UNIVERSIDADE DO SUL DE SANTA CATARINA

ÂNIMA EDUCAÇÃO

MARCEL VIEIRA RIBEIRO

SAGI A: APLICAÇÃO ROBÔ PARA ADMINISTRAÇÃO DE SERVIDORES DE DISCORD

Tubarão

2023

MARCEL VIEIRA RIBEIRO

SAGI A: APLICAÇÃO ROBÔ PARA ADMINISTRAÇÃO DE SERVIDORES DE DISCORD

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade do Sul de Santa Catarina, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Aran Bey Tcholakian Morales

Tubarão

2023

MARCEL VIEIRA RIBEIRO

SAGI A: APLICAÇÃO ROBÔ PARA ADMINISTRAÇÃO DE SERVIDORES DE DISCORD

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade do Sul de Santa Catarina, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação.

Florianópolis, 13 de junho de 2023.

Professor e orientador Aran Bey Tcholakian Morales, Dr.
Universidade do Sul de Santa Catarina

Professor Saulo Popov Zambiasi, Dr.
Universidade do Sul de Santa Catarina

Professor Flávio Ceci, Dr.
Universidade do Sul de Santa Catarina

RESUMO

Neste projeto, é abordado o desenvolvimento de um protótipo de aplicação bot para Discord, explorando os desafios associados a esta tarefa. A motivação se deu pela crescente popularidade do Discord, que apesar de inicialmente ser famoso entre os gamers, se consolidou como uma plataforma de comunicação online, utilizada por pessoas de todo o mundo para situações cotidianas ou que representam diferentes momentos da vida como trabalho, lazer ou estudar. O que pode resultar na exposição dos usuários a vulnerabilidades ou ataques de cibercriminosos. Com isso em mente, foram implementadas funcionalidades administrativas e recreativas no bot, permitindo a sua interação com os membros dos servidores em que é convidado. Desta maneira, um bot interativo pode aprimorar a experiência dos usuários em suas interações, permitindo maior monitoria e segurança além de agregar valor às suas comunidades. Ao decorrer do desenvolvimento, também foi identificada a necessidade de criar um ambiente de desenvolvimento adequado, além de destacar a importância das ferramentas e tecnologias selecionadas.

Palavras-chave: *Bot*. Aplicativo. *Discord*. Plataforma.

ABSTRACT

This project addresses the development of a bot application prototype for Discord, exploring the challenges associated with this task. The motivation was due to the growing popularity of Discord, which despite being initially famous among gamers, has established itself as an online communication platform, used by people from all over the world for everyday situations or that represent different moments of life such as work, leisure or study. Which may result in users being exposed to vulnerabilities or attacks by cybercriminals. With that in mind, administrative and recreational functionalities were implemented in the bot, allowing its interaction with the members of the servers in which it is invited. In this way, an interactive bot can improve the users' experience in their interactions, allowing greater monitoring and security, in addition to adding value to their communities. During development, the need to create an adequate development environment was also identified, in addition to highlighting the importance of the selected tools and technologies.

***Keywords**: Bot. Application. Discord. Platform.*

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Os games têm obtido grande importância no universo digital e na indústria do entretenimento, impulsionando o mercado, o que torna sua produção uma atividade promissora (SCHWAB, 2004). Desde o ano de 2014 empresas de consultoria como a Newzoo publicam em seus sites, resultados anuais da indústria de jogos, contendo previsões de crescimento da receita do mercado global de games. Isto permeia, não só na imprensa, como também no meio acadêmico. Como é o exemplo do trabalho intitulado “A Representatividade das Mulheres nos Games”, realizado por Paula Casagrande Bristot, Eliane Pozzebon e Luciana Bolan Frigo no programa de Pós-Graduação em Tecnologias da Informação e Comunicação (PPGTIC) da Universidade Federal de Santa Catarina (UFSC) e publicado pela XVI SBGames em novembro de 2017. Bristot, Pozzebon e Frigo (2017, p. 862) descrevem em sua pesquisa que a Newzoo “é líder global de inteligência de mercado especializada em games” e que “em 2016 a indústria de games gerou uma receita superior a 90 bilhões de dólares, com cerca de 2,1 bilhões de jogadores ao redor do mundo”.

Em menos de uma década depois, as previsões da Newzoo têm se popularizado através da mídia com chegada de valores duas vezes maiores. Segundo a publicação em julho de Wijman (2022, p. 1, tradução nossa), “nosso último relatório anual [julho de 2022] mostra que serão 3.2 bilhões de jogadores no mundo. E eles contribuirão para uma receita de até $196.8 bilhões ao final de 2022, com crescimento de 2.1% ao ano”.

Wijman (2022, p. 1, tradução nossa) também explica em sua publicação que o mercado tende a crescer ainda mais nos próximos anos, mesmo diante de condições econômicas desafiadoras. E chama o mercado de games de “até certo ponto, ‘à prova de recessão’”. Embora alguns *gamers* sensíveis ao preço possam gastar menos em tempos econômicos mais difíceis, o engajamento permanecerá alto. Hoje, a monetização gratuita (free-to-play) e os serviços de assinatura significam que os jogos estão mais democratizados do que nunca (WIJMAN, 2022)

Para Bristot, Pozzebon e Frigo (2017, p. 868):

> O game é a mídia que proporciona maior interação e tem grande poder de influência, pelo fato do jogador estar no controle do personagem. Essa experiência de controle proporciona uma maior identificação com o personagem e permite que o jogador se projete e se coloque na situação dele tendo maior sensação de participação.

Mas Martoni, da Silva e Pagani (2018, p. 5) apontam um problema de comunicação comum nos games cooperativos ou *multiplayer*: “a atenção dos jogadores é desviada para os textos, o que pode prejudicar a equipe durante a partida. Por conta disso, muitos jogadores preferem utilizar ferramentas de comunicação externas ao jogo”. O universo de games ou jogos eletrônicos online é vasto, e apesar de já existir softwares (programas) para comunicação, em 2015 Jason Citron e Stan Vishnevskiy criaram o Discord, software que facilitou conversas de texto, voz ou vídeo (DISCORD COMPANY, 2023, tradução nossa). Greenberg (2019, p. 12, tradução nossa) aponta que “Discord é a maior plataforma de chat sobre games do mundo, com 250 milhões de usuários”.

Ao embarcar nesta pesquisa fez-se necessário compreender um pouco do público alvo e do crescimento da plataforma Discord.

1.1 PROBLEMÁTICA

É de conhecimento público o crescimento do Discord e que está também associado a impactos no mercado de games. Em 2018 a preferência pelo discord não parou de crescer, o aplicativo ganhou novos 45 milhões de usuários registrados, um total de 130 milhões de usuários naquele ano e registrava 15 bilhões de mensagens por mês (KIMBALL, 2018). Atualmente, a plataforma se encontra com 150 milhões de usuários ativos por mês, 19 milhões de servidores ativos por semana e 4 bilhões de minutos de conversas diárias nos servidores (DISCORD COMPANY, 2022).

Uma outra perspectiva de seu crescimento foi quando houve o anúncio de parceria com a Sony o que resultaria na integração do canal de comunicação com a PlayStation Network (PSN). Feito por Jim Ryan, presidente da Sony Interactive

Entertainment (SIE), o anúncio informava também que a empresa teria feito investimentos minoritários na plataforma (PALMEIRA, 2021).

Mas com o crescimento e popularização também vieram ameaças de cibercrime como afirma Paula e Nazaré (2020, p. 3), as pessoas estão correndo um risco diferente a exposição ao coronavírus, que é a exposição à vulnerabilidades nas aplicações e ataques de cibercriminosos, exploradas através da utilização de *softwares* de videoconferências.

Segundo a Brewster (2019, tradução nossa) em sua publicação no site da *Forbes,* no início de 2019 o FBI investigou comunidades de Discord cujos membros poderiam estar envolvidos em atividades criminosas, como os "*HellsGate*" que oferecia milhares de contas online diferentes ou os "*SentryMBA*" que oferecia softwares com credenciais falsas e que poderiam automatizar processos de inserção de nomes de usuários e senhas em vários sites.

Desde então percebe-se diversas atualizações de segurança na plataforma. Diante desse cenário de ameaças e riscos, surge o questionamento: como gerenciar um servidor de Discord, tendo em mente ferramentas auxiliares que ofereçam maior moderação sobre os membros e mensagens a fim de mitigar ou evitar tais problemas?

1.2 OBJETIVOS

Esta seção apresenta os objetivos geral e específicos do presente estudo.

1.2.1 Objetivo Geral

O presente projeto objetiva desenvolver uma ferramenta com funcionalidades capazes de impactar na administração de um servidor de Discord, permitindo que a moderação do servidor tenha respostas ágeis para resolver problemas dos membros e ter maior controle dos canais de texto e voz.

1.2.2 Objetivos Específicos

- Estabelecer ferramentas e tecnologias tanto essenciais quanto adicionais para o desenvolvimento do bot como linguagem, softwares e bibliotecas;
- elaborar e classificar funcionalidades que possam gerenciar membros e mensagens em servidores de Discord;
- concluir o protótipo e disponibilizá-lo em código aberto.

1.3 JUSTIFICATIVA

Como afirma Abhinand e Balasubramanian (2022, p. 213, tradução nossa): “Discord continua a melhorar seus recursos, eficiência e modelo de negócios e só deve se tornar uma entidade maior no futuro”. O Discord também fornece um portal para os desenvolvedores inovarem, com recursos para integrar bots ou aplicações, cada uma utilizando bibliotecas já existentes para várias linguagens de programação. Abhinand e Balasubramanian (2022, p. 213, tradução nossa) explica ainda que usando sua API, uma variedade de aplicações podem ser criadas e cada uma pode ser adicionada a um número ilimitado de guildas (termo técnico para servidor) com a exigência de passar por um processo de autorização.

Um bom exemplo de como as aplicações impactam e somam à experiência dos usuários é o desenvolvimento do *Kanmani*, uma ferramenta que facilita a entrada dos alunos no processo de estudo, auxiliando na produtividade somando parâmetros de bem-estar durante a pandemia do COVID-19 (ABHINAND; BALASUBRAMANIAN, 2022, p. 215, tradução nossa).

Ou ainda quando aprimoram a própria plataforma. Raglianti et al. (2021, p. 1, tradução nossa) desenvolveram o DISCORDANCE, uma ferramenta que permite superar problemas de pesquisa explorando o histórico completo de mensagens de um servidor de Discord, com uma alta taxa de transferência e volatilidade.

Estudar as possibilidades de aplicações auxiliares de Discord para diferentes áreas tem sido um desafio intrigante para todos que se interessam, especialmente aos acadêmicos de cursos de tecnologia como ciências da computação. Tendo em vista estas possibilidades, foi decidido explorar ferramentas e funcionalidades adicionais que possam ajudar administradores de servidores de Discord nas

questões de gerenciamento de membros e mensagens com o objetivo de facilitar sua segurança e administração no geral.

1.4 ESTRUTURA DA MONOGRAFIA

O presente estudo está estruturado em quatro capítulos e as considerações finais, sendo que o primeiro capítulo é composto pela introdução, com os objetivos, problemática e justificativa, que conduzirão os estudos. O segundo capítulo contém a revisão bibliográfica que fundamenta teoricamente o trabalho permitindo uma ampla noção da plataforma baseando-se no meio acadêmico. No terceiro capítulo está o método que conduz a pesquisa e no quarto capítulo está detalhada a implementação a ser experimentada em ambiente simulado. Para finalizar são apresentadas as conclusões e o referencial teórico.

2 REVISÃO BIBLIOGRÁFICA

Ao longo do tempo, os avanços tecnológicos têm impactado a natureza e o uso da nossa linguagem bem como a forma como nos comunicamos, o que remonta a era das redes sociais. A tecnologia de impressão, por exemplo, não apenas padronizou a ortografia, mas também contribuiu para o aumento da alfabetização. O surgimento de novos meios tecnológicos para produzir, gravar ou transmitir a linguagem ajudaram a moldar a linguagem escrita e falada e a relação entre elas (BARON, 2009, p.65, tradução nossa).

Para Baron (2009, p. 69, tradução nossa), *Computer Mediated Communication* (CMC) é definido como “a noção de que computadores podem ser utilizados como dispositivos com propósitos de transmitir linguagem em rede, entre si”. “Chat” ou bate-papo, é um local CMC síncrono para manter conversas com vários participantes. Este modelo como é conhecido agora nasceu em 1988, quando Jarkko Oikarinen, na época estudante na Universidade de Oulu na Finlândia, escreveu um programa que veio a ser conhecido como Internet Relay Chat (IRC). No início dos anos 1990, o IRC tornou-se conhecido do público em geral, servindo como modelo para programas de bate-papo mais genéricos disponíveis por meio de provedores de Internet como America Online (BARON, 2009, p. 76, tradução nossa).

Em seu trabalho, Tori, Kirner e Siscoutto (2006) abordam definições a respeito de ambiente virtual:

> Representações da realidade ou da imaginação sempre fizeram parte da vida do ser humano permitindo-o expressar-se ao longo do tempo, desde desenhos primitivos, figuras e pinturas até cinema, passando por jogos, teatro, ópera, ilusionismo e outras expressões artísticas. O uso do computador potencializou e convergiu tais formas de expressão, viabilizando a multimídia, que envolve textos, imagens, sons, vídeos e animações, e mais recentemente a hipermídia, que permite a navegação não linear e interativa por conteúdos multimídia. Ao mesmo tempo, os vídeo-games ganharam um espaço extraordinário, explorando a interação.

Adiante, Recuero (2012, p. 21) define que estas ferramentas tecnológicas de comunicação e expressão “permitem a interconexão entre as pessoas, mantendo canais permanentes de circulação de informação: são as redes sociais amplificadas pela mediação do computador”. As conexões online foram inicialmente concebidas como um substituto para o contato face a face, quando este último era impraticável. Mas sob esse aspecto, a mensagem de texto se tornou rapidamente a forma

preferida por muitos. Descobriu-se que a rede ou mundo da conectividade, é adequada exclusivamente para a vida corrida e sobrecarregada que ela possibilita (TURKLE, 2011, p. 13). O que é destacado é a mudança sutil no uso do ambiente, anteriormente com uma premissa pessoal e posteriormente foi levado a uma situação nova, a informação passou a ser construída, trocada e valorada. Onde "o que você está fazendo?" passa a ser "o que está acontecendo?" (RECUERO, 2012, p. 21). Moraes (2015, p. 8) lembra que “a ferramenta tecnológica em questão reúne, em um só dispositivo, canais de comunicação, informação e entretenimento, itens mais do que suficientes para atrair os usuários.”

É necessário apontar como tais tecnologias impactam nossas vidas também de forma negativa, quando autores apontam o uso das redes sociais de forma a influenciar as informações e podendo até serem utilizadas para gerar desinformação ou confusão. Como aponta Jardine (2019, p. 32), informando que os esforços para influenciar informações possuem longa história, mas atualmente a escala potencial de operações de influência é decisivamente afetada pelas novas plataformas digitais. Plataformas como Facebook, Twitter, WhatsApp, Telegram e Viber, que possuem um grande número de usuários. A crescente sofisticação da inteligência artificial (IA) e dos algoritmos de aprendizado de máquina também aponta para uma nova mudança qualitativa potencial nas operações de influência. Geralmente, as pessoas tendem a confiar um pouco menos na palavra escrita do que no áudio e, em particular, na mídia de vídeo (JARDINE, 2019).

Dentro do contexto de plataformas de redes sociais e o impacto em nossa comunicação, é examinado a plataforma Discord, que foi a escolhida para a realização deste trabalho.

2.1 DISCORD E COMUNIDADE

Vários programas de software que permitem voz sobre protocolo de Internet ou VoIP (*voice over internet protocol*) foram desenvolvidos no início dos anos 2000. Os mais populares na época eram Ventrilo, TeamSpeak e Mumble, lançados por volta de 2002 a 2005 e são usados principalmente para jogos. O Discord é um dos mais utilizados atualmente e é considerado um descendente destes (BANKOV, 2019, tradução nossa). Segundo Mello (2021) estes programas “transmitem e

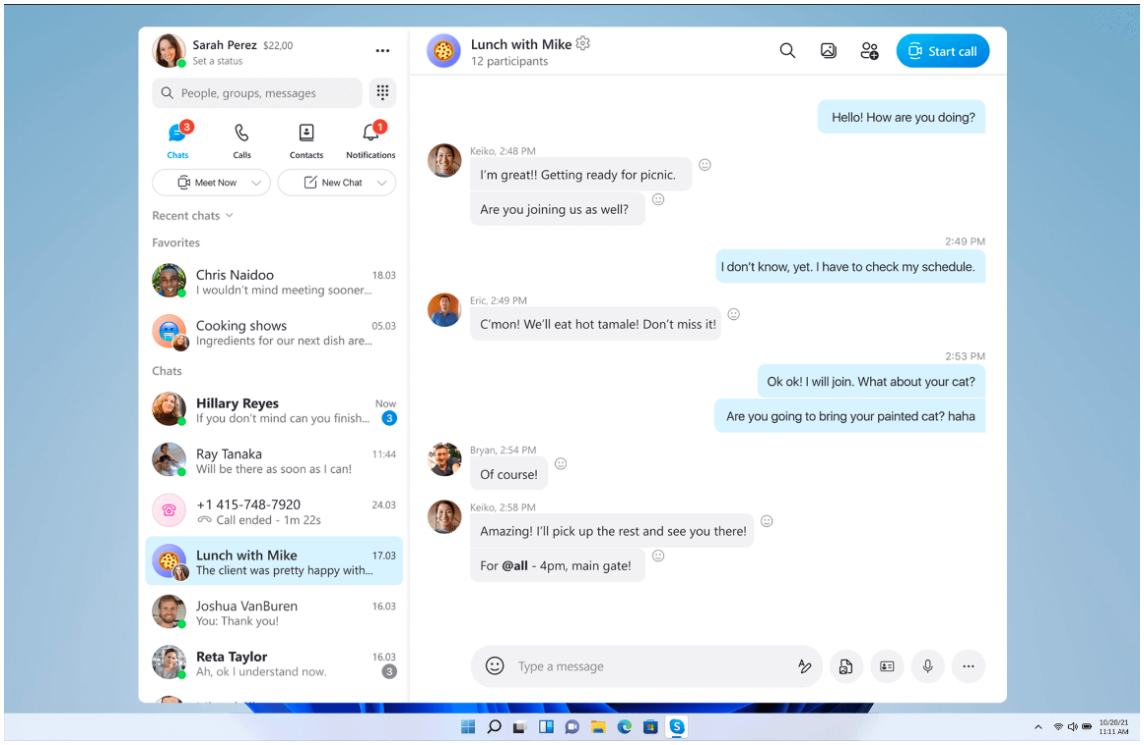
recebem mensagens de áudio, vídeo e texto através do envio de pacotes de informação, que reduzem o seu impacto na velocidade e estabilidade da conexão entre dois protocolos de internet".

O discord é uma das maiores plataformas para gamers do mundo e está disponível gratuitamente em várias versões para computadores (Windows, MacOS e Linux) e também para dispositivos móveis (Android e iOS), além de uma versão para navegadores. E apesar de ter começado a vender jogos como uma plataforma de distribuição digital similar a *Steam*, seu foco primário reside nos grupos de mensagem por texto (BANKOV, 2019, tradução nossa). Também existem outros benefícios para assinantes do seu plano mensal chamado *Discord Nitro*, mas não foram abordadas nenhuma destas vantagens neste trabalho.

Steam é o nome de um programa de software planejado inicialmente como um canal para distribuição de atualizações e correções de bugs para os títulos de jogos da Valve. Mais tarde, se tornou uma plataforma de distribuição digital ou uma loja online para muitos desenvolvedores e organizações de jogos (BANKOV, 2019, tradução nossa). O Discord diferentemente da Steam, além de permitir a comunicação através da linguagem escrita e falada, também possibilita a comunicação por videoconferência e compartilhamento de tela. As plataformas de videoconferência permitem realizar, em tempo real, conexões entre pessoas que estejam em lugares diferentes, provendo uma comunicação por meio de áudio e vídeo. As mais comuns são o Skype, Hangouts e o Discord (FARIAS et al., 2019). Sistemas de videoconferência são uma variação de "sistemas de sala" (room systems), utilizam softwares e webcam em vez de hardwares dedicados. (ROESLER et al., 2012).

Foi identificado informações a respeito da nomenclatura mais utilizada sobre a estrutura do Discord em um de seus blogs. "Servidor": os servidores são os espaços no Discord. São criados por comunidades específicas e grupos de amigos. A grande maioria dos servidores é pequena e reservada para convidados. Alguns servidores maiores são públicos. Qualquer usuário pode começar um novo servidor gratuitamente e convidar os amigos para participar. "Canal": os servidores do Discord são organizados em canais de voz e texto, que geralmente são dedicados a assuntos específicos e podem ter regras diferentes (DISCORD, 2023).

Figura 1 - Foto demonstrativa do Skype



Fonte: Adaptado de MICROSOFT[1]

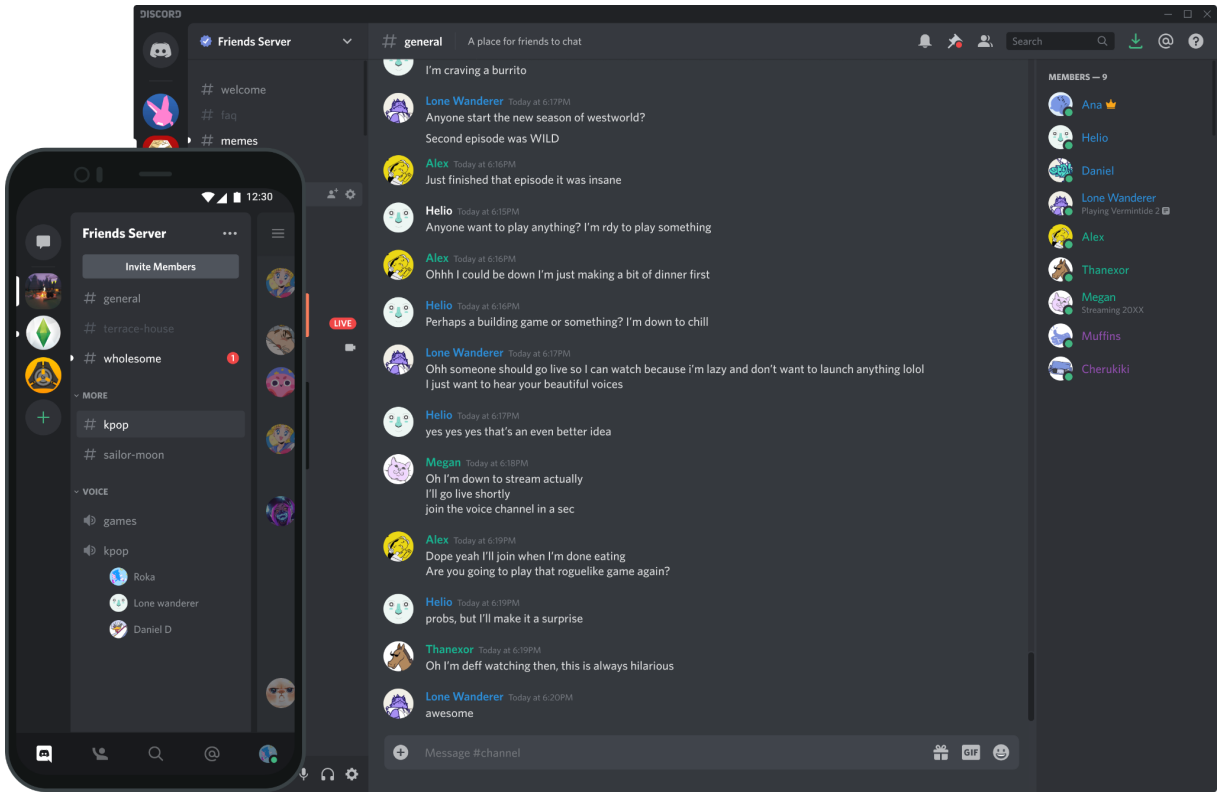
Konstantinou e Epps (2017, p. 03) apontam algumas funcionalidades adicionais oferecidas pelo Discord:

> O uso de convites para entrar no canal; o nível de verificação exigido; um filtro integrado para conteúdo explícito que ajuda na moderação do canal; um log de auditoria para rastrear a atividade no canal; e notificações para membros individuais ou todos os participantes do canal, dependendo do nível de autorização.

Discord apresenta comodidade por necessitar apenas de uma conta para participar em vários servidores da comunidade. Em cada um deles, os assuntos podem ser divididos em diversos segmentos predispostos pelos moderadores, são os chamados canais (JACQUES FILHO, 2018). Os servidores cadastrados ficam exibidos em barra no lado esquerdo, o conteúdo e os canais de cada servidor em área de maior extensão ao centro e os participantes em barra no lado direito. Os canais são pré-definidos entre dois tipos de envio: através de texto ou em conversa por áudio (JACQUES FILHO, 2018).

[1] Blogs do Skype disponíveis no site da microsoft.

Figura 2 - Foto demonstrativa do Discord



Fonte: Adaptado de DISCORD[2]

Jacques Filho (2018) destaca ainda que “de acordo com os moderadores de cada servidor, as seções existentes podem ser abertas ou restritas, nas quais apenas membros com credenciais podem acessar”. O sistema de cargos do Discord permite que os usuários criem uma hierarquia dentro do servidor, fazendo uma distinção entre administradores, moderadores e membros (GREENBERG, 2019, p. 13, tradução nossa). É crucial manter um ecossistema saudável na comunidade e resolver problemas prontamente para evitar que eles se tornem persistentes e criem uma comunidade tóxica. Mas um maior nível de engajamento só é alcançado quando há um forte senso de comunidade, uma boa moderação e o incentivo aos membros da comunidade altamente ativos (MILLINGTON, 2017).

2.1.1 Discord e Games

[2] Blog de segurança intitulado “O que é Discord?” disponível no site do Discord.

Os videogames ou jogos eletrônicos são considerados sistemas que envolvem jogadores em situações de conflito, definidas por regras previamente estabelecidas, gerando um resultado quantificável (SALEN; ZIMMERMAN, 2003).

Para os desenvolvedores de games, é exigido a aplicação de técnicas sofisticadas que atendam seu requisito de entretenimento, ou seja, conter diversão e que sua utilização seja agradável, criando a sensação de imersividade, que será obtida pela integração de aspectos artísticos e tecnológicos. Assim, desenvolver jogos digitais envolve muita interdisciplinaridade com outras ciências como Educação, Psicologia, Artes Plásticas, Letras, Design Gráfico e Música (BATTAIOLA, 2002). Já para os jogadores, estas questões se refletem em sua experiência uma vez que envolve três conceitos fundamentais: interação que consiste na capacidade do ambiente para responder às ações do usuário; envolvimento que está relacionado com a capacidade de manter a atenção do usuário e, a imersão que se refere a capacidade de fazer com que o usuário permaneça focado no ambiente simulado (MACHADO et. al, 2011, p. 255). Martoni, da Silva e Pagani (2018, p. 5) afirmam que “jogar em equipe, é necessário a comunicação entre os membros para, por exemplo, elaboração da estratégia, a análise da execução, as tomadas de decisões para mudanças na estratégia”.

A comunicação e a interação social são dois dos principais componentes que impulsionam a popularidade de jogos e continuarão a impactar a forma como os jogos são comercializados e jogados (BANKOV, 2019, tradução nossa) e por este motivo é ressaltado em diversos autores o uso de aplicativos de comunicação externas ao jogo. Por exemplo quando Martoni, da Silva e Pagani (2018, p. 2) demonstram em seu trabalho que:

> Devido à natureza do sistema de combate do jogo estudado, Final Fantasy XIV (FFXIV) ser um jogo de tempo real, é necessária a tomada de decisões rápidas e precisas [...]. No entanto, em encontros com inimigos mais poderosos, o jogador é constantemente sobrecarregado com informações: estado do personagem, estado do inimigo, marcas telegráficas de perigo, estado das habilidades, indicadores específicos de cada classes, dentre outras diversas mecânicas do jogo. [...] necessitando que os jogadores, organizados em grupo, definam, sigam, reflitam e, se necessário, mudem a estratégia do grupo. Isso torna a comunicação entre jogadores vital para o sucesso. No entanto o jogo possui apenas uma ferramenta de bate-papo por texto, o que pode ocasionar perda de tempo e tomadas de decisões rápidas exigidas em uma partida devido ao fato de que as entradas para movimentação e execução de habilidades dos personagens é realizada por meio do teclado e, durante o modo de digitação de mensagens no bate-papo, a execução de ações é bloqueada. Muitos jogadores contornam a barreira de comunicação por meio de aplicativos terceiros para

> conversação em voz, o que simplifica a transmissão de mensagens aos jogadores.

É importante ressaltar também que os usuários estão interessados em diversos tipos de jogos, mas uma parcela significativa da plataforma também está interessada em jogos baseados em blockchain e NFTs, chamados de *cryptogames*, como é o exemplo de um dos principais servidores Discord em janeiro de 2023: a comunidade do jogo *play-to-earn* (jogue para ganhar - dinheiro) Axie Infinity (SERADA, 2023, p. 58). Em um outro exemplo, vê-se o uso do Discord para o mercado específico de cripto games, esse canal no Discord oficial do CryptoKitties foi criado com o objetivo de promover vendas para outros membros do servidor, é um servidor gratuito e aberto a qualquer pessoa interessada no jogo (SERADA, 2023, p. 59).

2.1.2 Discord e Educação

Como visto anteriormente o uso do Discord por seus usuários foi fortemente abraçado pela comunidade gamer e jogos eletrônicos. Mas com o advento da pandemia principalmente, centros de educação do mundo todo passaram a buscar formas alternativas à modalidade presencial. Com esta saída, o uso do Discord para a área da educação foi popularizado.

Uma pesquisa realizada em parceria pela Universidade Estadual da Paraíba (UEPB) e o Instituto Federal de Educação, Ciência e Tecnologia da Paraíba (IFPB) mostra, não só os problemas de realizar monitorias, como também o interesse dos alunos em participar, apontando ferramentas de videoconferência como possível solução e os benefícios dessa opção: Existem alunos nas universidades que residem em cidades distantes da instituição. Para frequentar as aulas de monitoria precisam se deslocar até as instituições em horários que não coincidem com o horário letivo das suas aulas (FARIAS et al., 2019).

Wulanjani (2018) afirma que “a aplicação de ferramentas multimídia pode desenvolver aprendizes habilidosos porque a multimídia pode criar um ambiente de aprendizagem onde os alunos praticam suas habilidades linguísticas e adquirem a cultura alvo.”

Basso, Costa e Oliveira (2019, p. 35) afirmam que:

> Outros paradigmas podem ainda ser rompidos, como o senso comum de que a juventude não se interessa por ciência e que os celulares e computadores são instrumentos de redução das relações e da comunicação e ainda o uso de um aplicativo como o discord, projetado para comunidades de jogos, tornando-o ferramenta de comunicação para aprendizagem.

Com a vinda da pandemia, foi necessário uma resposta para que várias escolas utilizassem a modalidade de ensino à distância (EAD). Com o uso de ferramentas como aplicativos para reuniões por videoconferência, a tecnologia permitiu que professores e alunos continuassem suas aulas (SANTOS, 2020). Como é o exemplo dos canais de estudos do servidor de Discord *Science and Technology*, nomeado de *Science*, é dividido em áreas da ciência, tecnologia, engenharia e matemática. Essas áreas não são utilizadas com a mesma frequência que outras casuais. Mas é utilizado para conversas focadas e eficientes, já que os membros alertam sobre o cuidado necessário em postagens nestes locais, assim como a equipe do servidor, que modera quaisquer mensagens com fora do tema ali abordado (MELLO, 2021).

2.1.3 Discord e Trabalho

O uso do discord também pode se estender a questões de trabalho. Irani (2019, p. 03-08, tradução nossa) analisa que o uso de videoconferência para pesquisas (de campo) se tornou uma opção aceitável para entrevistas com metodologia quantitativa e que assemelha-se ao presencial. Em suas análises, em oportunidades ela descreve pontos positivos como a flexibilidade de cronograma e até questões psicológicas dos participantes. Já em sua análise de desafios, cita limitações como conexão estável de internet apontando que não é para todo o tipo de tópico de pesquisa e participantes.

Pode-se utilizar o Discord de várias maneiras, seu amplo uso permite desde uma conversa pessoal até reuniões de trabalho (FARIAS et al., 2019). E o isolamento social devido a pandemia pode ser um outro gatilho para o uso de aplicações como Discord e Snapchat. (Moschogianni, 2022, tradução nossa).

2.2 DESENVOLVIMENTO DE APPS TERCEIROS E BOTS

Redes sociais como o Facebook disponibilizam dados por diversas razões, entre elas, promover a rede através de aplicativos de terceiros. São produzidos externamente, sem ligação à rede social e integrados à plataforma do Facebook através de API (Application programming interface) (VASCONCELLOS, 2013). Cater (2013) ao tratar sobre API, investiga que alguns atribuem a origem das APIs àqueles que buscaram utilizar os serviços de uma empresa de novas maneiras lucrativas. No entanto, as APIs *web* como as que são conhecidas hoje, têm uma história mais curta. Empresas como Salesforce, eBay, Flickr, Amazon, Facebook e Google Maps têm liderado o desenvolvimento das modernas APIs da web. Esse novo tipo de API permite que os desenvolvedores utilizem os serviços internos de outras aplicações da web de maneiras diferentes e para diferentes propósitos, impulsionando uma nova era de parcerias online sociais e comerciais (CARTER, 2013).

Segundo Vlist et al. (2022, p. 2, tradução nossa) “as APIs têm sido objeto de estudo por acadêmicos em diversas disciplinas e áreas, incluindo estudos de mídia e comunicação, pesquisas em sistemas de informação (SI) e engenharia de software”. Elas possibilitam a comunicação e a troca de dados e funcionalidades entre sistemas de software. Acabam impulsionam a economia das plataformas e atuam como elementos de infraestrutura centrais que sustentam os grandes ecossistemas de aplicativos e serviços criados por terceiros e parceiros (VLIST, 2022, p. 2, tradução nossa). Muitos desses aplicativos e serviços podem ser bots. Correspondem exatamente à interface do usuário de um navegador hipermídia, apesar de não se assumir que todos os aplicativos são navegadores por conta do estilo. A interface de conector genérica oculta os detalhes do aplicativo do servidor, e por isso um agente de usuário pode ser dentre algumas opções: um robô automatizado realizando recuperação de informações ou buscando dados específicos (FIELDING, 2000, p. 103).

2.2.1 Bots

Os bots são programas versáteis que possuem uma vasta variedade de utilidades, mas os relatórios mostraram que os bots são a maioria silenciosa da Internet. Bilhões desses agentes de software moldam nossa experiência na web, influenciando a maneira como aprendemos, negociamos, trabalhamos e interagimos uns com os outros online (ZEIFMAN, 2014).

Alguns bots possuem boas intenções, aqueles que catalogam páginas da internet para mecanismos de busca, auxiliam na moderação de salas de chat e protegem contra conteúdo indesejado na Wikipedia; outros são maliciosos, buscando quebrar senhas, atacar websites e espalhar spam. Mas em geral, eles são programados para realizar tarefas repetitivas e entediantes sem serem supervisionados e raramente estabelecem interações com indivíduos humanos (MURGIA et al., 2016).

Um tipo de bot são os chatbots. Brandtzaeg e Følstad (2017, p. 2) explicam que embora nos últimos anos tenha havido um aumento do interesse em chatbots por parte de plataformas de mensagens, as interfaces de linguagem natural não são novidade nas áreas de ciência da computação e estudos da Internet. Esses bots, que são capazes de interagir com os usuários, são empregados para lidar com o atendimento ao cliente, poupando recursos humanos e acelerando a resolução de problemas comuns (BRANDTZAEG; FØLSTAD, 2017). Dessa forma, o usuário não precisa procurar em uma extensa lista de Perguntas Frequentes (FAQ) para obter ajuda. No entanto, em algumas ocasiões, esses bots não são tão eficazes e acabam tendo um efeito ruim, é importante perceber que a forma como o chatbot conduz a conversa tem influência nas respostas dos pedidos feitos aos usuários (ADAM; WESSEL; BENLIAN, 2020. p. 439).

3 MÉTODO

Para Michels (2010, p. 78), “A ciência é constituída de um conhecimento racional, metódico e sistemático, capaz de ser submetido à verificação, buscado através de métodos e técnicas diversas”. Também considera-se que pesquisa científica é a realização completa de uma investigação, desenvolvida e redigida de acordo com as normas de metodologia consagradas pela ciência (MICHELS, 2010).

Por isso, seguindo uma sequência lógica encadeada proveniente de um método, obtendo os mesmos resultados caso repetida, uma pesquisa pode ser considerada científica. Assim, a informação criada contribui para o conhecimento apurado e também para sua reforma ou produção (MICHELS, 2010).

Andrade (2001) define a pesquisa científica como um conjunto de procedimento sistematizado, pautado no raciocínio lógico, cujo objetivo é encontrar soluções para os problemas propostos empregando métodos científicos, corroborado por Silva e Menezes (2001, p. 19), as quais afirmam que “pesquisar, significa de forma bem simples, procurar respostas para indagações propostas” e “a pesquisa é realizada quando se tem um problema e não se tem informações para solucioná-lo.”

3.1 CARACTERIZAÇÃO DO TIPO DE PESQUISA

A natureza do presente trabalho é a pesquisa aplicada, que para Meireles (2014): “objetiva a aplicação dos conhecimentos básicos, produz produtos, processos e patentes; e gera novas tecnologias e conhecimentos resultantes do processo de pesquisa.”.

Para tal utilizou-se na pesquisa a abordagem qualitativa, que segundo Roesch (2007) busca explorar um ambiente além de levantar e definir problemas. Meireles (2014) diz que “interpretação dos fenômenos e a atribuição de significados são básicas no processo de pesquisa qualitativa […] Não requer o uso de métodos e técnicas estatísticas.”.

Também utilizou-se a abordagem quantitativa que, para Oliveira (2001, p. 117):

> Significa quantificar opiniões, dados, nas formas de coleta de informações, assim como também com o emprego de recursos e técnicas estatísticas desde a mais simples, como porcentagem, média, moda, mediana e desvio padrão, até as de uso mais complexo, como coeficiente de correlação, análise de regressão, etc.

Quanto aos procedimentos adotados, a pesquisa leva em consideração a bibliografia de diversos autores, principalmente para a elaboração da fundamentação teórica, foram utilizados como fonte de pesquisa: livros, artigos, monografias, teses, sites e portais de tecnologia, revistas e redes eletrônicas. E destaca-se autores pesquisadores de extrema relevância para as áreas de ciências da computação e comunicação, como Naomi S. Baron (2003), Raquel Recuero (2012) e Roy T. Fielding (2000). Meireles (2014). Meireles (2014) aponta que este tipo de pesquisa é um “Levantamento desenvolvido a partir de material já elaborado (livros e artigos científicos), geralmente, como fundamento para outros tipos de pesquisa.”.

Como objetivo de pesquisa foi utilizada a pesquisa exploratória que é a investigação da pesquisa empírica com as finalidades de desenvolver hipóteses e com isso aumentando a familiaridade da pesquisa com o ambiente. (MARCONI; LAKATOS, 2002, p. 85). Para Reis (2008, p. 55), “A pesquisa exploratória é o primeiro passo de qualquer pesquisa, que acontece quando o tema escolhido é pouco explorado e o pesquisador precisa incorporar características inéditas e buscar novas abordagens”.

4 IMPLEMENTAÇÃO

Neste capítulo é tratado do desenvolvimento do bot. Mas para iniciar o desenvolvimento do experimento da ferramenta é necessário conferir e instalar os pré requisitos no sistema.

4.1 PRÉ REQUISITOS E INSTALAÇÕES

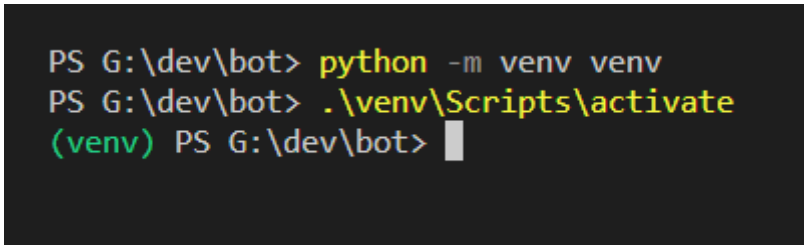
Para dar início na programação do bot, o ideal é preparar um ambiente com boas condições e que forneça tudo que é requisitado para seu desenvolvimento. Utilizou-se o Visual Studio Code, um editor de código popular que foi escolhido por sua praticidade de instalação no sistema operacional *Windows 64bits*.

Foi optado por utilizar a linguagem de programação Python, instalada através da Microsoft Store. Então no terminal do *VScode* foi conferido se a versão instalada corresponde a atual, para isso, foi utilizado o comando: *python --version*.

Com o VScode preparado e o *python* instalado, foi criado um diretório no windows reservado para o projeto e foi dado o nome de "*bot*". Utilizando o powershell criou-se e executou-se o *Venv* (Figura 3), ambiente virtual para instalar as bibliotecas do *python* futuramente utilizadas no projeto. Com o comando "*scripts\activate*" no ambiente virtual ativa-se o ambiente para instalar o *nextcord*, a principal biblioteca para o desenvolvimento do projeto. Para fazer sua instalação, utilizou-se o comando: pip install nextcord.

Com o nextcord instalado pode-se criar o primeiro arquivo de código em Python. Então foi criado um novo arquivo de nome "main" navegando no menu Arquivo e na opção novo arquivo do VScode. Nele, foi onde foi escrito todo o código e empregado a maior parte do tempo de desenvolvimento.

Figura 3 - Instalando e ativando o *Venv*



Fonte: Adaptado de VSCODE[3]

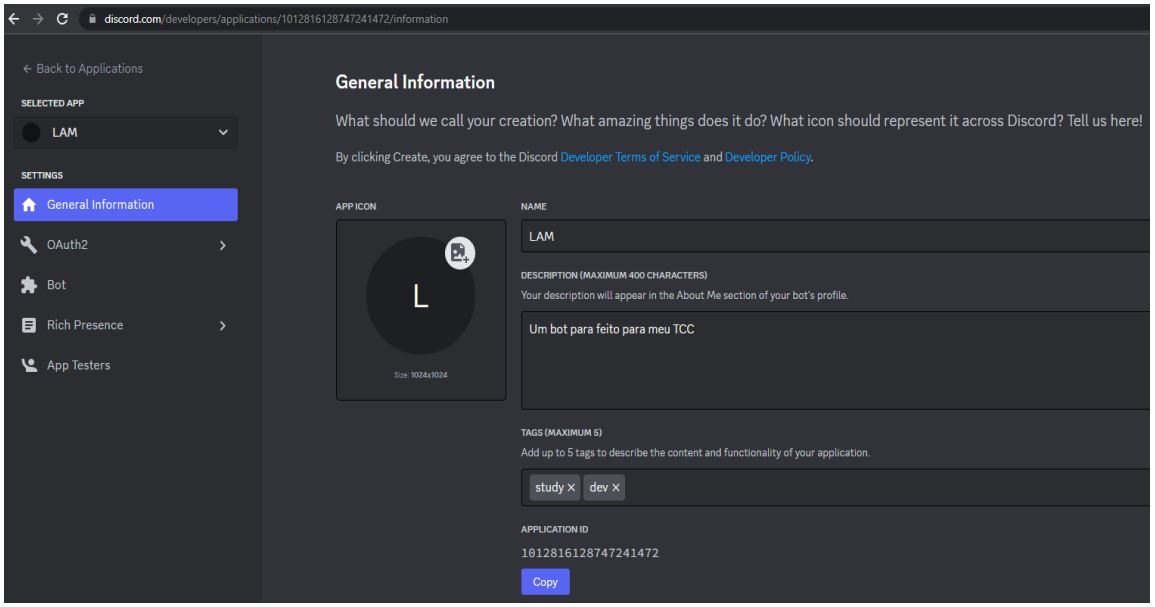
[3] Terminal powershell no programa Visual Studio Code.

4.2 CRIAÇÃO DA APLICAÇÃO ROBÔ

A aplicação e o bot são coisas diferentes: a aplicação é o projeto em si, a ideia e identificação a respeito da criação, já o bot é como uma interface que interage com os usuários e os servidores aplicando as funcionalidades. O discord oferece em seu site, "*dev portal*" ou portal para desenvolvedores, a possibilidade de criar uma aplicação. O nome completo da página é Discord Developer Portal, na parte de aplicações é onde foi criado o experimento deste trabalho e pode-se criar novas aplicações. Foi dado o nome de LAM. Ao acessar a aplicação tem-se algumas áreas de interesse como informações gerais, autenticação e bot (Figura 4).

Nas informações gerais pode-se alterar o nome, descrição e tags da aplicação. A autenticação é a área que pode ser criado um link com OAuth2 para os usuários convidar o bot para seus servidores, neste caso, para fim de testes em um servidor de testes que foi criado.

Figura 4 - Página de informações gerais do app



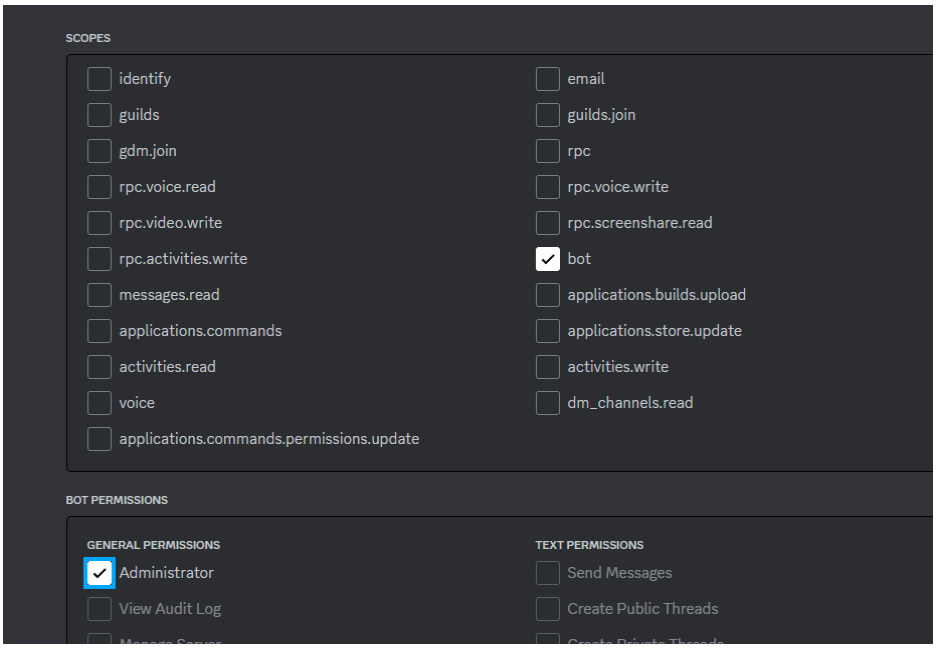
Fonte: DEVELOPER PORTAL (2022)

O Discord oferece na página de Gerador de URL OAuth2, a criação de um link de convite para seu aplicativo escolhendo escopos e permissões de que ele precisa para funcionar. Com isso pode-se compartilhar a URL com outras pessoas. Neste caso, utilizou-se para adicionar o bot em um servidor de teste.

O Discord faz com que quando um bot entre em um servidor, ganhe um cargo automatizado pela ferramenta. Este cargo com nome do app, possui permissões que podem ser configuradas no portal. Então no dev portal, quando é configurado as permissões do bot, é preciso dizer que o escopo é bot e possua o cargo máximo

com acesso a todas as funcionalidades e permissões dentro do servidor e por isso assinala-se “Bot” para SCOPES e “Administrator” para BOT PERMISSIONS como demonstra a figura 5.

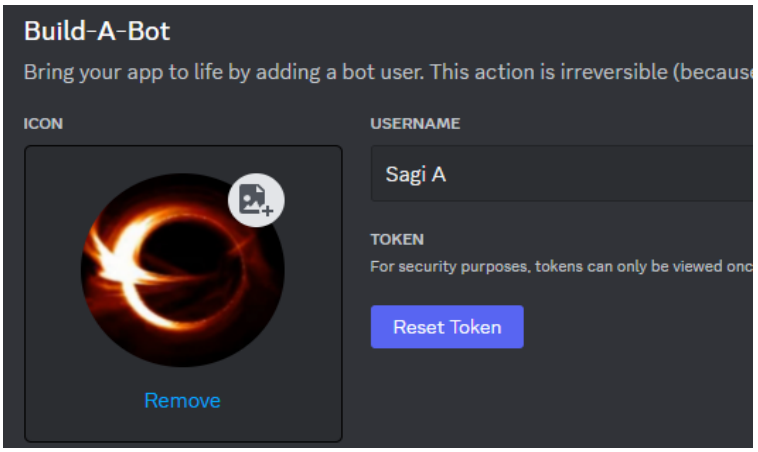
Figura 5 - Configurando gerador de URL OAuth2



Fonte: DEVELOPER PORTAL (2022)

Na área de bot é onde foi dado o nome do bot: “Sagi A”, em homenagem ao buraco negro da Via Lactea. Também foi colocada sua foto mais recente. No dev portal, é também onde se adquire o *token*, que é a chave de segurança para autenticação do bot com o Discord. Por motivos de segurança, o discord faz com que o token só possa ser visualizado uma vez, assim que criado. Caso o desenvolvedor esqueça ou perca, precisará gerar um novo.

Figura 6 - Configurações do perfil do bot e OAuth2 token

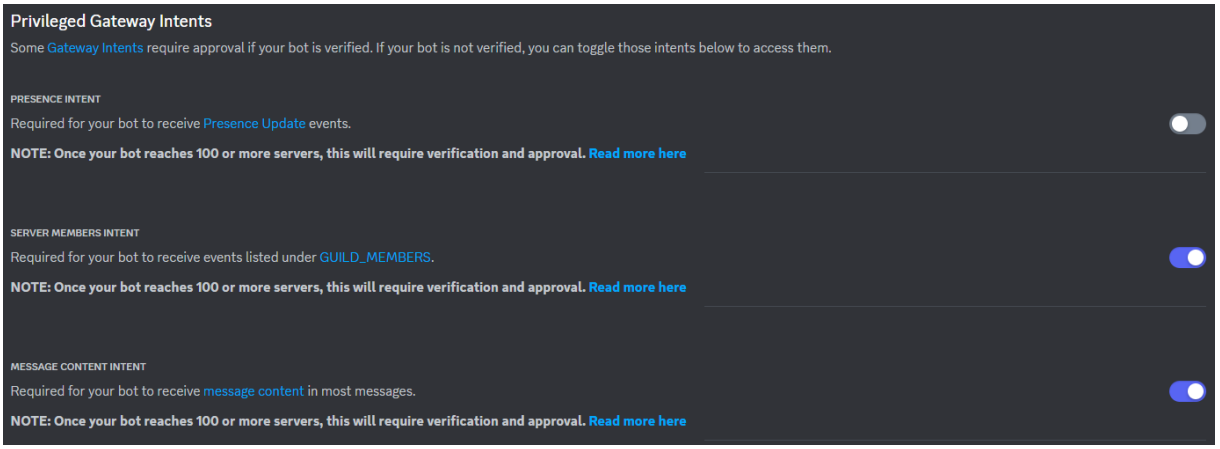


Fonte: DEVELOPER PORTAL (2022)

Para que o bot administrativo realize as funções de moderação sem problemas, é necessário dar permissões: Intents privilegiados e cargo com permissão de administrador.

Por serem sistemas de restrições de acesso, *Gateways Intents* são também um método que o Discord utiliza para diminuir a carga computacional. De modo geral, as ferramentas precisam especificar eles para ter acesso a informações de eventos e comandos relacionados a servidores, membros e mensagens. Com a ajuda do Nextcord, não é preciso especificar cada um dos Intents que é utilizado porque ele faz isso por padrão para todos, com exceção dos 3 Intents privilegiados: Intents.members, *Intents.presences*, and *Intents.message_content*. "*Privileged Gateway Intents*" são algumas regras específicas e restritas para as bibliotecas e exigem mais etapas manuais. Essas intenções são chamadas de intenções privilegiadas e precisam ser manualmente ativadas tanto no portal, quanto no código.

Figura 7 - Configuração de Privileged intents



Fonte: DEVELOPER PORTAL (2022)

Neste experimento, necessita-se que o bot monitore os membros e a listagem dos membros do servidor, por exemplo, quando alguém entra ou sai do servidor, ou quando se faz preciso que o bot realize uma ação punitiva em alguém, além de monitorar as mensagens em todos os canais em que o bot tenha acesso no servidor. Para tal, é utilizado o intents *members=True* e *message_content=True*.

Na medida que este trabalho foi sendo desenvolvido, foi percebido um bug com os Intents, como não é proposto no trabalho não foi investigado a razão do problema, mas acredita-se que é relacionado a biblioteca Nextcord que passa atualmente por atualizações. Para receber a informação "content" dentro de "message" é necessário o Intent "*Message Content*", entretanto mesmo com este e todos os outros Intents relacionados a mensagens como *guilds* (*server*) e *dm* (*direct*

message) não foi possível capturar o conteúdo da mensagem, apenas parte da informação da mensagem como por exemplo o ID e o seu autor. Até a data de finalização deste trabalho, não foi observado outra alternativa a não ser habilitar todos os intents com o método "*all*" para efetuar assim a captura de forma correta das informações, como demonstra a Figura 8.

Figura 8 - Correção do Intents com o método all

```
7
8   bot = commands.Bot(intents=nextcord.Intents.all())
9   #bot = commands.Bot(intents=nextcord.Intents(guild_messages=True, messages=True,
10
```

Fonte: APÊNDICE B

4.3 ARQUIVO MAIN

Neste arquivo, foi construído tudo que se refere ao bot em si, graças a biblioteca nextcord não é requerido implementar requisições HTTP para comunicar-se com api do Discord. Com isso o código fica menos confuso e menor, facilitando a leitura, programação e refatorações futuras.

Nos itens abaixo é descrito melhor a estrutura do código fonte bem como sobre as classes e métodos nele contidos.

4.3.1 Estrutura do código

No topo de todo código fonte, estão localizadas as importações das bibliotecas python. Para que a aplicação funcione corretamente, se utilizam algumas que são indispensáveis e outras opcionais com propósito de auxiliar o desenvolvimento.

- *Asyncio* - para possibilidade de utilização de subrotinas "*coroutines"* ou "corrotinas" em portugues. Uma corrotina emprega o conceito de simultaneidade utilizado para simplificar o código que é executado de forma assíncrona.
- *Random* - Para auxiliar com cálculos matemáticos randômicos utilizados nos comandos.
- *Nextcord* - Esta é a biblioteca principal do projeto do bot para Discord, com ela pode-se ter acesso a muitas classes que implementam requisições para a

API do Discord sem que tenha-se muito trabalho. É com ela que é feito todas as funcionalidades do bot.

- *Datetime* - Para padronização de *timestamp* ou registro de data e hora.
- *Commands* - Utilizou-se este módulo para especificar quais métodos trabalham com eventos do Discord.
- *Application_checks* - Para configurar e restringir permissões nos comandos.
- *SlashOption* - Para que os comandos possam ser melhor customizados oferecendo mais opções para o usuário.
- *Optional* - Para que as opções sejam opcionais.

Após a importação das bibliotecas está a declaração do bot passando por parâmetro as suas *Intents*, aplicando as permissões configuradas anteriormente.

Figura 9 - Bibliotecas python e instância do Bot

```
1  import asyncio
2  import random
3  import nextcord
4  from nextcord.ext import commands, application_checks
5  from nextcord import SlashOption
6  from typing import Optional
7
8  bot = commands.Bot(intents=nextcord.Intents(message_content=True, members=True).default())
```

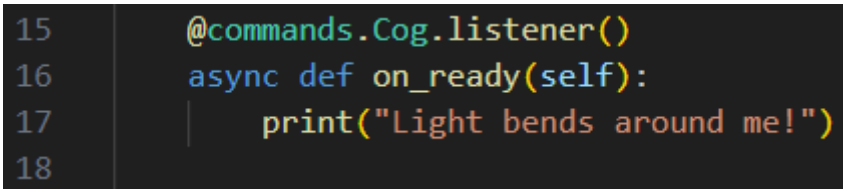
Fonte: APÊNDICE A

Logo abaixo estão as classes Sagi e Aux.

Sagi (APÊNDICE A, linha 13) é a classe e parte principal de todo o código, é instanciada através do método "*setup*" no final do código que aciona o método de inicialização (*constructor*) chamado __init__ recebendo o bot por parâmetro. Aqui é onde estão todas as funcionalidades que foram classificadas entre *listeners* e *commands*:

- *Listeners* - São métodos relacionados a eventos do Discord. Uma mensagem postada em um canal ou um membro que acaba de chegar disparam eventos, pode-se capturá-los e manipulá-los conforme nossa necessidade. Para atribuir a função de método *listener*, é utilizado o decorador *@commands.Cog.listener()*.

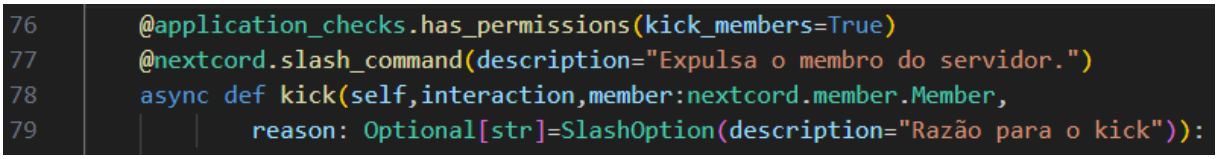
Figura 10 - Exemplo de método *listener:* on_ready.



Fonte: APÊNDICE A

- *Slash commands* - São o que compõe o sistema de comandos para aplicações Discord. Neste sistema interativo o usuário digita o comando utilizando o caractere barra na frente e pode passar informações em opções que o método do comando receberá por parâmetros. Por este motivo utilizou-se a biblioteca *SlashOption*. E para que as opções não sejam obrigatórias, utiliza-se a biblioteca *Optional*. Para atribuir a função de método *command*, utilizou-se o decorador *@nextcord.slash_command()* passando por parâmetro a descrição do comando para o usuário visualizá-la. E utilizou-se o decorador *@application_checks.has_permissions* para atribuir restrições de permissões no uso de cada comando.

Figura 11 - Exemplo de *slash command:* kick.



Fonte: APÊNDICE A

A classe Aux (APÊNDICE A, linha 186) é onde estão localizados os métodos auxiliares com algoritmos de autoria do autor, separados para uma melhor organização e podendo ser reutilizados no futuro. As funcionalidades do bot fazem uso destes métodos para atenderem seus objetivos.

Na parte final do código é onde se encontram os métodos de configuração (*setup*) e execução do bot (*bot.run*), contendo o token chave que autentica a aplicação junto ao portal. Por motivos de segurança, o Discord alerta que os tokens só podem ser visualizados uma vez: quando criados. Em caso de perda do acesso ao token, é requerido gerar um novo. Na Figura 12, pode-se visualizar onde ele está grifado em vermelho e utilizado do mesmo para execução do bot com o método bot.run(DISCORD_APP_TOKEN).

Figura 12 - Parte final do código e execução do bot

Fonte: APÊNDICE A

4.3.2 Funcionalidades

Para este trabalho foi desenvolvido no bot Sagi diversas funcionalidades, pode-se classificá-las em 4 listeners e 10 slash commands. Sendo 3 listeners com propósito administrativo e 1 recreativo, e 8 comandos administrativos e 2 recreativos como demonstra o Quadro 1 abaixo com mais detalhes. Destaca-se que a classificação que foi dada para estas funcionalidades se deve também em relação ao seu uso, ou seja, o público alvo. É sugerido o uso de ferramentas administrativas apenas para usuários administradores. Deste modo, podem ser evitados muitos problemas por mal utilização da ferramenta e suas funcionalidades.

Quadro 1 - Funcionalidades do bot Sagi

Tipo	Nome	Propósito	Descrição
Listener	on_ready()	administrativo	Evento disparado quando o bot muda o status para online.
Listener	on_message()	recreativo	Evento disparado quando uma mensagem é postada em um dos canais em que o bot tem acesso.
Listener	on_member_join()	administrativo	Evento disparado quando um membro entra no servidor.
Listener	on_member_remove()	administrativo	Evento disparado quando um membro sai do servidor.
Comando	/clear	administrativo	Comando para apagar mensagens no canal utilizado.
Comando	/getuser	administrativo	Comando para capturar as informações do usuário.
Comando	/kick	administrativo	Comando para expulsar um membro.

Tipo	Nome	Propósito	Descrição
Comando	/ban	administrativo	Comando para banir um membro.
Comando	/unban	administrativo	Comando para remover o banimento de um membro.
Comando	/getban	administrativo	Comando para capturar as informações do usuário banido.
Comando	/banlist	administrativo	Comando para listar os banimentos do servidor.
Comando	/ping	administrativo	Comando para verificar a latência do bot.
Comando	/oi	recreativo	Comando para interagir com o bot e receber um cumprimento randômico.
Comando	/bola8	recreativo	Comando para interagir com o bot e receber uma resposta randômica com temática de sorte.

Fonte: APÊNDICE A.

Neste quadro, as funcionalidades foram divididas em tipo: comando ou listener; nome, propósito: administrativo a.k.a moderação ou recreativo; e uma breve descrição. Abaixo será abordado cada funcionalidade com maior profundidade.

4.3.2.1 Listeners

- on_ready

O “on ready” é uma funcionalidade que está situada na linha 18 do código fonte, é do tipo listener com propósito administrativo, disparada quando o bot é executado e fica online. Com o comando “*print*” em python, envia uma mensagem no console confirmando que o evento aconteceu. Sua presença online pode ser percebida em todos os servidores que o cliente bot é membro, através da lista de membros.

- on_message

Um destaque especial para esta funcionalidade situada na linha 22 do tipo listener: No presente trabalho ela foi implementada com um propósito recreativo exatamente igual ao comando “/oi”, em que o bot retorna uma mensagem aleatória de cumprimento com o método “*choise*” da biblioteca “*random*”, dentre as opções temos: 'Olá!!', 'Heyyyy!', 'Ola amg vc eh um amg!', 'Why hello there!!', 'AAAAAAAAA YUPII'. Entretanto ela também pode ser utilizada com fins administrativos como por

exemplo para filtros de linguagem no conteúdo das mensagens, uma opção para isso é capturar o conteúdo da mensagem utilizando o “message.content” e filtrar utilizando a biblioteca “*regex*”.

- on_member_join

Funcionalidade situada na linha 29 do código fonte, do tipo listener com propósito administrativo que é disparada quando um membro entra no servidor e então com o método “*channel.send*” envia uma mensagem de boas vindas ao canal definido como “sistema” pelos donos do servidor, por padrão é o primeiro canal criado. O método que captura o “canal sistema” é na verdade uma requisição que a biblioteca Nextcord realiza através das informações de servidor a partir do membro: “*member.guild.system_channel*”

- on_member_remove

Evento semelhante ao on_member_join e é uma funcionalidade que está localizada na linha 34 do código fonte. É do tipo listener e tem o propósito administrativo que é disparada quando um membro sai do servidor (por quaisquer razões) e então com o método “*channel.send*” envia uma mensagem de boas vindas ao canal definido como “sistema” pelos donos do servidor.

4.3.2.2 Slash Commands (comandos)

- /clear

Funcionalidade situada na linha 40 do código fonte tipo *slash_command* com propósito administrativo. Apaga mensagens do canal onde foi utilizada. Opcionalmente pode ser inserido um número de mensagens a serem apagadas no canal, de máximo de 500 mensagens por vez e caso este número não for inserido e o comando executado serão apagadas 20 mensagens por padrão. Este comando executa o método “*create_task*” da biblioteca *asyncio* que agenda a execução de uma tarefa de co-rotina. No caso, para executar um método na classe Aux (auxiliar) criado pelo autor a fim de organização de código. O método em questão se chama “*delete_messages_from_channel*” da linha 188 que recebe por parâmetro informações do comando e o número de mensagens a serem deletadas no canal (Figura 13).

Dessa maneira conseguiu-se fazer uma requisição para a API para capturar o histórico de mensagens do canal com a seguinte linha de código: “*interaction.channel.history(limit=amount).flatten()*”. Uma simples estrutura de repetição permite acessar cada mensagem do histórico e excluir uma por uma com o

método “*message.delete()*” e adicionar um pequeno *delay* (temporizador) utilizando ambos o método “*message.delete(delay=0.1)*” quanto o “*asyncio.sleep(1.1)*” para que o bot não sofra com excesso de requisição à API do Discord, o que pode ocasionar problemas de conectividade (timeout - banimento temporário).

Destaca-se uma diferença para esta funcionalidade de característica excepcional: o uso de uma *task* (tarefa) somado a abordagem de investigar o histórico de mensagens possibilita que a operação de apagar mensagens aconteça sem que novas mensagens sejam apagadas por descuido, tendo em vista que a funcionalidade trabalha em com uma instância do histórico, ou seja, o que o histórico era antes das novas mensagens existirem.

Figura 13 - Método delete_messages_from_channel

```
185
186 class Aux():
187     @staticmethod
188     async def delete_messages_from_channel(interaction,amount):
189         channel_history = await interaction.channel.history(limit=amount).flatten()
190         for message in channel_history:
191             await asyncio.sleep(1.1)
192             await message.delete(delay=0.1)
193
```

Fonte: APÊNDICE A

- /getuser

Comando localizado na linha 57 do código fonte, é do tipo *slash_command* e possui propósito administrativo. Captura informações do usuário baseado no ID ou menção do usuário inserido e exibe em uma mensagem personalizada em formato de *Embed* do Discord de acordo com o tipo do usuário: membro ou não membro. Basicamente é possível exibir mais informações exclusivas caso o usuário seja membro. Por este motivo, foi decidido construir um algoritmo com lógica única para tentar buscar pelo usuário como tipo “*Member*” com o método “*guild.fetch_member*”, caso não consiga, é feito outra tentativa desta vez como tipo “*User*” e se ainda assim não conseguir é porque o usuário não existe. Após isso, carrega-se algumas informações como “*guilds*”, “*roles*” e “*ismember*” para o método auxiliar, que vai confeccionar a mensagem personalizando as informações a serem exibidas, através da Classe *Embed.* Ao final do método é retornado a instância do “*embed*” que é enviado pelo método “*interaction.send*” para o canal.

- /kick

Kick é uma das funcionalidades core do Sagi que re-implementa o método guild.kick. Está situada na linha 79 do código fonte tipo *slash_command* com

propósito administrativo. Pode ser utilizada para expulsar membros do servidor. É requerido inserir o membro por menção ou ID e opcionalmente a razão ou motivo. O comando tenta expulsar o membro com o método "*member.kick*" e caso falhe é informado que o membro não existe ou seja, o usuário não é um membro do servidor em que foi digitado o comando.

- /ban

Ban assim como o kick é uma funcionalidade muito importante do Sagi que re-implementa o método guild.ban, está situada na linha 92 do código fonte, do tipo *slash_command* e com propósito administrativo. É utilizada para banir usuários do servidor. É requerido inserir o usuário por menção ou ID (caso seja membro ou apenas ID caso não membro) e opcionalmente a razão ou motivo. É verificado se o usuário já está banido no servidor, para isso utiliza-se o método de criação do autor na classe Aux, de nome "*banverification*" na linha 208, que busca os banimentos do servidor com o método "*guild.bans*". Com um laço é verificado usuário por usuário nesta lista e comparado ao usuário do comando, caso não tenha sido banido é executado o banimento com o método "*guild.ban*" enviando o usuário e a razão. Após o banimento é executado o método "*create_task*" da biblioteca *asyncio* que agenda a exclusão de mensagens com método da classe Aux chamado "*clear_when_ban*". O método possui uma lógica criada pelo autor que captura os canais usando o "*interaction.guild.text_channels*" para produzir um histórico de cada canal. Isso é feito com uma lógica contida em um laço que elabora uma lista de histórico de mensagens de acordo com o número de mensagens a serem excluídas recebido por parâmetro. Com o histórico pronto, a lógica de exclusão de mensagens é semelhante a do comando clear, possui apenas a diferença de comparar se o autor de cada mensagem é o usuário banido e em caso positivo a mensagem é excluída com o método *message.delete* como demonstra a Figura 14.

Figura 14 - Método clear_when_ban

```
194     @staticmethod
195     async def clear_when_ban(interaction,messages_per_channel,user):
196         channels = interaction.guild.text_channels
197         channel_histories = [
198             await message.history(limit=messages_per_channel).flatten()
199             for message in channels
200             ]
201         for history in channel_histories:
202             for message in history:
203                 if message.author == user:
204                     await asyncio.sleep(1.1)
205                     await message.delete(delay=0.1)
```

Fonte: APÊNDICE A

Destaca-se três diferenças nesta funcionalidade que, apesar de possuir limitações, são características extremamente ricas e poderosas: i) É uma funcionalidade para banir qualquer usuário, ou seja, membro ou não, mas o bot precisa possuir conhecimento prévio do usuário (estar em servidores mutuamente ao menos uma vez). ii) Pode-se apagar em até 500 mensagens daquele usuário por canal, em todos os canais. iii) Assim como o comando clear, o uso de uma task somado a abordagem de investigar o histórico de mensagens possibilita que a operação de apagar mensagens sem apagar novas mensagens, permitindo a manutenção e o uso do chat simultaneamente.

Utilizando as formas convencionais (sem o uso do Sagi) não é possível banir não membros e ao mesmo tempo simplesmente deletar todas as mensagens do usuário naquele servidor de Discord de uma só vez, o que pode frustrar muitos usuários administradores que desejam banir contas de spam. Por isso abordou-se a situação com respostas menos limitantes re-implementando uma situação que já existia com limitações.

- /unban

Comando reverso ao ban, está localizado na linha 114 do código fonte, é do tipo slash_command com propósito administrativo. É utilizada para remover o banimento de usuários do servidor. É requerido inserir o usuário por ID e opcionalmente a razão ou motivo. Quando o método recebe o ID por parâmetro, o Python e o Nextcord se encarregam de transformar esta informação em uma instância de "*User*" deste usuário e por meio do método *guild.unban*, foi possível remover o banimento do usuário passando sua instância por parâmetro.

Destaca-se que não foi proposto para este trabalho uma ferramenta de *log* e monitoria automatizada, mas esta razão ou motivo pode ser explorado por ferramentas que possuem estas funcionalidades do tipo *log*. Apesar disso, a implementação desse comando não existe no ambiente nativo da plataforma senão por outros bots. A plataforma oferece a remoção de bans nativamente apenas na aba de listagem de usuários banidos do servidor.

- /getban

Funcionalidade do tipo slash_command do tipo administrativa que está localizada na linha 126 do código fonte. É utilizada para capturar as informações do banimento do usuário no servidor e precisa do ID do usuário como requisito. Executa o método *guild.fetch_ban* passando o usuário por parâmetro. Com esta instância de ban é possível confeccionar um Embed que é usado na mensagem para o usuário. O embed é feito com várias informações do usuário banido como seu "*@mention*"

nome, avatar, ID, motivo do ban, data do ban, etc. Caso a requisição do método fetch_ban falhe é indicado que o usuário não existe ou não foi banido do servidor.

Para a data do banimento, foi utilizado um algoritmo de criação do autor para retornar a data do banimento. Este método está na classe Aux, tem o nome "*bandate*" e está na linha 216 do código fonte (Figura 15). Com uma lógica peculiarmente única ele captura a listagem de entradas de "*Auditlog*" do tipo *Ban,* através do *Auditlog* do servidor (funcionalidade nativa da plataforma oferecida com a API do Discord e pelo Nextcord). Então, é feito uma lógica de *comprehension* em Python (que mistura estruturas de repetição e condição na mesma linha) com esta lista, varrendo todos os bans e verificando em qual dos bans é o do usuário. Caso houver sucesso retorna-se a data formatada, caso contrário, a entrada do Ban não está no Auditlog, o que significa que estava armazenada a mais de 45 dias e expirou após esse tempo e por isso retorna-se esta informação.

Figura 15 - Método bandate

```
215     @staticmethod
216     async def bandate(interaction, userid):
217         bans = []
218         async for entry in interaction.guild.audit_logs(action=nextcord.AuditLogAction.ban):
219             bans.append("{0.target.id} {0.created_at} {0.target}".format(entry))
220         banentry = []
221         [banentry.append(ban) for ban in bans if int(ban[0:18]) == userid]
222         if banentry != []:
223             return banentry[0][19:29]
224         else:
225             return "+45 days"
```

Fonte: APÊNDICE A

- /banlist

Funcionalidade do tipo slash_command do tipo administrativa que está localizada na linha 144 do código fonte. É utilizado para listar todos os banimentos no servidor. Executa o método *guild.bans* para capturar a lista de bans com o máximo de 500 usuários. A partir desta lista de entradas de ban é confeccionado um Embed com um design único criado pelo autor, que lista os usuários e suas informações em duas colunas, estas colunas são configuráveis com as variáveis "len80" e "len60". Dentre as informações existe o nome e o ID do usuário.

- /ping

Funcionalidade do tipo slash_command do tipo administrativa que está localizada na linha 169 do código fonte. É utilizado para verificar a latência do bot em tempo real. A mensagem em resposta é do tipo *ephemeral*, que significa que somente o usuário pode visualizá-la.

- /oi

Funcionalidade do tipo slash_command do tipo recreativa que está localizada na linha 173 do código fonte. Responde com uma mensagem de cumprimento aleatória de uma lista em "*hardcode"*, utilizando o método *choice* da biblioteca *random*.

- /bola8

Funcionalidade do tipo slash_command do tipo recreativa que está localizada na linha 178 do código fonte. Responde com uma mensagem aleatória de uma lista em "*hardcode"*, utilizando o método *choice* da biblioteca *random*. As respostas tem uma temática de sorte, a funcionalidade se assemelha ao uso de uma moeda para responder perguntas mundanas.

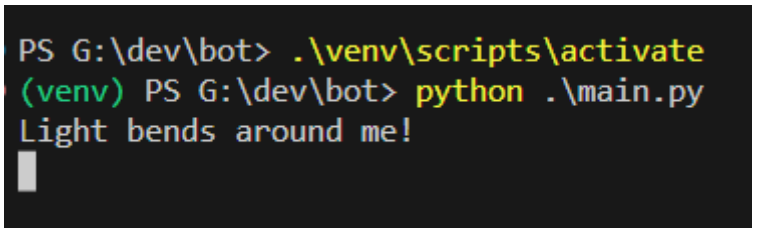
4.4 TESTES DAS FUNCIONALIDADES

Esta parte demonstrativa foi dividida em algumas situações: preparativos para execução do programa, execução dos testes de eventos (*listeners*) e de comandos (*slash_commands*).

4.4.1 Preparação e execução do programa

Utilizando a ferramenta VSCode, no powershell, navegou-se até o diretório em que o bot está instalado utilizando o comando "cd G:\dev\bot". Foi ativado o venv digitando o comando "venv\scrpit\activate" e executado o arquivo main com o comando "*python main.py"* como demonstra a Figura 16.

Figura 16 - Executando o bot



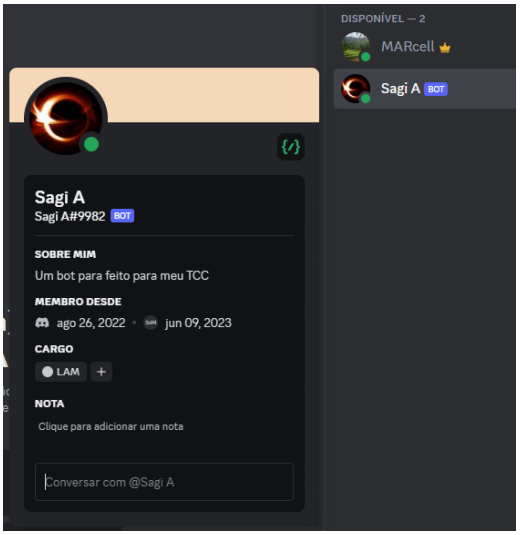
Fonte: Adaptado de VSCODE[4]

A partir deste momento, a primeira funcionalidade *on_ready* já foi executada e a mensagem "Light bends around me" aparece no console. Além disso, pode-se

[4] Terminal Powershell no programa Visual Studio Code.

conferir o perfil do Sagi online na lista de membros nos servidores em que o bot foi convidado (Figura 17). Com isso, é dado sequencia para os próximos eventos!

Figura 17 - Presença do Sagi online

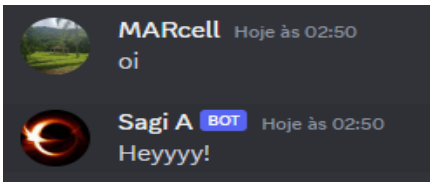


Fonte: Adaptado de DISCORD[5]

4.4.2 Teste dos eventos ou *listeners*

On_message - Para este evento ser ativado e a funcionalidade disparar uma mensagem, o usuário precisa enviar uma mensagem começando com a palavra “oi” (Figura 18).

Figura 18 - Cumprimentando de Sagi

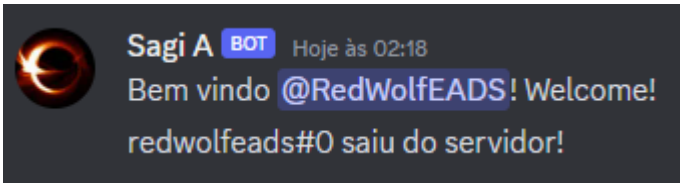


Fonte: Adaptado de DISCORD[6]

On_member_join - A ideia é que quando surge um novo membro no servidor o bot automaticamente irá enviar uma mensagem desejando boas vindas no canal configurado como “sistema”. Por padrão é o único canal quando se cria um novo servidor. No exemplo, o membro que acaba de entrar é o RedWolfEADS, como demonstra a Figura 19.

[5] Perfil do Sagi A na plataforma Discord.
[6] Exemplo de on_message em canal de texto no Discord.

Figura 19 - Mensagem boas vindas e saída do servidor



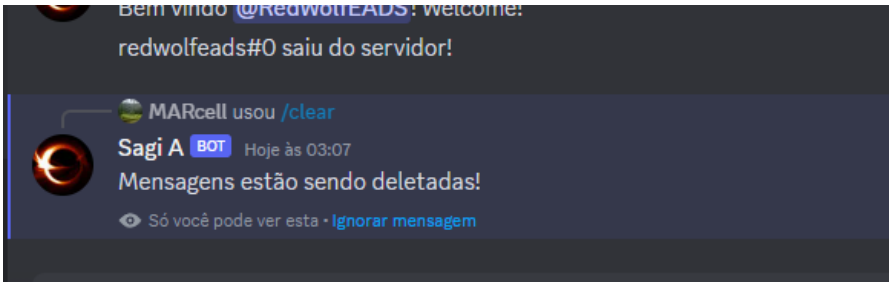
Fonte: Adaptado de DISCORD[7]

On_member_remove - Da mesma maneira, quando um membro sai do servidor o bot automaticamente posta uma mensagem no canal sistema notificando que tal usuário saiu (Figura 19).

4.4.3 Teste dos comandos ou *slash_commands*

/clear - Neste comando, o objetivo é apagar mensagens no canal, podendo inserir o número de mensagens a serem apagadas como demonstra a Figura 20.

Figura 20 - Apagando mensagens

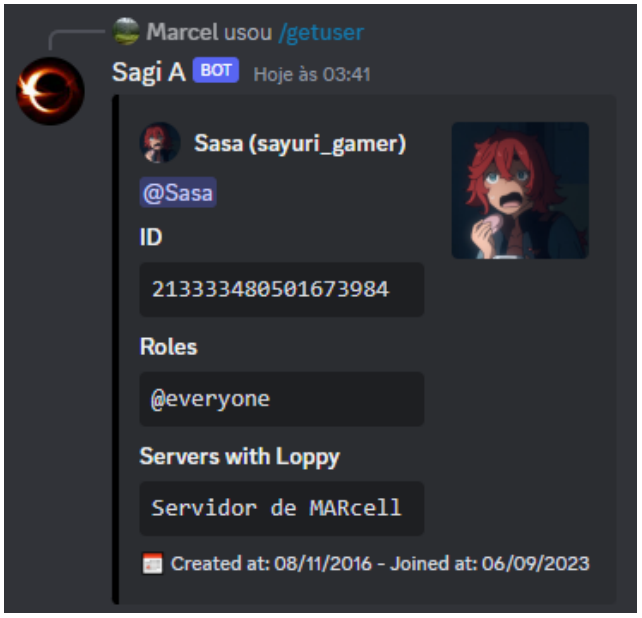


Fonte: Adaptado de DISCORD[8]

/getuser - Ao usar o getuser, é inserido um ID e recebe-se informações deste usuário, no exemplo da Figura 21, é mostrado as informações da @Sasa.

[7] Exemplo de on_member_join e on_member_remove em canais de texto no Discord.
[8] Exemplo do comando “clear” em canais de texto no Discord.

Figura 21 - Informações do usuário



Fonte: Adaptado de DISCORD[9]

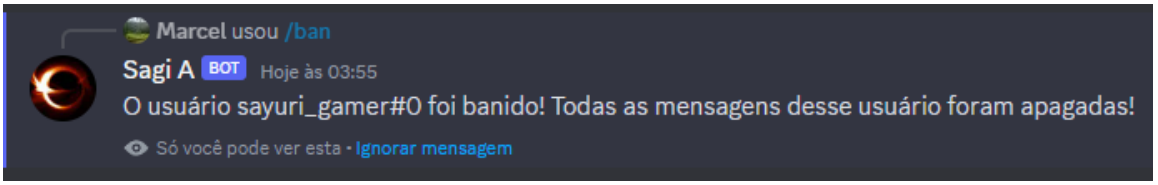
/kick - Usando o kick, é inserido a menção da membro Sasa para expulsá-la do servidor (Figura 22).

Figura 22 - Expulsando o membro

Fonte: Adaptado de DISCORD[10]

/ban - Ao usar o ban, inseriu-se o ID de Sasa para bani-la do servidor, deste modo pôde-se evidenciar que ela foi banida mesmo fora do servidor (Figura 23).

Figura 23 - Banindo um usuário

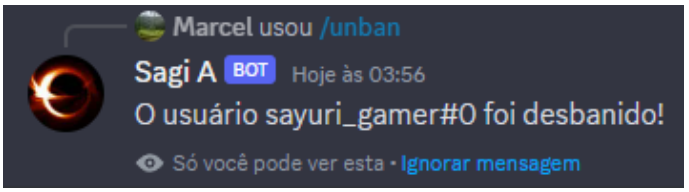


Fonte: Adaptado de DISCORD[11]

[9] Exemplo do comando getuser.
[10] Exemplo do comando kick.
[11] Exemplo do comando ban.

/unban - Inseriu-se o ID da Sasa que estava banida para remover o banimento do servidor. (Figura 24).

Figura 24 - Removendo o ban do usuário

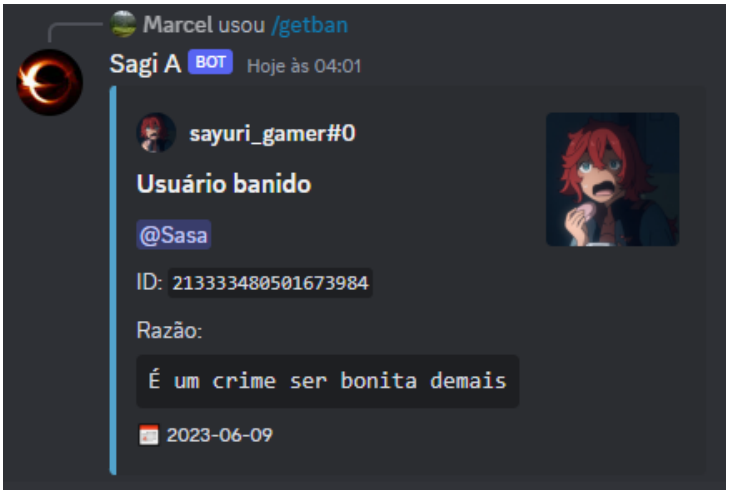


Fonte: Adaptado de DISCORD[12]

Neste momento, ambos os usuários Sasa e RedwolfEADS foram banidos para os próximos testes.

/getban - No comando getban foi inserido o ID de Sasa para capturar as informações do seu ban.

Figura 25 - Capturando informações do banimento

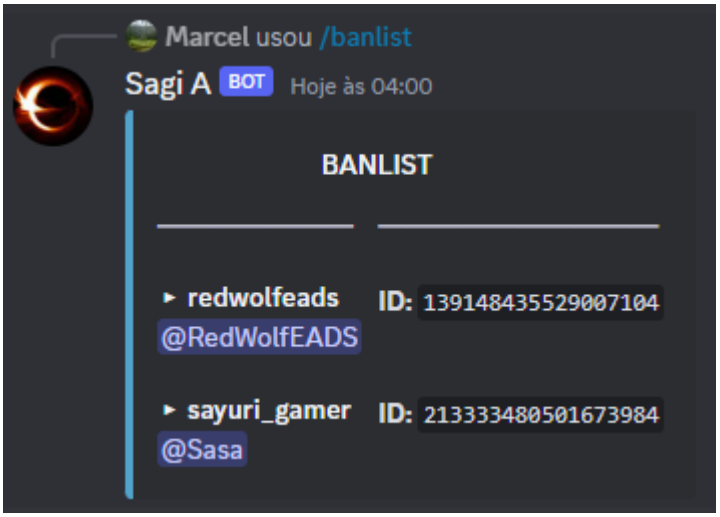


Fonte: Adaptado de DISCORD[13]

/banlist - Ao usar este comando, o bot posta uma mensagem com a lista de todos os usuários banidos no servidor.

[12] Exemplo do comando unban.
[13] Exemplo do comando getban.

Figura 26 - Listando os usuários banidos



Fonte: Adaptado de DISCORD[14]

/ping - Ao utilizar este comando, o bot responde informando sua latência.

Figura 27 - Mostrando a latência do bot

Fonte: Adaptado de DISCORD[15]

/oi - Ao usar este comando, o bot responde com uma mensagem randômica de cumprimento.

/bola8 - Da mesma maneira que o comando “oi”, o bot responde com uma mensagem randômica, mas desta vez em uma temática de sorte, como em uma brincadeira.

Figura 28 - Comandos oi e bola8

Fonte: Adaptado de DISCORD[16]

[14] Exemplo do comando banlist.
[15] Exemplo do comando ping.
[16] Exemplo dos comandos oi e bola8.

5 CONSIDERAÇÕES FINAIS

Neste trabalho, foi explorado o desenvolvimento de um aplicativo bot para o Discord com todos os desafios associados a essa tarefa. A motivação se deu pela crescente popularidade do Discord, que apesar de ter sido inicialmente famosa entre gamers, se consagrou como uma plataforma de comunicação online, utilizada por pessoas do mundo todo para situações cotidianas como trabalho, lazer ou estudo. O que pode resultar em exposição dos usuários a vulnerabilidades ou ataques de cibercriminosos. Para isso, foi implementado funcionalidades administrativas e recreativas no bot, permitindo sua interação com os membros dos servidores em que for convidado.

Ao desenvolver este projeto foi percebida a praticidade do portal de desenvolvedores, Discord Developer Portal. Ele oferece recursos para criar e configurar a aplicação e o bot, incluindo a geração de tokens de autenticação. Essa integração com APIs e plataformas externas é um aspecto relevante no desenvolvimento de software, pois permite criar soluções que se conectam a diferentes serviços e ampliam suas funcionalidades.

Também foi destacado a importância de preparar um ambiente para desenvolvimento, isso inclui a escolha do editor de código, a instalação da linguagem de programação e a criação de um ambiente virtual. Essas etapas foram e são cruciais para garantir um ambiente estável e adequado para o desenvolvimento de software. Muito é dito na academia que a utilização de bibliotecas e ferramentas adequadas simplifica o processo de desenvolvimento, mas foi possível notar com mais clareza nesta experiência. A biblioteca principal utilizada foi o Nextcord, que fornece acesso às classes e funcionalidades necessárias para interagir com a API do Discord. Além disso, outras bibliotecas como asyncio e random foram mencionadas, mostrando a importância de selecionar as ferramentas certas para facilitar o desenvolvimento.

Em relação às funcionalidades do bot, é esperado que ao classificá-las em administrativas e recreativas possa causar uma boa impressão de como as ferramentas possam ser poderosas no auxílio aos usuários. Da mesma forma, espera-se que sejam amigáveis o suficiente para os mais recentes na plataforma. Seus objetivos incluem ouvir eventos específicos, como mensagens enviadas ou membros entrando ou saindo de um servidor, e também a interação com comandos.

Essas funcionalidades demonstram uma capacidade de personalização e automação que os bots podem oferecer, agregando valor aos servidores e proporcionando diferentes experiências aos usuários.

Destaca-se também as falhas e limitações encontradas durante o desenvolvimento, como problemas com os "Intents" da biblioteca Nextcord. Essas limitações podem exigir ajustes e soluções alternativas para garantir o funcionamento adequado do bot. Identificar e solucionar essas limitações faz parte do processo de desenvolvimento de software e o que nos move a buscar e experimentar por melhorias continuamente.

REFERÊNCIAS

ADAM, M.; WESSEL, M.; BENLIAN, A. **Ai-based chatbots in customer service and their effects on user compliance**. **Electron Markets.** n. 31, 2021. p. 427–445. Disponível em: https://link.springer.com/article/10.1007/s12525-020-00414-7 Acessado em: jun. 2023.

ABHINAND, G; BALASUBRAMANIAN, Roshni. Study on the Development and Implementation of Ubiquitous Bots for the Discord Interface. In: **International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT)**, v. 8, p. 212-221, 2022. Disponível em: https://ijsrcseit.com/CSEIT228137 Acessado em: nov. 2022.

ANDERSON, Maya. **Discord And The Harbormen Gaming Community**. jun. 2019.

BASSO, Silvia Elianede Oliveira; COSTA, Maria Luisa Furlan; OLIVEIRA, Patrícia L. L. Mertzig Gonçalves de. Educação a distância e olimpíada nacional de história do brasil: êxito da modalidade um programa inovador. In: **Colloquium Socialis**, Universidade do Oeste Paulista (UNOESTE), Presidente Prudente: UNOESTE, v. 3, n. 2, p. 32-36. 2019. Disponível em: https://journal.unoeste.br/index.php/cs/article/view/2599 Acessado em: nov. 2022.

BANKOV, Boris. The Impact of Social Media on Video Game Communities and the Gaming Industry. In: Conference: **Information and Communication Technologies in Business and Education (ICTBE)**. University of Economics, Varna, 2019.

BARON, Naomi S. Language of the Internet. In: **The Stanford Handbook for Language Engineers**. Stanford Center for the Study of Language and Information. Stanford, ed. Ali Farghali, p. 59-127, 2003. Disponível em: https://dra.american.edu/islandora/object/auislandora%3A25157 Acessado em: jun. 2023.

BATTAIOLA, André Luiz; Elias, Nassim Chamel; Domingues, Rodrigo G. et al Desenvolvimento de um Software Educacional com Base em Conceitos de Jogos de Computador In: **XIII Simpósio brasileiro de informática na educação**. São Leopoldo: SBC, 2002, p. 282-290.

BRANDTZAEG, Petter B.; FØLSTAD, Asbjørn. **Why people use chatbots**. SINTEF. Forskningsveien, Oslo, Norway. 2017.

BREWSTER, Thomas. **Discord**: The $2 billion gamer's paradise coming to terms with data thieves, child groomers and FBI investigators. Forbes, 2019. Disponível em: https://www.forbes.com/sites/thomasbrewster/2019/01/29/discord-the-2-billion-gamers-paradise-coming-to-terms-with-data-thieves-child-groomers-and-fbi-investigators/#26f268ca3741 Acessado em: nov. 2022.

BRISTOT, Paula Casagrande; POZZEBON, Eliane; FRIGO, Luciana Bolan. A representatividade das mulheres nos games. In: **XVI Simpósio brasileiro de informática na educação**. Programa de Pós-Graduação em Tecnologias da Informação e Comunicação, UFSC. Curitiba. 2017.

CARTER, Meg. **A Brief History of API-Based Web Applications.** Smarterbear Blog. 2013. Disponível em: https://smartbear.com/blog/a-brief-history-of-api-based-web-applications Acessado em: jun. 2023.

DISCORD COMPANY. **Sobre o Discord, nossa missão e valores**. Disponível em: https://discordapp.com/company Acessado em: fev. 2023.

DEVELOPER PORTAL, Discord. **Find the perfect feature set for your game in our Game SDK, and sign up for Server Commerce to start selling directly in your server**: Get started by creating a new application. Disponível em: https://discord.com/developers/ Acessado em: out. 2022.

FARIAS, Ricardo de S. et al. Ferramentas de videoconferência para a realização de monitoria a distância. In: **VIII Congresso brasileiro de informática na educação**. Brasília, DF: CBIE, 2019, p. 355-364.

FIELDING, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures**. Tese (Doctor of Philosophy in Information and Computer Science). University of California, Irvine, USA, 2000.

GREENBERG, Miriam. **A study of community engagement on discord for game marketing**: Case TJR Games Oy. Monografia (Bachelor of Business Administration, International Business). Kajaani University of Applied Science, KAMK. Kajaani, Finland. 2019. Disponível em: https://www.theseus.fi/bitstream/handle/10024/266217/Thesis_Greenberg_Miriam.pdf Acessado em: nov. 2022.

IRANI, Elliane. **The Use of Videoconferencing for Qualitative Interviewing**: Opportunities, Challenges, and Considerations. Frances Payne Bolton School of Nursing, Case Western Reserve University. Cleveland, Ohio. 23 nov. 2018 Disponível em: https://journals.sagepub.com/doi/10.1177/1054773818803170 Acessado em: fev. 2023.

JACQUES FILHO, Edu Fernandes Lima. **(RE)Faça você mesmo:** práticas de modding e a circulação midiática na série de jogos Arma. Tese (Doutorado em Ciências da Comunicação) Universidade do Vale do Rio dos Sinos, São Leopoldo, 2018. Disponível em: http://www.repositorio.jesuita.org.br/handle/UNISINOS/7311 Acesso em: out. 2022.

JARDINE, Eric. Beware Fake News: How Influence Operations Challenge Liberal. In: **Governing Cyberspace during a Crisis in Trust**: An essay series on the economic potential — and vulnerability — of transformative technologies and cyber security. tdahCentre for International Governance Innovation, p. 31-35, 2019. Disponível em: https://www.jstor.org/stable/resrep26129.8 Acessado em: jun. 2023.

KIMBALL, David V. **What I've Learned from Running a 10,000 Member Discord Server**. 2018. Disponível em: https://medium.com/@davidvkimball/what-ive-learned-from-running-a-10-000-memberdiscord-server-60c4f613a63d Acessado em: nov. 2022.

KONSTANTINOU, Georgios; EPPS, Julien. **Facilitating online casual interactions and creating a community of learning in a first-year electrical engineering course.** School of Electrical Engineering and Telecommunications. The University of New South Wales (UNSW), Sydney, Australia, dez. 2017. Disponível em: https://www.researchgate.net/publication/322408524_Facilitating_online_casual_interactions_and_creating_a_community_of_learning_in_a_first-year_electrical_engineering_course Acessado em: jun. 2023.

MACHADO, Liliane dos Santos; MORAES, Ronei Marcos de; NUNES, Fatima de Lourdes dos Santos; COSTA, Rosa Maria Esteves Moreira da. Serious games baseados em realidade virtual para educação médica. **Revista brasileira de educação médica**. Associação Brasileira de Educação Médica, v. 35, n. 2, Brasília - DF. jan. 2011.

MARCONI, Marina de Andrade; LAKATOS, Eva Maria. **Técnicas de pesquisas**. 5. ed. São Paulo: Atlas, 2002.

MARTONI, Caio Pereira; SILVA, André Constantin da; PAGANI, Carlos Eduardo. **Proposta para melhoria da jogabilidade do jogo Final Fantasy XIV para pessoas com TDAH**: o Ambiente Stimulus. Instituto Federal de Educação, Ciência e Tecnologia do Estado de São Paulo (IFSP). Hortolândia, São Paulo, 2018.

MEIRELES, Gustavo S. C. **Metodologia científica e tecnológica**: Classificação e métodos científicos. 2014. Disponível em: http://professor.pucgoias.edu.br/SiteDocente/admin/arquivosUpload/10107/material/Classifica%C3%A7%C3%A3o%20da%20pesquisa.pdf Acesso em: nov. 2022.

MELLO, Aruan Pereira. **Comunicação da ciência:** o aplicativo Discord e o servidor "Science and Technology". Monografia (Bacharelado em Biblioteconomia). Faculdade de Biblioteconomia e Comunicação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2021.

MICHELS, Leandro. **Metodologia científica, tecnológica, comunicação e expressão**. Universidade Federal de Santa Maria, 2010.

MILLINGTON, Richard. What Motivates Community Members To Stay Engaged In Online Communities? In: **Community Management FeverBee Blog**. dez. 2017. Disponível em: https://www.feverbee.com/stay-engaged/ Acessado em: jun. 2023.

MORAES, Lívia Assad de. Comunicação móvel, ambientes virtuais e espaços de interação. In: **XXXVIII Congresso Brasileiro de Ciências da Comunicação**. Intercom – Sociedade Brasileira de Estudos Interdisciplinares da Comunicação. Universidade Federal Fluminense, Niterói, Rio de Janeiro, set. 2015.

MOSCHOGIANNI, Georgia. Investigating factors Impacting Discord and Snapchat Use Behavior: Perspective from Swedish users. In: Journal of Digitovation and Information System. Department of Architecture, Aristotle University, Thessaloniki, Greece, 2022. Disponível em: https://jdiis.de/index.php/jdiis/article/view/51 Acessado em: nov. 2022.

MURGIA, Alessandro; JANSSENS, Daan; DEMEYER, Serge; VASILESCU, Bogdan. Among the Machines: Human-Bot Interaction on Social Q&A Websites. In: **CHI EA '16**: Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems. San Jose, CA, USA, 2016. p. 1272-1279. Disponível em: https://dl.acm.org/doi/10.1145/2851581.2892311 Acessado em: 2023.

OLIVEIRA, Silvio Luiz de. **Tratado de metodologia científica**. São Paulo: Pioneira Thomson Learning. p. 117. 2001.

PALMEIRA, Carlos. **Sony anuncia investimento e integração do Discord com a PSN**. Tecmundo, 2021. Disponível em: https://www.tecmundo.com.br/voxel/216627-sony-anuncia-investimento-integracao-discord-psn.htm Acesso em: out. 2022.

PAULA, Thiago Miranda de; NAZARÉ, Tiago Bittencourt. **ZOOMBOMBING: Vulnerabilidades na plataforma Zoom Meetings**. Doctum Cataguases, 2020. Disponível em: https://dspace.doctum.edu.br/handle/123456789/3763 Acessado em: nov. 2022.

RAGLIANTI, Marco; NAGY, Csaba; MINELLI, Roberto; LANZA, Michele. DiscOrDance: Visualizing Software Developers Communities on Discord. In: **VISSOFT conference**. Software Institute (USI), Lugano, Switzerland, 2021. Disponível em: https://discordance.si.usi.ch/files/Raglianti2021a.pdf Acessado em: nov. 2022

REIS, Linda G. **Produção de monografia da teoria à prática**. 2 ed. Brasília: SENAC-DF, 2008.

RECUERO, Raquel. A Economia do Retweet: Redes, Difusão de Informações e Capital Social no Twitter. Contracampo. **Revista eletrônica do Programa de Pós-Graduação em Comunicação. Edição 24, Sociabilidades em rede.** e-ISSN 2238-2577. Niterói, Rio de Janeiro, v. 24, n. 1, jul. 2012. Disponível em: https://periodicos.uff.br/contracampo/issue/view/992 Acessado em: abr. 2023.

ROESCH, Sylvia Maria Azevedo. Projetos de estágio e de pesquisa em administração: guia para estágios, trabalhos de conclusão, dissertações e estudos de casos. 3. ed. São Paulo: Atlas, 2007.

ROESLER, V. et al. Mconf: an open source multiconference system for web and mobile devices. In: **Multimedia - A Multidisciplinary Approach to Complex Issues**. Rijeka, Croatia: InTech, 2012.

SALEN, Katie; ZIMMERMAN, Eric. **Rules of play**: Game design fundamentals. Cambridge: MIT Press. 2004.

SANTOS, Hugo Miguel Ramos dos. Os desafios de educar através da Zoom em contexto de pandemia: investigando as experiências e perspetivas dos docentes portugueses. In: **Práxis Educativa**. Ponta Grossa. 2020. v. 15, p. 1-17. Disponível em: https://revistas.uepg.br/index.php/praxiseducativa/article/view/15805 Acessado em: nov. 2022.

SCHWAB, Brian. **AI game engine programming**. Hingham: Charles River Media. 2004.

SERADA, Alesha. Does #selling sell? Analyzing content of CryptoKitties traders' talk on Discord. In: **7th International GamiFIN Conference**. University of Vaasa, Lapland, Finland, 2023. p. 57-66.

SILVA, Edna; MENEZES, Estela. **Metodologia da pesquisa e elaboração de dissertação.** 3. ed. Florianópolis: UFSC. 2001.

TORI, Romero; KIRNER, Claudio; SISCOUTTO, Robson. Fundamentos e Tecnologia de Realidade Virtual e Aumentada. **Livro do Pré-Simpósio, VIII Symposium on Vitual Reality**. Belém, 2006. Editora SBC - Sociedade Brasileira de Computação, Porto Alegre, 2006.

TURKLE, Sherry. **Alone together**: why we expect more from technology and less from each other. New York, Basic Books, 2011.

VASCONCELLOS, Bruno C. de. **Desenvolvimento de um Aplicativo Integrado ao Facebook e ao Sistema Do Projeto IGUAL para Compartilhamento e Recomendação de Objetos de Aprendizagem**. Monografia (Especialização em Sistemas Distribuídos com Ênfase em Banco de Dados) Universidade Federal do Pampa, Bagé, 2013. Disponível em: https://dspace.unipampa.edu.br/handle/riu/67 Acessado em: Maio. 2023.

VLIST, Fernando. N. van der; HELMOND, Anne; BURKHARDT, Marcus; Seitz, Tatjana. **API Governance**: The Case of Facebook's Evolution. Social Media + Society, 2022. Disponível em: https://doi.org/10.1177/20563051221086228 Acessado em: jun. 2023.

WIJMAN, T. The games market will show strong resilience in 2022, growing by 2.1% to reach $196.8 billion. In: **Newzoo Consulting**. Disponível em: https://newzoo.com/insights/articles/the-games-market-will-show-strong-resilence-in-2022 Acesso em: Set. 2022.

WULANJANI, Arum Nisma. Discord application: turning a voice chat application for gamers into a virtual listening class. In: **2nd English language and literature international conference (ELLiC)**. v. 2, p. 115-119. awd:Anais ELLIC 2018.

ZEIFMAN, Igal. 2014 Bot Traffic Report: Just the droids you were looking for. In: **Incapsula, Imperva blog**. dez. 2014. Disponível em:https://www.imperva.com/blog/bot-traffic-report-2014/ Acessado em: jun. 2023.

APÊNDICE A

```
import asyncio
import random
import nextcord
from nextcord.ext import commands, application_checks
from nextcord import SlashOption
from typing import Optional

#Intents modificado para .all(), habilitando todos os Intents.
bot = commands.Bot(intents=nextcord.Intents.all())
#bot = commands.Bot(intents=nextcord.Intents(guild_messages=True,
messages=True, dm_messages=True, message_content=True,
members=True).default())


class Sagi(commands.Cog):
    def __init__(self, bot:commands.bot):
        self.bot = bot

    @commands.Cog.listener()
    async def on_ready(self):
        print("Light bends around me!")

    @commands.Cog.listener()
    async def on_message(self, message):
        if not message.author == bot:
            if message.content.startswith("oi"):
                frase = ['Olá!!','Heyyyy!','Ola amg vc eh um amg!','Why hello
there!!','AAAAAAAAA YUPII 🐱‍🏍']
                await message.channel.send(random.choice(frase))

    @commands.Cog.listener()
    async def on_member_join(self, member:nextcord.member.Member):
        channel = member.guild.system_channel
        await channel.send(f"Bem vindo {member.mention}! Welcome!")

    @commands.Cog.listener()
    async def on_member_remove(self, member:nextcord.member.Member):
        channel = member.guild.system_channel
        await channel.send(f"{member} saiu do servidor!")
```

```
    @application_checks.has_permissions(manage_messages=True)
    @nextcord.slash_command(description="Apaga N mensagens no canal, máximo de
500 mensagens por vez (20 caso não informado).")
    async def clear(self, interaction,
        amount: Optional[int]=SlashOption(description="Informe o número de
mensagens.")):
        if amount and amount > 500: amount = 500
        if amount == 0:
            message = "Você está de brincadeira comigo? 🤡"
            return await interaction.send(message, ephemeral=True)
        if amount <= 500:
            if amount == None:
                amount = 20
            try:

asyncio.create_task(Aux.delete_messages_from_channel(interaction,amount))
                message = "Mensagens estão sendo deletadas!"
                return await interaction.send(message, ephemeral=True)
            except Exception as e:
                print(e)

    @nextcord.slash_command(description="Mostra as informações do usuário, se
for possível.")
    async def getuser(self, interaction, user:nextcord.user.User):
        guilds = []
        for guild in user.mutual_guilds:
            guilds.append(f"{guild.name}")
        try:
            member = await interaction.guild.fetch_member(user.id)
            if member:
                roles = []
                for role in member.roles:
                    roles.append(f"{role.name}")
                return await interaction.send(embed=await
Aux.userEmbedGen(member, guilds, roles, ismember=True))
        except:
            try:
                user = await self.bot.fetch_user(user.id)
                return await interaction.send(embed=await
Aux.userEmbedGen(user, guilds, roles=None, ismember=False))
            except Exception as e:
```

```
                print(e)
                message=f'{user} não existe.'
                return await interaction.send(message, ephemeral=True)

    @application_checks.has_permissions(kick_members=True)
    @nextcord.slash_command(description="Expulsa o membro do servidor.")
    async def kick(self,interaction,member:nextcord.member.Member,
            reason: Optional[str]=SlashOption(description="Razão para o
kick")):
        try:
            await member.kick(reason=reason)
            message = f"O membro {member} foi expulso."
            return await interaction.send(message, ephemeral=True)
        except Exception as e:
            print(e)
            message = f"{member} não é um membro."
            return await interaction.send(message, ephemeral=True)

    @nextcord.slash_command(name='ban',description="Bane um usuário do
servidor e apaga uma quantidade de suas mensagens por canal.")
    @application_checks.has_permissions(ban_members=True)
    async def ban(self, interaction, user:nextcord.user.User,
        reason:Optional[str] = SlashOption(description="Razão do ban"),
        messages_per_channel:Optional[int] = SlashOption(description="Número
de mensagens a serem apagadas (500 máx).")):
        try:
            if messages_per_channel and messages_per_channel > 500:
                messages_per_channel = 500
            banverify = await Aux.banverification(interaction,user.id)
            if banverify:
                message = "O usuário já está banido."
                return await interaction.send(message, ephemeral=True)
            await interaction.guild.ban(user, reason=reason,
delete_message_seconds=0)
            if messages_per_channel != None:

asyncio.create_task(Aux.clear_when_ban(interaction,messages_per_channel,user))
                message = f"O usuário {user} foi banido! Todas as mensagens
desse usuário foram apagadas!"
            else:
                message = f"O usuário {user} foi banido!"
            await interaction.send(message, ephemeral=True)
```

```
        except Exception as e:
            print(type(e).__name__, e.args)

    @application_checks.has_permissions(ban_members=True)
    @nextcord.slash_command(description="Desbane um membro do servidor,
passando seu ID.")
    async def unban(self,interaction,user: nextcord.user.User,
        reason:Optional[str] = SlashOption(description="Razão da remoção do
banimento")):
        try:
            await interaction.guild.unban(user, reason=reason)
            message = f"O usuário {user} foi desbanido!"
            return await interaction.send(message, ephemeral=True)
        except Exception as e:
            print(e)
            message = "O usuário não está banido!"
            return await interaction.send(message, ephemeral=True)

    @nextcord.slash_command(description="Mostra informações do usuário banido,
passando seu ID.")
    async def getban(self, interaction, user:nextcord.user.User):
        try:
            ban = await interaction.guild.fetch_ban(user)
            date = await Aux.bandate(interaction, user.id)
            message = nextcord.Embed(title='Usuário banido', colour=0x52a3cc)
            message.set_author(name=ban.user, icon_url=ban.user.avatar)

message.add_field(name="",value=f"{ban.user.mention}",inline=False)
            message.add_field(name="",value=f"ID:
`{ban.user.id}`",inline=False)

#message.add_field(name=f"Hash",value=f"{ban.user.mention}",inline=True)
            message.add_field(name='', value=f" Razão: ```{ban.reason}```",
inline=False)
            message.set_thumbnail(url=ban.user.avatar)
            message.set_footer(text="📅 {}".format(date))
            return await interaction.send(embed=message)
        except:
            message = "Não foi possivel encontrar este usuário ou não esxiste
bans no server!"
            await interaction.send(message, ephemeral=True)
```

```
    @nextcord.slash_command(description="Mostra a lista de usuários banidos
neste servidor.")
    async def banlist(self, interaction):
        bans = await interaction.guild.bans(limit=500).flatten()
        rawbans = [entry for entry in bans]
        if len(rawbans) > 0:
            lagger = 0
            for ban in rawbans:
                if len(ban.user.name) > lagger:
                    lagger = len(ban.user.name)
            len80 = int(lagger*80/100)
            len60 = int(lagger*60/100)
            len60 = len60 if len60 > 4 else 4
            banlist = [[ban.user,ban.user.mention,str(ban.user.id)] for ban in
rawbans]
            mensage = nextcord.Embed(colour=0x52a3cc)
            mensage.set_author(name=len80*' '+'BANLIST', icon_url='')
            mensage.add_field(name="",value=len60*"—",inline=True)
            mensage.add_field(name="",value=10*"—",inline=True)
            for ban in banlist:
                mensage.add_field(name="",value=f"",inline=False)
                mensage.add_field(name=f"▸
{ban[0].name}",value=f"{ban[1]}",inline=True)
                mensage.add_field(name="",value=f"**ID:**
`{ban[2]}`",inline=True)
            return await interaction.send(embed=mensage)
        return await interaction.send("Parabéns! A lista de bans do server
está vazia!")


    @nextcord.slash_command()
    async def ping(self, interaction):
        await interaction.send(f"pong! {round(self.bot.latency*1000)} ms",
ephemeral=True)

    @nextcord.slash_command()
    async def oi(self, interaction):
        frase = ['Olá!!','Heyyyy!','Ola amg vc eh um amg!','Why hello
there!!','AAAAAAAAA YUPII 🐱🏍']
        await interaction.send(random.choice(frase))

    @nextcord.slash_command()
```

```
    async def bola8(self, interaction):

        frases = ["Tente depois", "Tente novamente", "Sem opinião", "É o seu
destino", "A sorte está lançada", "Uma chance em duas",
                  "Faça sua pergunta novamente", "Na minha opinião sim",
"Claro que sim", "Sim absolutamente", "Você pode contar com isso",
                  "Sem dúvida", "Muito provável", "Sim", "Começou bem", "Não
é", "Improvável", "Não sonhe", "Não conta", "Impossível"]
        await interaction.send(random.choice(frases))


class Aux():
    @staticmethod
    async def delete_messages_from_channel(interaction,amount):
        channel_history = await
interaction.channel.history(limit=amount).flatten()
        for message in channel_history:
            await asyncio.sleep(1.1)
            await message.delete(delay=0.1)

    @staticmethod
    async def clear_when_ban(interaction,messages_per_channel,user):
        channels = interaction.guild.text_channels
        channel_histories = [
            await message.history(limit=messages_per_channel).flatten()
            for message in channels
            ]
        for history in channel_histories:
            for message in history:
                if message.author == user:
                    await asyncio.sleep(1.1)
                    await message.delete(delay=0.1)

    @staticmethod
    async def banverification(interaction,userid):
        bans = await interaction.guild.bans(limit=1000).flatten()
        for ban in bans:
            if ban.user.id == userid:
                return True
        return None

    @staticmethod
```

```
    async def bandate(interaction, userid):
        bans = []
        async for entry in
interaction.guild.audit_logs(action=nextcord.AuditLogAction.ban):
            bans.append("{0.target.id} {0.created_at}
{0.target}".format(entry))
        banentry = []
        [banentry.append(ban) for ban in bans if int(ban[0:18]) == userid]
        if banentry != []:
            return banentry[0][19:29]
        else:
            return "+45 days"

    @staticmethod
    async def userEmbedGen(user, guilds, roles, ismember):
        message = nextcord.Embed(description=user.mention, colour=user.colour)
        message.add_field(name='ID', value=f'```{user.id}```', inline=False)
        message.set_author(name=f"{user.display_name} ({user.name})",
icon_url=user.avatar)
        if ismember:
            if roles:
                message.add_field(name=f'Roles', value=f"```{',
'.join(roles)}```", inline=False)
            message.set_thumbnail(url=user.display_avatar)
            message.set_footer(text=f"📅 Created at:
{user.created_at.strftime('%m/%d/%Y')} - Joined at:
{user.joined_at.strftime('%m/%d/%Y')}")
        else:
            message.set_thumbnail(url=user.avatar)
            message.set_footer(text=f"📅 Created at:
{user.created_at.strftime('%m/%d/%Y')}")
        if guilds:
            message.add_field(name=f'Servers with Loppy', value=f"```{',
'.join(guilds)}```", inline=False)
        message.set_image(url=user.banner)
        return message

DISCORD_APP_TOKEN =
'MTAxMjgxNjEyODc0NzI0MTQ3Mg.Gfc2w_.A3nHbSaU2gkWTzaMhTL-3ndxGhT3ANaAMoUniQ'

def setup(bot):
    bot.add_cog(Sagi(bot))
```

```
setup(bot)
bot.run(DISCORD_APP_TOKEN)
```

APÊNDICE B

```
import asyncio
import random
import nextcord
from nextcord.ext import commands, application_checks
from nextcord import SlashOption
from typing import Optional

#Intents modificado para .all(), habilitando todos os Intents.
bot = commands.Bot(intents=nextcord.Intents.all())
#bot = commands.Bot(intents=nextcord.Intents(guild_messages=True,
messages=True,dm_messages=True, message_content=True,members=True).default())
```