Web chat prático e privado:
Estudo de caso de comunicação virtual

João Hudson de Lacerda Oliveira

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, 2023

João Hudson de Lacerda Oliveira

Web chat prático e privado

Monografia apresentada ao curso Ciência da Computação
do Centro de Informática, da Universidade Federal da Paraíba,
como requisito para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Raoni Kulesza

Julho de 2023

Catalogação na publicação
Seção de Catalogação e Classificação

O48w Oliveira, Joao Hudson de Lacerda.
Web chat prático e privado: estudo de caso de comunicação virtual / Joao Hudson de Lacerda Oliveira. - João Pessoa, 2023.
52f. : il.

Orientação: Raoni Kulesza.
TCC (Graduação) - UFPB/CI.

1. Web chat online. 2. Comunicação privada. 3. Transmissão de mídia ilimitada. 4. Servidor de baixo custo. 5. Acessibilidade. 6. Praticidade. I. Kulesza, Raoni. II. Título.

UFPB/CI CDU 004.773.6

Elaborado por Michelle de Kássia Fonseca Barbosa - CRB-738

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

Trabalho de Conclusão de Curso de Ciência da Computação intitulado ***Web chat prático e privado: Estudo de caso de comunicação virtual*** de autoria de João Hudson de Lacerda Oliveira, aprovada pela banca examinadora constituída pelos seguintes professores:

Prof. Dr. Raoni Kulesza
CI/UFPB

Prof. Dr. Carlos Eduardo Coelho Freire Batista
CI/UFPB

Prof. Dr.Marcelo Iury De Sousa Oliveira
CI/UFPB

Coordenador(a) do Curso Ciência da Computação
Leandro Carlos de Souza
CI/UFPB

João Pessoa, 14 de julho de 2023

Centro de Informática, Universidade Federal da Paraíba
Rua dos Escoteiros, Mangabeira VII, João Pessoa, Paraíba, Brasil CEP: 58058-600
Fone: +55 (83) 3216 7093 / Fax: +55 (83) 3216 7117

"Program designers have a tendency to think of the users as idiots who need to be controlled. They should rather think of their program as a servant, whose master, the user, should be able to control it. If designers and programmers think about the apparent mental qualities that their programs will have, they'll create programs that are easier and pleasanter — more humane — to deal with."

(John McCarthy)

DEDICATÓRIA

Este trabalho é dedicado a todas as pessoas que já passaram raiva com software ruim.

AGRADECIMENTOS

Gostaria de agradecer a minha fámilia: meu avô Neco pelo enorme apoio durante boa parte da minha graduação, a minha namorada Edina(que me motivou bastante a manter o foco), a minha mãe Aucilene, minha avó Francisca, a meu primo Phelipe, meu irmão Matheus, meu amigo Marcello, meus colegas Genival e Luciane, e muitas outras pessoas que contribuíram com esse projeto, incluindo todos os participantes das pesquisas feitas. Gostaria também de agradecer aos professores que mais contribuíram com o meu conhecimento a cerca do assunto: Ulysses Oliveira, Alan Kelon, Chrisitan Azambuja, Ewerton Monteiro e Andrei Formiga.

RESUMO

Na atualidade, é comum mencionar vazamento de dados, espionagem e restrições em chats virtuais. Isso pode gerar desconforto ao utilizá-los para discussões sensíveis. Outro aspecto incômodo são as limitações na comunicação, como censura, restrições de tamanho de mídia, perda de qualidade e complexidade de instalação e cadastro. Com base nesses problemas, surgiu a ideia de desenvolver o sistema de chat SL Chat, que é um Web Chat online de fácil acesso, disponível em qualquer dispositivo com navegador e em diversos idiomas. A ferramenta oferece conversas totalmente privadas, sem necessidade de cadastro, e liberdade total no envio de informações, sejam textos ou mídias. Para avaliar a proposta, foi conduzida uma pesquisa qualitativa com alguns usuários.

Palavras-chave: Web Chat Online, Comunicação Privada, Trasmissão de Mídia Ilimitada, Servidor de Baixo Custo, Acessibilidade e Praticidade.

ABSTRACT

Nowadays, it is common to mention data leakage, spying and restrictions on virtual chats. This can cause discomfort when using them for sensitive discussions. Another uncomfortable aspect is the communication limitations, such as censorship, media size restrictions, loss of quality and complexity of installation and registration. Based on these problems, the idea arose to develop the SL Chat chat system, which is an easily accessible online Web Chat, available on any device with a browser and in several languages. The tool offers completely private conversations, without the need for registration, and total freedom in sending information, whether texts or media. To evaluate the proposal, a qualitative research was conducted with some users.

Key-words: Online Web Chat, Private Communication, Unlimited Media Transmission, Low Cost Server, Accessibility and Practicality.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

DTO - Data Transfer Object

E/S - Entrada e Saída

RAM - Randoom Access Memory

BBF - Backend For Frontend

P2P - Peer to Peer

UFPB - Universidade Federal da Paraíba

HTML - HyperText Markup Language

CSS - Cascading Style Sheets

SPA - Single Page Application

API - Application Programming Interface

REST - Representational State Transfer

MB - Megabytes

JSON - JavaScript Object Notation

CI - Centro de Informática

OOP - Object Oriented Programming

Conteúdo

1 INTRODUÇÃO

Nos chats virtuais que usamos na atualidade é bem comum ouvirmos falar de dados de conversas vazadas, espionagem e temas parecidos. Dado esse contexto algumas pessoas podem se sentir desconfotáveis ao usarem essas aplicações para se comunicarem sobre determinados assuntos.

Outro fator que pode incomodar algumas pessoas são as limitações na comunicação, seja por censura, limites de tamanho de mídias que podem ser enviados, perda de qualidade das mídias, ou complexidade no momento de instalar ou se cadastrar numa nova aplicação dessas.

Se baseando nesses problemas nasceu a ideia de construir um chat que os resolvesse e service de alternativa nessas ocasiões. SL Chat é um Web Chat online que propõe ser fácil de acessar, acessível por qualquer dispositivo que disponha de um browser, em qualquer idioma, com conversas totalmente privadas, sem a necessidade de se cadastrar e liberdade total no envio de informações, sejam textos ou mídias.

Este trabalho se propõe a estudar o processo de desenvolvimento desta aplicação, sua proposta e seus resultados.

1.1 Objetivos

1.1.1 Objetivos Gerais

O trabalho tem o objetivo de desenvolver uma aplicação web chat que solucione problemas de privacidade, limitações no envio de mídia, que seja prático e de fácil acesso.

1.1.2 Objetivos Específicos

Especificamente falando pretende-se:

- Analisar limitações e problemas encontrados nas tecnologias atuais para comunicação virtual;
- Analisar a proposta da aplicação em questão para os problemas encontrados;
- Discorrer sobre as estratégias técnicas usadas para solucionar os problemas propostos;
- Falar sobre as tecnologias usadas na solução;

- Analisar a Arquitetura da Solução;
- Analisar os requisitos necessários e desejáveis para a aplicação;
- Analisar resultados da pesquisa qualitativa referente a satisfação dos usuários sobre a aplicação em seu estado mais recente.

1.2 Estrutura da Monografia

Este trabalho está estruturado da seguinte forma: na seção 1 é apresentada a introdução. Na seção 2 são apresentados as bases teóricas e tecnologias exploradas no projeto. Na seção 3 são apresentados os trabalhos relacionados, que serviram de referência para a construção deste. Na seção 4 é apresentada a especificação da aplicação. Na seção 5 é apresentada a metodologia utilizada. Na seção 6 é apresentada a arquitetura da aplicação. Na seção 7 são apresentados os resultados e a análise deles.

2 FUNDAMENTAÇÃO TEÓRICA

A comunicação gratuita através das redes sociais costumam deixar algumas pessoas desconfiadas: "como eles ganham dinheiro com isso?". Acredita-se que através da análise das nossas conversas essas empresas conseguem nos direcionar anúncios mais apropriados e fazer muio mais dinheiro com publicidade. Além de casos que a empresa parece analisar as conversas para poder aplicar certas políticas restritivas de sua comunidade. Algumas pessoas não gostam muito dessa ideia e podem preferir, em alguns casos, ter um pouco mais de privacidade, a depender do teor da conversa.

Para trazer uma boa alternativa nos casos mencionados, a aplicação deve ser prática, tanto em sua usabilidade quando em seu acesso, construir uma sistema de comunicação por meio de salas privadas com compartilhamento de link acaba tornando possível que usuários se comuniquem de forma bem descompromissada, sem cadastros, sem vínculos permanentes, a ideia básica seria de que o usuário apenas cria-se uma sala, informaria seu nome(ou apelido) e compartilharia o link com outro(s) usuário(s). Com uma aplicação Web isso seria ainda mais simples, pois a mesma não precisa ser instalada e poderia ser executada em qualquer plataforma/dispositivo que tenha um browser. Se a aplicação não armazenar as conversas, os usuários podem ficar mais confortáveis sabendo que seus dados não serão explorados. Temos então a proposta incial para construção da aplicação. As tecnologias exploradas no projeto serão apresentadas a seguir.

2.1 SocketIO

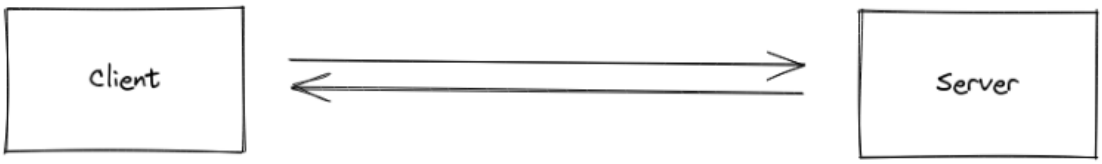

Figura 1: Comunicação Bidirecional

Socketio[16] é uma biblioteca orientada a eventos feita em cima do protocolo WebSocket[5], ela possibilita uma comunicação de baixa latência, bidrecional e baseada em eventos entre cliente e servidor. A biblioteca está disponível para disversas linguagens, como: JavaScript, TypeScript, Java, Golang, Python. Existem versões da biblioteca tanto para o cliente quanto para o servidor, trazendo uma forma prática e consistente de se usar a comunicação bidirecional de baixa latência do WebSocket.

Importante enfatizar que a biblioteca não é uma implementação direta do protocolo WebSocket[5], mas sim uma abstração construída em cima desse protocolo. Sendo assim um servidor Socketio[16] deve se comunicar com um cliente Socketio[16] e não com um que use diretamente o protocolo WebSocket, e vice e versa.

Socketio[16] possui algumas vantagens em relação a implementações diretas do protocolo WebSocket[5]:

- **Reconexão Automática:** Sob algumas condições particulares, a conexão WebSocket[5] entre o servidor e o cliente pode ser interrompida com ambos os lados desconhecendo o estado quebrado do link. É por isso que o Socketio[16] inclui um mecanismo de pulsação, que verifica periodicamente o status da conexão. E quando o cliente eventualmente é desconectado, ele se reconecta automaticamente com um atraso exponencial de back-off, para não sobrecarregar o servidor.
- **Buffer de Pacote:** Os pacotes são automaticamente armazenados em buffer quando o cliente é desconectado e serão enviados na reconexão.
- **HTTP long-polling fallback:** A conexão retornará à sondagem longa HTTP caso a conexão WebSocket não possa ser estabelecida.

2.2 NodeJS

Figura 2: NodeJS

NodeJS[1] é um ambiente de execução JavaScript[8] para desktop, ele possibilita a execução de código JavaScript[8] diretamente no desktop sem o intermédio de um browser, usando a V8, VM do JavaScript[8] do Google Chrome[2]. Isso permite que o desenvolvedor faça uso não só da praticidade da linguagem, como também de seu E/S não bloqueante, útil para casos com muitas requisições.

2.2.1 E/S Não Bloqueante

Figura 3: Loop de Eventos

[1]https://nodejs.org/en/about
[2]https://www.google.com/intl/pt-BR/chrome/

JavaScript[8] segue um modelo de desenvolvimento assíncrono, onde uma única thread executa o código mas **não** aguarda por E/S algum. Por exemplo, sempre que existe alguma entrada de dados, essa entrada é processada por uma função que fica aguardando os dados chegarem, todo o resto do código continua executando independente dessa função. Esse mecanismo é denomiando *Event Loop* e consiste num loop de eventos que executa essas funções conforme vão sendo enfileirdas.

Além de simplificar o desenvolvimento pelo contexto single thread, essa abordagem permite um grande ganho de performance na E/S. O modelo padrão de E/S cria várias threads, bloqueia sempre que houver alguma E/S e retorma quando é concluída, isso acaba reduzindo o desempenho geral do sistema quando ocorre com muita frequência. O ganho de performance é considerável quando a aplicação faz uso intensivo de E/S.

2.3 Express

Express[3] é um framework leve, flexível e prático para construção de servidores http em NodeJS[7]. Ele fornece os recursos básicos para construção de um servidor desse tipo, isso inclui:

- Sistema de rotas completo.
- Sistema de verbos.
- Servidor de arquivos estáticos.
- Serialização e desserialização de dados de requisição e resposta.

[3]https://expressjs.com/pt-br/

2.4 UUID

UUID[4] é um identificador universalmente exclusivo utilizado para identificação de qualquer coisa no mundo da computação. O UUID[4] é um número de 128 bits representado por 32 dígitos hexadecimais, exibidos em cinco grupos separados por hifens, na forma textual 8-4-4-4-12 sendo um total de 36 caracteres (32 caracteres alfanuméricos e 4 hifens).

Quando gerado de acordo com os métodos padrões, os UUIDs[4] são únicos, para fins práticos, sem depender sua singularidade de uma autoridade central de registro ou coordenação entre as partes que os geram, ao contrário da maioria dos outros esquemas de numeração. Embora a probabilidade de um UUID[4] ser duplicado não seja zero, ele está próximo o suficiente para ser insignificante.

Assim, qualquer pessoa pode criar um UUID[4] e usá-lo para identificar algo com quase certeza de que o identificador não duplica um que já tenha sido ou será criado para identificar outra coisa. As informações rotuladas com UUIDs[4] por partes independentes podem, portanto, ser posteriormente combinadas em um único banco de dados ou transmitidas no mesmo canal, com uma probabilidade insignificante de duplicação.

Exemplo: 123e4567-e89b-12d3-a456-426655440000

2.5 SPA

Single Page Applications (SPA)[5] são aplicações cuja funcionalidade está concentrada em uma única página. Ao invés de recarregar toda a página ou redirecionar o usuário para uma página nova, apenas o conteúdo principal é atualizado de forma assíncrona, mantendo toda a estrutura da página estática.

O interessante desse modelo é que ele permite construir aplicações com respostas mais rápidas, transições de telas suaves e customizadas. É particularmente interessante para aplicações que precisem manter referências a um mesmo objeto em todas as telas, como por exemplo um socket que se mantém conectado.

[4]https://pt.wikipedia.org/wiki/Identificador_Único_universal
[5]https://www.devmedia.com.br/ja-ouviu-falar-em-single-page-applications/39009

2.6 ES6

JavaScrit[8] é uma linguagem de programação que nasceu com a proposta de deixar as páginas da web mais dinâmicas, e que inicialmente apresentava muitos desafios para os desenvolvedores devido a falta de padronização da mesma entre os navegadores. Depois de alguns anos de desenvolvimento da web, os desenvolvedores foram percebendo a necessidade de algumas mudanças e padronizações nessa linguagem, a proposta que teve um impacto bastante significativo no desenvolvimento de projetos grandes com a linguagem foi o padrão ES6[6], que dentro muitas outras features inclui:

- Sistema de módulos que permite importar código de outro arquivo, sem ter o intermédio do html da página.
- Definição mais precisa de classes para melhor uso do paradigma OOP.

[6]https://medium.com/@matheusml/o-guia-do-es6-tudo-que-você-precisa-saber-8c287876325f

3 TRABALHOS RELACIONADOS

3.1 WhatsApp

Figura 4: WhatsApp

Uma rede social extremamente utilizada para troca de mensagens, principalmente aqui no Brasil, é o WhatsApp[7], ele permite troca de mensagens em tempo real, envio de mídias, ligações, etc... A base para iniciar a comunicação com outros usuários é o contato(número de telefone) da pessoa, que é vinculado a sua conta, esse vínculo é feito na etapa de cadastro da conta e é usado para identificação do usuário, durante essa fase de cadastro a aplicação

[7]https://www.whatsapp.com/

solicita ao usuário que insira um código enviado para seu número, para confirmar que é a pessoa mesmo quem está cadastranto aquele telefone. As conversas persistem mesmo offline e são armazenadas localmente, é um aplicativo útil mesmo para conexões fracas e instáveis, ele sabe lidar com problemas de queda de rede. Outro ponto chave dessa aplicação é que ela se propõe a ser bastante privada, usando uma criptografia ponta a ponta[2].Entretanto existem alguns pontos sobre essa aplicação:

- **Suspeita de Espionagem:** Não existem provas concretas de que o WhasApp[6] faça qualquer tipo de espionagem sobre os dados de seus usários, e como maior argumento contra essa possibilidade está a já mencionada criptografia ponta a ponta[2], entretanto já houveram certas repercusões sobre essa possibilidade como foi noticiado em uma matéria do G1[8]. Verdade ou não, algumas pessoas podem ficar desconfortáveis em converar sobre assuntos muito privados depois desse tipo de repercusão.

- **Limite de Tamanho no Envio de Mídias:** A aplicação limita o tamanho das mídias que os usuários podem enviar, por exemplo, caso tente enviar um vídeo muito longo ele será cortado.

- **Redução na Qualidade das Mídias:** Provavelmente com o intuito de reduzir o tamanho do arquivos, as mídias costumam perder qualidade quando enviadas por essa aplicação.

[8]https://g1.globo.com/distrito-federal/noticia/coincidencia-diz-diretor-do-whatsapp-sobre-anuncios-relacionados-as-mensagens-trocadas-no-app.ghtml

3.2 Direct Instagram

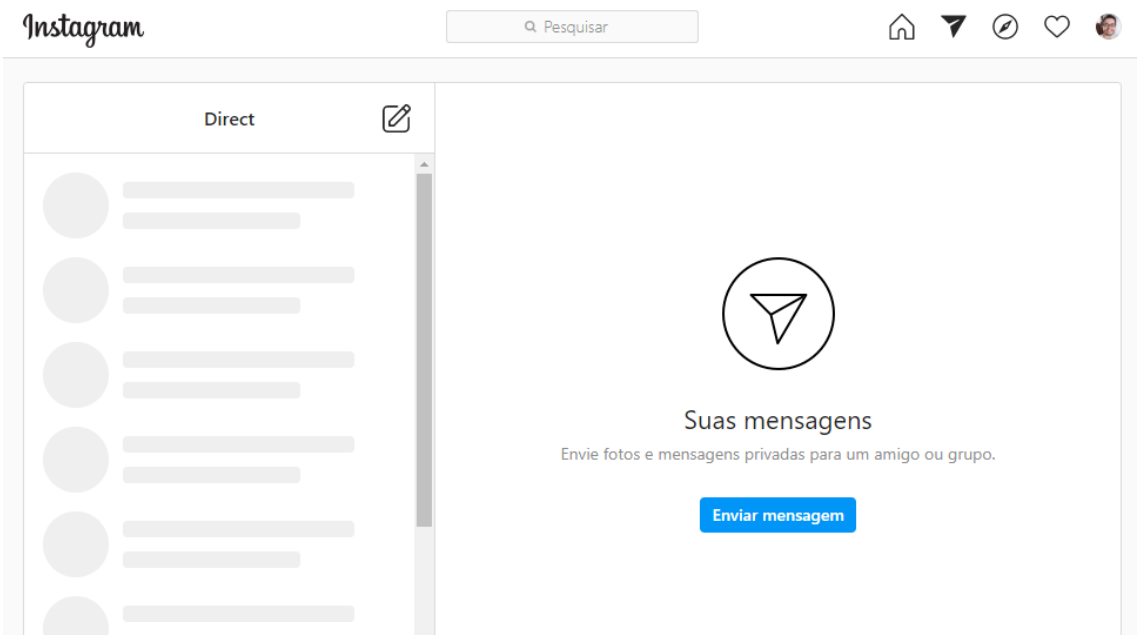

Figura 5: Direct do Instagram

O Direct do Instagram[9] é o módulo de mensagens da rede social Instagram[10]. Ele permite funcionalidades bem semelhantes as do WhatsApp[6]. Com as diferenças de que: ele está vinculado ao próprio Instagram, o que torna mais prático compartilhar conteúdo da referida rede social; armazena uma pequena parte mais recente da conversa localmente e por tanto é menos prático em situações de instabilidade; a conexão com outros usuários se da por meio da própria rede social Instagram, onde as pessoas são identificadas por suas contas que devem ser feitas preenchendo algumas informações pessoais.

[9]https://about.instagram.com/pt-br/features/direct
[10]https://www.instagram.com/

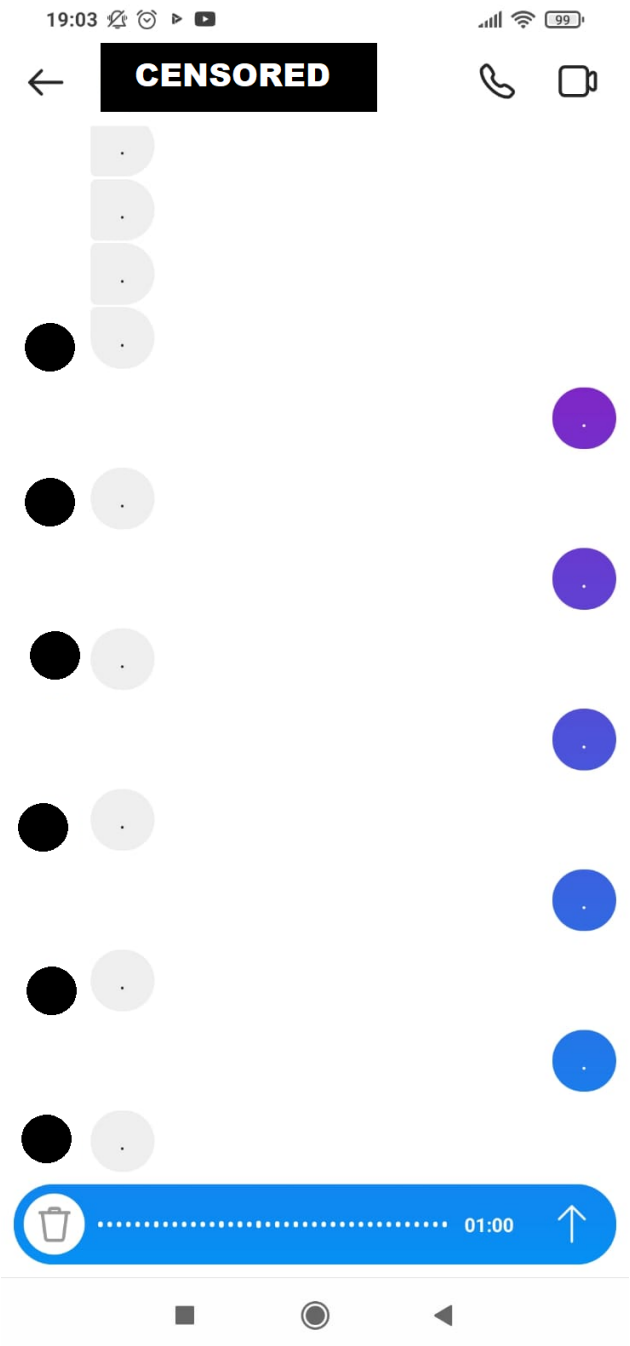

Figura 6: Limite de Áudio

No caso do Instagram também temos o problema com o limite do tamanho das mídias enviadas, é até bem evidente isso quando tentamos enviar um áudio muito longo e aplicação pausa ele propositalmente ao completar **1 minuto**, o que pode ser frustrante para alguns usuários.

Figura 7: Punição por Não Seguir Diretrizes

A privacidade parece não ser 100% visto que a rede social aplica certas políticas de restrição em publicações e no próprio Direct, isso não só pode deixar algumas pessoas desconfortáveis por terem suas conversas possivelmente analisadas, como também limita a comunicação dentro do que a rede social considera aceitável [11].

[11]https://www.techtudo.com.br/listas/2020/10/oito-tipos-de-mensagens-que-voce-nao-deve-enviar-no-instagram-direct.ghtml

3.3 Messenger

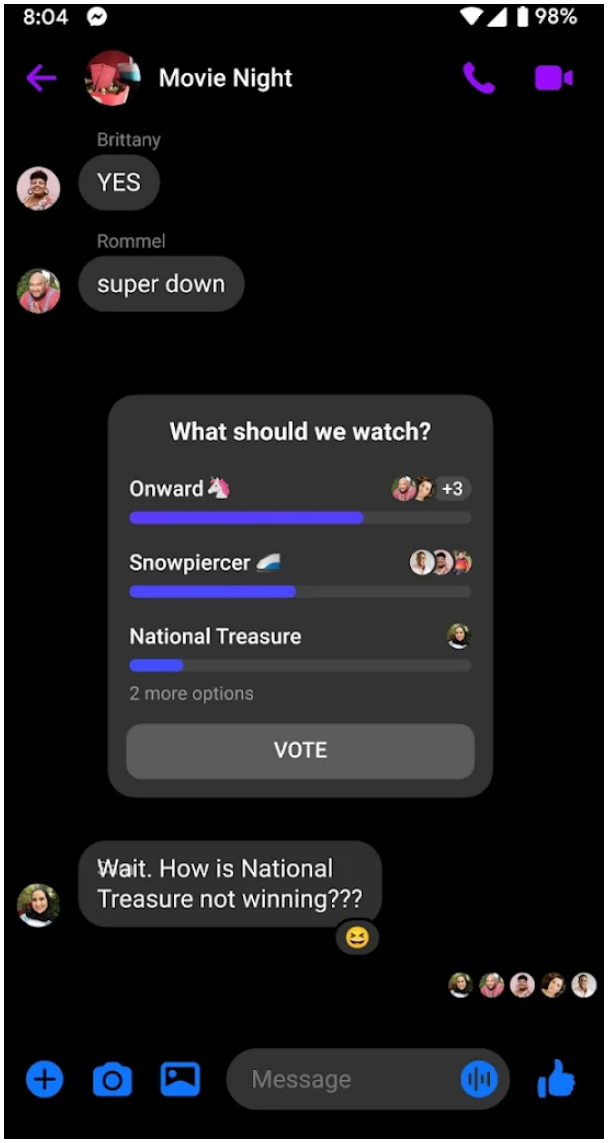

Figura 8: Messenger

O Messenger[12] é um aplicativo de mensagens do Facebook, ele seria uma extensão dessa rede social para troca de mensagens e funciona em um aplicativo à parte. Seu propósito é semelhante ao do Direct do Instagram, a forma de identificar os usuários também, usando a conta do Facebook[13] como base. Possui limitações semlhantes as do Direct.

[12]https://www.messenger.com/
[13]https://www.facebook.com/

4 FUNCIONALIDADES E USABILIDADE

4.1 Levantamento das Principais Funcionalidades

Analisaremos agora as funcionalidades necessárias e desejáveis que devem ser implementadas na aplicação. Como se trata de um web chat, podemos fazer uma levantamento inicial dessas funcionalidades analisando outras aplicações semelhantes, como o WhatsApp[6]. É importante observar que a ideia proposta não pretende criar uma concorrência com o WhatsApp[6], mas sim trazer uma alternativa mais conveniente para alguns momentos, uma vez que as conversas serão temporárias(sem persistência), é comum até mesmo usar esses outros chats para convidar alguém para a conversa.

No WhatsApp[6] podemos encontrar algumas principais funcionalidades como:

- Mensagens de texto.
- Envio e captura de imagens.
- Envio e gravação de vídeos.
- Envio e gravação de áudio.
- Chamada de voz e de vídeo.
- Envio de arquivos.
- Comunicação a partir do número de telefone de outros usuários, onde as conversas são persistidas.
- Mútiplos Idiomas.

4.2 Avaliação das Funcionalidades

Dadas essas funcionalidades, podemos analisar quais delas se encaixam melhor na ideia proposta e quais são indispensáveis pelos usuários. Avaliando funcionalidades a serem consideradas e suas prioridades:

- **Mensagem de Texto:** A funcionalidade mais básica de qualquer chat não pode ficar de fora, sua implementação é bastante simples e constitue a forma mais básica de comunicação.

- **Envio de Mídia:** Englobando o envio de áudio, imagem, vídeo e arquivos no geral, esta é uma funcionalidade bastante útil e que pode deixar o chat mais rico. Principalmente das mídias, pois é muito comum usuários enviarem áudio, imagens e vídeos uns para os outros.

- **Gravação de Áudio:** Muito importante ou até mesmo um recurso necessário. Muitas pessoas fazem uso desse recurso e com muita frequência, algums usam inclusive por não saber escrever, o que acaba se tornando também um recurso que amplia a acessibilidade.

- **Captura de Imagem:** Bastante desejável, afinal é comum pessoas tirarem fotos no momento da conversa e enviarem para outras. Entretanto, o browser dos dispositivos móveis(lugar onde essa funcionalidade seria mais usada) costumam disponibilizar a câmera do próprio dispositivo na hora de fazer uploads de arquivos, o que acaba fazendo com que essa fucionalidade não seja tão crucial.

- **Gravação de Vídeo:** O mesmo da funcionalidade acima se aplica a esta.

- **Chamada de Áudio:** Desejável, muitas pessoas usam a chamada do WhatsApp[6] para se comunicarem e evitar os custos das operadoras de telefone. Entretanto o custo da comunicação desse tipo de funcionalidade pode ser mais alto e sua implementação mais complexa, dado que a latência de uma comunicação em tempo real deve ser bem baixa.

- **Chamada de Vídeo:** Bastante apreciada também, e com a possibilidade de ver quem está do outro lado, essa tem grandes vantagens porém a sua desvantagem é semelhante a da chamada de áudio, no entanto ainda maior, devido a transmissão do vídeo elevar os custos em comparação a de áudio puro.

- **Comunicação feita a partir de:** Uma vez que a ideia desta aplicação não pretende vincular o usuário de forma permanente à conversa, e também não pretende tornar burocático seu primeiro acesso com cadastros complexos. A ideia para se identificar e iniciar uma conversa foi diferente, sem necessidade de cadastro, o usuário precisaria apenas criar uma sala e compartilhar ela com quem fosse conversar, via link, a pessoa que entra na sala precisa apenas informas seu nome.

- **Mútiplos Idiomas:** Para tornar mais acessível, é útil prover vários idiomas para a aplicação.

As análises acima foram feitas de forma objetiva, buscando estabelecer o custo de implementação das funcionalidades, e sua necessidade por parte dos usuários. Saber o quão

útil e necessária é uma funcionalidade para quem a utiliza, geralmente é alcançado através de pesquisas com estas pessoas(elicitação de requisitos). É importante notar que levamos em conta tanto a utilidade e nececssidade dos requisitos quanto seu custo de desenvolvimento, o custo também precisa ser considerado na decisão de implementar uma funcionalidade. Como o WhatsApp[6] é uma aplicação muito popular e amplamente utilizada, é comum já ter uma boa noção da utildade dessas funcionalidades antes mesmo da pesquisa, afinal é provável que você também seja um dos usuários, mas no decorrer da implementação incial(prototipagem) foram feitas pesquisas diretas com outros usuários através de testes práticos de um protótipo da aplicação com algumas dessas funcionalidades implementadas, isso para ter um levantamento mais confiável do que seria se fosse feito apenas com a opinião do desenvolvedor.

A prototipagem não só pode ajudar nas primeiras pesquisas e no levantamento dos requisitos, como também pode ajudar a entender o nível de complexidade para implementar algumas funcionalidades. Ela deve ser um projeto básico com apenas algumas funcionalidades implementadas e que seja feito sem muito rigor para simplificar.

Uma vez validado os requisitos com base o WhatsApp[6] e na ideia original da aplicação, os requisitos estabelecidos foram:

- Mútiplos Idiomas.
- Mensagem de Texto.
- Criação e Compartilhamento de Salas Temporárias.
- Envio de Mídia.
- Envio de Arquivos.
- Gravação de Áudio.

A lista acima está em ordem de prioridade para sua implementação, embora a **gravação de áudio** possa ser mais importante que, por exemplo, **envio de arquivos**, a implementação do envio de áudio pode reutilizar a implementação do envio de mídias e de arquivos, por tanto, seria implementado depois. Sobre os **mútiplos idiomas**, embora não seja a funcionalidade mais crucial de todas, o suporte a este mecanismo deve ser implementado no início para evitar muita refatoração, já que ele afeta praticamente todo o texto presente na aplicação, etapas iniciais de prototipagem não precisam se preocupar com este requisito, mas o desenvolvimento oficial da aplicação sim. Para saber se um requisito aparentemente menos importante deve ser priorizado, deve ser analisado o quanto ele impacta outros requisitos e o quanto esses requisitos podem depender e tirar proveito dele.

5 METODOLOGIA

Para implementar a ideia proposta, começamos pela definição das principais estratégias e analisando quais são as principais tecnologias mais adequadas para implementá-las. Isso inclui a linguagem de programação, plataforma, bibliotecas, frameworks...

Para esse caso em questão, precisamos de uma comunicação bidirecional, ou seja, o servidor também precisa ter a capacidade de enviar mensagens para o usuário quando desejar. Precisamos de tecnologias que sejam eficientes para considerar um grande número de mensagens sendo rapassadas para vários outros usuários, broadcast de mensagens, que nesse caso seria a ação de um usuáro enviar uma mensagem para outros usuários por intermédio do servidor.

Também precisamos que o cliente possa ser executado no maior número de dispositivos possível para trazer o fator de acessibilidade, além de que ele não seja burocrático para se acessar, sobretudo no primeiro acesso(que é onde fica a primeira impressão). Um cliente que não precise de cadastro e que não precise ser instalado pode atender bem a este caso.

5.1 Tecnologias

Atendendo a esta ideia, a plataforma escolhida para o client foi a web, pois é possível acessá-la de qualquer dispositivo sem problemas, sem ter que desenvolver vários projetos, sem a complexidade de publicar em cada loja de aplicativos e sem exigir que o usuário instale nada. A linguagem de programação escolhida foi o JavaScript[8], por ser a linguagem nativa do browser.

A plataforma para backend escolhida foi o nodejs[7], devido ao seu E/S não bloqueante, ele é muito otimizado para o tipo de tarefa proposta, broadcast de mensagens de um usuário para outro. Esse tipo de operação gera muito mais sobrecarga no E/S de rede que em poder de processamento, o que torna essa plataforma uma ótima opção para a aplicação em questão.

Para comunicação foi usada a biblioteca Socketio[16], uma abstração leve e produtiva construída em cima do protocolo WebSocket, que é excelente para comunicação bidirecional, útil nesse caso já que o servidor precisa repassar a mensagem de um cliente para outro. Socketio é também bastante escalável.

5.2 Tradução

Para ampliar a acessibilidade da ideia proposta, a aplicação foi concebida desde o início com o intuito de ter mútiplos idiomas. Apesar de existir o recurso de tradução integrada em alguns browsers, existem muitas desvantgens quanto a contar com este recurso para tradução:

- Não é garantido que o browser do cliente terá esse recurso.
- A tradução dinâmica das páginas pode não ser tão clara, afinal traduzir linguagem natural é uma tarefa complexa para algoritmos.
- A tradução dinâmica pode distorcer as mensagens do chat, principalmente porque podem haver gírias ou até mesmo palavras escritas propositalmente em outro idioma.

Dado estes pontos, uma tradução nativa da própria aplicação seria mais interessante para trazer uma acessibilidade de qualidade. A estratégia para implementar este mecanismo foi a seguinte:

- **Dicionários:** Usando uma estratégia semelhante a do android[10], foi implementado um mecanismo basedo em dicionários, onde teríamos tabelas escritas no formato json, que fariam o mapeamento de *chaves lógicas*, que iriam referenciar textos por contexto na aplicação, para os textos propriamente ditos. Haveria um dicionário para cada idioma. Uma outra vantagem dessa forma de tradução, é que daria para escolher frases diferentes que façam mais sentido em cada idioma por contexto.
- **Endpoint de Tradução:** Os dicionários seriam fornecidos para o client através de um endpoint que usaria o cabeçalho *Accept-Languages* para escolher o idioma correto e retornar o dicionário correto em formato json. Isso ainda abriria possibilidade para reaproveitar a tradução em outros clientes, caso fossem desenvolvidos(como mobile por exemplo).
- **Definição Dinâmica de Textos:** Para usar os textos dos dicionários o cliente atribui dinamicamente a cada elemento gráfico os textos, após a solicitação do dicionário, usando as chaves lógicas que referenciam cada contexto da frase. Isso permite ainda desacoplar a frase do código, e a produção textual de cada frase ficaria centralizada nos dicionários.

5.3 Tratamento de Erro

Para implementar um tratamento de erro eficaz não me dediquei a pesquisar por um "padrão infalível"que fosse muito usado, ao invés disso eu olhei para minha aplicação e pensei numa estratégia que resolvesse o problema da melhor forma possível.

Claro que quando vamos ter uma ideia inicial partimos de algo que já conhecemos, um padrão bem comum de se encontrar é um tratamento de erro onde o servidor retorna um objeto json contendo uma mensagem de erro, e o frontend simplesmente exibe diretamente. Isso geralmente é encontrado em sistemas com modelo BFF, mas pode ser visto em outros tipos também.

No caso desta aplicação, essa não seria uma boa alternativa, estamos falando de um contexto com mútiplos idiomas, a mensagem de erro vinda direto do backend seria inútil. Então continuousse a análise de um método mais eficaz para este caso, a ideia é que se não faz sentido se referenciar direto a mesagem, poderia dar um significado mapeavel para ela, um código de erro. O sistema de erros basicamente iria disparar códigos de erros pré-estabelecidos, então bastaria mapear estes erros para suas respectivas mensagens, temos assim o início da estratégia de tratamento de erro para esta aplicação.

Outros pontos a se considerar:

5.3.1 De onde virão os códigos de erro?

Os códigos de erro podem vir tanto do backend quanto do próprio frontend, a final de contas, podem ocorrer alguns problemas no frontend também, como por exemplo, negar acesso a um recurso que o client precise usar. Para dar suporte a isso e considerando também os erros que podem vir do próprio socket(afinal de contas, estamos usando um socket que mantém sua conexão), poderíamos ter 3 tipos de mensagens: erro do socket, código de erro do backend, código de erro do frontend. De forma a evitar conflitos, os códigos de erro do backend foram prefixados com *SLLERR* e os do frontend com *LOCAL_ERROR*

5.3.2 Como será feito o mapeamento?

O mapeamento dos erros nas mensagens também foi feito pensando na forma mais eficaz, a princípio poderia se pensar em colocar um erro por classe, tentar mapear o código na classe através de algum padrão complexo, como *Chain of Responsability*[9], mas na verdade qualquer abordagem dessas acrescentaria muita complexidade de código desnecessária. Uma

forma muito mais simples é criar um mapa. Então foi construído um mapa que teria o mapeamento dos códigos de erros em geral para as chaves do dicionário que contém a tradução, isso em um arquivo. Em outro arquivo teria uma única função que faria a decodificação do erro usando o mapa, caso o erro não se encontre no mapa, é algum erro desconhecido e não tratado, então a função retornaria a mensagem de erro desconhecido do dicionário.

Desta forma foi solucionado o tratamento de erro em mútiplos idiomas com apenas uma função e um mapa em um arquivo a parte, algo que pode ser chamado de *solução elegante*[1]. Para adicionar um novo erro, bastaria criar uma constante com o código do erro, seja no backend ou frontend, adicionar na tabela de erros, e por fim, adicionar as mensagens de erro em seus respectivos dicionários e idiomas. Um fluxo simples e sem necessidade de adicionar mais lógica. O uso de mapas é particularmente útil em situações em que é necessário fazer mapeamentos discretos e finitos, onde uma tabela poderia resolver, ele pode ser encarado como uma função discreta. A vantagem do uso deles é que a estrutura de dados de um mapa pode ter o custo O(1) e ainda permitir que você importe esses dados de arquivos externos ao seu código(ex.: json), deixando-o mas limpo.

5.4 Broadcast de Mensagens de Texto

Chegando na parte principal, a estratégia para envio das mensagens de texto seriam bastante simples, basta associar cada socket a um usuário no backend (isso é feito de forma eficiente associando o id do socket ao objeto do usuário por meio de um mapa), e também a sua sala, assim quando enviar uma mensagem através do socket, o servidor reenvia estas mensagens para todos os outros usuários da mesma sala.

Para implementar a abstração de sala foi usado o uuid[4] v4, a versão 4 de um algoritmo de geração de ids únicos, e um mapa extra de usuário por sala(para reduzir o custo da busca do usuário por sala). Uma vez implementado o broadcast de mesagens, a parte visual consiste basicamente em receber o payload das mensagens vinda do backend com o nome do usuário, seu id, e o texto da mensagem, daí adicionar isso na lista visual de mensagens.

5.5 Broadcast de Mídia

Esta foi de longe a parte mais complexa de todo o projeto, o que iria ser apenas "a cereja do bolo"acabou virando o "recheio".

5.5.1 Estratégia Inicial

Como uma primeira tentiva no envio de mídia foi necessário pesquisar mais a respeito da tecnologia usada na aplicação, Socketio[16]. A tecnologia permite envio de texto e dados binários, entretanto, no formato de dados binários a adição de mais metadados ficaria complexa, já que a mensagem é composta pelo nome do evento, que indica quem vai receber a mensagem, e pelo payload, que para incluir metadados geralmente é usado json, neste caso em particular o json era preferível para poder identificar quem era o usuário. Dado esse contexto, uma primeira abordagem foi usando a formatação base64 para incorporar os dados binários no payload json.

Uma observação importante é que no protocolo do socketio, cada mensagem é um evento e quando ela é enviada, só é recebida pelo ouvinte quando é carregada totalmente na memória RAM. Pensando em arquivos de mídia grandes isso poderia ser um problema, o consumo de memória poderia aumentar muito, o tamanho dos buffers precisaria aumentar também, gerando um alto consumo de memória.

Outra desvantagem se encontra no uso da formatação em base64, que acaba aumentando consideravelmente o tamanho dos arquivos enviados.

5.5.2 Streaming de Mídia

A segunda estratégia buscava contornar este problema, o uso de stream de dados é uma estratégia usada para contornar este tipo de problema, onde você não deseja carregar de tudo de uma vez na memória. A princípio existiam 2 ideias para implementar o stream de mídia nesse contexto:

- **Uso de Stream por API Rest:** Esta estratégia seria a princípio um jeito fácil. Daria para trabalhar com dados binários no body, usar metadas na query, path ou headers, e ainda fazer stream dos dados do body para o servidor; por fim, daria para retornar para o usuáro uma url onde ele poderia acessar o recurso. **As vantagens** dessa abordagem seriam: remoção do base64, que reduziria o tamanho das mensagens; uso de url simples ao invés do base64 direto no html, que deixaria a página mais leve; facilidade no compartilhamento do recurso, que acaba se resumindo à url. **As desvantagens** desta abordem seriam: necessidade de banco de dados para armazenar as mídias, isso por si só já seria uma grande desvantagem considerando o alto custo de armazenamento que iria causar na aplicação; Persistência das mídias, que acabaria com a proposta de não persistir de forma alguma as mensagens dos usuários.

- **Uso de Stream P2P via Socketio:** Esta estratégia por outro lado, tornaria possível fazer o envio de mensagens sem necessidade de armazená-las no backend, isso funcionaria da seguinte forma: O arquivo seria convertido em base64, o base64 seria particionado em pedaços de tamanho fixo, os pedaços seria enviados em várias mensagens identificando o usuário e seu progresso no envio. Uma vez que todos os pedaços sejam recebidos, o receptor montaria novamento o base64 adicionarioa a tela para visualização. **As desvantagens** consistiram basicamente no tamanho do base64, na poluição da página com todos estes dados em string bruta, e no trabalho de particionar strings tão grandes.

Por manter intacta a proposta de solução para o problema(não persistir as mensagens), o stream P2P acabou sendo escolhido. Mas ainda haviam alguns problemas que precisam ser resolvidos, então novas estratégias foram elaboradas.

5.5.3 Melhorando Armazenamento Local

Uma das primeiras melhorias na estratégia anterior de stream P2P, foi a conversão do base64 em blob, por ser um armazenamento binário ele ocupava menos espaço, além de ter a possibilidade de gerar uma url local para o recurso, evitando assim acrescentar os dados brutos diretamente no html da página, o efeito seria parecido com o da primeira estratégia, onde uma url era compartilhada para acessar o recurso(porém nesse caso ela só serve para o usuário local). Por fim, os blobs são gerenciados pelo browser, que pode escolher a forma mais eficiente de armazená-los, podendo até poupar memória RAM do dispositivo.

5.5.4 Melhorando Stream P2P

Um dos problemas no stream implementado até o momento foi o fato dele trabalhar particionando texto base64 bruto, pensando nisso surgiu a ideia de particionar arquivos binários no lugar, a formatação em texto para enviar os dados dentro do payload json só precisaria ocorrer no momento do envio e do recebimento. A partir disso a partição ficou em cima da abstração de arquivos, onde os dados eram fatias de blobs do arquivo a ser enviado, então no momento de enviar a mensagem, eles eram convertido em base64, no momento de receber eles eram convertidos de volta em binário e então concatenados aos demais pedaços até que fosse conluída a transmissão. No final do envio, todos os pedaços binários eram unificados em um uníco blob e sua url era gerada para ser compartilhada na tela.

5.5.5 Otimizando Mensagens

A formatação base64 usando as primitvas de conversão do browser esatavam aumentando muito o tamanho do array binário, isso porque a codificação era feita a partir da representação em string dele, que por si só já poderia ocupar até 4x o tamanho original do array. Uma melhoria inicial foi trocar o base64 pela própria representação em string do array binário.

Dado que essa primeira melhoria não fez grande diferença e que o aumento no tamaho ainda era exagerado, uma segunda estratégia foi usada. Pensando em termos de representação binária, uma forma eficiente de representar os dados binários em texto, seria usar a menor quantidade de texto possível para representar os bytes, para isso foi levantado um conjunto de 256 caracteres que não precisassem de caractere de escape para ser representados dentro de um json e que fossem representados com o menor número de bytes possível. Com o auxílio do ChatGPT[15] foi levantado esse conjunto de caracteres, que foi então usado para codificar e decodificar os bytes transmitidos. A fórmula é simples, cada caractere seria mapeado para um byte de 0 a 255.

Dessa vez o ganho foi considerável, um arquivo agora poderia variar, em sua versão codificada, de 100% a 200% de seu tamanho original, em média ocuparia 164%, dado que mais da metade dos caracteres do conjunto utilizado ocupam 2 bytes. Importante ressaltar que a codificação de caracteres padrão é a UTF-8, por tanto os caracteres podem ter de 1 a 4 bytes de tamanho. Em termos de tempo, o ganho de velocidade observado na transmissão de mídia foi de cerca de 64.4%.

6 ARQUITETURA

6.1 Modelo de Comunicação

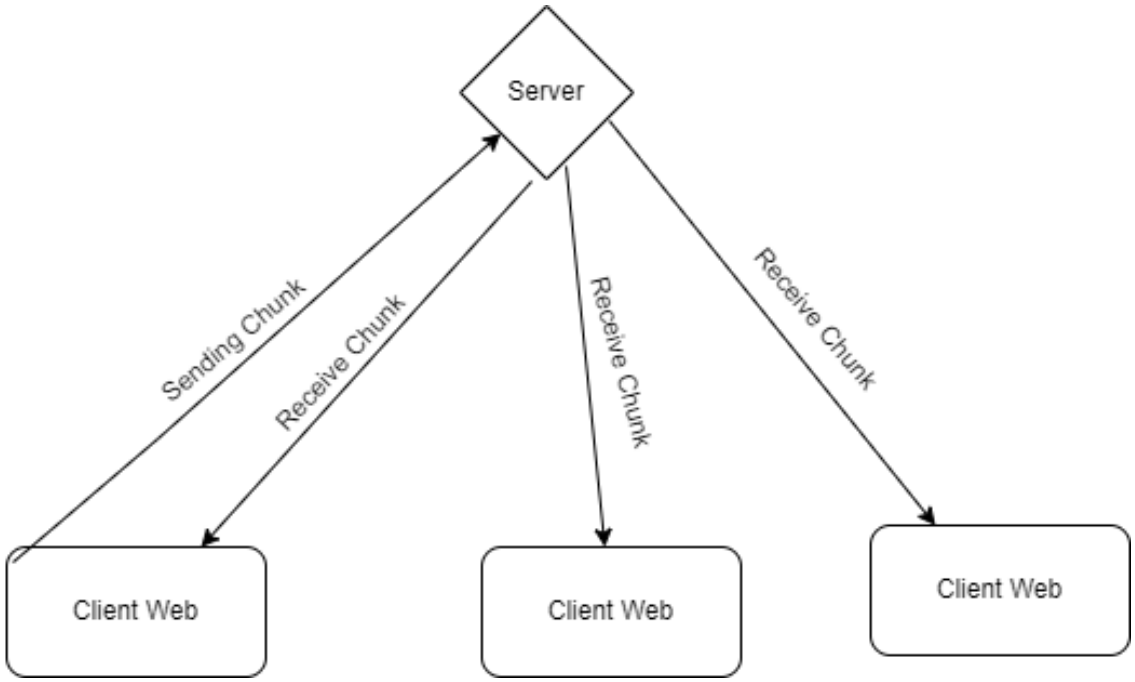

Figura 9: Envio de Mídia

Na aplicação a troca de mensagens segue um modelo com estratégia P2P, como já mencionado. Cada cliente envia uma mensagem e os demais recebem, em caso de texto todo o texto é enviado, em caso de mídia ela seria enviada em partes(ou pedaços). O fluxo funciona basicamente com um cliente enviando uma mensagem para o servidor, e ele reenviando ela em seguida para todos da sala, incluindo o próprio cliente. O servidor faz apenas o repasse, sem persistir ou processar as mensagens.

6.2 Proposta de Arquitetura

A arquiteura atual do sistema é composta por um cliente web e um servidor monolito, que também é responsável por servir os arquivos do cliente. Foi elaborado desta forma inicialmente por simplicidade, está sendo mantido assim por conta de limitações de infraestrutura. No momento a infraestrutura da aplicação permite apenas um contêiner, o que dificulta uma arquiteura mais elaborada.

Posteriormente a ideia é que a arquiteura evolua para poder escalar mais, uma vez que mude de infraestrutura e torne isso possível, o resultado deve ser algo assim:

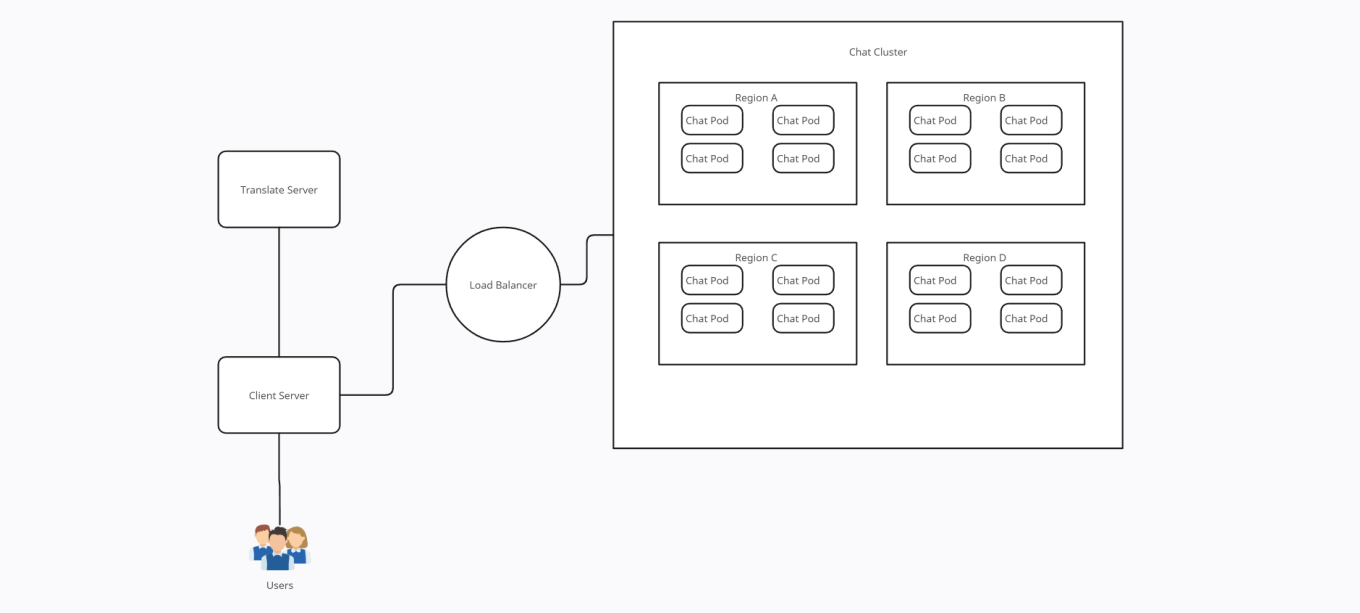

Figura 10: Futura Arquitetura

Esta nova arquitetura deve permitir que a aplicação escale muito mais futuramente, ela é composta por:

- **Client Server:** O servidor de arquivos estáticos do cliente exclusivamente para ele, o que permite remover do servidor do chat a sobrecarga de ter que entregar o cliente, e que não tem tanta necessidade de escalar, já que só é acessado quando a aplicação é carregada.
- **Translate Server:** O servidor que fornece os dicionários de cada idioma para o cliente. Separá-lo do cliente web deixa-o mais flexível e retutilizável. Seu acesso ocorre na mesma frequência do **Client Server** e por tanto, sua estrutra é simples.
- **Chat Cluster:** Um cluster com vários pods(instâncias) do **Chat Server**, onde cada servidor de chat fica totalmente isolado um do outro, permitindo assim uma escala horizontal eficiente. Outro ponto importante é a separação por região, a ideia é que seriam usados 2 critérios para escolher qual servidor o usuário seria conectado: disponibiliade e proximidade. O servidor mais vazio da região mais próxima seria preferido, distribuindo bem os usuários e garantindo uma melhor latência. A escolha do servidor seria feita durante a criação da sala, para garantir a comunicação, os demais usuários que entrarem na sala iriam automaticamente para o servidor correto(no qual a sala se encontra), isso poderia ser feito colocando a referência ao servidor correto no link compartilhado.

7 AVALIAÇÃO DA PROPOSTA

Esta seção tem como objetivo analisar a proposta prática deste trabalho por meio de uma pesquisa qualitativa com usuários.

Para avaliar a aplicação desenvolvida como um software de qualidade, dado que seu uso é feito por pessoas(e não por outros sistemas, como é o caso de micro serviços), uma ótima forma de fazer isso é pondo seu uso em prática e solicitar avaliação dos usuários. A pesquisa foi feita usando o Google Forms e foi distribuída para diferentes tipos de pessoas: acadêmicos, jovens comuns, pessoas mais velhas e com pouco contato com tecnologia...

7.1 Pesquisa

A maioria das questões são por escala de 1 a 5 para facilitar o questionário em termos de satisfação, uma questão perguntando se o usuário usaria a aplicação, e por fim uma questão opcional onde o usuário poderia informar um comentário adicional. Esses foram os resultados:

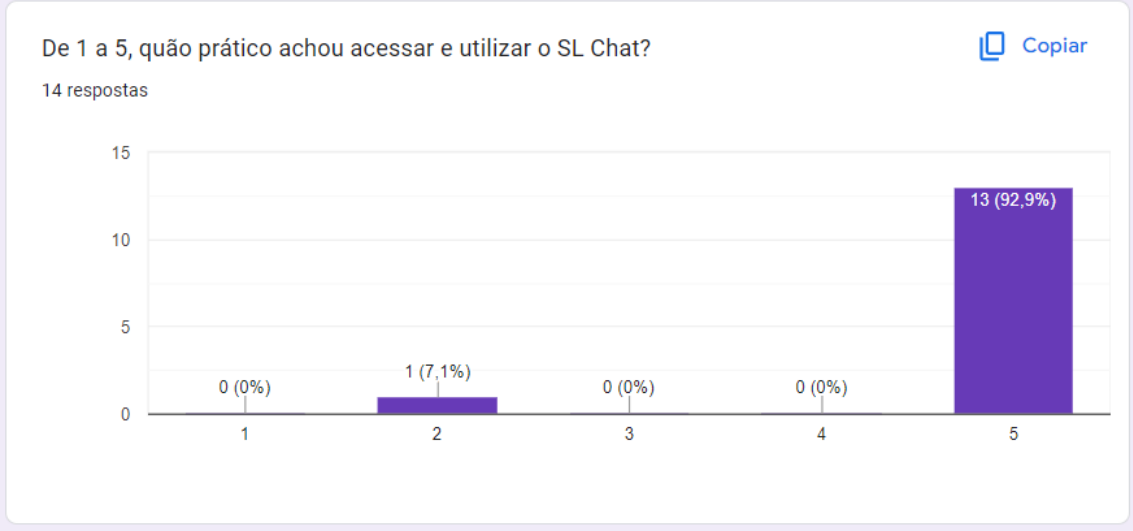

Figura 11: Resultado da Pesquisa Q1

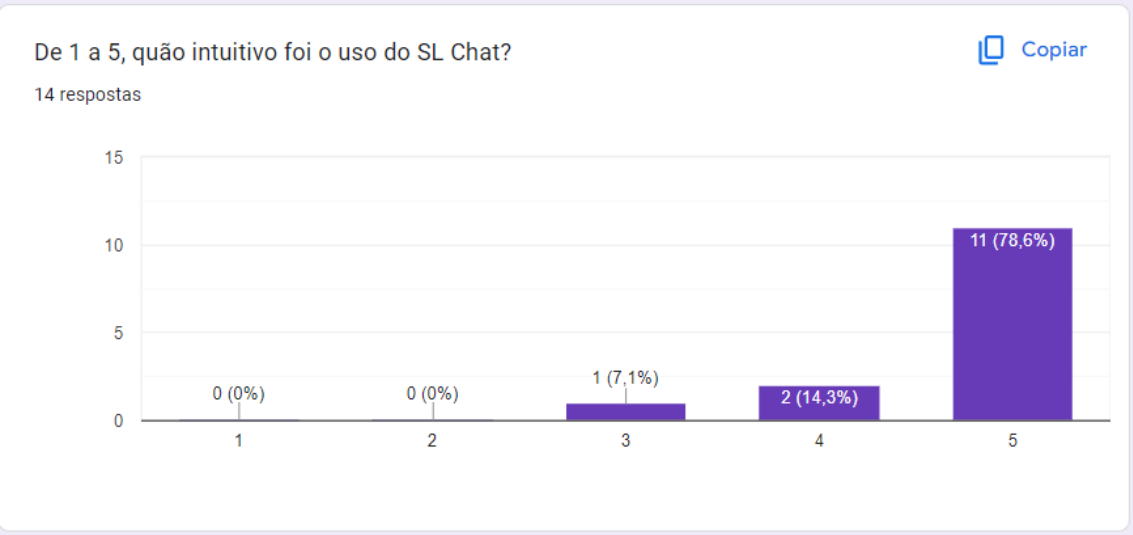

Figura 12: Resultado da Pesquisa Q2

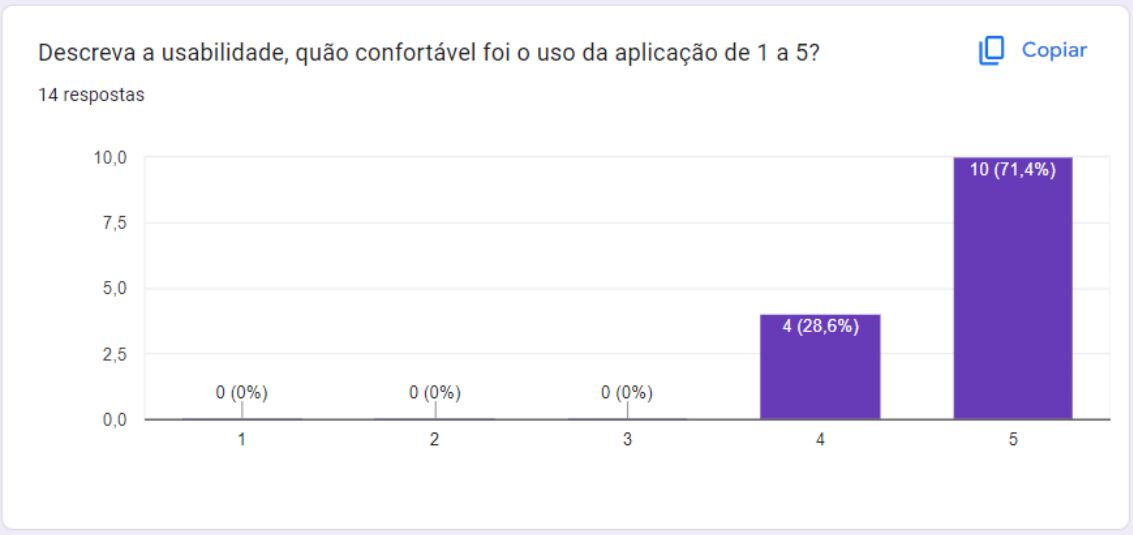

Figura 13: Resultado da Pesquisa Q3

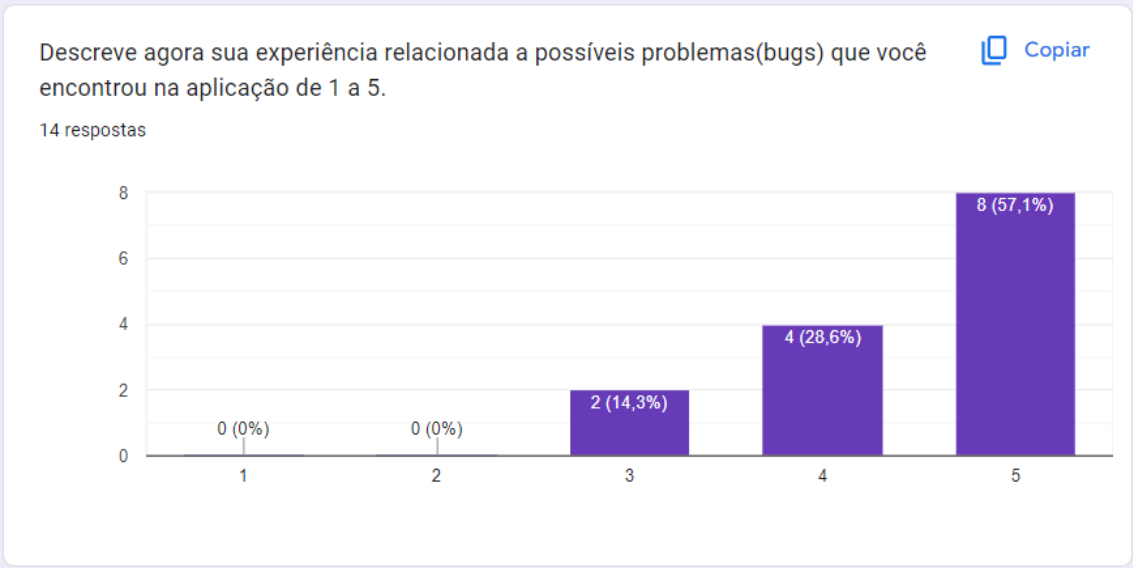

Figura 14: Resultado da Pesquisa Q4

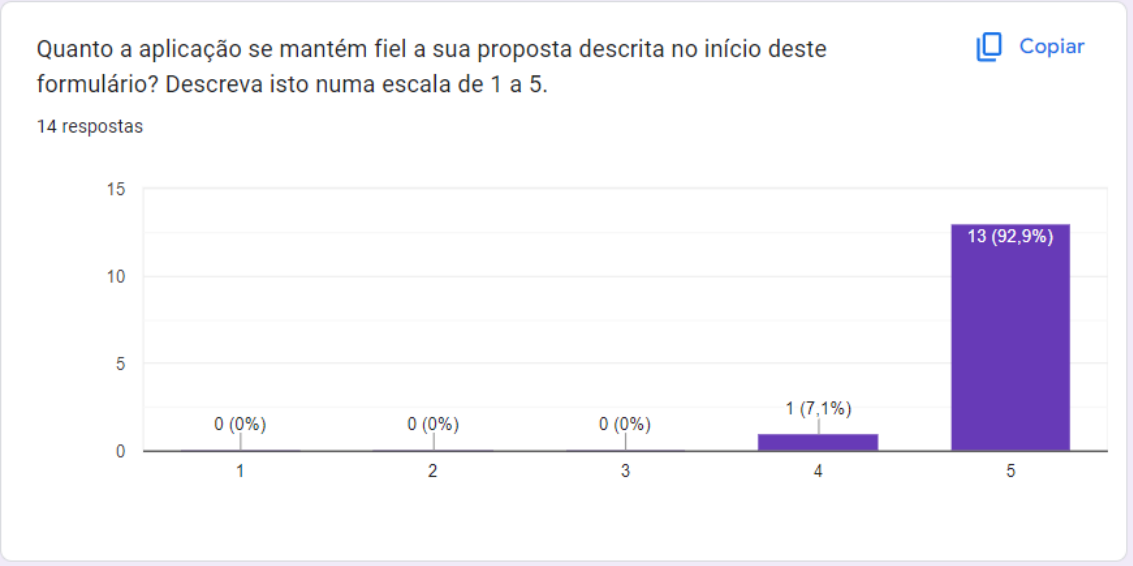

Figura 15: Resultado da Pesquisa Q5

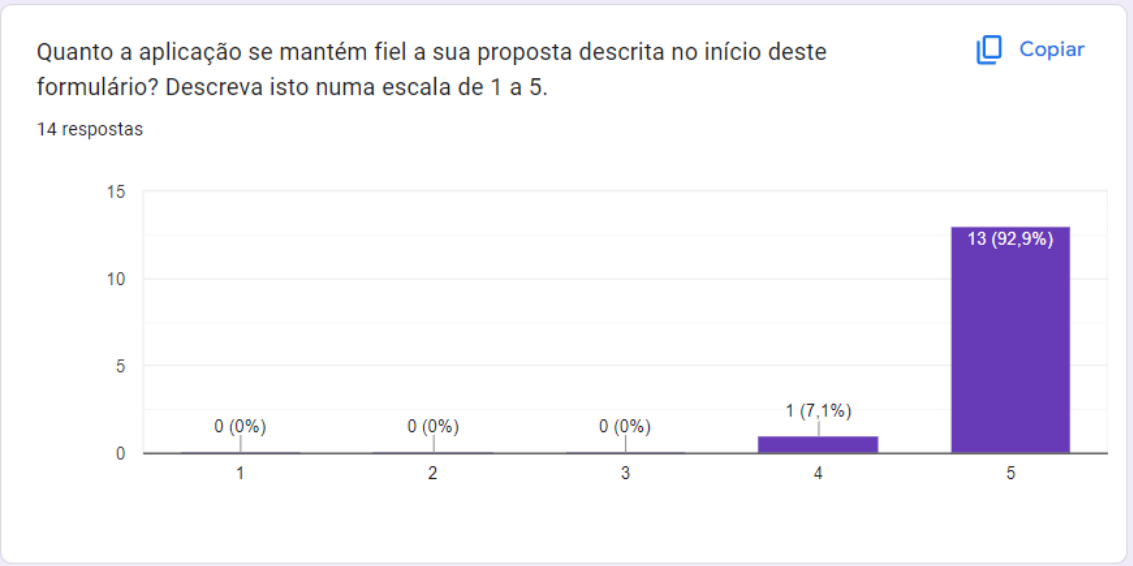

Figura 16: Resultado da Pesquisa Q5

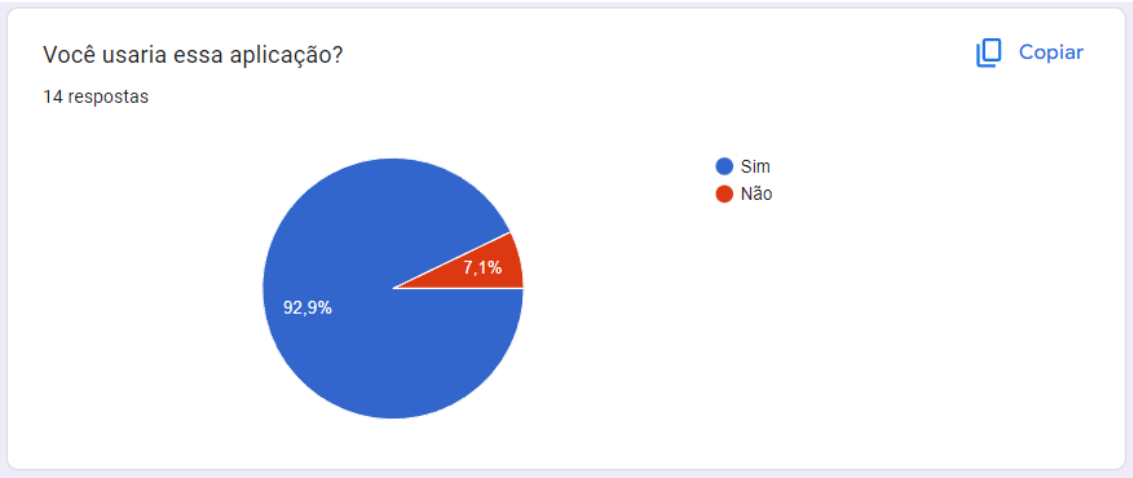

Figura 17: Resultado da Pesquisa Q6

7.2 Análise de Resultados

Resumindo as notas da aplicação em médias gerais de 1 a 5, e a porcentagem das pessoas que afirmaram que usariam:

- **Praticidade:** 4.8
- **Intuitivo:** 4.7
- **Usabilidade:** 4.7
- **Bom Funcionamento/Ausência de Bugs:** 4.4
- **Fidelidade a Proposta:** 4.9
- **Porcentagem das Pessoas que Usariam:** 92.9%

Um ponto importante a se considerar é que a pesquisa não foi feita no melhor ambiente possível, possivelmente sua avaliação teria sido melhor, em especial na parte referente a bugs. Isso por conta da infraestrutura escolhida para publicação da aplicação.

7.2.1 Dificuldades

Infelizmente uma escolha ruim na cloud trouxe um viés negativo para a pesquisa sobre a aplicação, a cloud back4app[18] trouxe recentemente um modelo de deploy com docker incluindo um plano gratuito, bastante atrativo e útil, mas trouxe um certo prejuízo na pesquisa por ainda está em **beta**:

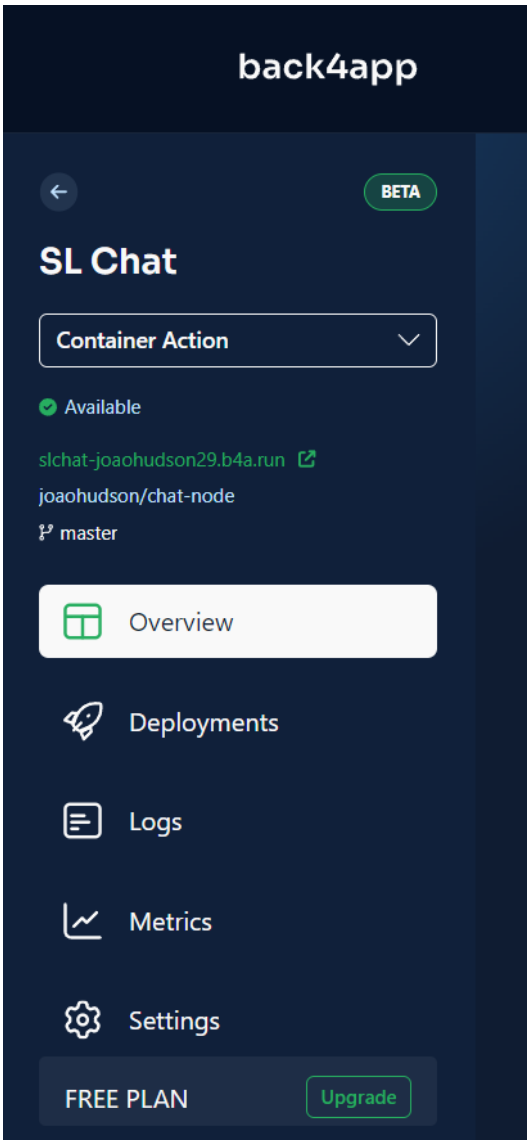

Figura 18: Cloud em Beta

Um comentário em particular, um dos que pontuaram no quesito *ausência de bug* com nota 3, descreveu bem o cenário onde a cloud gerava instabilidade:

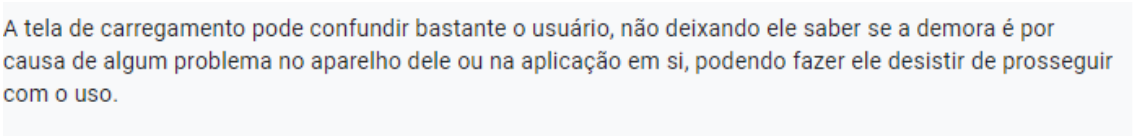

Figura 19: Crítica de Loop Infinito

O que aconteceu nesse caso foi um problema com o gateway da cloud, onde o mesmo retornava *GATEWAY_TIMEOUT* em alguns dos arquivos estáticos do cliente. O problema desse tipo de erro é que o cliente não consegue ser totalmente carregado e não tem um jeito prático de contornar via software. O suporte da cloud foi notificado e trabalharam em cima deste problema, porém o resultado acabou permanecendo na pesquisa.

7.3 Consumo de Recursos

Em termo de consumo de recursos computacionais, as observações inciais são bem positivas.

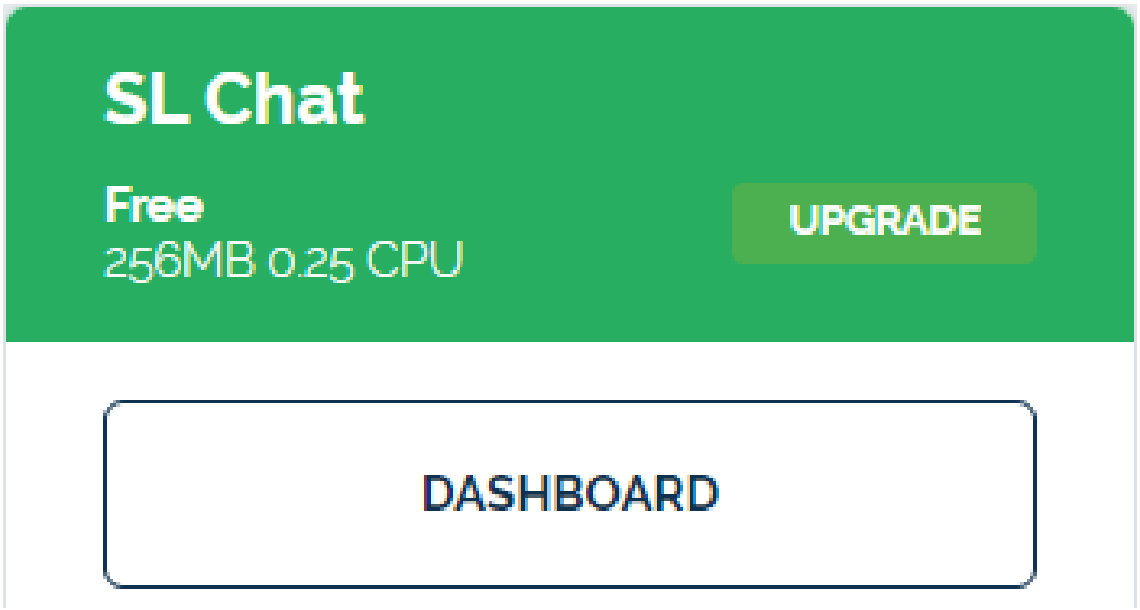

Figura 20: Servidor Bem Simples

Com apenas **0.25GHZ** de velocidade de CPU e **256MB** de memória RAM. Temos uma máquina bem simples, que apesar de tudo, suportou vários chats paralelos e envio de mídia superior a quantidade de memória do próprio servidor.

8 CONCLUSÕES E TRABALHOS FUTUROS

Nesse trabalho foi apresentada uma proposta para construção de um Web Chat Online, totalmente privado, livre(sem necessidade de cadastro) e com envio ilimitado de mídias. O objetivo da aplicação SL Chat é trazer uma solução para situações onde o usuário se sinta desconfortável com seus dados e suas mensagens salvas pelas empresas, além de cenários onde o tamanho das mídias sejam limitados, ou mesmo o processo burocrático de cadastro e instalações de aplicativos. De uma maneira geral a aplicação busca desburocratizar alguns cenários de comunicação mais privados. Provendo não necessariamente uma concorrência com os chats já existentes, mas uma alternativa para certas circunstâncias.

A aplicação está disponível para uso limitado, devido a sua infraestrutura e arquiteura atual. Para o futuro espera-se efetuar a troca de cloud para uma infraestrutura mais robusta, estável e que traga suporte a mútiplos serviços. Então seguir com a implementação da nova arquitetura escalonável, para permitir que a aplicação atinja um público bem maior e em várias regiões diferentes, com uma boa latência. Espera-se também aumentar o número de idiomas suportados, que no momento são apenas 3: português, ingês e espanhol.

REFERÊNCIAS

[1] *Algoritmo Elegante.* https://www.ime.usp.br/~pf/algoritmos/aulas/footnotes/cs-concepts.html.

[2] *A criptografia de ponta-a-ponta é um recurso de segurança que protege os dados durante uma troca de mensagens, de forma a que o conteúdo só possa ser acessado pelos dois extremos da comunicação.* https://pt.wikipedia.org/wiki/Criptografia_de_ponta-a-ponta.

[3] *ES6 é a sexta versão do padrão da linguagem JavaScript.* http://es6-features.org/#Constants.

[4] *Um identificador único universal (do inglês universally unique identifier - UUID) é um número de 128 bits usado para identificar informações em sistemas de computação.* https://pt.wikipedia.org/wiki/Identificador_%C3%BAnico_universal.

[5] *WebSocket é uma tecnologia que permite a comunicação bidirecional por canais full-duplex sobre um único soquete Transmission Control Protocol (TCP).* https://pt.wikipedia.org/wiki/WebSocket.

[6] Brian Acton, Jan Koum: *WhatsApp.* Versão Web disponível em: <https://web.whatsapp.com/> Acesso em: Nov 2009. https://web.whatsapp.com/.

[7] Dahl, Ryan: *Nodejs is an open-source, cross-platform JavaScript runtime environment.* Disponivel em: <https://nodejs.org/en> Acesso em: 27 May 2009. https://nodejs.org/en.

[8] Eich, Brendan: *JavaScript é uma linguagem de programação interpretada estruturada, de script em alto nível com tipagem dinâmica fraca e multiparadigma.* Mais informações disponível em: <https://pt.wikipedia.org/wiki/JavaScript>, Acesso em: 4 Dec 1995. https://pt.wikipedia.org/wiki/JavaScript.

[9] Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides: *Chain Of Responsability.* https://pt.wikipedia.org/wiki/Chain_of_Responsibility.

[10] Google: *Mútiplos Idiomas na Aplicação Android.* Disponível em: <https://developer.android.com/training/basics/supporting-devices/languages?hl=pt-br> Acesso em: 30 Apri 2009. https://developer.android.com/training/basics/supporting-devices/languages?hl=pt-br.

[11] Holowaychuk, TJ: *Express.js é um framework para Node.js que fornece recursos mínimos para construção de servidores web.* Disponível em: <https://expressjs.com/> Acesso em: 16 Nov 2010. `https://expressjs.com/`.

[12] Hykes, Solomon: *Docker é um conjunto de produtos de plataforma como serviço que usam virtualização de nível de sistema operacional para entregar software em pacotes chamados contêineres.* Disponível em: <https://www.docker.com/> Acesso em: 12 Mar 2013. `https://www.docker.com/`.

[13] Linus Torvalds, Junio Hamano: *Git é um sistema de controle de versões distribuído, usado principalmente no desenvolvimento de software, mas pode ser usado para registrar o histórico de edições de qualquer tipo de arquivo.* Disponível em: <https://git-scm.com/> Acesso em: 07 Apr 2005. `https://git-scm.com/`.

[14] Microsoft: *TypeScript is JavaScript with syntax for types.* Disponível em: <https://www.typescriptlang.org/>, Acesso em: 1 Ago 2021. `https://www.typescriptlang.org/`.

[15] OpenAI: *ChatGPT é um assistente virtual inteligente no formato chatbot online com inteligência artificial desenvolvido pela OpenAI, especializado em diálogo.* Disponível em: <https://chat.openai.com/> Acesso em: Jun 2020. `https://chat.openai.com/`.

[16] Rauch, Guillermo: *Is an event-driven library for real-time web applications.* Disponível em: <https://socket.io/pt-br/> Acesso em: 4 Nov 2010. `https://socket.io/pt-br/`.

[17] Schlueter, Isaacf Z.: *npm é um gerenciador de pacotes para o Node.JS.* Disponível em: <https://www.npmjs.com/> Acesso em: 12 Jan 2010. `https://www.npmjs.com/`.

[18] Tomazic, Nik: *Low-code backend to build modern apps.* `https://www.back4app.com/`.