FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS – FATECS
CURSO DE ENGENHARIA DA COMPUTAÇÃO

Pablo Felipe Santos Carneiro
RA: 21804056

Victor Augusto de Sousa Azevedo
RA: 21805359

SISTEMA DE INTEGRAÇÃO ENTRE MENSAGEIROS

BRASÍLIA
2022

PABLO FELIPE SANTOS CARNEIRO
VICTOR AUGUSTO DE SOUSA AZEVEDO

SISTEMA DE INTEGRAÇÃO ENTRE MENSAGEIROS

Trabalho de Conclusão de Curso (TCC) apresentado como um dos requisitos para a conclusão do curso de Engenharia de Computação do CEUB– Centro Universitário de Brasília

Orientador (a)**: Me. Francisco Javier de Obaldia Diaz**

BRASÍLIA
2022

PABLO FELIPE SANTOS CARNEIRO
VICTOR AUGUSTO DE SOUSA AZEVEDO

SISTEMA DE INTEGRAÇÃO ENTRE MENSAGEIROS

Trabalho de Conclusão de Curso (TCC) apresentado como um dos requisitos para a conclusão do curso de Engenharia de Computação do CEUB – Centro Universitário de Brasília

Orientador (a)**: Me. Francisco Javier de Obaldia Diaz**

Brasília, 2022.

BANCA EXAMINADORA

_____
Me. Francisco Javier De Obaldía Díaz
Orientador (a)

_____
Dra. Ingrid Maria Dittert
Examinador (a)

_____
Me. Tiago Leite Pereira
Examinador (a)

Sistema de integração entre mensageiros
Integration system for messengers

Pablo Felipe Santos Carneiro e Victor Augusto Sousa Azevedo[1],
Francisco Javier de Obaldia Diaz[2], Ingrid Maria Dittert[3], Tiago Leite Pereira[4]

RESUMO

A popularização da internet e principalmente do *smartphone* tornou possível a comunicação instantânea entre pessoas através de várias aplicações chamadas mensageiros. Porém, esta variedade de *software* também trouxe uma fragmentação do mercado de mensageiros devido às preferências pessoais dos usuários. Este projeto busca trazer uma solução viável, simples e acessível para a unificação de grupos de diferentes mensageiros, permitindo a comunicação de usuários de plataformas diferentes. O objetivo é atingido pelo método de pontes (*bridges*) representadas e realizadas por *bots*[1] em cada plataforma trabalhando em conjunto para espelhar o conteúdo de uma plataforma nas outras, assim garantindo a sincronia dos grupos e permitindo a comunicação em tempo real entre usuários de plataformas diferentes.

PALAVRAS-CHAVE: Ponte. Mensageiro. Integração

ABSTRACT

The popularization of the internet and especially of the smartphone enabled people to communicate instantaneously through various applications called messengers. Nevertheless, this variety of software also brought a fragmentation of the messenger scene, due to each user's preferences. This project seeks to bring a viable, simple and accessible solution to the unification of chat rooms in different messengers. This objective is reached by the use of bridges, represented and performed by bots in each platform working together to mirror the contents of one platform in the others, thus ensuring the synchronization of chat rooms and allowing real-time communication between users of different platforms.

KEYWORDS: Bridge. Messenger. Integration

---

[1] CEUB, alunos.
[2] CEUB, orientador.
[3] CEUB, primeira examinadora.
[4] CEUB, segundo examinador.

1 INTRODUÇÃO

A proliferação da *internet* e dos *smartphones* trouxe uma série de facilidades às vidas de seus usuários, como os mensageiros instantâneos. Trouxe, também, mercados vibrantes e competitivos para todo o tipo de *software* para os quais há demanda, inclusive para a área dos mensageiros. Devido à variedade de soluções existentes neste mercado, se vê presente uma fragmentação no uso destes aplicativos, que possuem funcionalidades e usabilidade parecidas, o que torna a sua base de usuários o principal atrativo de uma solução de mensageiro instantâneo. Este projeto busca ser uma solução simples para a unificação de grupos em diferentes plataformas, garantindo a sincronia de mensagens entre grupos interconectados por *bots* em suas respectivas plataformas.

Há iniciativas com propostas similares: o Matrix, por exemplo, é um padrão aberto descentralizado interoperável com outros sistemas de comunicação. Oferece, a diversas plataformas, sistemas de *bridging* com seus serviços de grupos e salas de conversa, oferecendo uma experiência integrada ao usuário. (matrix.org, setembro de 2022).

Também pode-se citar o projeto Matterbridge, que possui foco na plataforma Mattermost, mas que também permite o uso do método de *bridging* entre outras plataformas, e que se diferencia deste projeto na implementação, já que usa Go como linguagem de programação, e não Node.js/Typescript, como este (mattermost.com, setembro de 2022).

Há, também, várias iniciativas menores que cumprem, a grosso modo, o mesmo objetivo geral deste projeto. Muitas, porém, não contam com atualizações há algum tempo, não funcionam mais, não têm suporte pleno a todas as funcionalidades das plataformas que abrangem, e na maioria dos casos só possuem suporte a duas plataformas. Esta iniciativa busca oferecer uma sistema modular com suporte a múltiplas plataformas e capacidade de suportar uma alta quantidade de grupos ao mesmo tempo, permitindo o uso do sistema por vários usuários diferentes.

Aqui, busca-se uma solução acima de tudo simples de se configurar e utilizar, funcionando de tal forma que um usuário comum, sem conhecimento técnico, possa utilizar uma instância já hospedada do projeto, e, caso queira hospedar sua própria, deve também ser, dentro do possível, simples de se configurar. Além disso, o sistema foi construído de forma a facilitar o desenvolvimento do suporte a novas plataformas no futuro. Esta modularidade permite que não haja dependência de plataformas específicas para o funcionamento do *software*, oferecendo uma série de funções e métodos agnósticos que devem ser chamadas pelos módulos de plataforma, permitindo que se use a infraestrutura do projeto de forma independente, acelerando o desenvolvimento do suporte a novas plataformas.

Este projeto foi desenvolvido em TypeScript para Node.js versão 16.14. Sendo este o requisito mínimo de

software para funcionar, versões inferiores ou superiores ao descrito do Node.js não são suportadas para o funcionamento do projeto.

2 REVISÃO BIBLIOGRÁFICA

A fundamentação teórica para o desenvolvimento deste projeto tem como base as referências bibliográficas sobre o estudo e efetivação do modelo de *bridging* por *bots* junto à documentação de bibliotecas das respectivas plataformas na linguagem de programação Typescript.

2.1 TypeScript

É uma linguagem de programação desenvolvida pela Microsoft que amplia o JavaScript, através de um compilador que transforma código TypeScript em JavaScript, preservando assim a compatibilidade com as *engines* presentes no mercado, como o Node.js e todos os navegadores modernos. A sua utilização proporciona funcionalidades de tipagem como interfaces, *enums* e especificação de tipos de variáveis e parâmetros. Isto permite que se detectem erros de tipagem em tempo de compilação, o que não ocorre no JavaScript, que, por ser uma linguagem interpretada e com tipagem fraca, somente apresentará estes potenciais problemas em tempo de execução (typescriptlang.org, novembro de 2022).

2.2 JSON

O *JavaScript Object Notation* (JSON, Notação de Objeto JavaScript) é um formato leve de transferência de dados. Criado com o intuito de ser compreensível para leitura e escrita humana, fornece uma maneira simples de converter objetos JS em texto para envio e leitura, potencialmente a *softwares* escritos em outras linguagens. É baseado em um subconjunto da linguagem de programação JavaScript que segue o padrão ECMA-262. Na realidade o JSON é um formato de texto independente da linguagem própria para o seu uso. Pode ser utilizado desde a família C até as mais recentes como o Python. Essas características tornam o JSON o ideal para a função que ele cumpre (json.org, novembro de 2022).

2.3 Node.js

O Node é um programa de código aberto, multiplataforma que atua como plataforma de *back-end*. Trata-se de um ambiente de execução JavaScript, operando uma *engine* e executando código JavaScript em um ambiente de servidor (nodejs.org, novembro de 2022).

2.4 NPM

O *Node Package Manager* (NPM, Gerenciador de Pacotes do Node) gerencia a instalação e desinstalação de pacotes escritos pela comunidade NPM em um projeto Node.js, permitindo a adição de novas funcionalidades ao projeto (npmjs.com, novembro de 2022).

2.5 Discord.js

Documentação necessária da API (*Application Programming Interface*) do Discord dedicada a fornecer a interface necessária para o desenvolvimento de

bots para Discord. O discord.js é um pacote NPM que utiliza a API oficial da plataforma, e é recomendado oficialmente pelo Discord para o desenvolvimento de aplicações de *bot* em JavaScript/TypeScript. Suas características de orientação a objeto, flexibilidade, riqueza de funcionalidades, rapidez e eficiência a tornam uma poderosa ferramenta baseada totalmente em *promise*[9], indispensável para o desenvolvimento do módulo responsável pelo suporte à plataforma Discord (discord.js.org, novembro de 2022).

2.6 Discord.js *Guide* pelos desenvolvedores do Netlify.

Guia completo de como desenvolver um *bot* no Discord, sendo que os principais tópicos do guia são: como criar, organizar e expandir a lista de comandos disponíveis e implementados pelo *bot*, tutorial inicial de como programar o *bot* para enviar uma mensagem, até como fazê-lo reagir a algum comando ou mensagem ou enviar uma mensagem anexada como uma resposta (discordjs.guide, setembro de 2022).

2.7 *Telegraf.js framework documentation.*

Documentação do *framework* Telegraf. Telegraf é um pacote npm para o desenvolvimento de *bots* para Telegram (telegraf.js.org, novembro de 2022).

2.8 FS (File System Module)

O File System é um uma biblioteca nativa do Node.js que permite trabalhar com arquivos de sistema do próprio computador. Com ele é possível ler, criar, atualizar, deletar e renomear arquivos através de métodos fornecidos pela biblioteca (nodejs.org, novembro de 2022)..

2.9 Queue

Queue é uma biblioteca disponível no NPM que oferece uma implementação prática de filas. Um objeto do tipo Queue, fornecido pela biblioteca, possui métodos similares aos de um *array* comum, como *push*, que insere uma função na fila. Funções inseridas na fila são executadas na ordem FIFO (*First In, First Out*, ordem de chegada) automaticamente. Esta biblioteca foi utilizada para implementar as filas de envio de cada Conexão[2], melhorando a confiabilidade e resiliência do sistema, pois garante-se que as mensagens serão distribuídas na ordem em que chegam ao Distribuidor[3], e que sempre haverá somente uma mensagem sendo distribuída simultaneamente por Conexão (npmjs.com, novembro de 2022).

3 METODOLOGIA DO TRABALHO

O primeiro passo para o desenvolvimento da aplicação consistiu na pesquisa por projetos existentes que cumprissem objetivos similares. Há alguns projetos muito grandes e complexos, que fogem do escopo deste, e há outros pequenos, normalmente abandonados, que não cumprem todos

os objetivos deste projeto ou simplesmente não funcionam mais. Não obstante, esta experiência serve como base para novas ideias e métodos de implementação.

Há alguns métodos de se atingir a proposta deste projeto, e o método de *bridging* por *bots* foi o escolhido após o estudo entre os demais. É o mais simples de ser implementado e utilizado: consiste no uso de *bots* que espelham o conteúdo de um grupo em outro de outra plataforma. Foi considerado o mais apropriado para o projeto pois, além de ser de fácil implementação, também não requer conhecimento técnico por parte do usuário final - basta convidar os *bots* aos grupos e repassar um código de sincronia de um para o outro - e oferece uma experiência totalmente dentro do ecossistema da plataforma do usuário.

Posteriormente, o desenvolvimento se deu focando, primeiramente, nas plataformas Telegram e Discord, priorizando-se as funcionalidades básicas.

A etapa seguinte deu-se pela implementação de funcionalidades mais complexas envolvendo mídias, arquivos e outras funcionalidades das plataformas suportadas .

Por fim, foram realizados testes e obtidos resultados para aferir a velocidade média do funcionamento de todas as funcionalidades do projeto.

4 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

O projeto foi feito de forma a permitir o uso modular das plataformas: no diretório `platforms`, ficam os arquivos referentes às plataformas que se deseja fazer uso (os “módulos”). Estes arquivos devem ter um objeto chamado manifest[6] que implemente a interface `PlatformManifest`, especificada no arquivo `utils.ts`. Este objeto possui as seguintes propriedades:

```
interface PlatformManifest {
  name: string,
  start: Function,
  needsToken?: boolean;
  send: Function,
  ready?: boolean;
}
```

- `name`: especifica o nome da plataforma, por exemplo, `'telegram'`.
- `start`: função que inicia o bot da respectiva plataforma, por exemplo, `(token: string) => start(token)`, em que `start` é uma função implementada no mesmo arquivo módulo da plataforma.
- `send`: função que recebe o grupo/sala de destino e uma instância de `MessageObject`. Por exemplo:

```
async (
group: string,
message:
utils.MessageObject
): Promise<void> =>
send(group, message)
```

O módulo da plataforma deve conter os procedimentos necessários para converter a

mensagem em texto, apresentando o remetente e a plataforma de origem corretamente formatados conforme o funcionamento da plataforma em questão. Neste caso, este trabalho é feito pela função `send`, que tem o mesmo nome da propriedade que ocupa, e deve estar implementada no mesmo arquivo módulo da plataforma.
- `needsToken`: indica se a propriedade `start` requer um token como parâmetro. Se `true`, este token deve estar presente no arquivo `.env`.
- `ready`: inicialmente `undefined`, deve ser alterado para `true` pela rotina da plataforma quando tudo estiver pronto para enviar e receber mensagens.

Esta abordagem permite que o desenvolvimento e adição de plataformas seja completamente modular, ou seja, nenhuma lógica referente a qualquer plataforma está *hard-coded* no projeto, e seus módulos podem ser retirados e adicionados à pasta *platform* sem qualquer alteração no resto do projeto, assim como o carregamento destes arquivos é feito de forma automática pelo projeto. Algumas plataformas requerem *tokens* dos *bots* que serão controlados pelo programa. Estes *tokens* devem estar presentes no arquivo de variáveis de ambiente.

Quando o programa é iniciado, ele tenta ler dois arquivos que podem ou não estar presentes na raiz do projeto:

- `.env`: o arquivo de variáveis de ambiente, se usado, deve estar populado seguindo a lógica `<plataforma>=<token>` para todas as plataformas que se pretende utilizar. O nome da plataforma deve ser o mesmo declarado em seu `manifest`. O *token* indicado deve ser o código de *bot* fornecido pela aplicação, que permitirá a identificação e uso de uma conta *bot* existente para os fins do projeto.
- `config.json`: este arquivo é gerado pelo próprio programa, salvando as configurações de Conexões, permitindo que se preserve estes dados entre sessões diferentes.

Com a leitura dos *tokens* do arquivo de variáveis de ambiente, o projeto assume controle das contas *bot* às quais os *tokens* pertencem. Estas contas já precisam existir neste momento, e cabe ao interessado em hospedar o projeto criá-las da maneira que a plataforma espera e instrui.

Com o projeto devidamente configurado e em execução, deve-se então configurar as Conexões entre os grupos que se deseja integrar. O usuário interessado deve, então, convidar os *bots* das respectivas plataformas a estes grupos. Então, deve-se enviar o comando “`/start`“ na conversa em que o *bot* se encontra, para que seja gerado um código de Conexão. Este código deve ser enviado no grupo da outra plataforma, e, assim, o sistema de Conexões registra a configuração na

memória e, para persistência por múltiplas sessões, no arquivo `config.json`. Feito este processo, as mensagens enviadas nestes grupos serão encaminhadas para os outros grupos da Conexão. O código gerado tem validade de 5 minutos e pode ser reutilizado em quantas plataformas for necessário, repetindo-se este processo e aditando à memória e ao arquivo de Conexões. Também é criada uma fila de envio para cada nova Conexão, garantindo a ordem de envio de mensagens e capacidade de lidar com grandes quantidades de mensagens simultaneamente.

Com pelo menos uma Conexão feita, esta contendo ao menos dois grupos de plataformas diferentes, o projeto passa a cumprir seu objetivo de integrar estes grupos, encaminhando todas as mensagens de um para o outro.

O processo de recebimento e encaminhamento de mensagens pode ser separado em etapas que abordam desde a obtenção da mensagem recebida, em um objeto proveniente do *framework* da plataforma de origem, potencialmente com mídias e texto. Estas informações são extraídas de forma a gerar um objeto padronizado, para que este seja distribuído às outras plataformas. As etapas de recebimento são definidas como:

4.1 Primeira etapa, módulo da plataforma de origem

Receber o objeto original da mensagem detectada, no formato da plataforma. Os dados relevantes são extraídos do objeto original para gerar um novo objeto, padronizado, para manuseio pelo resto do programa. Este trabalho de extração e padronização do objeto original deve ser feito pelo módulo da plataforma em que se originou a mensagem. O novo objeto deve seguir um padrão esperado pelo resto do projeto, definido pela seguinte interface TypeScript:

```typescript
interface MessageObject {
  platform: string;
  type: string;
  id: string;
  content: {
    text: string;
    images: Array<Attachment>;
  };
  reply_to: MessageObject | false;
  sender: {
    id: string;
    user: string;
    name: string;
    isAdmin?: boolean;
  };
  group: string;
}
```

- `platform`: plataforma de origem da mensagem;
- `type`: de qual tipo se trata a mensagem, podendo ser um texto, imagem, vídeo, sticker e áudio;
- `content`: conteúdo da mensagem, seja ele composto por texto, mídia ou arquivos.
- `reply_to`: caso a mensagem seja uma resposta a alguma outra, indica a mensagem respondida, senão, é `false`.
- `sender`: indica o emissor da

mensagem na plataforma de origem. Estes dados serão utilizados para a formatação da mensagem encaminhada às outras plataformas;
- `group`: identifica o grupo no qual se originou a mensagem.

Nesta fase, também é feito o *download* de mídias ou arquivos presentes na mensagem. Estes arquivos são baixados em uma pasta temporária e têm uma vida útil de um minuto: após este tempo, são excluídos da pasta de arquivos temporários. Este diretório também é limpo quando o programa é iniciado.

Com o objeto criado, a mensagem agora possui uma representação padronizada e está pronta para ser encaminhada para o Distribuidor.

4.2 Segunda etapa, distribuidor

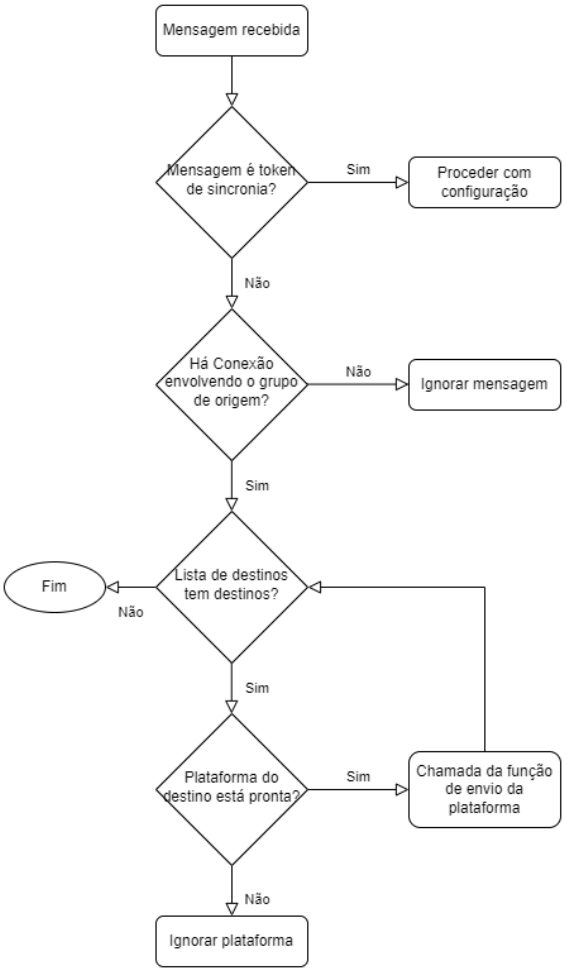
Consiste no recebimento da mensagem, já no formato padronizado, pelo Distribuidor e o cumprimento de seu papel. Esta parte do programa é implementada no arquivo `distributor.ts`, e tem a função de receber mensagens e distribuí-las aos destinatários corretos, conforme as configurações de Conexões.

Quando o Distribuidor recebe uma mensagem ele busca, na lista de Conexões, a Conexão que inclui o grupo de origem da mensagem. Caso não haja tal Conexão, a mensagem é ignorada. O procedimento completo realizado pelo distribuidor é representado na figura 1.

Com a Conexão correspondente devidamente identificada, o código referente ao encaminhamento da mensagem aos destinos corretos é adicionado à fila da Conexão. Quando chega a vez desta mensagem, ela é encaminhada para cada um dos outros grupos presentes no objeto referente à Conexão, com exceção do grupo de origem. Este encaminhamento se dá através da função `send` presente no manifesto da plataforma em questão.

A próxima, terceira e última etapa é realizada para todos os grupos presentes na Conexão, exceto o de origem.

(Figura 1. Fluxograma do distribuidor)



(Fonte: Autor, 2022)

4.3 Terceira etapa, módulo da plataforma de destino

O objeto padronizado repassado pelo Distribuidor terá suas informações extraídas e repassadas à função do *framework* da plataforma. Isto também inclui o trabalho de gerar o cabeçalho, indicando o remetente e a plataforma de origem da mensagem que está sendo enviada.

4.4 Metodologia para testes e resultados

No funcionamento prático do projeto, a experiência de uso nas mensagens de texto não apresenta de forma perceptível qualquer tempo de latência adicional quando comparado à da própria mensagem recebida na aplicação mensageira original, com isso, para uma abordagem mais precisa, foram feitos testes de pacotes de rede no programa analisador de tráfego de redes *Wireshark* para verificar se há diferença considerável de tempo no recebimento da mensagem dentro desse ambiente de análise e coleta.

Os testes foram realizados em um computador portátil Lenovo Ideapad S145 81V7S00100, que compõe um processador Ryzen 5 de terceira geração modelo 3500u, 12GB ram 2666mhz, que utiliza uma placa de rede WLAN Qualcomm Atheros QCA9377 com protocolo de conexão Wi-Fi 4(IEEE 802.11ac) nas bandas 2.4/5Ghz. Este foi o dispositivo escolhido pois a maioria dos usuários das plataformas as utilizam no sistema *mobile*[7] que por sua vez utiliza adaptadores *wireless* para a comunicação.

O cálculo de adição de atraso é feito através da diferença de tempo, em milissegundos, entre os *logs*[5] colhidos no programa Wireshark do primeiro pacote recebido da plataforma original (primeira linha) e o primeiro pacote colhido da plataforma destino (última linha).

4.5 Testes com mensagens de texto

Na primeira parte da realização dos testes foi utilizada, como remetente, uma conta do Discord, e, como destinatários, outra conta no Discord e uma conta do Telegram, esta última que receberia a mensagem indiretamente através do funcionamento do projeto. Assim, é possível comparar os tempos e medir a performance do sistema. Neste ambiente, foram obtidos os resultados apresentados na tabela 1:

(Tabela 1. Diferença em milissegundos entre as plataformas Discord e Telegram com mensagens de texto)

Discord para Telegram	Atraso adicionado, em milisegundos (ms):
Teste 1:	222 ms
Teste 2:	197 ms
Teste 3:	195 ms
Teste 4:	278 ms
Teste 5:	234 ms

(Fonte: Autor, 2022)

Com os dados obtidos da tabela 1, conclui-se que a média de atraso adicionado nas condições do teste é de

225 milissegundos, apresenta um desvio padrão de ±30,28 milissegundos.

(Figura 2. Imagem respectiva a tabela 1, Discord para Telegram. Teste 3)

0.000000	162.159.133.234
0.050487	192.168.2.6
0.195747	149.154.175.50

(Fonte: Autor, 2022)

(Figura 3. Imagem respectiva a tabela 1, Discord para Telegram. Teste 4)

1.091153	162.159.133.234
1.133440	192.168.2.6
1.369504	149.154.175.50

(Fonte: Autor, 2022)

Também realizou-se o teste no sentido contrário, com o remetente no Telegram e dois destinatários: um no mesmo grupo e outro no grupo integrado, na plataforma Discord, obtendo a tabela 2:

(Tabela 2. Diferença em milissegundos entre as plataformas Telegram e Discord com mensagens de texto)

Telegram para Discord	Atraso adicionado, em milisegundos (ms):
Teste 1:	371 ms
Teste 2:	324 ms
Teste 3:	407 ms
Teste 4:	450 ms
Teste 5:	386 ms

(Fonte: Autor, 2022)

Assim, a média de atraso adicionado é de 388 milissegundos e apresenta ±66,32 milisegundos de desvio padrão.

(Figura 4. Imagem respectiva a tabela 2, Telegram para Discord. Teste 2)

0.585218	149.154.175.50
0.586182	192.168.2.6
0.756353	149.154.175.50
0.909056	162.159.133.234

(Fonte: Autor, 2022)

(Figura 5. Imagem respectiva a tabela 2, Telegram para Discord. Teste 4)

0.000000	149.154.175.50
0.048233	192.168.2.6
0.450522	162.159.133.234

(Fonte: Autor, 2022)

4.6 Testes com mídia de imagem

Na segunda parte da realização de testes, com mensagens de mídias, foi utilizada como remetente uma conta do Telegram, e como destinatários outra da mesma plataforma e uma no grupo integrado do Discord. O teste foi realizado com uma foto, de 6,52kB, da logo do CEUB, apresentada na figura 8. Neste ambiente, foram obtidos os resultados da tabela 3.

(Figura 8: Imagem da logo do CEUB referencial para o teste)

(Fonte: CEUB, 2022)

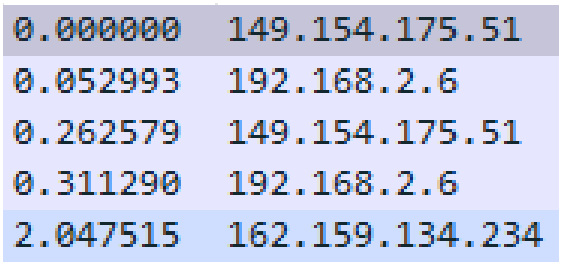
(Tabela 3: Diferença em milissegundos entre as plataformas Telegram e Discord com mídias)

Telegram para Discord Imagem	Atraso adicionado, em milisegundos (ms):
Teste 1:	1944 ms
Teste 2:	1947 ms
Teste 3:	1945 ms
Teste 4:	1685 ms
Teste 5:	1843 ms

(Fonte: Autor, 2022)

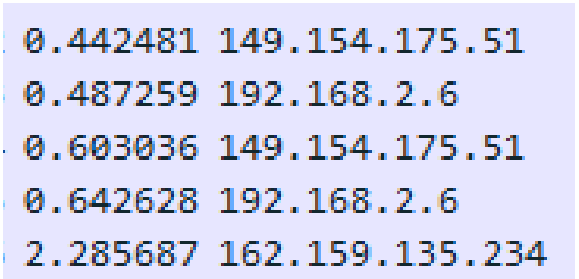
Assim, a média de atraso adicionado é de 1873 milissegundos , com um desvio padrão de 101,93 milisegundos.

(Figura 6. Imagem respectiva a tabela 3, Telegram para Discord. Teste 2)



(Fonte: Autor, 2022)

(Figura 7. Imagem respectiva a tabela 3, Telegram para Discord. Teste 5)



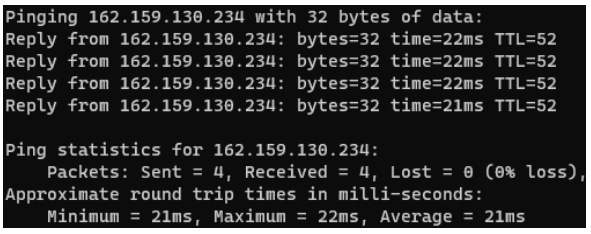
(Fonte: Autor, 2022)

Nota-se uma flutuação considerável nos valores obtidos. Esta flutuação se dá por fatores alheios ao projeto, na forma com a qual os servidores da plataforma lidam com mídias.

4.7 Análise dos pacotes

É importante ressaltar que boa parte da diferença entre as mensagens geradas do Discord e Telegram se dá pela diferença de servidores, tipo de conexão e suas respectivas localizações, que acabam aumentando a latência do pacote como um todo. A fim de se realizar um estudo mais profundo dos resultados, foi feito um teste de *ping*[8] através do terminal de comandos do Windows para verificar a latência padrão dos servidores. O endereço 162.159.130.234 é advindo da análise dos pacotes do Discord. Este servidor possui uma média de 21 milissegundos de *ping* como representado na figura 9:

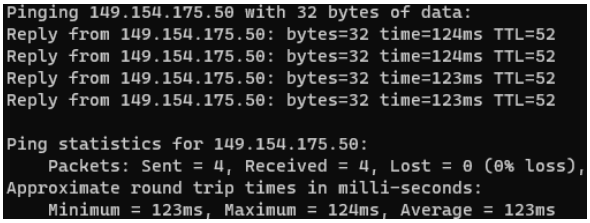
(Figura 9. Teste de ping do servidor 162.159.130.234)



(Fonte: Autor, 2022)

Já o servidor 149.154.175.50 vem da análise dos pacotes do Telegram, e apresenta uma média de *ping* de 123 milissegundos, como representada na figura 10:

(Figura 10: Teste de ping do servidor 149.154.175.50)



(Fonte: Autor, 2022)

Esta discrepância de mais de 100 milissegundos se dá, em sua maior parte, pela geolocalização dos respectivos servidores. Com a utilização de uma extensão de navegador[4] *iPvFoo* foi possível identificar o nome das conexões dos servidores para a pesquisa da sua localização.

(Figura 11: Resultados obtidos com iPvFoo com a conexão do Discord)

	discord.com	162.159.136.232
	brazil8681.discord.media	162.159.138.234 S
	cdn.discordapp.com	162.159.133.233 ↻
	gateway.discord.gg	162.159.134.234 S
	status.discord.com	162.159.136.232 ↻

(Fonte: Autor, 2022)

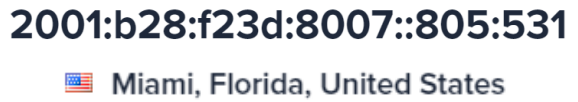
Assim sendo, o servidor do Discord, conforme a figura 11, é localizado no continente sul-americano, propriamente no Brasil.

(Figura 12: Resultados obtidos com iPvFoo com a conexão do Telegram)

	web.telegram.org	2001:67c:4e8:f004::9
	t.me	2001:67c:4e8:f004::9
	telegram.me	2001:67c:4e8:f004::9
	zws1-1.web.telegram.org	2001:b28:f23d:8007::805:532
	zws1.web.telegram.org	2001:b28:f23d:8007::805:531
	zws2.web.telegram.org	2001:67c:4e8:f004::9
	zws4.web.telegram.org	2001:67c:4e8:f004::9
	zws5.web.telegram.org	2001:b28:f23f:9::852:438

(Fonte: Autor, 2022)

(Figura 13: Resultado de pesquisa do IP zws1 obtido no iPvFoo)



(Fonte: ipinfo.io, 2022)

O servidor do Telegram, como apresentado nas figuras 12 e 13, o servidor zws1 (conexão primária realizada nos testes) é localizado no continente norte-americano, nos Estados Unidos da América.

Durante os testes também foi notada a diferença de modernidade de protocolos usados por cada plataforma: o Telegram utiliza o protocolo SSL enquanto o Discord utiliza o TLSv1.2, protocolo mais moderno, oriundo do próprio SSL.

5 CONSIDERAÇÕES FINAIS

O projeto cumpre os objetivos pretendidos e apresenta utilidade real, realizando de forma satisfatória sua missão de integrar grupos em plataformas diferentes, melhorando e facilitando a comunicação entre usuários com preferências diferentes de plataforma mensageira. A performance do projeto sofre grandes variações a depender de uma série de fatores, principalmente relacionados ao plano contratado para a hospedagem do projeto e latência do provedor do usuário com os demais servidores. Porém, mesmo nestas situações, o projeto continua oferecendo uma experiência razoável aos usuários interessados.

Enquanto este projeto apresenta uma solução viável para um problema real, podendo ser de grande utilidade para melhorar o cenário da comunicação na internet contemporânea, deve-se atentar a alguns aspectos: o impacto social de seu uso indiscriminado e as adequações à legislação pertinente.

As facilidades proporcionadas pelo uso do projeto podem ter um impacto na disseminação de mentiras e notícias falsas: torna-se automática a disseminação de informações de uma plataforma para outra, o que pode, em casos de mau uso, ter até mesmo um impacto socialmente negativo. Porém, não é viável, do ponto de vista do escopo deste projeto, controlar o uso da aplicação. É importante lembrar que as mensagens replicadas de uma plataforma a outra precisam obedecer às regras de ambas as plataformas, portanto ainda estão sujeitas a pedidos de remoção judiciais e da própria plataforma, cabendo a estes o papel de regular o conteúdo transmitido nas redes.

Outra consideração importante é a adequação à legislação relevante, especialmente à LGPD. A aplicação não coleta dados pessoais, e, por mais que o projeto tenha acesso a todas as mensagens trocadas nos grupos em que opera, nenhuma informação sobre as mensagens é armazenada a longo prazo. Porém, é importante salientar que é imprescindível obter o consentimento de todos os presentes nos grupos e salas de conversa em que se pretende fazer uso deste projeto. Não fazê-lo seria uma violação de privacidade.

AGRADECIMENTOS

Agradecemos ao nosso professor orientador, cujo apoio e conhecimentos foram fundamentais para o desenvolvimento deste projeto, e a todo o corpo docente que nos auxiliou na nossa trajetória acadêmica. Também agradecemos aos nossos familiares, cujo apoio foi imprescindível para a continuidade e permanência no curso, e a todos os queridos amigos que fizemos no processo.

REFERÊNCIAS

Discord.js. (https://discord.js.org/#/docs/discord.js/main/general/welcome, 2022).
Desenvolvedores do Netlify, Discord.js *Guide.* (https://discordjs.guide/#before-you-begin, 2022).
Telegraf.js framework documentation. (https://telegraf.js.org, 2022).
Matrix. (https://matrix.org, 2022).
JSON - JavaScript Object Notation (https://www.json.org/json-en.html, 2022).
Red Hats. What is an API? (https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces, 2022).
Node.js. (https://nodejs.org/en/docs, 2022).
Portal Discord. (https://discord.com/developers/docs/topics/community-resources#libraries, 2022).
ipinfo.io (https://ipinfo.io/2001:b28:f23d:8007::805:531, 2022)
TypeScript: JavaScript with Syntax for

Types.
(https://www.typescriptlang.org, 2022)
Introducing JSON
(https://www.json.org, 2022)
Node.js
(https://nodejs.org, 2022)
Node Package Manager
(https://npmjs.com, 2022)

GLOSSÁRIO

1. *Bot*: no âmbito deste projeto, uma conta *bot* é um usuário de uma plataforma que apresenta comportamento controlado por *software*, respondendo a eventos internos ou externos à plataforma.
2. Conexão: configuração com dois ou mais grupos, de plataformas diferentes, com o intuito de fornecer ao Distribuidor informações quanto ao destino de mensagens originadas em grupos presentes na Conexão.
3. Distribuidor: parte deste projeto responsável por distribuir mensagens para os destinos corretos.
4. Extensão de navegador: pequeno módulo de programa adicionado para nova funcionalidade ou personalizar um navegador web.
5. *Log*: série de registros de acontecimentos gravados em um certo período de tempo.
6. *Manifest*: objeto que indica as principais funções do módulo da plataforma para uso pelo resto do programa.
7. *Mobile*: dispositivo móvel.
8. *Ping*: comando do protocolo de mensagens de controle da Internet para testar a conectividade entre dispositivos, também utilizado para verificar o tempo de resposta entre eles.
9. *Promise* (Promessa): na programação JavaScript, uma promessa é um objeto que representa a incerteza quanto ao estado de conclusão de uma operação ou função assíncrona que retorna algum valor. Enquanto a operação assíncrona não é finalizada, o estado da promessa é *pending*. Conforme a operação evolui, a promessa pode ser resolvida (*resolved*, sucesso) ou rejeitada (*rejected*, falha).