    {
        ConnectedSockets.Add(websocket);
        await HandleWebSocketConnection(websocket);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Erro: {ex.Message}");
    }
}
private async Task HandleWebSocketConnection(WebSocket websocket)
{
    var buffer = new byte[1024 * 512];
    var messageBuilder = new StringBuilder();

    while (websocket.State == WebSocketState.Open)
    {
        WebSocketReceiveResult result;
        do
        {
            result = await websocket.ReceiveAsync(new ArraySegment<byte>(buffer),
CancellationToken.None);
            var messageChunk = Encoding.UTF8.GetString(buffer, 0, result.Count);
            messageBuilder.Append(messageChunk);
        }
        while (!result.EndOfMessage);

        if (result.MessageType == WebSocketMessageType.Text)
        {
            var message = messageBuilder.ToString();
            messageBuilder.Clear(); // Clear the StringBuilder for the next message
            // Console.WriteLine($"Evento Recebido:
{message}");

            await eventsController.HandleEvent(websocket, message, ConnectedSockets);
        }
        else if (result.MessageType == WebSocketMessageType.Close)
        {
            await websocket.CloseAsync(WebSocketCloseStatus.NormalClosure,
string.Empty, CancellationToken.None);
            ConnectedSockets.Remove(websocket);
        }
    }
}

```

```

        Array.Clear(buffer, 0, buffer.Length);
    }
}
}

```

6.3 Cliente

Em relação a comunicação em via sockets, podemos destacar a classe **Client.cs**, no qual manipula os métodos inerentes a comunicação com o servidor e consequente com os outros nós.

```

using System.Collections.ObjectModel;
using System.Diagnostics;
using System.Net.WebSockets;
using System.Text;
using ChatApp.ViewModel;
using CommunityToolkit.Mvvm.ComponentModel;

namespace ChatApp;
public partial class WebSocketClient : ObservableObject
{
    private ClientWebSocket _clientWebSocket;
    public WebSocketClient(){}
    public async Task Connect(string uri)
    {
        _clientWebSocket = new ClientWebSocket();
        await _clientWebSocket.ConnectAsync(new Uri(uri), CancellationToken.None);
        Console.WriteLine("Connected to WebSocket server.");
    }

    public async Task SendEvent(string eventType, string eventData)
    {
        var message = $"{eventType}/{eventData}";
        await SendMessage(message);
    }

    private async Task SendMessage(string message)
    {
        byte[] messageBytes = Encoding.UTF8.GetBytes(message);
        int bufferSize = 8192; // Tamanho do buffer que você deseja usar para fragmentos
        int offset = 0;

        while (offset < messageBytes.Length)
        {
            int chunkSize = Math.Min(bufferSize, messageBytes.Length - offset);
            bool endOfMessage = (offset + chunkSize) == messageBytes.Length;

            var buffer = new ArraySegment<byte>(messageBytes, offset, chunkSize);

```

```

        await _clientWebSocket.SendAsync(buffer, WebSocketMessageType.Text,
endOfMessage, CancellationToken.None);

        offset += chunkSize;
    }

    Console.WriteLine($"Sent message: {message}");
}

public async Task Receive(ObservableCollection<Mensagem> mensagens)
{
    var buffer = new byte[1024 * 512];
    var messageBuilder = new StringBuilder();

    while (_clientWebSocket.State == WebSocketState.Open)
    {
        WebSocketReceiveResult result;
        do
        {
            result = await _clientWebSocket.ReceiveAsync(new
ArraySegment<byte>(buffer), CancellationToken.None);
            var messageChunk = Encoding.UTF8.GetString(buffer, 0, result.Count);
            messageBuilder.Append(messageChunk);
        }
        while (!result.EndOfMessage);

        var receivedMessage = messageBuilder.ToString();
        messageBuilder.Clear();
        Console.WriteLine($"Received message: {receivedMessage}");
        if (!string.IsNullOrEmpty(receivedMessage))
        {
            string[] msgTratada = receivedMessage.Split(new char[] { '/' }, 2);
            string tipoEvento = msgTratada[0];
            string[] dados = msgTratada[1].Split(new char[] { '$' }, 2);
            switch (tipoEvento)
            {
                case "message":
                    var msgTexto = new Mensagem
                    {
                        Conteudo = dados[1],
                        Timestamp = DateTime.Now,
                        Emissor = dados[0]
                    };
                    mensagens.Add(msgTexto);
                    break;
                case "imagemessage":
                    string documentsPath =
Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
                    FileHandling.DecodeBase64ToFile(dados[1], documentsPath, "jpg");

```



```

        if (!string.IsNullOrEmpty(documentsPath))
        {
            var msgImagem = new Mensagem
            {
                Emissor = dados[0],
                FilePath = documentsPath + ".jpg",
                Timestamp = DateTime.Now,
                Tipo = MensagemTipo.Imagem
            };
            mensagens.Add(msgImagem);
        }
        break;
    default:
        break;
    }
}
}
}

public async Task Disconnect()
{
    await _clientWebSocket.CloseAsync(WebSocketCloseStatus.NormalClosure, "Client
disconnected.", CancellationToken.None);
    _clientWebSocket.Abort();
    Console.WriteLine("Disconnected from WebSocket server.");
}
}
}

```

6.4 Interface

O framework .NET MAUI funciona através de pares entre documentos .xaml e .cs, no qual são respectivamente os componentes, e o backend que relaciona a função destes componentes. É um framework que utiliza a arquitetura MVVM (Model-View-View-Model), no qual utiliza-se uma classe que integra todas as principais funções visando a facilidade de manutenção e controle do código.

TelaInicial.xaml

```

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="ChatApp.TelaInicial"
    xmlns:viewmodel="clr-namespace:ChatApp.ViewModel"
    BackgroundColor="#21315A"
    x:DataType="viewmodel:MainViewModel"
    xmlns:local="clr-namespace:ChatApp">
    <VerticalStackLayout VerticalOptions="Center">
        <Label

```

```

        Text="Ecochat"
        Margin="10"
        VerticalOptions="Center"
        HorizontalOptions="Center"
        FontSize="35"/>
    <Label
        Text="Conectando aqueles que querem tornar o mundo melhor"
        FontSize="20"
        VerticalOptions="Center"
        HorizontalOptions="Center"
        Margin="10"/>
    <Image Source="terra.gif"
        Aspect="AspectFit"
        WidthRequest="250"
        HeightRequest="250"/>
    <Button Pressed="NavegarChat" Text="Acessar Chat" Margin="3" WidthRequest="500"/>
</VerticalStackLayout>
</ContentPage>

```

TelaInicial.xaml.cs

```

using ChatApp.ViewModel;
namespace ChatApp
{
    public partial class TelaInicial : ContentPage
    {
        public TelaInicial()
        {
            InitializeComponent();
        }
        async void NavegarChat(object sender, EventArgs args)
        {
            await Navigation.PushAsync(new AppShell());
        }
    }
}

```

7. BIBLIOGRAFIA

CONCEITO. Chat. Disponível em: <<https://conceito.de/chat>>. Acesso em: 25 maio 2024.

DESKMANAGER. Chatbot: o que é? Disponível em: <<https://deskmanager.com.br/blog/chatbot-o-que-e/>>. Acesso em: 25 maio 2024.

WALLARM. Transmission Control Protocol (TCP). Disponível em: <<https://www.wallarm.com/what/transmission-control-protocol-tcp>>. Acesso em: 25 maio 2024.

BUNNY.NET. What is Transmission Control Protocol (TCP)? Disponível em: <<https://bunny.net/academy/network/what-is-transmission-control-protocol-tcp/>>. Acesso em: 25 maio 2024.

BUNNY.NET. What is User Datagram Protocol (UDP) and how does it work? Disponível em: <<https://bunny.net/academy/network/what-is-user-datagram-protocol-udp-and-how-does-it-work/>>. Acesso em: 25 maio 2024.

MICROCHIP. Berkeley Sockets. Disponível em: <<https://developerhelp.microchip.com/xwiki/bin/view/applications/tcp-ip/sockets-ports/#HBerkeleySockets>>. Acesso em: 25 maio 2024.

ResearchGate. TCP client-server socket flow. Disponível em: <https://www.researchgate.net/figure/TCP-client-server-socket-flow_fig6_370038185>. Acesso em: 25 maio 2024.

GEEKSFORGEEKS. UDP client-server using connect: C++ implementation. Disponível em: <<https://www.geeksforgeeks.org/udp-client-server-using-connect-c-implementation/>>. Acesso em: 25 maio 2024.

MICROSOFT. Tutorial: create a new app with Visual C#. Disponível em: <<https://learn.microsoft.com/pt-br/visualstudio/get-started/csharp/tutorial-console?view=vs-2022>>. Acesso em: 26 maio 2024.

MICROSOFT. C# Guide. Disponível em: <<https://learn.microsoft.com/pt-br/dotnet/csharp/>>. Acesso em: 27 maio 2024.

WIKIPEDIA. C Sharp (programming language). Disponível em: <[https://en.wikipedia.org/wiki/C_Sharp_\(programming_language\)](https://en.wikipedia.org/wiki/C_Sharp_(programming_language))>. Acesso em: 27 maio 2024.

BITDEGREE. Java ou C#: qual é melhor aprender? Disponível em: <<https://br.bitdegree.org/tutoriais/java-ou-c-sharp>>. Acesso em: 27 maio 2024.

MICROSOFT. What is .NET MAUI? Disponível em: <<https://learn.microsoft.com/pt-br/dotnet/maui/what-is-maui?view=net-maui-8.0>>. Acesso em: 27 maio 2024.

MICROSOFT. .NET MAUI documentation. Disponível em: <https://learn.microsoft.com/pt-br/dotnet/maui/?view=net-maui-8.0&WT.mc_id=dotnet-35129-website>. Acesso em: 27 maio 2024.

8. FICHAS DE APS

FICHA ATIVIDADES PRÁTICAS SUPERVISIONADAS – APS

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

NOME: Diego Freire de Almeida

RA: N8059D-2

CURSO: Ciência da Computação

CAMPUS: Jundiaí

SEMESTRE: 5º Semestre

TURNO: Noturno

DATA	ATIVIDADE	TOTAL DE HORAS	ASSINATURA	
			ASSINATURA DO ALUNO	ASSINATURA DO PROFESSOR
08/05/2023	Discussão inicial no <u>Discord</u> para definição sobre o tema do trabalho	5		
09/05/2023	Pesquisa sobre o tema	3		
11/05/2023	Pesquisa sobre o tema	5		
13/05/2023	Pesquisa sobre o tema	4		
13/05/2023	Reunião do grupo para a definição do projeto do software	4		
15/05/2023	Elaboração do conteúdo do trabalho escrito	8		
16/05/2023	Elaboração do conteúdo do trabalho escrito	8		
17/05/2023	Pesquisa sobre a utilização do recurso MAUI (C#)	4		
16/05/2023	Desenvolvimento do Software	4		
17/05/2023	Desenvolvimento do Software	6		
18/05/2023	Desenvolvimento do Software	5		
19/05/2023	Desenvolvimento do Software	4		
20/05/2023	Reunião do grupo no laboratório de informática	2		
24/05/2023	Desenvolvimento do Software	5		
25/05/2023	Desenvolvimento do Software	5		
26/05/2023	Desenvolvimento do Software	3		
26/05/2023	Finalização e formatação do trabalho em norma ABNT	3		

TOTAL DE HORAS: 78 Horas

FICHA ATIVIDADES PRÁTICAS SUPERVISIONADAS – APS

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

NOME: Kaiiky de Lara Sales

RA: N9218H-8

CURSO: Ciência da Computação

CAMPUS: Jundiaí

SEMESTRE: 5º Semestre

TURNO: Noturno

DATA	ATIVIDADE	TOTAL DE HORAS	ASSINATURA	
			ASSINATURA DO ALUNO	ASSINATURA DO PROFESSOR
08/05/2023	Discussão inicial no <u>Discord</u> para definição sobre o tema do trabalho	5		
09/05/2023	Pesquisa sobre o tema	3		
11/05/2023	Pesquisa sobre o tema	5		
13/05/2023	Pesquisa sobre o tema	4		
13/05/2023	Reunião do grupo para a definição do projeto do software	4		
15/05/2023	Elaboração do conteúdo do trabalho escrito	8		
16/05/2023	Elaboração do conteúdo do trabalho escrito	8		
17/05/2023	Pesquisa sobre a utilização do recurso MAUI (C#)	4		
16/05/2023	Desenvolvimento do Software	4		
17/05/2023	Desenvolvimento do Software	6		
18/05/2023	Desenvolvimento do Software	5		
19/05/2023	Desenvolvimento do Software	4		
20/05/2023	Reunião do grupo no laboratório de informática	2		
24/05/2023	Desenvolvimento do Software	5		
25/05/2023	Desenvolvimento do Software	5		
26/05/2023	Desenvolvimento do Software	3		
26/05/2023	Finalização e formatação do trabalho em norma ABNT	3		

TOTAL DE HORAS: 78 Horas

FICHA ATIVIDADES PRÁTICAS SUPERVISIONADAS – APS

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

NOME: Leonardo de Souza Rodrigues

RA: F344HB-2

CURSO: Ciência da Computação

CAMPUS: Jundiá

SEMESTRE: 5º Semestre

TURNO: Noturno

DATA	ATIVIDADE	TOTAL DE HORAS	ASSINATURA	
			ASSINATURA DO ALUNO	ASSINATURA DO PROFESSOR
08/05/2023	Discussão inicial no Discord para definição sobre o tema do trabalho	5		
09/05/2023	Pesquisa sobre o tema	3		
11/05/2023	Pesquisa sobre o tema	5		
13/05/2023	Pesquisa sobre o tema	4		
13/05/2023	Reunião do grupo para a definição do projeto do software	4		
15/05/2023	Elaboração do conteúdo do trabalho escrito	8		
16/05/2023	Elaboração do conteúdo do trabalho escrito	8		
17/05/2023	Pesquisa sobre a utilização do recurso MAUI (C#)	4		
16/05/2023	Desenvolvimento do Software	4		
17/05/2023	Desenvolvimento do Software	6		
18/05/2023	Desenvolvimento do Software	5		
19/05/2023	Desenvolvimento do Software	4		
20/05/2023	Reunião do grupo no laboratório de informática	2		
24/05/2023	Desenvolvimento do Software	5		
25/05/2023	Desenvolvimento do Software	5		
26/05/2023	Desenvolvimento do Software	3		
26/05/2023	Finalização e formatação do trabalho em norma ABNT	3		

TOTAL DE HORAS: 78 Horas

FICHA ATIVIDADES PRÁTICAS SUPERVISIONADAS – APS

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

NOME: Nicolas Pimenta da Silva

RA: N86357-9

CURSO: Ciência da Computação

CAMPUS: Jundiá

SEMESTRE: 5º Semestre

TURNO: Noturno

DATA	ATIVIDADE	TOTAL DE HORAS	ASSINATURA	
			ASSINATURA DO ALUNO	ASSINATURA DO PROFESSOR
08/05/2023	Discussão inicial no Discord para definição sobre o tema do trabalho	5		
09/05/2023	Pesquisa sobre o tema	3		
11/05/2023	Pesquisa sobre o tema	5		
13/05/2023	Pesquisa sobre o tema	4		
13/05/2023	Reunião do grupo para a definição do projeto do software	4		
15/05/2023	Elaboração do conteúdo do trabalho escrito	8		
16/05/2023	Elaboração do conteúdo do trabalho escrito	8		
17/05/2023	Pesquisa sobre a utilização do recurso MAUI (C#)	4		
16/05/2023	Desenvolvimento do Software	4		
17/05/2023	Desenvolvimento do Software	6		
18/05/2023	Desenvolvimento do Software	5		
19/05/2023	Desenvolvimento do Software	4		
20/05/2023	Reunião do grupo no laboratório de informática	2		
24/05/2023	Desenvolvimento do Software	5		
25/05/2023	Desenvolvimento do Software	5		
26/05/2023	Desenvolvimento do Software	3		
26/05/2023	Finalização e formatação do trabalho em norma ABNT	3		

TOTAL DE HORAS: 78 Horas

FICHA ATIVIDADES PRÁTICAS SUPERVISIONADAS – APS

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

NOME: Vitor dos Santos Rosa

RA: G521CE-3

CURSO: Ciência da Computação

CAMPUS: Jundiá

SEMESTRE: 5º Semestre

TURNO: Noturno

DATA	ATIVIDADE	TOTAL DE HORAS	ASSINATURA	
			ASSINATURA DO ALUNO	ASSINATURA DO PROFESSOR
08/05/2023	Discussão inicial no <u>Discord</u> para definição sobre o tema do trabalho	5		
09/05/2023	Pesquisa sobre o tema	3		
11/05/2023	Pesquisa sobre o tema	5		
13/05/2023	Pesquisa sobre o tema	4		
13/05/2023	Reunião do grupo para a definição do projeto do software	4		
15/05/2023	Elaboração do conteúdo do trabalho escrito	8		
16/05/2023	Elaboração do conteúdo do trabalho escrito	8		
17/05/2023	Pesquisa sobre a utilização do recurso MAUI (C#)	4		
16/05/2023	Desenvolvimento do Software	4		
17/05/2023	Desenvolvimento do Software	6		
18/05/2023	Desenvolvimento do Software	5		
19/05/2023	Desenvolvimento do Software	4		
20/05/2023	Reunião do grupo no laboratório de informática	2		
24/05/2023	Desenvolvimento do Software	5		
25/05/2023	Desenvolvimento do Software	5		
26/05/2023	Desenvolvimento do Software	3		
26/05/2023	Finalização e formatação do trabalho em norma ABNT	3		

TOTAL DE HORAS: 78 Horas