

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Christhian Rezende Vieira

Desenvolvimento de um *bot* de captura de mensagens do Telegram para inteligência de ameaças cibernéticas

Uberlândia, Brasil

2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Christhian Rezende Vieira

**Desenvolvimento de um *bot* de captura de mensagens do
Telegram para inteligência de ameaças cibernéticas**

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, como parte dos requi-
sitos exigidos para a obtenção título de Ba-
charel em Ciência da Computação.

Orientador: Rodrigo Sanches Miani

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2025

Christhian Rezende Vieira

Desenvolvimento de um *bot* de captura de mensagens do Telegram para inteligência de ameaças cibernéticas

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Uberlândia, Brasil, 14 de abril de 2025:

Rodrigo Sanches Miani
Orientador

Paulo Henrique Ribeiro Gabriel

Diego Nunes Molinos

Uberlândia, Brasil
2025

Agradecimentos

O desenvolvimento deste trabalho só foi possível graças à contribuição de diversas pessoas, cujo apoio e incentivo foram fundamentais para que eu chegasse até aqui. Expresso minha mais profunda gratidão a todos que, de alguma forma, fizeram parte dessa jornada.

Agradeço, em primeiro lugar, aos meus pais, que sempre foram meu porto seguro, confiando na minha capacidade e me incentivando a persistir em meus objetivos. Sou imensamente grato pelo apoio incondicional, pelo amor e pelos valores que me transmitiram ao longo da vida.

À minha irmã, por estar sempre ao meu lado, compartilhando momentos, experiências e aprendizados, e por me incentivar a seguir em frente, meu sincero agradecimento.

Aos meus amigos, pelo companheirismo, pelas palavras de encorajamento e pelo suporte ao longo dessa jornada acadêmica, minha gratidão.

Ao meu professor orientador Rodrigo Sanches Miani, agradeço pela oportunidade de desenvolver este trabalho, pela paciência, dedicação e pelos ensinamentos que contribuíram significativamente para sua realização.

Agradeço aos meus professores, desde as séries iniciais até a graduação, que, ao longo dos anos, compartilharam seus conhecimentos e contribuíram para minha formação, deixo meu reconhecimento e apreço.

Por fim, a todos que, de alguma forma, colaboraram para a concretização deste trabalho, meu mais sincero agradecimento.

Resumo

Este trabalho propõe o desenvolvimento de um sistema de backend para a captura e análise de mensagens provenientes de canais públicos do Telegram, com o objetivo de apoiar a inteligência de ameaças cibernéticas. Para isso, foi criado um bot utilizando o framework NestJS, a linguagem TypeScript, o ORM Prisma e o banco de dados PostgreSQL, com contêinerização via Docker. A aplicação realiza a coleta de mensagens de maneira automática e manual, permitindo seu armazenamento e posterior classificação quanto à relevância para o contexto de cibersegurança, por meio de técnicas de aprendizado de máquina supervisionado. Durante os experimentos, foram coletadas 760 mensagens, das quais aproximadamente 16% foram classificadas como relevantes. Os resultados demonstram a viabilidade da aplicação para auxiliar na identificação de conteúdos maliciosos em redes sociais, propondo também sua futura integração com outras plataformas de comunicação.

Palavras-chave: Telegram, NestJS, BackEnd, PostgreSQL, Coleta e análise de mensagens maliciosas, Cibersegurança

Abstract

This work proposes the development of a backend system for the collection and analysis of messages from public Telegram channels, aiming to support cyber threat intelligence. To achieve this, a bot was created using the NestJS framework, the TypeScript language, the Prisma ORM, and the PostgreSQL database, with containerization via Docker. The application collects messages both automatically and manually, enabling their storage and subsequent classification regarding their relevance to the cybersecurity context through supervised machine learning techniques. During the experiments, 760 messages were collected, of which approximately 16% were classified as relevant. The results demonstrate the feasibility of the application in assisting the identification of malicious content on social networks, also proposing its future integration with other communication platforms.

Keywords: Telegram, NestJS, BackEnd, PostgreSQL, Collection and analysis of malicious messages, Cybersecurity

Lista de ilustrações

Figura 1 – Arquitetura do sistema	24
Figura 2 – Diagrama entidade relacionamento do banco de dados	26
Figura 3 – Execução do comando docker-compose build	44
Figura 4 – Execução do comando docker-compose up	44
Figura 5 – Execução do <i>script</i> run-docker-migrations.sh	45
Figura 6 – Requisição HTTP POST à rota telegram/ <i>join-channel</i>	46
Figura 7 – Requisição HTTP POST à rota telegram/ <i>start-messages-capture</i>	47
Figura 8 – Funcionamento da rota telegram/ <i>start-messages-capture</i>	47

Lista de tabelas

Tabela 1 – Comparação da rapidez entre bancos de dados	21
Tabela 2 – Exemplo de informações de canais armazenadas no banco de dados . .	33
Tabela 3 – Exemplo de mensagens armazenadas no banco de dados	34
Tabela 4 – Exemplo de mensagens classificadas armazenadas no banco de dados .	35

Lista de abreviaturas e siglas

API	Interface de Programação de Aplicação (<i>Application Programming Interface</i>)
TOR	<i>The Onion Router</i>
HTTP	Protocolo de Transferência de Hipertexto (<i>Hypertext Transfer Protocol</i>)
ORM	Mapeamento Objeto-Relacional (<i>Object Relational Mapping</i>)
SGBD	Sistema Gerenciador de Banco de Dados
YAML	YAML não é linguagem de marcação (<i>YAML Ain't Markup Language</i>)
CTI	Inteligência de Ameaças Cibernéticas (<i>Cyber Threat Intelligence</i>)
BERT	<i>Bidirectional Encoder Representations from Transformers</i>
CACs	Canais de Atividade Cibercriminosa
VCS	Sistema de Controle de Versão (<i>Version Control System</i>)
DTO	Objeto de Transferência de dados (<i>Data Transfer Object</i>)
JSON	Notação de Objeto Javascript (<i>Javascript Object Notation</i>)
SUS	Sistema Único de Saúde
SQL	Linguagem de Consulta Estruturada (<i>Structured Query Language</i>)
NOSQL	Não Apenas Linguagem de Consulta Estruturada (<i>No Structured Query Language</i>)

Sumário

1	INTRODUÇÃO	11
2	FUNDAMENTAÇÃO TEÓRICA	14
2.1	<i>Deep Web e Dark Web</i>	14
2.2	Telegram	16
3	TRABALHOS RELACIONADOS	17
4	METODOLOGIA	19
4.1	NestJs	19
4.2	Banco de dados e PostgreSQL	20
4.3	Prisma	21
4.4	Docker e Docker Compose	22
5	DESENVOLVIMENTO	23
5.1	Visão geral	23
5.2	Arquitetura e estruturação do sistema	24
5.3	Banco de dados e tabelas	25
5.4	Configuração do projeto	26
5.4.1	Controle de versões e repositórios de código	26
5.4.2	Configuração geral e instalação das bibliotecas	27
5.5	Construção de serviços para operações no banco de dados	28
5.6	Captura de mensagens em canais do Telegram	28
5.6.1	Conexão com a conta do Telegram	29
5.6.2	Ingresso em canais e grupos do Telegram	29
5.6.3	Captura de mensagens	30
5.7	API para acesso ao sistema	30
5.8	Classificação das mensagens	30
5.9	Contêinerização da aplicação	31
5.10	Documentação e análise dos resultados	32
6	ESTUDO DE CASO	33
6.1	Coleta e armazenamento das mensagens	33
6.2	Classificação das mensagens	34
7	CONCLUSÃO	36

	REFERÊNCIAS	37
	APÊNDICES	40
	APÊNDICE A – CONSTRUÇÃO DO SISTEMA	41
	APÊNDICE B – FUNCIONAMENTO DO SISTEMA	44
B.1	Contêinerização da aplicação	44
B.2	Uso da API	45

1 Introdução

A *Deep Web* e a *Dark Web* são ambientes frequentemente associados à privacidade e ao anonimato. Esses espaços permitem o acesso a conteúdos não disponíveis na *Surface Web*¹, incluindo fóruns e mercados que abordam temas sensíveis ou de natureza ilícita (HOPE, 2017). Na *Dark Web*, especialmente por meio da rede TOR², ocorrem diversas atividades criminosas, como a venda de dados, a distribuição de *malwares* e outros crimes cibernéticos (KAUR; RANDHAWA, 2020).

Diante disso, os cibercriminosos encontraram no aplicativo de mensagens Telegram uma alternativa para esses mercados clandestinos. O número de links para grupos do mensageiro compartilhados em fóruns da *Dark Web* aumentou de 170.000 em 2020 para mais de um milhão em 2021 (REDAÇÃO, 2021). Esse interesse pelo Telegram deve-se às opções de privacidade e anonimato em chats secretos, bem como à possibilidade de comunicação sem a necessidade de compartilhar o número de telefone. Além disso, a plataforma permite o envio e recebimento de grandes volumes de arquivos, em diferentes formatos, e facilita a localização de grupos públicos e privados sobre diversos temas, com capacidade de até 200.000 membros (REDAÇÃO, 2021).

Desta forma, o ambiente do Telegram, assim como a *Deep Web* e a *Dark Web*, é um campo ideal para a coleta de dados que podem ser usados para a análise de ameaças cibernéticas. A natureza descentralizada e, muitas vezes, anônima dessas plataformas favorece a disseminação de conteúdos ilícitos, sendo fundamental a identificação de mensagens nesse espaço para antecipar possíveis ataques, prevenir campanhas criminosas e proteger a integridade de usuários e organizações. O monitoramento e extração de informações de fóruns e *marketplaces* da *Dark Web*, aliado ao uso de técnicas como Processamento de Linguagem Natural (PLN) e aprendizado de máquina, permitem categorizar e identificar conteúdos perigosos, contribuindo para o aumento da segurança online (FILHO, 2023).

A relevância da identificação dessas mensagens é inegável, tanto para a sociedade quanto para a academia. A crescente dependência da internet em todos os setores da vida cotidiana tem ampliado o alcance das ameaças cibernéticas. A Cybersecurity Ventures (CV) estimou que, em 2021, o cibercrime causou um prejuízo anual de aproximadamente US\$ 6 trilhões globalmente, valor que aumentou ao longo dos anos e confirmou uma previsão feita em 2015 pela própria CV, quando os prejuízos foram estimados em US\$ 3 trilhões (CISOADVISOR, 2020). No Brasil, em 2018, a PSafe registrou cerca de 120,7 milhões de links maliciosos detectados, ressaltando a dimensão local dessa ameaça (TEI-

¹ Camada da internet mais conhecida e acessível a todos, sendo a parte da internet que está indexada por mecanismos de busca

² Rede de comunicação descentralizada que visa proteger a privacidade e o anonimato na internet

[XEIRA, 2022](#)).

Diversos trabalhos acadêmicos investigam a identificação de posts maliciosos na *Dark Web* e outros ambientes, abordando diferentes técnicas e métodos. Como o foco deste trabalho é o gerenciamento de bases de dados e a construção de um sistema *backend* para a identificação de mensagens maliciosas em canais e grupos do Telegram, pode-se tomar como exemplo a dissertação de ([AZEVEDO, 2020](#)), na qual são analisados diferentes modelos de referência para o desenvolvimento de aplicações de inteligência de ameaças cibernéticas e sua aplicabilidade para o compartilhamento de dados. Para suportar a funcionalidade de armazenamento de dados, a aplicação deve possuir uma estrutura de armazenamento capaz de sustentar a quantidade de entrada e saída de dados necessária para o bom funcionamento das aplicações. As principais estruturas de armazenamento existentes são bancos de dados relacionais, bancos de dados não relacionais e estruturas customizadas, utilizadas em *softwares* proprietários.

O objetivo geral deste trabalho é desenvolver um sistema de *backend* para a identificação de mensagens maliciosas em canais e grupos do Telegram, utilizando um *bot* criado na plataforma, com o intuito de fornecer uma ferramenta eficiente para auxiliar na análise e categorização de conteúdos que representem ameaças cibernéticas. Para isso, o sistema foi responsável por estruturar e gerenciar um banco de dados para armazenar os conteúdos das mensagens. Além disso, foi implementada uma API para permitir o ingresso em canais e grupos do Telegram pelo *bot* e a execução da coleta das mensagens de forma manual, facilitando a análise desses conteúdos por parte de analistas de segurança e pesquisadores. Este trabalho visa, portanto, não apenas detectar e identificar conteúdos maliciosos, mas também criar uma estrutura escalável que permita a integração com técnicas de aprendizado de máquina, a fim de melhorar a precisão e eficiência da detecção de ameaças.

Durante o desenvolvimento deste trabalho, foram utilizadas diversas ferramentas de *software* cuidadosamente selecionadas para garantir a eficiência e qualidade do sistema em desenvolvimento. A análise detalhada dessas ferramentas será apresentada no capítulo referente ao método. O banco de dados PostgreSQL ([GROUP, 2024](#)) foi empregado para o gerenciamento e armazenamento dos dados que representam as mensagens, oferecendo alta escalabilidade, confiabilidade e eficiência ao fazer isso de forma otimizada. A ferramenta Docker ([ANDRÉ, 2023](#)) foi escolhida para armazenar e instalar as ferramentas no ambiente de desenvolvimento por meio de contêineres, de forma a facilitar o desenvolvimento e o deploy da aplicação. A linguagem Typescript, juntamente com o framework NestJs, foram escolhidos para o desenvolvimento do sistema e da API, permitindo o ingresso do *bot* em canais e grupos do Telegram, além da execução da coleta das mensagens, garantindo grande suporte e integração com as tecnologias previamente mencionadas, além de alta escalabilidade. Todas essas ferramentas foram discutidas, testadas e escolhidas

após comparações com outras opções disponíveis no mercado, visando garantir o melhor resultado possível.

O sistema proposto foi desenvolvido em conformidade com os Termos de Uso e as Diretrizes da API do Telegram. Todas as interações realizadas com a plataforma Telegram respeitam as políticas estabelecidas quanto à coleta, uso e armazenamento de dados. A implementação, baseada na biblioteca GramJS, limita-se ao acesso autorizado a canais públicos e segue as boas práticas recomendadas pela documentação oficial da API, não realizando qualquer forma de coleta automatizada não permitida ou ações que infrinjam os limites de uso da plataforma.

Este trabalho é estruturado em sete capítulos, junto desta introdução. O Capítulo 2 apresenta a fundamentação teórica, explorando os conceitos essenciais para a compreensão do estudo. O Capítulo 3 discute os trabalhos relacionados, analisando pesquisas e projetos que contribuem para a contextualização deste trabalho. No Capítulo 4, são detalhadas a metodologia adotada e as tecnologias e ferramentas essenciais para o desenvolvimento. O Capítulo 5 apresenta todo o desenvolvimento do sistema, desde as configurações iniciais até as regras de negócio necessárias para sua execução. O Capítulo 6 descreve os experimentos conduzidos e avalia os resultados obtidos. Por fim, o Capítulo 7 apresenta as conclusões, destacando as principais contribuições do estudo e apontando possíveis direções para pesquisas futuras.

2 Fundamentação Teórica

Este capítulo apresenta os principais conceitos fundamentais para a compreensão e o desenvolvimento deste trabalho. A seguir, cada um desses conceitos será explorado em detalhes.

2.1 *Deep Web* e *Dark Web*

A *Deep Web* refere-se a uma parte da Internet que não é indexada por motores de busca tradicionais, como Google e Bing, devido ao uso de autenticações ou protocolos não acessíveis por esses buscadores. Ela fica em uma camada mais baixa do que a superfície e é lá que estão cerca de 90% de todos os sites. Essa parte da web oculta é tão grande, que é impossível saber exatamente quantas páginas ou sites estão ativos simultaneamente. Os sites da *Deep Web* podem estar protegidos por senhas ou outras camadas de segurança, enquanto outros podem simplesmente informar aos mecanismos de busca para não pesquisar neles. Sem links visíveis, essas páginas são mais ocultas por outras razões. Na *Deep Web* como um todo, o conteúdo oculto é geralmente mais seguro e legal, abrangendo desde postagens de blogs e novos *layouts* de páginas da web ainda não implementados, até as páginas que você acessa ao usar serviços de bancos digitais, que fazem parte da *Deep Web* (KASPERSKY, 2025).

Essas páginas não apresentam riscos significativos para o computador ou para a segurança em geral. A maioria desses sites é mantida oculta dos navegadores de Internet para proteger informações e a privacidade dos usuários, como contas financeiras com dados bancários e de aposentadoria, contas de mensagens de redes sociais e e-mail, bancos de dados de empresas privadas, informações sigilosas do SUS, como documentação médica e arquivos de processos jurídicos (KASPERSKY, 2025). Ao se aprofundar ainda mais na *Deep Web*, começa-se a aproximar de alguns perigos. Para certos usuários, partes da *Deep Web* representam um meio de burlar restrições e acessar serviços de *streaming* não oferecidos em suas regiões, por exemplo. Outros buscam fazer o *download* de músicas pirateadas ou filmes que ainda nem chegaram aos cinemas.

A *Dark Web*, por sua vez, é um subconjunto da *Deep Web*, significativamente menor do que a web de superfície, onde os sites não estão indexados por mecanismos de busca e são acessíveis apenas por redes específicas, como a TOR (*The Onion Router*), que permite o anonimato ao criptografar e retransmitir dados por vários nós. Essas plataformas se tornaram associadas a atividades ilícitas, como o tráfico de dados e a venda de substâncias ilegais, o que as torna um ponto de interesse para a análise de ciberameaças. Quando se fala em segurança, os perigos da *Dark Web* são diferentes daqueles da *Deep Web*. Embora

não seja fácil se deparar com atividades virtuais ilegais, essas tendem a ser mais extremas e perigosas se você procurar ([KASPERSKY, 2025](#)). Recentemente, houve um aumento nas iniciativas de monitoramento automatizado desses ambientes para identificar ameaças emergentes.

A *Deep Web* surgiu em meados dos anos 1990, quando a internet começou a se popularizar e o surgimento de uma rede centralizada levou à criação de novas estratégias para proteger determinadas informações. Para evitar toda essa exposição surgiu a *Deep Web* ([ANDRION, 2021](#)). Já a *Dark Web* teve início com o lançamento do navegador Freenet em 2000. Um trabalho de tese de Ian Clarke, estudante da Universidade de Edimburgo, na Escócia, a Freenet foi concebida como uma forma de comunicação anônima, troca de arquivos e interação online ([BELCIC; NELSON, 2020](#)).

No começo de 2024, houve um aumento nas iniciativas de monitoramento automatizado desses ambientes para identificar ameaças emergentes. Um exemplo recente de ameaças desse tipo ocorreu em março de 2024, quando o grupo de *hackers* denominado R00TK1T fez um anúncio indicando que o próximo alvo de um ataque cibernético seria a gigante de alimentos Nestlé. Na ameaça, publicada na *Dark Web*, a gangue faz o seguinte alerta: “Atenção, pessoal! Somos o R00TK1T, hoje revelamos nossas intenções malévolas para com a empresa global Nestlé. Preparem-se, o caos é a nossa promessa.”. A mensagem enfatiza a determinação dos *hackers* em perturbar a tranquilidade virtual da Nestlé, sinalizando uma ameaça futura às defesas de segurança cibernética da empresa. Com isso, nota-se a importância de se aumentar o monitoramento e segurança desses ambientes ([CISOADVISOR, 2024](#)).

Com o crescente número de usuários no Telegram, não é surpreendente que algumas pessoas estejam utilizando a plataforma de forma ilícita ([MELIANI, 2024](#)). Uma pesquisa conduzida pela empresa de inteligência cibernética Cyberint, em parceria com o Financial Times, revela a existência de uma rede crescente de *hackers* que utilizam o Telegram para disseminar dados vazados e realizar outras atividades ilegais. Os analistas notaram que essas ações possuem semelhanças com as transações de venda de dados realizadas na *Dark Web*. “Testemunhamos, recentemente, um aumento de mais de 100% no uso do Telegram por cibercriminosos”, relata Tal Samra, analista de ameaças digitais da Cyberint. Entre os dados compartilhados na plataforma, a Cyberint encontrou listas de e-mails e senhas, números de cartão de crédito, cópias de passaporte, credenciais para serviços como Netflix, *malwares* e guias para ataques ([ALECRIM, 2021](#)).

Desta forma, neste trabalho, o Telegram é relevante, pois os dados coletados incluem mensagens de grupos e canais que podem estar relacionados a atividades criminosas. Muitos desses grupos operam em ambientes anônimos e fechados, onde informações sobre fraudes, ataques cibernéticos e outros crimes são compartilhadas. O estudo dessas mensagens pode ajudar a identificar padrões e classificar conteúdos potencialmente perigosos

utilizando inteligência artificial.

2.2 Telegram

O Telegram é um aplicativo de mensagens instantâneas baseado na nuvem, lançado em 2013 pelos irmãos Nikolai e Pavel Durov. Ele se destaca por sua abordagem voltada para segurança, privacidade e velocidade, sendo uma alternativa popular a outros aplicativos de mensagens como o WhatsApp (DUROV; DUROV, 2025b).

O Telegram utiliza um protocolo proprietário chamado MTProto para criptografar comunicações e garantir segurança e privacidade dos usuários. Embora as mensagens comuns não sejam criptografadas de ponta a ponta por padrão, o aplicativo oferece o modo "Chat Secreto", que implementa criptografia de ponta a ponta e impede o armazenamento de mensagens nos servidores do Telegram (NASCIMENTO, 2023).

Uma das características mais notáveis do Telegram é sua API aberta, que permite a criação de *bots* e integração com aplicações externas. A API do Telegram fornece suporte para envio e recebimento de mensagens, gerenciamento de grupos e canais, além da possibilidade de automação de tarefas (DUROV; DUROV, 2025a). No contexto deste trabalho, a API do Telegram é utilizada para capturar mensagens de canais relacionados à *Deep Web* e *Dark Web* e salvá-las em um banco de dados, possibilitando futuras análises sobre crimes cibernéticos e outros comportamentos ilícitos.

3 Trabalhos Relacionados

Neste capítulo, são apresentados alguns dos trabalhos que estão relacionados com esta monografia. O objetivo desses estudos é investigar e explorar os conteúdos relacionados à *Deep Web* e *Dark Web* utilizando diferentes métodos e ferramentas para coleta e análise de dados. Esses trabalhos visam compreender a natureza das atividades ilícitas, a disseminação de informações sensíveis, além de explorar como as tecnologias podem ser empregadas para detectar, monitorar e prevenir crimes cibernéticos nesses ambientes ocultos.

O trabalho de (FILHO, 2024) utiliza dados coletados de fóruns da *Dark Web* e realiza a análise da presença e incidência dos Indicadores de Comprometimento (IoCs), com o objetivo de melhorar a compreensão sobre o uso da *Dark Web* como fonte de dados para CTI (Cyber Threat Intelligence). O estudo coletou 26.575 posts de dois fóruns da *Dark Web*, utilizando ferramentas de extração de dados desenvolvidas pelos próprios autores, além da ferramenta ioc-finder. Durante a análise, foram investigados dez tipos de IoCs, como URLs, e-mails, domínios e endereços IP, com o intuito de identificar padrões relacionados a possíveis atividades cibernéticas maliciosas.

Como resultado, o trabalho contribuiu para a compreensão do papel dos fóruns da *Dark Web* na disseminação de conteúdos maliciosos, fornecendo insumos valiosos para o desenvolvimento de ferramentas de detecção e caracterização de ameaças cibernéticas. Esse estudo oferece contribuições significativas que alinham-se com os objetivos deste trabalho, que também foca na extração e análise de conteúdos maliciosos.

O projeto DarkGram (ROY et al., 2024) se destaca por conduzir uma análise em larga escala de canais dedicados a atividades ilícitas no Telegram. Entre fevereiro e maio de 2024, os pesquisadores monitoraram 339 Canais de Atividades Ciberdelitivas (CACs) no Telegram, que, em conjunto, somavam mais de 23,8 milhões de seguidores. Para identificar e analisar conteúdos maliciosos nesses canais, foi desenvolvido o *framework* DarkGram, baseado no modelo BERT, alcançando uma precisão impressionante de 96% na detecção de postagens maliciosas. Durante o período de estudo, o DarkGram permitiu a análise quantitativa de 53.605 postagens compartilhadas nos CACs.

A análise revelou que os CACs no Telegram distribuem uma ampla gama de conteúdos ilícitos, como credenciais comprometidas, *softwares* e mídias piratas, ferramentas de *hacking*, *malwares*, kits de *exploits* e esquemas de engenharia social. Embora grande parte desse conteúdo seja disponibilizado gratuitamente, os administradores dos canais frequentemente utilizam promoções e sorteios para engajar os usuários, além de impulsionar as vendas de conteúdos ciberdelitivos premium por meio de bots. Este estudo

destaca a importância de monitorar e analisar plataformas de comunicação emergentes, como o Telegram, para identificar e mitigar atividades ilícitas, oferecendo *insights* valiosos para pesquisadores e profissionais de segurança da informação. O DarkGram valida a relevância deste trabalho, demonstrando que a análise automatizada de canais do Telegram é viável e pode gerar impacto significativo na detecção de atividade ilícitas.

O estudo de (SILVA; FORNASIER; KNEBEL, 2020) explora as complexidades dos ambientes da *Deep Web* e *Dark Web*, analisando suas implicações sociais e jurídicas. O objetivo principal da pesquisa é discutir as dimensões éticas, sociais e legais dessas redes, considerando tanto seu uso para práticas legítimas de privacidade quanto para atividades ilícitas. Através de uma análise qualitativa, o estudo examina legislações relevantes, os desafios enfrentados na investigação de crimes cibernéticos e o impacto social dessas redes. Os principais resultados destacam a necessidade de atualização legislativa e de uma maior cooperação entre os setores público e privado para combater o cibercrime, sem comprometer direitos fundamentais como privacidade e liberdade de expressão. Este estudo complementa o presente trabalho ao abordar o contexto legal e ético das atividades na *Deep Web* e *Dark Web*, fornecendo uma base teórica crucial sobre os desafios e repercussões legais que podem surgir na coleta e uso de dados dessas redes. Este conhecimento é essencial para garantir que este trabalho seja realizado em conformidade com as leis vigentes, respeitando os direitos dos usuários.

4 Metodologia

Neste capítulo são apresentadas e explicadas as principais tecnologias e ferramentas essenciais para a execução deste trabalho.

4.1 NestJs

O desenvolvimento de aplicações modernas requer *frameworks* que facilitem a estruturação do código, promovam boas práticas e forneçam ferramentas para escalabilidade e manutenção. O NestJS é um desses *frameworks*, sendo uma opção robusta para a construção de aplicações *backend* em Typescript ou Javascript. Ele é uma ferramenta de código aberto baseada no Node.js e construída sobre o *framework* Express.js ou, opcionalmente, Fastify. Foi projetado para oferecer uma estrutura modular e escalável, sendo altamente inspirado em conceitos do *framework* Angular, como injeção de dependências e modularização (MYŚLIWIEC, 2025).

O NestJs adota uma arquitetura modular, composta por diversos componentes fundamentais:

- **Módulos:** São a base da organização da aplicação, permitindo dividir funcionalidades em partes independentes e reutilizáveis;
- **Controladores:** Responsáveis por lidar com as requisições HTTP e direcioná-las para os serviços apropriados;
- **Serviços:** Contêm a lógica de negócios e podem ser injetados em controladores ou outros serviços, promovendo reutilização e desacoplamento;
- **Middlewares e Guards:** Permitem manipulação de requisições antes de chegarem aos controladores, sendo úteis para autenticação, *logging* e validação.

Neste trabalho, o NestJS é utilizado para a criação de uma aplicação escalável e eficiente para coletar e armazenar dados de mensagens de canais do Telegram relacionados à *Deep Web* e *Dark Web*. A escolha deste *framework* se deu devido à sua capacidade de organização modular, suporte a TypeScript e compatibilidade com o ORM Prisma e o banco de dados PostgreSQL. Além disso, sua integração facilitada com bibliotecas como GramJS possibilita uma comunicação eficiente com a API do Telegram. A escalabilidade do NestJS decorre de sua arquitetura baseada em microsserviços, que permite a distribuição de carga entre diferentes instâncias e a adoção de estratégias como balanceamento de carga e processamento assíncrono. Além disso, o suporte nativo a injeção de

dependências e a separação de responsabilidades facilita a manutenção e a expansão da aplicação conforme a demanda cresce. Sendo assim, sua adoção neste trabalho possibilita um desenvolvimento mais seguro, organizado e alinhado com boas práticas de engenharia de *software*.

Esse *framework* possui uma funcionalidade denominada *schedule*, onde é possível fazer a execução de tarefas agendadas em intervalos específicos, similar aos *cron jobs* dos sistemas Unix. Ela é útil para automações recorrentes, como limpeza de registros no banco de dados, envio de e-mails ou notificações periódicas, sincronização de dados com serviços externos ou até a execução de tarefas assíncronas em horários específicos (GADO, 2024). Dessa forma, nesse trabalho as *schedules* são utilizadas para ajudar na captura de mensagens de forma automática e recorrente, o que será abordado nos capítulos posteriores.

Junto com o NestJs, foi utilizado o npm, um gerenciador de pacotes para NodeJs. O npm consiste em um cliente de linha de comando e um banco de dados online de pacotes públicos e privados pagos. Ele permite instalar, compartilhar e gerenciar dependências de código de forma eficiente (SOFTWARE, 2024).

4.2 Banco de dados e PostgreSQL

Os bancos de dados desempenham um papel fundamental no armazenamento e recuperação de informações em sistemas computacionais. Eles são sistemas estruturados para armazenar, gerenciar e recuperar informações de maneira eficiente. Os bancos de dados relacionais, como o PostgreSQL, organizam os dados em tabelas interligadas por meio de relações, garantindo integridade e consistência dos dados (ALMEIDA, 2024).

Para selecionar o banco de dados ideal para o desenvolvimento desse trabalho, foram feitas análises de rapidez e eficiência em como os dados são inseridos nessas bases de dados. Esse teste foi feito utilizando um computador rodando em um sistema operacional Linux Mint 21.1 Cinnamon, com processador AMD Ryzen 5 PRO 4650GE, 16GB de memória RAM DDR4 com velocidade de 3200Mhz, 20GB de memória swap e para armazenamento um SSD NVME de 1TB com velocidade de leitura de 2400MB/s e 1900MB/s de gravação.

Os bancos de dados analisados foram selecionados de acordo com sua popularização no mercado e seu suporte à armazenamento de dados de texto. Dessa forma, os bancos escolhidos para análise foram o PostgreSQL, o MongoDB e o Cassandra, sendo respectivamente os 2 últimos bancos de dados não-relacionais.

Para a realização do teste, foram inseridos 17675 registros em cada base de dados e contabilizada a duração dessa operação. Cada registro contém dados como sua identifi-

cação (id), o texto completo simulando um post em uma rede social, a data de criação do registro, um campo lógico para identificar se o post possui índices de comprometimento (IoCs) e vários outros demais campos para informar os índices de comprometimento, se houver.

A partir da análise, foi escolhido o PostgreSQL como o banco de dados ideal para o desenvolvimento deste trabalho, em consequência da rapidez de inserção de novos registros durante o teste. A tabela 1 ilustra essa comparação.

Tabela 1 – Comparação da rapidez entre bancos de dados

Banco de dados	Rapidez
PostgreSQL	24 segundos
MongoDB	3 minutos e 13 segundos
Cassandra	maior que 30 minutos

Durante o teste, outras questões foram analisadas, como o suporte, a quantidade e qualidade de fóruns e assuntos disponíveis na internet, a dificuldade de instalar e executar e a quantidades de *softwares* disponíveis que oferecem suporte à esses bancos de dados. O PostgreSQL se sobressaiu em todas essas análises, muito em consequência da sua popularidade no mercado e o tempo desde seu lançamento oficial.

No contexto dos sistemas *backend*, o PostgreSQL é amplamente utilizado devido à sua escalabilidade e eficiência no gerenciamento de grandes volumes de dados. Ele é um Sistema de Gerenciamento de Banco de Dados Relacional (SGBD) open-source altamente escalável e extensível. Essa ferramenta permite a estruturação de bases de dados robustas para armazenar informações extraídas de canais do Telegram, como textos e datas, garantindo integridade e eficiência no acesso às informações. Sua compatibilidade com o ORM Prisma facilita a modelagem e manipulação dos dados, permitindo consultas otimizadas e escalabilidade do sistema.

Existe um conceito denominado *migrations*, que refere-se a um conjunto de scripts ou comandos que facilitam a criação e modificação de um banco de dados. Esse mecanismo permite o versionamento das alterações realizadas, garantindo consistência entre os ambientes, evolução do esquema do banco de dados e maior automação no gerenciamento das mudanças. No projeto, as *migrations* são criadas utilizando o ORM Prisma.

4.3 Prisma

O Prisma é um ORM (*Object-Relational Mapping*) moderno para aplicações Node.js e TypeScript. Ele facilita a interação com bancos de dados, fornecendo uma abordagem tipada e eficiente para manipulação de dados (INC, 2025). Ao contrário de ORMs tra-

dicionais, o Prisma gera automaticamente um cliente baseado no esquema do banco de dados, garantindo maior segurança e facilidade no desenvolvimento.

Neste trabalho, o Prisma é utilizado como ORM para interagir com o banco de dados PostgreSQL, facilitando a criação e leitura de dados coletados das mensagens do Telegram. Sua integração com NestJS permite um desenvolvimento mais ágil e seguro, garantindo escalabilidade ao sistema.

4.4 Docker e Docker Compose

Docker é uma plataforma de virtualização baseada em contêineres que permite empacotar, distribuir e executar aplicações de forma isolada do ambiente subjacente ([ANDRÉ, 2023](#)). A tecnologia de contêineres proporciona maior portabilidade e eficiência no desenvolvimento de *software*, pois elimina conflitos de dependências e facilita a replicação de ambientes.

O Docker Compose é uma ferramenta que permite definir e gerenciar aplicações multicontêiner com Docker por meio de um arquivo YAML. Ele simplifica a orquestração de contêineres, facilitando a inicialização e configuração de ambientes de desenvolvimento e produção ([HABBEMA, 2024](#)).

Juntas, as duas ferramentas são utilizadas para garantir a consistência do ambiente de execução da aplicação, permitindo a fácil replicação e implantação em diferentes máquinas e servidores. Com o Docker, a aplicação é empacotada em contêineres isolados, enquanto o Docker Compose orquestra e configura a comunicação entre esses contêineres, garantindo que todos os componentes necessários, como o banco de dados PostgreSQL e o serviço backend com NestJS, funcionem corretamente em conjunto. Isso assegura que o ambiente de desenvolvimento seja idêntico ao ambiente de produção, minimizando problemas relacionados a discrepâncias de configuração.

5 Desenvolvimento

Neste capítulo, será detalhado o desenvolvimento da aplicação para a recuperação das mensagens de canais e grupos do Telegram, o armazenamento dessas informações e a integração do sistema de classificação de conteúdos maliciosos. O objetivo é fornecer uma visão abrangente e detalhada das etapas e metodologias empregadas ao longo deste trabalho, permitindo uma compreensão mais sólida do processo de identificação de conteúdos maliciosos no Telegram.

5.1 Visão geral

A execução do trabalho proposto foi feita com base em estudo das documentações das ferramentas utilizadas e em referências bibliográficas, sendo dividida nas seguintes etapas:

- **Arquitetura e estruturação do Sistema:** organização do sistema como um todo, desde a estruturação de pastas e arquivos até a organização de seus componentes e funcionalidades;
- **Banco de dados e tabelas:** criação das tabelas no banco de dados e seus esquemas por meio do ORM Prisma;
- **Configuração do projeto:** configuração geral do projeto, incluindo instalação e utilização de VCS, bibliotecas, *lints*, *logger*, variáveis de ambiente, *pipes*, *filters*, *erros personalizados*, integração do Prisma com o NestJs, entre outros;
- **Construção de serviços para operações no banco de dados:** integração do sistema com o banco de dados por meio de serviços criados para a execução de operações utilizando o ORM Prisma;
- **Captura de mensagens em canais do Telegram:** definição das regras de negócios envolvida para a captura de mensagens em canais do Telegram, abrangendo desde a comunicação inicial com o mensageiro até a coleta e armazenamento das mensagens no banco de dados;
- **API para acesso ao sistema:** desenvolvimento da API para a execução de funcionalidades do sistema de forma manual;
- **Classificação das mensagens:** integração ao sistema de categorização automatizada das mensagens coletadas, utilizando técnicas de aprendizado de máquina para identificar conteúdos potencialmente maliciosos;

- **Contêinerização da aplicação:** integração com o ambiente Docker para garantir portabilidade e a replicação do sistema em diferentes ambientes;
- **Documentação e análise dos resultados:** apresentação e discussão dos resultados obtidos, com foco na captura de mensagens em canais e grupos do Telegram.

Todos os códigos e implementações apresentados, bem como os demais materiais utilizados neste trabalho, estão disponíveis no repositório <<https://github.com/christhian12rv/deteccao-incidentes-telegram>> para consulta e acompanhamento.

5.2 Arquitetura e estruturação do sistema

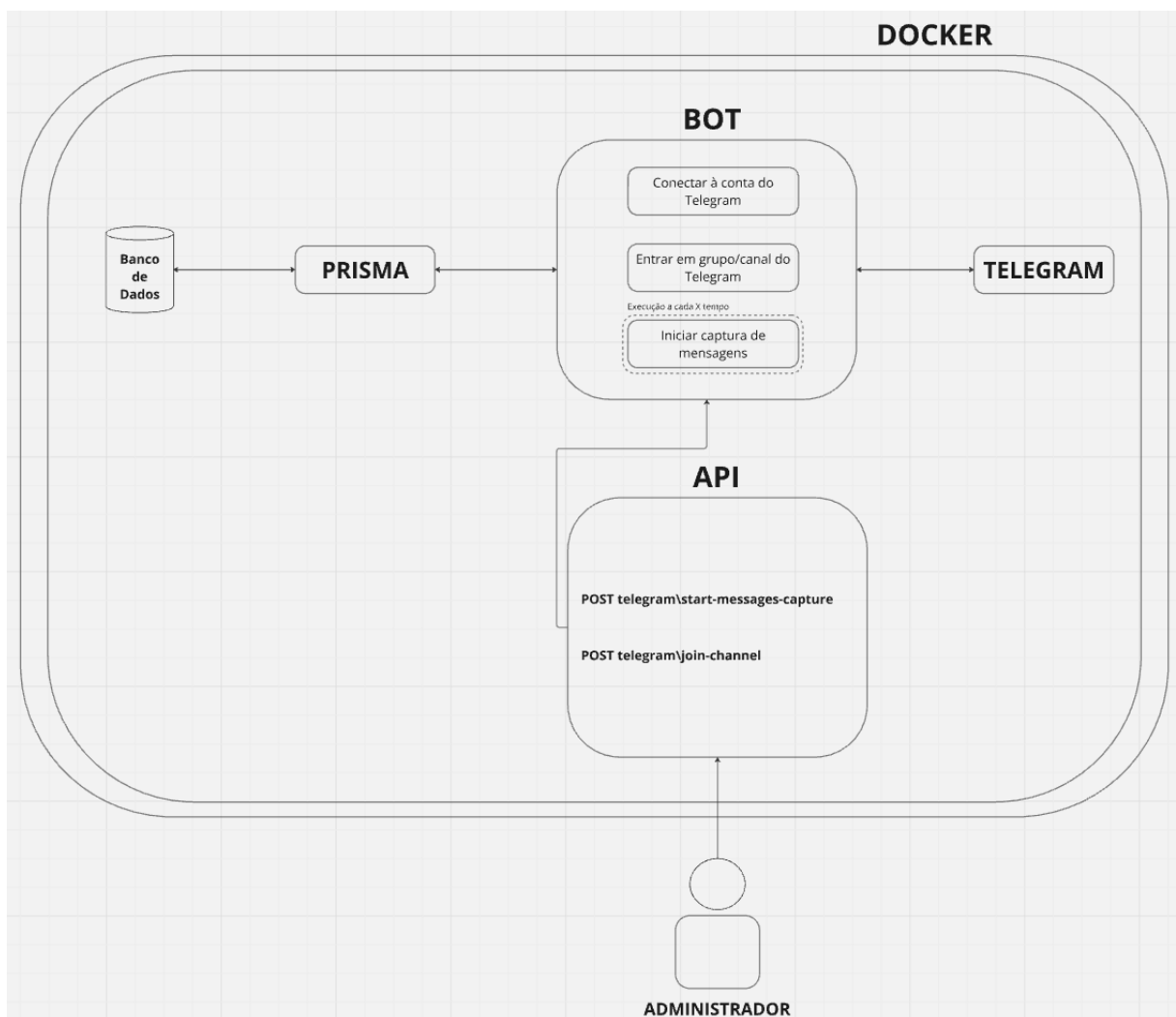


Figura 1 – Arquitetura do sistema. (Fonte: o Autor)

A Figura 1 apresenta a arquitetura geral do sistema. De modo geral, seu funcionamento é simples, com cada componente específico trabalhando em conjunto para alcançar

o objetivo final. O fluxo do sistema será explicado com base na Figura 1, sendo estruturado da seguinte forma:

- **Administrador**: responsável por fazer requisições à API quando necessário;
- **API**: contém as rotas da aplicação, que, ao serem requisitadas, acionam os serviços necessários do *bot*;
- **Bot**: implementa a lógica de negócios geral do sistema, incluindo a conexão com a conta do Telegram, a entrada em canais e a captura de mensagens, entre outros processos essenciais para o funcionamento da aplicação;
- **Telegram**: plataforma de mensagens onde estão registrados os canais e de onde as mensagens são capturadas pelo *bot*;
- **Prisma**: ORM que permite a comunicação entre o *bot* e o banco de dados;
- **Banco de dados**: responsável pelo armazenamento das mensagens capturadas no Telegram, bem como informações sobre os canais e outros dados relevantes;
- **Docker**: responsável pela contêinerização da aplicação.

5.3 Banco de dados e tabelas

Foram criadas três tabelas no banco de dados, *telegram_channel*, *telegram_message* e *telegram_classified_messages*, que armazenam informações sobre os canais, as mensagens e a classificação das mensagens do Telegram, respectivamente. As duas primeiras tabelas foram modeladas utilizando o ORM Prisma, definidas no arquivo *schema.prisma* da aplicação e a terceira tabela utilizando comandos SQL diretamente no banco de dados. O diagrama entidade relacionamento do banco de dados é apresentado na Figura 2.

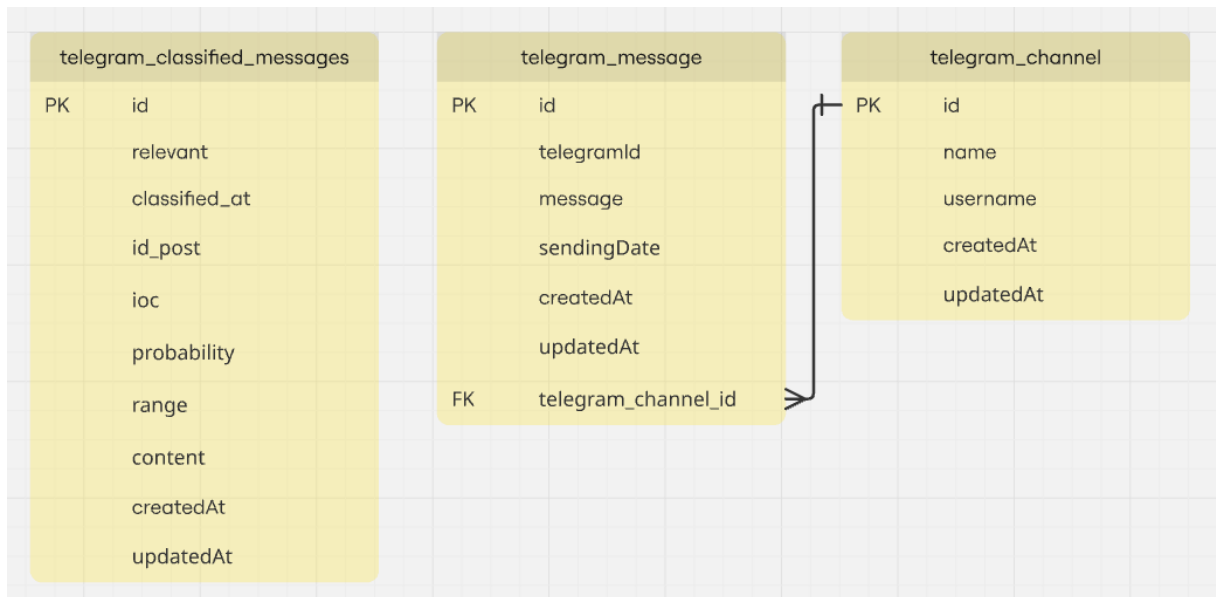


Figura 2 – Diagrama entidade relacionamento do banco de dados. (Fonte: o Autor)

Com o Prisma, é possível criar e aplicar migrações de forma automática utilizando comandos simples, garantindo o versionamento adequado das alterações no banco de dados. As migrações são armazenadas na pasta *migrations*, permitindo um controle estruturado da evolução do esquema do banco.

5.4 Configuração do projeto

Nesta seção, serão detalhadas as configurações gerais do projeto, incluindo a utilização do Git, a instalação das bibliotecas essenciais, a utilização de *lints*, *logger*, variáveis de ambiente, *pipes*, *filters* e *custom errors*, além da integração do Prisma com o NestJs, entre outros aspectos relevantes para a estruturação e funcionamento da aplicação.

5.4.1 Controle de versões e repositórios de código

Para um melhor controle das alterações realizadas no projeto ao longo do tempo, utilizou-se a ferramenta Git, um Sistema de Controle de Versão (VCS), que possibilita o acesso a versões anteriores dos arquivos alterados no código-fonte. Esse processo permite o acompanhamento e controle do desenvolvimento do projeto, proporcionando segurança ao realizar as modificações no código. Além da ferramenta Git, utilizou-se a plataforma GitHub, um repositório online de código que possibilita a utilização do Git.

5.4.2 Configuração geral e instalação das bibliotecas

Todo o desenvolvimento do projeto, exceto a classificação das mensagens, foi realizado utilizando o *framework* NestJs juntamente com a linguagem de programação Typescript e o npm, um gerenciador de pacotes para NodeJs. As principais dependências instaladas no projeto estão descritas abaixo:

- **telegram**: cliente do GramJs, uma biblioteca para manuseamento da api do Telegram;
- **nestjs**: juntamente com essa dependência, foram instaladas várias outras relacionadas para o funcionamento adequado do *framework* NestJs no projeto;
- **prisma**: Dependência necessária para o funcionamento do ORM Prisma;
- **typescript** e **ts-node**: para a utilização correta da linguagem de programação Typescript, é necessário instalar sua dependência. O ts-node foi instalado para permitir a execução do Typescript com um único comando, transpilando e executando o código de forma automática;
- **zod**: biblioteca que valida e transforma dados em aplicações TypeScript. Com ela foi possível fazer validações no conteúdo recebido do usuário por meio da API;
- **eslint**: o eslint é um linter, uma ferramenta de análise estática desenvolvimento para Javascript e Typescript. Com ele é possível definir e controlar regras de estilos e formatação do código, garantindo um código padronizado e a captura e correção de problemas e erros relacionados à formatação, quando identificados;
- **dotenv**: biblioteca que permite gerenciar variáveis de ambiente no projeto.

Também foi utilizada no projeto a funcionalidade *logger* do próprio NestJs para um melhor monitoramento e depuração da aplicação, permitindo registrar eventos, capturar erros e acompanhar o fluxo de execução, facilitando a identificação de problemas e a manutenção do sistema. Além disso, foi criada uma classe de erro customizável denominada *Validation Error* para auxiliar na captura de erros da aplicação.

No NestJs, existe uma funcionalidade chamada pipe, que é utilizada para manipular ou validar dados de entrada em controlares ou manipuladores de rotas. Neste projeto, foi criado um pipe customizável utilizando a biblioteca zod para a validação da entrada do usuário por meio da API nos controladores. A validação também é feita nas classes de serviço *TelegramChannel* e *TelegramMessage*, pois essas fazem operações no banco de dados e há dados que precisam ser validados antes de fazer as operações.

Outra funcionalidade do NestJs são os *filters*, que são utilizados para tratar exceções de forma centralizada e personalizada, permitindo capturar, tratar e formatar

as respostas de erro antes que elas sejam enviadas ao cliente. Assim, em conjunto com o *ZodValidationPipe*, foi criado o *ZodFilter* para capturar exceções da biblioteca zod e formatá-las para serem enviadas de volta ao usuário.

A integração do Prisma com o NestJs foi feita criando um serviço do NestJs, o qual estabelece a conexão ao banco de dados de forma automática assim que o serviço é iniciado no sistema.

Por fim, as variáveis de ambiente são definidas por meio do arquivo *.env* criado na raiz do projeto. Elas são capturadas com a biblioteca *dotenv*, exportadas por meio do arquivo *config.js* e utilizadas na aplicação.

5.5 Construção de serviços para operações no banco de dados

Para realizar as operações no banco de dados por meio do ORM Prisma, foram criados, com o *framework* NestJS, dois módulos e dois serviços, sendo cada conjunto de módulo e serviço relacionado a uma tabela do banco de dados. Toda a regra de negócio das operações foi implementada nos respectivos serviços. Nesse projeto, foi necessária apenas a operação de inserção de dados no banco de dados, sendo assim, apenas o método *create* foi implementado em ambos os serviços.

Por meio do conceito de DTO (*Data Transfer Object*), é possível definir estruturas de dados que serão enviadas e recebidas entre o cliente e o servidor, servindo para validação e transformação dos dados antes de chegarem ao serviço ou banco de dados, garantindo que as requisições sigam um formato esperado. Assim, neste projeto foram criados dois DTOs, relacionados ao método *create* de ambos os serviços, para fazer a comunicação entre outras partes do sistema e também entre o usuário, quando necessário, garantindo uma maior estabilidade e segurança da aplicação.

5.6 Captura de mensagens em canais do Telegram

Nesta seção, é explicada toda a regra de negócios para a captura de mensagens em canais do Telegram, desde a comunicação e conexão inicial com o mensageiro até a coleta e armazenamento das mensagens no banco de dados.

Em um primeiro momento, foi realizada uma análise da API oficial do Telegram e da biblioteca GramJs. Verificou-se que os *bots* criados por meio da ferramenta *Bot Father* disponibilizada pelo Telegram não possuem a funcionalidade de entrar automaticamente em grupos e canais. Assim, para atingir o objetivo deste trabalho, optou-se por utilizar uma conta de usuário oficial do Telegram associada a um número de telefone real, sendo escolhida a conta do autor deste trabalho.

Seguindo com o desenvolvimento, as regras de negócio envolvendo a conexão e a coleta de mensagens foram implementadas por meio do serviço *telegram.service*, onde foram criados métodos para a conexão com a conta do Telegram, a inscrição ou ingresso em canais e grupos, e a coleta das mensagens. Cada uma dessas partes do projeto será explicada nas subseções seguintes.

5.6.1 Conexão com a conta do Telegram

Para a conexão entre o sistema e a API do Telegram, é necessário criar um *Telegram Application*, o qual deve ser feito seguindo a documentação disponível no seguinte endereço: <https://core.telegram.org/api/obtaining_api_id>. Após a criação, são liberadas duas chaves: *api_id* e *api_hash*, as quais são necessárias para o funcionamento da API.

A conexão entre a API do Telegram e o sistema é estabelecida inicialmente ao fornecer as duas chaves mencionadas para a criação de um objeto do tipo *TelegramClient* fornecido pela biblioteca *GramJs*. Após a criação desse *client*, sempre que o sistema for iniciado ou a conexão for interrompida, é necessário estabelecer uma nova conexão chamando o método *connect()*. Em seguida, é necessário realizar a autenticação ao usuário do Telegram, informando o número de telefone e a senha da conta. Quando a conexão for estabelecida e aberta, será solicitado, via terminal da aplicação, o código de autenticação enviado para a conta do Telegram. Após a inserção desse código, caso a configuração esteja correta, a API estará pronta para o uso das funções e requisições.

A biblioteca *GramJs* fornece uma funcionalidade chamada *Store Sessions*, que permite informar uma *string* para a criação de um objeto do tipo *StoreSession*. Esse objeto é então passado como parâmetro na criação do objeto *TelegramClient*. Com isso, é possível salvar a sessão atual do usuário do Telegram em uma pasta, cujo nome é definido pela *string* fornecida. Dessa forma, não é necessário realizar a autenticação novamente sempre que uma nova conexão é estabelecida no sistema.

5.6.2 Ingresso em canais e grupos do Telegram

Para a coleta das mensagens nos canais do Telegram, é necessário conhecer o nome de usuário do canal. Dessa forma, o uso da tabela *telegram_channel* se torna relevante, pois ela armazena o nome do canal e seu respectivo nome de usuário.

Por meio do método *joinChannel()* criado no serviço, é possível fazer com que o *bot* entre automaticamente em um canal específico do Telegram, possibilitando a captura das mensagens, a partir do nome de usuário do canal. Além disso, caso ainda não exista, um novo registro é inserido na tabela *telegram_channel* no banco de dados, registrando o canal ingressado.

5.6.3 Captura de mensagens

A captura de mensagens dos canais registrados é realizada automaticamente por meio da funcionalidade *Schedule* do NestJs. A expressão *cronjob* necessária para executar a *schedule* de forma recorrente, é definida nas variáveis de ambiente da aplicação.

A coleta das mensagens é realizada canal por canal, a partir da consulta aos canais registrados no banco de dados e ingressados pelo *bot* no Telegram, salvando as novas mensagens, se houverem. O número limite de mensagens a serem lidas e salvas também é definido nas variáveis de ambiente da aplicação, evitando a sobrecarga excessiva de operações. Para evitar a duplicidade de mensagens no banco de dados, é feita uma verificação da última mensagem salva, com base na data e hora de envio no Telegram, além do canal que está sendo analisado. Dessa forma, apenas as novas mensagens serão salvas, prevenindo problemas relacionados à duplicação.

Após a coleta das mensagens, as novas serão salvas no banco de dados, contendo informações sobre o seu identificador no canal do Telegram, o conteúdo da mensagem e a data e hora de envio. A data e hora são previamente formatadas com a utilização da biblioteca *moment-timezone*, considerando o fuso horário adequado. Exemplos e resultados do funcionamento serão apresentados no próximo capítulo.

5.7 API para acesso ao sistema

Como especificado na subseção anterior, a captura de mensagens é realizada de forma automática por meio de *cronjobs*. No entanto, foi desenvolvida uma API com requisições HTTP para executar funcionalidades de maneira manual. Foram criadas duas rotas no sistema, as quais são descritas abaixo:

- **POST /telegram/start-messages-capture:** essa rota permite executar manualmente a captura das mensagens do Telegram pelo *bot*. Não é necessário fornecer nenhum parâmetro além da execução da rota por meio do método HTTP POST.
- **POST /telegram/join-channel:** essa rota permite ao *bot* ingressar em canais e grupos do Telegram. É necessário fornecer no corpo da requisição um objeto do tipo JSON com o atributo *channelUsername*, que se refere ao nome de usuário do canal do Telegram.

5.8 Classificação das mensagens

A classificação das mensagens foi realizada utilizando o modelo supervisionado proposto por (FILHO, 2023), que tem como objetivo classificar postagens com base em

sua relevância. Cada mensagem recebeu um valor entre 0 e 1, refletindo o grau de relevância do seu conteúdo. Para fins de análise, foram consideradas apenas as mensagens com *probability* superior a 0.7, pois essas são classificadas como mais relevantes para a detecção de ameaças cibernéticas. Essa filtragem permite um estudo mais direcionado, concentrando-se nos conteúdos que apresentam maior potencial de impacto nos resultados da pesquisa.

Para integrar esse modelo supervisionado ao sistema desenvolvido neste trabalho, foi criado um novo arquivo de classificação de mensagens dentro do projeto já existente de categorização de postagens da *Dark Web*, desenvolvido por (FILHO, 2023). As únicas alterações feitas no projeto referem-se ao SQL necessário para capturar as informações das mensagens no banco de dados, uma vez que anteriormente a captura era voltada exclusivamente para posts da *Dark Web*. Dessa forma, torna-se possível realizar uma classificação precisa e detalhada das ameaças cibernéticas presentes no Telegram, possibilitando análises posteriores por pesquisadores de segurança e outros profissionais da área.

5.9 Contêinerização da aplicação

Para auxiliar na execução e desenvolvimento do sistema, foi integrado um ambiente Docker. Para isso, foram criados dois arquivos na pasta raiz da aplicação, os quais são descritos e detalhados abaixo:

- **Dockerfile:** trata-se de um arquivo de texto utilizado para criar imagens personalizadas para contêineres Docker. Ele automatiza a construção de imagens, especificando as configurações necessárias para executar uma aplicação. Neste trabalho, foi empregado para criar um contêiner com a instalação do NodeJs necessária para rodar a aplicação. Por meio desse arquivo, são copiados os arquivos essenciais para a execução dentro do contêiner, seguidos pela instalação das dependências do projeto, criação das *migrations* do ORM Prisma, compilação e execução da aplicação.
- **docker-compose.yaml:** esse arquivo é utilizado para definir e configurar os serviços, redes e volumes de um aplicativo Docker. Nele, foram configurados dois serviços: um para rodar o sistema, denominado app, e outro para a instalação e criação de um banco de dados local dentro do contêiner denominado db. A criação do banco de dados local facilita o desenvolvimento e os testes pelos desenvolvedores durante o trabalho na aplicação, embora seja possível usar um banco de dados externo, se desejável.

Para facilitar a execução das *migrations* no banco de dados local do contêiner, foi criado o arquivo *run-docker-migrations.sh*. Esse script realiza a conexão com o contêiner

Docker que está relacionado ao banco de dados e executa diretamente as *migrations*, proporcionando um processo de desenvolvimento mais rápido e eficiente.

5.10 Documentação e análise dos resultados

Com o desenvolvimento das etapas anteriores, os resultados alcançados são avaliados e explicados através de experimentos, com foco na coleta de mensagens em canais e grupos do Telegram, por meio de análises tanto visuais quanto conceituais.

Dessa forma, no próximo capítulo serão apresentadas uma visão clara sobre os dados coletados e sua relevância no contexto da segurança cibernética, com o objetivo de contribuir significativamente para futuras investigações e auxiliar na criação de estratégias de prevenção e mitigação de riscos.

6 Estudo de caso

O objetivo deste capítulo é apresentar os resultados obtidos com a implementação e execução do sistema proposto a partir de testes e experimentos. O sistema foi desenvolvido com o objetivo de coletar e armazenar mensagens de canais públicos do Telegram, a fim de auxiliar em análises futuras relacionadas à detecção de possíveis crimes cibernéticos.

Para a execução dos testes, foram escolhidos 8 canais públicos do Telegram relacionados à conteúdos de ameaças cibernéticas e realizada a coleta das mensagens em cada um. Foram coletadas e analisadas o total de 760 mensagens, sendo as 100 últimas enviadas de cada canal. As mensagens que não contêm conteúdo textual, como por exemplo vídeos, imagens, gifs, entre outros não entraram na análise. Estavam disponíveis mensagens de vários anos anteriores desde o período deste trabalho, como 2016, até a data atual do teste, 24 de abril de 2025. Isso se deve ao motivo de não haver novas mensagens enviadas no grupo por muito tempo. Abaixo é exemplificada a coleta de uma mensagem em cada um de 5 dos 8 canais testados.

6.1 Coleta e armazenamento das mensagens

Após a integração bem-sucedida com a API do Telegram, por meio da biblioteca GramJS, foi possível realizar a coleta automatizada de mensagens de canais públicos previamente selecionados. A Tabela 2 apresenta um exemplo real de informações de canais do Telegram salvas no banco de dados PostgreSQL. Já a Tabela 3 apresenta um exemplo real das mensagens armazenadas, contendo os principais campos estruturados conforme o modelo definido no capítulo e seção 5.3.

Tabela 2 – Exemplo de informações de canais armazenadas no banco de dados

id	name	username
1	Cyber Security News	@Cyber_Security_Channel
2	Dark web sites	@Dark_web_sites
3	Android Security & Malware	@androidMalware
4	IT Security Alerts	@itsecalert
5	Cloud and Cybersecurity	@cloudandcybersecurity

Tabela 3 – Exemplo de mensagens armazenadas no banco de dados

id	telegramId	message	telegramChannelId
1	127	The Worst Hacks of 2024 1. China's Salt Typhoon Telecom Breaches 2. Snowflake Customer Breaches 3. Change Healthcare Ransomware Attack. Did we miss anything in the list? Cyber_Security_Channel	1
2	842	The BEST Telegram BOT <link>	2
3	321	A Blueprint of Android Activity Lifecycle <link>	3
4	1121	Extremely critical RCE in PHPMailer! If you got ANY PHP application -> check it ASAP! The POC exploit code is in the wild - we expect large scale attacks tryign to abuse this flaw shortly! (Severity: high) Further Information: <link> alert severityhigh vulnerability zeroday php phpmailer Thanks for your anonymous reports through <link> - help us and report vulnerabilities which affect the mass! Forward this to your admin colleagues & friends!	4
5	131	If you're subscribed to receive emails from Yescertified.com, I will be sharing a 30% off coupon code for my Linux and AWS course bundle sometime tomorrow May 15th, 2024. Want to know what's included in my Linux and AWS course bundle and the benefits of signing up? See this link <link>. To receive the email with the half off coupon code, make sure to subscribe to my emails using the subscribe box at <link> and don't forget to add <email> to your safe senders list.	5

6.2 Classificação das mensagens

A classificação das mensagens foi realizada utilizando um modelo de aprendizado de máquina treinado previamente com um conjunto de dados rotulado, como explicado na seção 5.8. Esse modelo classifica as mensagens em relevantes ou não relevantes, com a probabilidade daquele tipo de conteúdo ser viável para análises considerando o potencial vínculo com atividades ilícitas ou sensíveis relacionadas ao cibercrime.

Durante o teste, foram coletadas 760 mensagens provenientes de canais públicos

do Telegram, das quais 125 foram classificadas como relevantes para o escopo do projeto (aquelas que possuem a coluna *probability* maior ou igual à 0.7), ou seja, aproximadamente 16% das mensagens. A Tabela 4 apresenta um exemplo das mensagens classificadas.

Tabela 4 – Exemplo de mensagens classificadas armazenadas no banco de dados

id	content	probability
1	worst hacks china salt typhoon telecom breaches snowflake customer breaches change healthcare ransomware attack miss anything list cyber security channel	0.835
2	best telegram bot	0.010
3	blueprint android activity lifecycle	0.011
4	extremely critical rce phpmailer got php application check asap poc exploit code wild expect large scale attacks tryign abuse flaw shortly severity high information alert severityhigh vulnerability zeroday php phpmailer thanks anonymous reports help report vulnerabilities affect mass forward admin colleagues friends	0.894
5	subscribed receive emails yescertified com sharing coupon code linux aws course bundle sometime tomorrow may want know included linux aws course bundle benefits signing see link receive email half coupon code make sure subscribe emails using subscribe box www yescertified com forget add support yescertified com safe senders list	0.043

Os dados obtidos indicam que o sistema é capaz de coletar e armazenar de forma eficiente os canais e mensagens no banco de dados. Após isso, o modelo supervisionado foi capaz de classificar de maneira adequada as mensagens armazenadas. Os resultados reforçam a viabilidade da aplicação proposta como ferramenta de apoio à pesquisa e ao monitoramento automatizado de conteúdos publicados em redes sociais associados ao cibercrime.

7 Conclusão

Diante da contínua evolução da tecnologia e do crescimento dos ataques cibernéticos, torna-se essencial o uso de estratégias e ferramentas eficientes para detectar, classificar e neutralizar ameaças digitais, especialmente em ambientes pouco controlados, como o Telegram, onde atividades ilícitas são frequentemente encontradas. Nesse contexto, favorável a práticas ilegais, este trabalho teve como objetivo contribuir para o desenvolvimento de uma aplicação capaz de capturar mensagens no Telegram e realizar sua posterior classificação, visando identificar e categorizar diferentes tipos de ameaças cibernéticas presentes nessas mensagens.

Dessa forma, foi proposto um sistema capaz de capturar mensagens provenientes de canais e grupos do Telegram, com o objetivo de realizar sua categorização. Para isso, foi criada uma aplicação de *bot* utilizando o *framework* NestJs, que executa periodicamente um algoritmo para capturar e armazenar as mensagens no banco de dados. Contudo, também é possível executar essa captura manualmente por meio de uma API construída juntamente com a aplicação. Posteriormente, foi criado um novo *script* de categorização das mensagens, integrado ao sistema desenvolvido por (FILHO, 2023), o que possibilitou uma análise mais eficiente dos conteúdos maliciosos provenientes do Telegram. Uma das limitações do projeto é a ausência de dados relacionados aos usuários que enviaram as mensagens nos canais do Telegram. Isso se deve à restrição da API do Telegram, que só permite a recuperação dessas informações caso o *bot* seja um administrador nos grupos.

Para trabalhos futuros, propõe-se a integração da aplicação com outras redes sociais, como o Whatsapp, ampliando assim a capacidade de análise e identificação de conteúdos maliciosos. Além disso, a implementação de notificações pelo Telegram, acionadas sempre que forem encontrados conteúdos maliciosos durante a classificação, contribuirá para uma análise mais ágil e eficiente sobre novas mensagens, facilitando o monitoramento em tempo real e a resposta rápida a possíveis incidentes de segurança.

Referências

- ALECRIM, E. Telegram vira alternativa à dark web para venda de dados roubados. 2021. Disponível em: <<https://tecnoblog.net/noticias/telegram-aumento-atividade-criminosa-dark-web/>>. Citado na página 15.
- ALMEIDA, M. Banco de dados relacionais: conhecendo conceitos, terminologias e ferramentas. 2024. Disponível em: <<https://www.alura.com.br/artigos/banco-dados-relacionais-conceitos-terminologias-ferramentas>>. Citado na página 20.
- ANDRÉ, A. Entendendo o docker: a plataforma de virtualização de contêineres que revoluciona o desenvolvimento de aplicativos. 2023. Disponível em: <<https://andersonandre.medium.com/entendendo-o-docker-a-plataforma-de-virtualiza%C3%A7%C3%A3o-de-cont%C3%AAneres-que-revoluciona-o-desenvolvimento-1c0670bc78bc>>. Citado 2 vezes nas páginas 12 e 22.
- ANDRION, R. Quem criou a deep web? 2021. Disponível em: <<https://canaltech.com.br/seguranca/quem-criou-a-deep-web-204611/>>. Citado na página 15.
- AZEVEDO, B. W. P. Modelo de referência para o desenvolvimento de aplicações de inteligência de ameaças cibernéticas e sua aplicabilidade para o compartilhamento de dados. 2020. Disponível em: <https://ppee.unb.br/wp-content/uploads/2023/01/PPEE_UNB_BruceAzevedo.pdf>. Citado na página 12.
- BELCIC, I.; NELSON, B. O que é dark web e como acessá-la? 2020. Disponível em: <<https://www.avast.com/pt-br/c-dark-web#:~:text=que%20pode%20encontrar.-,A%20hist%C3%B3ria%20da%20dark%20web,de%20arquivos%20e%20intera%C3%A7%C3%A3o%20online.>> Citado na página 15.
- CISOADVISOR. Prejuízos do cibercrime alcançam US\$ 6 trilhões em 2021. 2020. Disponível em: <<https://www.cisoadvisor.com.br/prejuizos-do-cibercrime-alcancam-us-6-trilhoes-em-2021/>>. Citado na página 11.
- _____. Grupo r00tk1t faz post ameaçador à nestlé na dark web. 2024. Disponível em: <<https://www.cisoadvisor.com.br/grupo-r00tk1t-faz-post-ameacador-a-nestle-na-dark-web/>>. Citado na página 15.
- DUROV, N.; DUROV, P. Bots: An introduction for developers. 2025. Disponível em: <<https://core.telegram.org/bots>>. Citado na página 16.
- _____. Telegram. 2025. Disponível em: <<https://telegram.org/>>. Citado na página 16.
- FILHO, S. A. de J. Identificação de posts maliciosos na dark web utilizando aprendizado de máquina supervisionado. 2023. Disponível em: <<https://repositorio.ufu.br/bitstream/123456789/41232/1/IdentificacaodePostsMaliciosos.pdf>>. Citado 4 vezes nas páginas 11, 30, 31 e 36.

- _____. Extração e análise de indicadores de comprometimento (iocs) em fóruns da dark web. 2024. Disponível em: <<https://ppgco.facom.ufu.br/defesas/identificacao-de-posts-maliciosos-na-dark-web-utilizando-aprendizado-de-maquina>>. Citado na página 17.
- GADO, W. Cron jobs, timeouts e intervals com nestjs. 2024. Disponível em: <<https://www.treinaweb.com.br/blog/cron-jobs-timeouts-e-intervals-com-nestjs>>. Citado na página 20.
- GROUP, P. G. D. Postgresql. 2024. Disponível em: <<https://www.postgresql.org/>>. Citado na página 12.
- HABBEMA, H. Docker: Compose. 2024. Disponível em: <<https://medium.com/@habbema/docker-compose-8c16d02585b4>>. Citado na página 22.
- HOPE, C. Surface web. 2017. Disponível em: <<https://www.computerhope.com/jargon/s/surface-web.htm>>. Citado na página 11.
- INC, P. D. Prisma. 2025. Disponível em: <<https://www.prisma.io/or>>. Citado na página 21.
- KASPERSKY. Deep web e dark web. 2025. Disponível em: <<https://www.kaspersky.com.br/resource-center/threats/deep-web>>. Citado 2 vezes nas páginas 14 e 15.
- KAUR, S.; RANDHAWA, S. Dark web: A web of crimes. 2020. Disponível em: <<https://link.springer.com/article/10.1007/s11277-020-07143-2>>. Citado na página 11.
- MELIANI, F. Telegram deve alcançar 1 bilhão de usuários ativos em um ano, segundo ceo. 2024. Disponível em: <<https://www.tudocelular.com/mercado/noticias/n220239/telegram-1-bilhao-usuarios-ativos-um-ano-ceo.html>>. Citado na página 15.
- MYŚLIWIEC, K. Documentação nestjs. 2025. Disponível em: <<https://docs.nestjs.com/>>. Citado na página 19.
- NASCIMENTO, D. imessage, whatsapp e telegram: qual é o melhor mensageiro? 2023. Disponível em: <<https://macmagazine.com.br/post/2023/08/13/imessage-whatsapp-e-telegram-qual-e-o-melhor-mensageiro/>>. Citado na página 16.
- REDAÇÃO. Telegram está sendo usado cada vez mais por cibercriminosos. 2021. Disponível em: <<https://securityleaders.com.br/telegram-esta-sendo-usado-cada-vez-mais-por-cibercriminosos/>>. Citado na página 11.
- ROY, S. S.; VAFA, E. P.; KHANMOHAMMADI, K.; NILIZADEH, S. Darkgram: explorando e mitigando o conteúdo cibercriminoso compartilhado nos canais do telegram. 2024. Disponível em: <https://arxiv.org/html/2409.14596v1?utm_source=chatgpt.com>. Citado na página 17.
- SILVA, F. V. da; FORNASIER, M. de O.; KNEBEL, N. M. P. Deep web e dark web: implicações sociais e repercussões jurídicas. 2020. Disponível em: <<https://revistas.unilasalle.edu.br/index.php/redes/article/download/6756/pdf>>. Citado na página 18.

SOFTWARE. O que é npm (node package manager) e para que serve? 2024. Disponível em: <<https://programae.org.br/termos/glossario/o-que-e-npm-node-package-manager-e-para-que-serve/>>. Citado na página 20.

TEIXEIRA, L. J. R. Investigação da darknet como fonte de dados para ciberataques. 2022. Disponível em: <<https://repositorio.ufu.br/bitstream/123456789/34740/1/Investiga%C3%A7%C3%A3oDarknetComo.pdf>>. Citado na página 12.

Apêndices

APÊNDICE A – Construção do Sistema

Neste apêndice, faz-se a cobertura pelas principais configurações e funcionalidades do sistema, onde são apresentados códigos e informações relevantes usadas no desenvolvimento do trabalho.

Para fazer a modelagem das tabelas do banco de dados foi o utilizado o ORM Prisma, onde por meio do arquivo *schema.prisma* faz-se a modelagem das tabelas *telegram_channel* e *telegram_message*. A modelagem da tabela *telegram_classified_messages* é feita diretamente no banco de dados por meio de código SQL.

```

1 generator client {
2   provider = "prisma-client-js"
3 }
4
5 datasource db {
6   provider = "postgresql"
7   url      = env("DATABASE_URL")
8 }
9
10 model TelegramChannel {
11   id                Int @id @default(autoincrement())
12   name              String?
13   username          String @unique
14   createdAt         DateTime @default(now())
15   updatedAt         DateTime @updatedAt
16   telegramMessage   TelegramMessage[]
17
18   @@map("telegram_channel")
19 }
20
21 model TelegramMessage {
22   id                Int @id @default(autoincrement())
23   telegramId        Int
24   message           String
25   sendingDate       DateTime
26   telegramChannelId Int
27   telegramChannel   TelegramChannel @relation(fields:
    [telegramChannelId], references: [id])
28   createdAt         DateTime @default(now())

```

```

29     updatedAt                DateTime @updatedAt
30
31     @@map("telegram_message")
32 }

```

Listing A.1 – Modelagem das tabelas utilizando o ORM Prisma

```

1 CREATE TABLE telegram_classified_messages (
2     id int4 NOT NULL DEFAULT
3         nextval('classified_tele_us_id_seq'::regclass),
4     relevant bool NULL,
5     classified_at timestamp NULL,
6     id_post int4 NULL,
7     ioc bool NULL,
8     probability float8 NULL,
9     "range" int4 NULL,
10    "content" varchar NULL,
11    CONSTRAINT telegram_classified_messages_pkey PRIMARY KEY (id)
);

```

Listing A.2 – Script de criação da tabela `telegram_classified_messages` no PostgreSQL

A estruturação de módulos e serviços para operações no banco de dados é feita por meio de arquivos com os respectivos nomes e também com o uso de DTOs para ajudar na validação de informações e criação de objetos.

```

+-- telegram-channel/
|   +-- dtos/
|   |   +-- CreateTelegramChannelDto.ts
|   +-- telegram-channel.module.ts
|   +-- telegram-channel.service.ts
|
+-- telegram-message/
|   +-- dtos/
|   |   +-- CreateTelegramMessageDto.ts
|   +-- telegram-message.module.ts
|   +-- telegram-message.service.ts

```

Listing A.3 – Estruturação de módulos e serviços para operações no banco de dados.

Para a parte de classificação das mensagens, é necessário realizar a consulta delas no banco de dados. É apresentado a seguir o trecho do código-fonte responsável pela captura das mensagens no banco de dados, garantindo que apenas aquelas ainda não classificadas sejam selecionadas. Caso novas mensagens sejam identificadas, o processo

segue com a execução do modelo supervisionado proposto, que realiza a classificação com base no conteúdo das mensagens.

```
#pega o ultimo id classificado
sql1 = "select count(1) from public.telegram_message where id not
       in (select distinct id_post from public.
          telegram_classified_messages)"
result_classificados = con.consultar(sql1)

# pega o ultimo id buscado
sql2 = "select count(1) from public.telegram_message"
result_buscados = con.consultar(sql2)

print(result_buscados)

#verifica se ja tem posts buscados
if result_buscados[0][0] == 0:
    logger.debug("Nao tem posts para serem classificados pois
                nenhum foi buscado.")
    logger.info("Tempo decorrido Total: {}".format(timedelta(
        seconds=time.time() - startClassification)))
    sys.exit(0)
```

Listing A.4 – Código-fonte necessário para busca das mensagens no banco de dados para classificação.

APÊNDICE B – Funcionamento do Sistema

No Apêndice B é apresentado o sistema em funcionamento após o processamento de alguns comandos e *scripts* necessários para a sua correta execução. Também é apresentado o funcionamento da API e os logs de suas respectivas rotas, indicando o sistema em execução.

B.1 Contêinerização da aplicação

Para utilizar a aplicação com contêinerização, é necessário primeiro fazer o *build* e, em seguida, inicializar o contêiner utilizando o Docker Compose. Essas etapas são realizadas com os comandos `docker-compose build` e `docker-compose up`, respectivamente.

```
christian@christian-desktop:~/Documents/UFRJ/Projects/deteccao-incidentes-telegram$ docker compose build
WARN[0000] /home/christian/Documents/UFRJ/Projects/deteccao-incidentes-telegram/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion
Compose now can delegate build to bake for better performances
Just set COMPOSE_BAKE=true
[+] Building 1.4s (14/14) FINISHED
=> [app internal] load build definition from Dockerfile
=> [app internal] load metadata for docker.io/library/node:alpine
=> [app internal] load .dockerignore
=> [app internal] transfer context: 82B
=> [app 1/8] FROM docker.io/library/node:alpine@sha256:dc4d28572e425f9d4c68a6f9c382fbcfec3fa2f8ef0b12cb1d96feabdb479e48
=> [app internal] load build context
=> [app internal] transferring context: 8.86kB
=> CACHED [app 2/8] WORKDIR /usr/src/app
=> CACHED [app 3/8] COPY package*.json ./
=> CACHED [app 4/8] RUN npm install
=> CACHED [app 5/8] COPY . .
=> CACHED [app 6/8] COPY env ./
=> CACHED [app 7/8] RUN npx prisma generate
=> CACHED [app 8/8] RUN npm run build
=> [app] exporting to image
=> [app] exporting layers
=> [app] writing image sha256:b0065138406a9ba9d4d44ce48c4cad5bb0eeea105727cb55af4388a03df03f83
=> [app] naming to docker.io/library/deteccao-incidentes-telegram-app
=> [app] resolving provenance for metadata file
[+] Building 1/1
✔ app Built
```

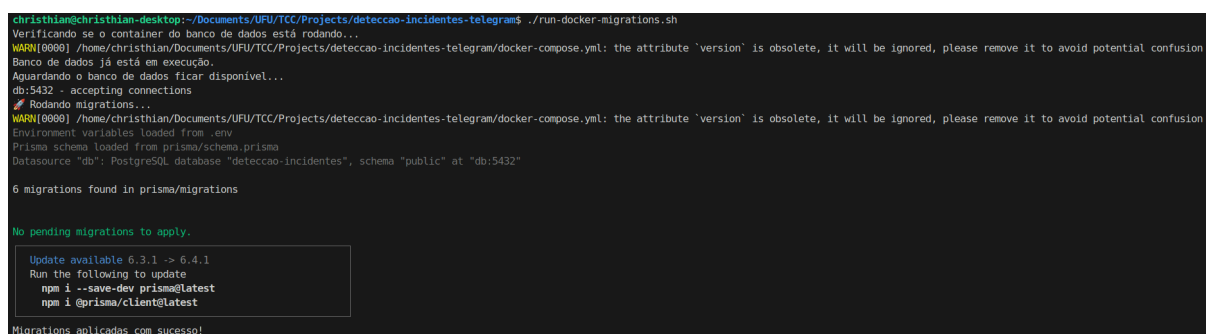
Figura 3 – Execução do comando `docker-compose build`. (Fonte: o Autor)

```
[10:16:38 PM] Starting compilation in watch mode...
deteccao-incidentes-telegram | [10:16:40 PM] Found 0 errors. Watching for file changes.
deteccao-incidentes-telegram |
deteccao-incidentes-telegram | [Nest] 51 - 03/03/2025, 10:16:41 PM LOG [NestFactory] Starting Nest application...
deteccao-incidentes-telegram | [Nest] 51 - 03/03/2025, 10:16:41 PM LOG [InstanceLoader] AppModule dependencies initialized +21ms
deteccao-incidentes-telegram | [Nest] 51 - 03/03/2025, 10:16:41 PM LOG [InstanceLoader] PrismaModule dependencies initialized +0ms
deteccao-incidentes-telegram | [Nest] 51 - 03/03/2025, 10:16:41 PM LOG [InstanceLoader] DiscoveryModule dependencies initialized +1ms
deteccao-incidentes-telegram | [2025-03-03T22:16:41.550] [INFO] - [Running gramJS version 2.26.21]
deteccao-incidentes-telegram | [Nest] 51 - 03/03/2025, 10:16:41 PM LOG [InstanceLoader] TelegramMessageModule dependencies initialized +16ms
deteccao-incidentes-telegram | [Nest] 51 - 03/03/2025, 10:16:41 PM LOG [InstanceLoader] TelegramChannelModule dependencies initialized +0ms
deteccao-incidentes-telegram | [Nest] 51 - 03/03/2025, 10:16:41 PM LOG [InstanceLoader] ScheduleModule dependencies initialized +0ms
deteccao-incidentes-telegram | [Nest] 51 - 03/03/2025, 10:16:41 PM LOG [InstanceLoader] TelegramModule dependencies initialized +1ms
deteccao-incidentes-telegram | [Nest] 51 - 03/03/2025, 10:16:41 PM LOG [RoutesResolver] TelegramController {/telegram}: +7ms
deteccao-incidentes-telegram | [Nest] 51 - 03/03/2025, 10:16:41 PM LOG [RouterExplorer] Mapped {/telegram/start-messages-capture, POST} route +5ms
deteccao-incidentes-telegram | [Nest] 51 - 03/03/2025, 10:16:41 PM LOG [RouterExplorer] Mapped {/telegram/join-channel, POST} route +1ms
deteccao-incidentes-telegram | [Nest] 51 - 03/03/2025, 10:16:41 PM LOG [NestApplication] Nest application successfully started +89ms
deteccao-incidentes-telegram | [Nest] 51 - 03/03/2025, 10:16:41 PM LOG Server running on port 3000...
```

Figura 4 – Execução do comando `docker-compose up`. (Fonte: o Autor)

Para garantir o funcionamento correto do sistema é necessário rodar as *migrations* criadas pelo Prisma dentro do contêiner. Para isso, basta executar o *script* `run-docker-`

migrations.sh. Caso novas migrações pendentes que precisem ser aplicadas, o *script* irá executá-las automaticamente.



```
christian@christian-desktop:~/Documents/UFU/TCC/Projects/deteccao-incidentes-telegram$ ./run-docker-migrations.sh
Verificando se o container do banco de dados está rodando...
WARN[0000] /home/christian/Documents/UFU/TCC/Projects/deteccao-incidentes-telegram/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion
Banco de dados já está em execução.
Aguardando o banco de dados ficar disponível...
db:5432 - accepting connections
🔥 Rodando migrations...
WARN[0000] /home/christian/Documents/UFU/TCC/Projects/deteccao-incidentes-telegram/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion
Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma
Datasource "db": PostgreSQL database "deteccao-incidentes", schema "public" at "db:5432"

6 migrations found in prisma/migrations

No pending migrations to apply.

Update available 6.3.1 -> 6.4.1
Run the following to update
npm i --save-dev prisma@latest
npm i @prisma/client@latest

Migrations aplicadas com sucesso!
```

Figura 5 – Execução do *script* run-docker-migrations.sh. (Fonte: o Autor)

B.2 Uso da API

Todos os testes da API, incluindo os apresentados nas figuras, foram realizados utilizando o Postman, um *software* que facilita aos desenvolvedores criar, compartilhar, testar e documentar APIs. O Postman permite aos usuários criar e salvar solicitações HTTP e HTTPS simples e complexas, além de ler suas respostas. Para facilitar e demonstrar o funcionamento da API, foi utilizado o *build* da aplicação por meio do Docker, utilizando o Docker Compose.

- **POST telegram/join-channel:** Como mostrado na Figura 6, foi escolhido para testes o ingresso no canal com nome de usuário *cybersecurityexperts*. Por meio do parâmetro *channelUsername* passado no corpo da requisição no formato JSON, informamos o canal e clicamos em *Send* para iniciar a requisição. Abaixo, pode-se ver o resultado da requisição e seu *status* 201, indicando sucesso. O corpo da resposta trás os dados criados no banco de dados na tabela *telegram_channel*, contendo os atributos *id*, *name*, *username*, *createdAt* e *updatedAt*, indicando respectivamente o identificador do registro, o nome do canal extraído da API do Telegram, o nome de usuário, a data de criação e a data de atualização.

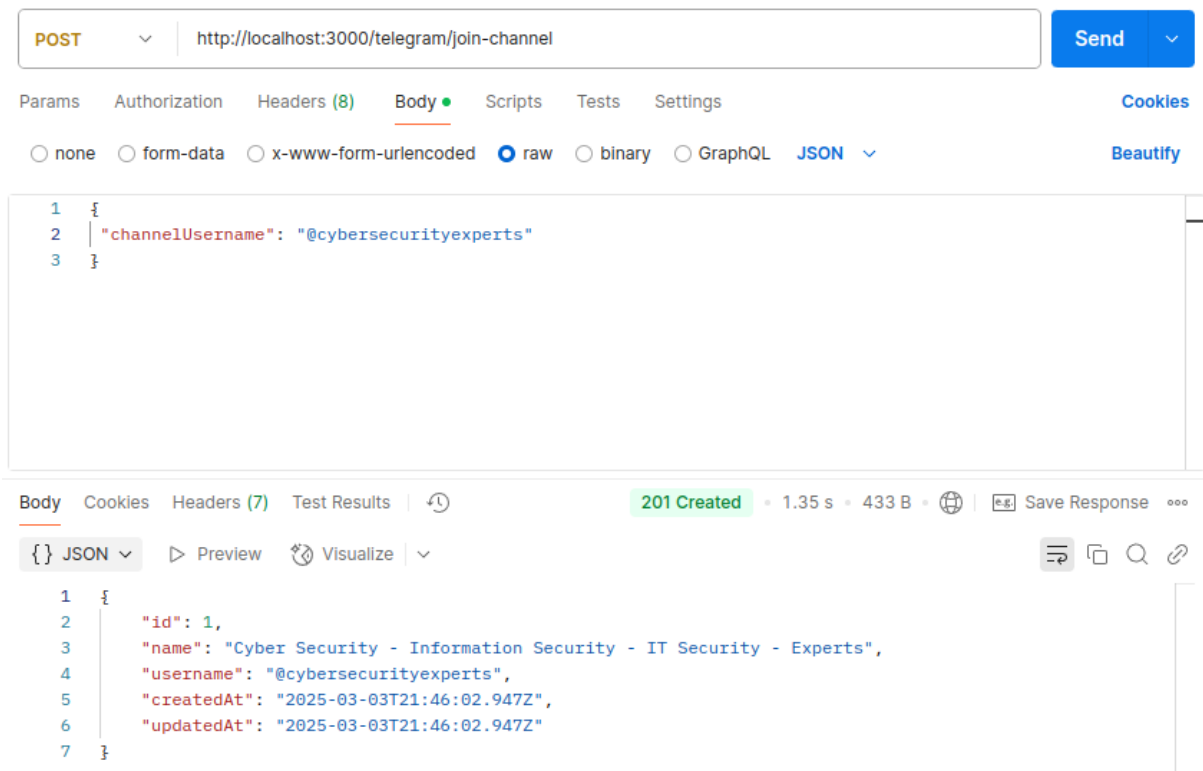


Figura 6 – Requisição HTTP POST à rota telegram/join-channel. (Fonte: o Autor)

- **POST telegram/start-messages-capture:** A partir das Figuras 7 e 8, é possível visualizar a chamada da requisição e o funcionamento da captura de mensagens, respectivamente. Não é necessário passar nenhum parâmetro, basta chamar a requisição por meio do método HTTP POST. Foram registrados no banco de dados e adicionados mais dois canais para proporcionar uma melhor visualização e entendimento do funcionamento da API.

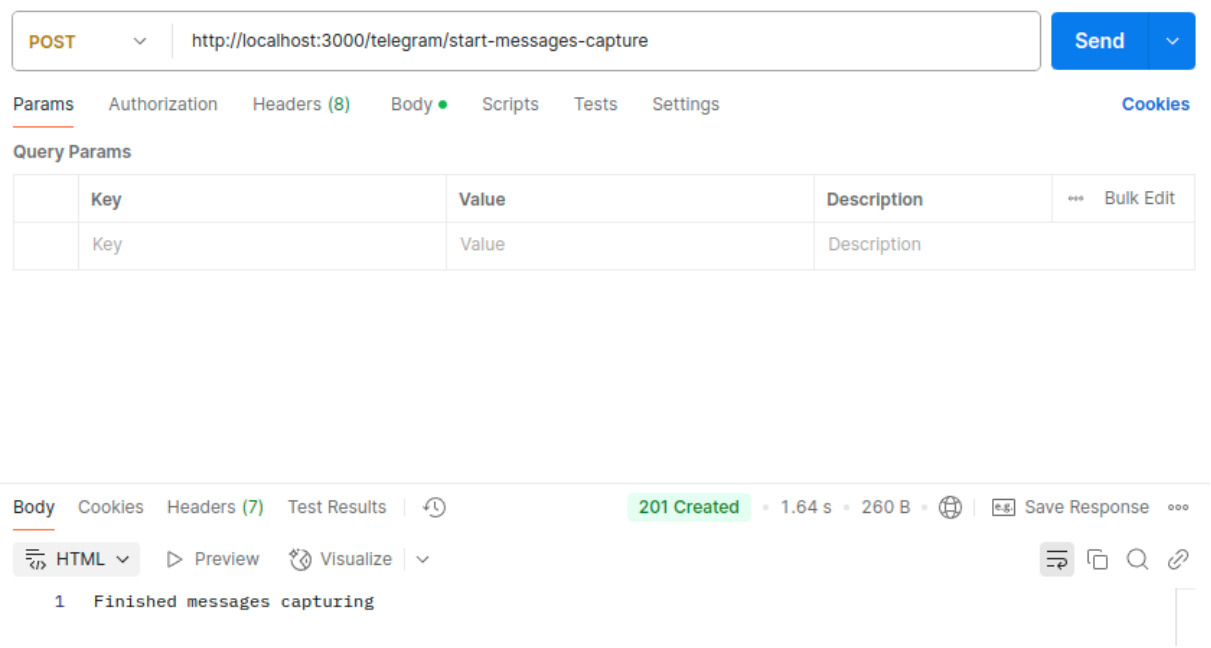


Figura 7 – Requisição HTTP POST à rota telegram/start-messages-capture. (Fonte: o Autor)

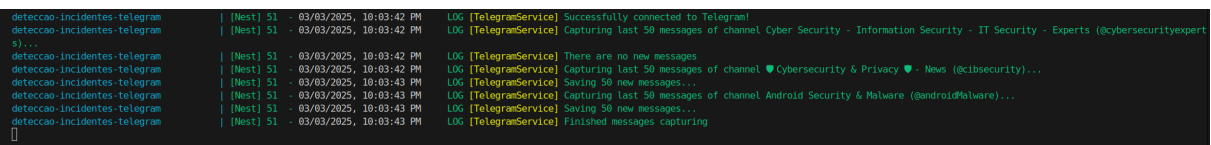


Figura 8 – Funcionamento da rota telegram/start-messages-capture. (Fonte: o Autor)