Curso de Sistemas de Informação

Universidade Estadual de Mato Grosso do Sul

SISTEMA COLETOR DE DADOS EM TEMPO REAL

Anderson Hiroshi Mori Correia

Profa. Dra. Raquel Marcia Müller

Dourados - MS 2024

Sistema Coletor de Dados em Tempo Real

Anderson Hiroshi Mori Correia

Este exemplar corresponde à redação final da monografia da disciplina Projeto Final de Curso devidamente corrigida e defendida por Anderson Hiroshi Mori Correia e aprovada pela Banca Examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Sistemas de Informação.

Dourados, 13 de novembro de 2024

Profa. Dra. Raquel Marcia Müller

C847s Correia, Anderson Hiroshi Mori

Sistema coletor de dados em tempo real / Anderson Hiroshi Mori Correia. –

Dourados, MS: UEMS, 2024.

55 p.

Monografia (Graduação) – Sistemas de Informação – Universidade Estadual de Mato Grosso do Sul, 2024.

Orientadora: Profa. Dra. Raquel Marcia Müller.

1. Fluxo de dados 2. Reddit 3. Visualização de dados I. Müller, Raquel Marcia

II. Título

CDD 23. ed. - 006.754

Ficha Catalográfica elaborada pela bibliotecária da Universidade Estadual de Mato Grosso do Sul (UEMS)

Bruna Peruffo Vieira – CRB 1/2959

Curso de Sistemas de Informação

Universidade Estadual de Mato Grosso do Sul

SISTEMA COLETOR DE DADOS EM TEMPO REAL

ANDERSON HIROSHI MORI CORREIA

Novembro de 2024

Banca Examinadora:

Profa. Dra. Raquel Marcia Müller (Orientador)

Área de Computação – UEMS

Prof. Dr. Diogo Fernando Trevisan

Área de Computação – UEMS

Prof. Dr. Jorge Marques Prates

Área de Computação – UEMS

Agradecimentos

Nesta seção, gostaria de expressar minha profunda gratidão a todas as pessoas que contribuíram para a realização deste trabalho.

Primeiramente, agradeço à minha orientadora, Profª. Drª. Raquel Marcia Müller, pela orientação, apoio e pelas valiosas contribuições na escrita e correção deste trabalho. Sua expertise e dedicação foram fundamentais não apenas para o desenvolvimento do projeto, mas também para meu crescimento acadêmico.

Agradeço também aos meus colegas, que sempre compartilharam ideias valiosas e contribuíram com discussões enriquecedoras, tornando esta jornada mais colaborativa e significativa.

Um agradecimento especial à minha esposa, Caroline de Gois Santos Mori, que sempre esteve ao meu lado, oferecendo amor, incentivo e compreensão. Seu apoio incondicional foi essencial para que eu pudesse superar os desafios e seguir em frente.

Por fim, quero expressar minha eterna gratidão aos meus pais, que nunca me deixaram desistir do ensino. Seu incentivo constante e crença em meu potencial foram pilares fundamentais em minha trajetória acadêmica.

Dourados, Outubro de 2024

Anderson Hiroshi Mori Correia

RESUMO

Um dos desafios atuais na área da Computação é lidar com o aumento contínuo de dados gerados constantemente por sistemas de tempo real e outros ambientes dinâmicos de processamento, especialmente de redes sociais como o *Reddit*, que são importantes fontes de grandes volumes de dados dinâmicos, conhecidos como *fluxo de dados*. Estes crescem de maneira contínua e indefinida ao longo do tempo. Esta pesquisa apresenta um estudo sobre técnicas de captura desses fluxos de dados, com o desenvolvimento de um sistema para coleta em tempo real, integrado a uma aplicação existente que facilita o armazenamento seguro dos dados em um banco de dados e suporta o carregamento, análise, processamento e visualização dos dados também em tempo real.

Palavras-chave: fluxo de dados, Reddit, visualização.

SUMÁRIO

1. Introdução **13**

1.1 Motivação e Justificativa....13

1.2 Objetivos....14

1.3 Metodologia....15

1.3.1 Escolha da Fonte de Dados....15

1.3.2 Criação do Robô....16

1.4 Organização do Texto....17

2. Revisão Bibliográfica 18

2.1 Fluxos de Dados....18

2.2 Tempo e Dados Coletados no Tempo....18

2.3 O Reddit....20

2.4 O que é o Reddit....21

2.5 O que é um *Bot* de Rede Social....22

2.6 O que é uma API....22

2.6.1 Como as APIs funcionam....24

2.6.2 APIs REST....24

2.7 Banco de Dados....25

2.7.1 PostgreSQL....26

2.8 Spring Boot....28

3. Desenvolvimento 30

3.1 Primeiro contato com o *Reddit*....30

3.2 Preparando o projeto....33

3.2.1 Entendendo algumas limitações....33

3.2.2 Criando o projeto....34

3.2.3 Arquitetura de desenvolvimento....34

3.3 Gerenciando a credencial....36

3.3.1 O access token....36

3.3.2 Codificando a classe de credencial....36

3.4 Otimizando os endpoints da API....37

3.4.1 A classe BaseReddit....37

3.5 Modelando as entidades....38

3.6 Iniciando a leitura das postagens....42

3.6.1 Controlador....42

3.6.2 Services....43

3.7 Salvando e classificando as postagens....46
3.7.1 O método fetchPosts....47
3.7.2 O método savePosts....47
3.7.3 Definindo as categorias....48
3.8 Executando o robô....49
3.9 Integração com o THSC....52
3.9.1 Implementação em C++....52

4. Resultados Obtidos 55
4.1 Desafios Encontrados....55
4.1.1 Penalidades da API....55
4.1.2 Desafios na Paginação de Postagens....55
4.2 Análise dos Resultados....56
4.2.1 Resultados do Robô....57
4.2.2 Linguagem de Programação....57
4.2.3 Banco de Dados....58
4.2.4 Frameworks....59
4.2.5 Ambiente de Desenvolvimento Integrado....60
4.2.6 Termos de Tecnologia....61
4.2.7 Sistema Operacional....62
4.3 Resultados do THSC....64

5. Conclusão Geral 66
5.1 Avaliação dos Objetivos e Resultados....66
5.2 Desafios e Adaptações....66
5.3 Contribuições e Aplicações Práticas....66
5.4 Reflexão Pessoal e Crescimento Profissional....67
5.5 Código Fonte....67
5.6 Conclusões Finais....67

LISTA DE FIGURAS

3.1 *Script* em *Python* que ilustra o primeiro contato com a API.....31
3.2 Resultado da execução do *script* usado para teste.....32
3.3 Diagrama de arquitetura do sistema coletor de dados.....35
3.4 Diagrama de Entidades-Relacionamentos (DER).....41
3.5 Progresso exibido no terminal durante a atividade do robô.....50
3.6 Resultado exibido no terminal ao encerrar o robô.....51
3.7 Código que realiza a extração de dados do *PostgreSQL* para carregamento e análise no THSC.....54

4.1 Mapa de cores das categorias "Linguagens de Programação"e "Termos de Tecnologia". 65
4.2 Mapa de cores e pontos representando valores totais e de categoria individual……65

Capítulo 1

1. INTRODUÇÃO

1.1 Motivação e Justificativa

Um dos desafios atuais da área da computação é conseguir lidar com o crescente aumento do volume de dados gerado continuamente por sistemas de tempo real e outros ambientes dinâmicos de processamento. Tais dados podem ser oriundos das mais diversas fontes, como produção científica e literária, notícias, monitoramento e sensoriamento remoto, simulações, experimentos, redes sociais, entre outras. Ainda, em várias áreas de aplicação, existe a necessidade de que estes dados, que crescem continuamente com o tempo, sejam processados e analisados à medida que são coletados. Tais dados são referenciados como fluxo de dados (AGGARWAL, 2007). A complexidade na análise e interpretação desses conjuntos de dados advém não somente do seu tamanho, mas também do número de atributos associados a cada instância ou elemento do conjunto.

Um atributo representa uma característica de um dado (HAN; KAMBER; PEI, 2011). Por exemplo, no âmbito das mídias sociais, uma única postagem, publicada através do *Reddit*, pode ser composta por mais de dez campos de informação, que identificam o autor do texto, a data e a relevância, entre outros atributos. Quando instâncias de dados possuem múltiplos atributos que podem ser interpretados como elementos de um conjunto *m*-dimensional, tais dados são denominados *multidimensionais*, em que *m* é o número de atributos envolvidos. Adicionalmente, atributos que representam uma localização, geralmente dada por latitude e longitude, são chamados dados geoespaciais, ou espaciais. Estes dados buscam ser representações da superfície terrestre e estão relacionados com seu posicionamento, ou localização no espaço geográfico, em outras palavras, podem ser posicionados em determinada região geográfica, tendo por base suas coordenadas. Assim, de acordo com essas informações, torna-se possível a análise do espaço geográfico.

Em aplicações de visualização, a localização espacial (ou geolocalização) se tornou um recurso importante, pois permite a criação de metáforas de visualização baseadas em mapas. Porém, embora seja um atributo intrínseco em dados multidimensionais gerados por mídias sociais atuais, como o *Reddit*, *X* ou o *Facebook*, a localização espacial nem sempre pode ser identificada diretamente, pois é opcional para o usuário. Ou seja, o usuário pode bloquear a identificação da localidade espacial de origem da postagem que está sendo feita.

Na ausência da localidade espacial, algumas técnicas para definição desse atributo podem ser utilizadas. A mais simples envolve a aplicação de um algoritmo randômico com pesos, que utiliza um conjunto de dados com as latitudes e longitudesdas localidades de interesse. Essa técnica é viável para os casos em que a região espacial considerada é conhecida, porém não pode ser utilizada em situações em que se requer uma maior precisão de localização.

A coleta de dados multidimensionais oriundos de mídias sociais é uma tarefa comum em vários setores, desde o ramo empresarial até o ambiente científico, com o objetivo de realizar análises dos mais diversos tipos. Essa demanda, inclusive, resultou na definição de uma nova área de pesquisa, a Ciência de Dados. A tarefa de coleta pode ser facilitada pelos próprios provedores, os quais disponibilizam bibliotecas específicas para serem utilizadas em aplicações que coletam seus dados. Tais aplicações são comumente denominadas de *robôs*, pois podem ficar em execução por um grande período, realizando a coleta dos dados de interesse, sem que seja necessária a interferência humana.

Um sistema coletor de dados foi desenvolvido com o objetivo de ser integrado à aplicação *Time Hierarchy Stream Cube* (THSC), desenvolvida por Müller(MüLLER, 2020), que usa uma estrutura de dados do tipo cubo de fluxos. Após a coleta, foram realizados estudos para verificar se as localizações espaciais foram ou não determinadas pelo usuário, pois tal informação é essencial para a integração com THSC. Em caso afirmativo, a localização foi utilizada como um atributo válido do dado, expressa através de uma latitude e uma longitude. Nos casos em que a localização espacial foi indeterminada, esta foi definida por um algoritmo randômico com pesos.

1.2 Objetivos

O objetivo geral deste projeto foi o desenvolvimento de uma aplicação para coleta de dados oriundos de redes sociais, aplicada ao *Reddit*, com determinação da localização espacial através de um algoritmo randômico de pesos, caso necessário. A aplicação foi integrada à estrutura do cubo de fluxos THSC, descrito por Müller (MüLLER, 2020), para permitir análise e visualização dos dados coletados em tempo real.

Os objetivos específicos alcançados foram:

1.1 Estudo dos conceitos relacionados a dados multidimensionais coletados no tempo, considerando trabalhos existentes na literatura específica.

1.2 Desenvolvimento de uma aplicação, um robô, para coleta de dados do *Reddit* em tempo real.

1.3 Armazenamento dos dados coletados em um banco de dados.

1.4 Teste do sistema desenvolvido usando metáforas de visualização para os dados, em tempo real, através de mapas de calor, através da aplicação do cubo de fluxos THSC(MüLLER, 2020).

1.3 METODOLOGIA

O desenvolvimento deste projeto envolveu três etapas principais, que foram definidas com o intuito de se alcançar os objetivos listados anteriormente. Essas etapas são descritas a seguir.A primeira etapa envolveu o estudo dos conceitos relacionados a dados multidimensionais coletados no tempo, considerando trabalhos existentes na literatura específica. Nessa fase, foram desenvolvidas algumas atividades iniciais, a primeira delas, o levantamento bibliográfico, com a busca de materiais relacionados ao tema nas plataformas de pesquisa. A leitura analítica foi realizada com o objetivo de ordenar e resumir as informações contidas nas fontes: todas as publicações foram lidas e analisadas na íntegra para validar aspectos como os objetivos dos achados e as principais conclusões.

Após a pesquisa bibliográfica inicial, que foi atualizada conforme o projeto se desenvolveu, foi feito um estudo para o desenvolvimento do robô, para a coleta dos dados em tempo real. Nessa fase, foi realizado um teste inicial de conexão com a API (*Application Programming Interface*) do *Reddit*, utilizando a linguagem *Python*, a qual forneceu a possibilidade de usar um módulo específico dedicado à utilização da API, facilitando assim o primeiro uso para qualquer iniciante.

1.3.1 Escolha da Fonte de Dados

Em uma segunda etapa, foi definida a rede social a ser utilizada como fonte de dados para o estudo e criação do robô. Inicialmente, a escolha era o *Twitter* (atual *X*), devido à sua ampla utilização no mundo e à disponibilidade de dados em tempo real que ele historicamente ofereceu. O (então) *Twitter*, sendo uma plataforma líder no compartilhamento de informações e opiniões, apresentava-se como um ambiente propício para a coleta de dados multidimensionais, especialmente aqueles relacionados a tópicos em constante evolução, como notícias, tendências e eventos globais.

No entanto, recentemente, a rede social passou por mudanças significativas em sua administração e propriedade, incluindo seu nome, o que levou a uma reavaliação de suas políticas e diretrizes de acesso à sua API. Tais mudanças resultaram em restrições mais rígidas quanto ao acesso e disponibilidade de recursos, incluindo também a cobrança de valores significativos pela coleta de dados.

Diante desse novo contexto, tornou-se desafiador e, em alguns casos, impraticável utilizar o *X* como a principal fonte de dados para a criação desse robô. Dessa forma, foi necessário adaptar a abordagem e buscar alternativas viáveis para concluir os objetivos deste projeto. Foi realizado um estudo da viabilidade de outras redes sociais e plataformas que pudessem fornecer dados relevantes e acessíveis para a análise pretendida. Após uma análise cuidadosa das opções disponíveis, a decisão foi de utilizar o *Reddit* como a nova fonte de dados principal. O *Reddit* é uma plataforma que oferece uma ampla gama de tópicos de discussão, conhecida por suas comunidades/*subreddits* ativas e diversificadas. Além disso, oferece um ambiente que permite a coleta de dados estruturados, incluindo postagens, comentários e informações de usuários. A mudança para o *Reddit* foi uma decisão fundamentada, que permitiu que o estudo continuasse a explorar o potencial das redes sociais como fonte de dados multidimensionais.

1.3.2 Criação do Robô

A terceira etapa envolveu a criação do robô e os testes de sua funcionalidade. Foi utilizado o acesso ao aplicativo *Reddit*, com a solicitação de acesso à API através da criação de uma aplicação, para assim obter os *tokens* de acesso e a partir daí extrair os dados. Para o desenvolvimento do robô propriamente dito, os estudos iniciais envolveram a definição da melhor linguagem de programação a ser utilizada. Como a aplicação fim (o THSC) foi desenvolvida em *C++*, era preciso que houvesse uma forma de integração entre o sistema coletor e a aplicação já existente. Nos estudos realizados, a linguagem *Python* foi uma das que se destacou, por conta das especificidades necessárias, como um módulo de requisições HTTP para acessar a API do *Reddit*. No entanto, com a busca em documentos de ajuda mais detalhados, optamos por utilizar *Java*.

A integração entre o robô coletor em *Java* e a aplicação THSC revelou-se uma escolha altamente viável, pois o propósito central do robô era efetuar a coleta de dados no aplicativo *Reddit*. Este processo foi simplificado pelo fato de que, para que a aplicação THSC funcionasse em perfeita harmonia com o robô, era necessário apenas que este armazenasse os dados coletados em um banco de dados.

1.4 Organização do Texto

Neste capítulo, abordamos a motivação e justificativa para o desenvolvimento do projeto. A sequência do texto apresenta os fundamentos computacionais e matemáticos utilizados para o desenvolvimento do robô coletor de dados proposto. São apresentados também, detalhes do desenvolvimento e testes que comprovam a funcionalidade da aplicação desenvolvida. Além desse, o texto está organizado em outros quatro capítulos, cujos conteúdos são sumarizados a seguir.

No Capítulo 2 são detalhados os conceitos principais que envolveram o desenvolvimento do projeto, abordando fluxos de dados, a relação entre dados e tempo, bem como os conceitos que envolvem o *Reddit* outras ferramentas computacionais utilizadas.

O Capítulo 3 descreve em detalhes as etapas de desenvolvimento do robô coletor de dados, que inclui o uso da API do *Reddit* e a modelagem do banco de dados utilizado. Esse capítulo também apresenta os testes iniciais realizados com o robô, usando os dados coletados e um relato dos principais desafios encontrados nessa etapa.

No Capítulo 4 são apresentados os resultados obtidos no projeto. Por fim, o Capítulo 5 conclui o texto, com a conclusão, que inclui uma discussão dos resultados e contribuições do trabalho realizado.

Capítulo 2

Revisão Bibliográfica

2.1 Fluxos de Dados

Nos últimos anos a evolução das tecnologias de *hardware* e *software* fizeram surgir uma grande quantidade de *fluxos de dados* trafegando continuamente através das redes de computadores, tornando sua análise um processo desafiador. Segundo Aggarwal (AG- GARWAL, 2007), fluxos de dados (*data streams*) são conjuntos de dados coletados no tempo, que podem crescer de forma rápida, contínua e indefinida.

Como o volume de dados subjacentes envolvido pode se tornar muito grande, fluxos de dados envolvem alguns desafios computacionais. Com o aumento do volume de dados, não é possível processá-los de forma eficiente, usando múltiplos passos: é preciso processar um dado item apenas uma vez. Isso traz restrições para a implementação de algoritmos, que precisam ser projetados para trabalhar em apenas um passo de análise dos dados. Ainda, como mostra a definição de Aggarwal, a componente temporal é inerente ao fluxo, caracterizando sua evolução ao longo do tempo, o que é chamado de *localidade temporal*. Portanto, além de trabalhar em apenas um passo de análise, os algoritmos devem considerar a evolução dos dados. Por esta razão, é importante analisar a relação existente entre os dados e o tempo.

2.2 Tempo e Dados Coletados no Tempo

O fenômeno fundamental do tempo sempre foi do interesse da humanidade (AIG- NER et al., 2011). Muitas teorias diferentes para caracterização da dimensão física do tempo tem sido desenvolvidas e discutidas há centenas de anos envolvendo várias áreas das ciências. O que é comum a todas essas teorias é que o tempo é unidirecional e dá ordem aos eventos. A principal questão envolvida diz respeito a como a dimensão física do tempo e os dados associados podem ser modelados de forma a facilitar a visualização interativa, utilizando sistemas computacionais, independente de um modelo particular.

Quando um atributo temporal está envolvido na coleta de dados, as abordagens (genéricas) mais conhecidas podem não ser as mais adequadas para o estabelecimento de um mapeamento visual direto entre múltiplas variáveis e a coordenada do tempo.

Ainda, é preciso considerar os relacionamentos existentes entre aspectos específicos, tais como os diferentes níveis de granularidade do tempo (minutos, horas, semanas, meses, anos, etc). Com o objetivo de auxiliar na tomada de decisão, Aigner et al. (AIGNER et al., 2011) propuseram uma abordagem sistemática para lidar com dados orientados ao tempo, baseada em três critérios principais: o dado, o tempo e a representação.

O *dado* pode ser subdividido em:

- Quadro de referência: considera o contexto do dado, através de duas formas, o dado espacial e o abstrato. Para dados espaciais, informações de espaço inerentes podem ser exploradas, a fim de encontrar um mapeamento adequado para visualizações, que incorpore a representação do tempo. Em dados abstratos, como nenhum mapeamento inicial é fornecido, se faz necessária uma representação espacial adequada dos dados, o que exige criatividade e experiência por parte do usuário. As dimensões da visualização podem ser utilizadas para expor o domínio do tempo e analisar influências e comportamentos dos dados.
- Número de variáveis: o dado pode ser mono ou multivariado.
- Nível de abstração: real ou abstrato.

O critério *tempo* pode ser subdividido em dois outros critérios: primitivas temporais e estruturas de tempo.

- Primitivas temporais: a coordenada do tempo pode ser composta por pontos ou intervalos de tempo. Um ponto pode ser considerado como um instante de tempo; os intervalos visam estender a noção de primitiva temporal, ou seja, especificam dois instantes ou um ponto acrescido de uma duração.
- Estruturas de tempo: podem ser classificadas em linear, cíclica ou ramificada. A estrutura linear corresponde à percepção comum de tempo, tal como uma coleção de primitivas temporais, ou seja, o tempo transcorre do passado para o futuro. O tempo cíclico é composto por um conjunto finito de primitivas temporais recorrentes, como as estações do ano. Estruturas ramificadas são modeladas como grafos: as primitivas temporais são representadas pelos vértices do grafo e as arestas direcionadas correspondem à ordem temporal.

O terceiro critério, *representação* de dados orientados ao tempo, pode ser descrito por meio da análise de dependência temporal e da dimensionalidade do espaço.

- Dependência temporal: pode ser estática ou dinâmica. Representações estáticas têm dados orientados ao tempo por meio de imagens, ou seja, não se alteram ao longo do tempo. Representações dinâmicas, por sua vez, utilizam a dimensão de tempo físico para acompanhar as relações temporais entre dados (as representações se alteram ao longo do tempo assim como uma animação).
- Dimensionalidade do espaço: pode ser classificada em bidimensional (2D) ou tridimensional (3D). É preciso determinar a necessidade de explorações visuais 2D ou 3D. Muitos pesquisadores argumentam a terceira dimensão é importante para expressar visualizações ao longo do tempo.

Em quaisquer circunstâncias, conjuntos de dados estarão relacionados a dois domínios de tempo: externo e interno. O tempo *externo* está definido fora do modelo de dados.

Ele é necessário para descrever como o conjunto de dados evolui ao longo do tempo. O tempo *interno* pode ser mapeado na dimensão temporal do modelo de dados e determina quando informações contidas nos dados são válidas. O tempo interno atua em cenários cujos instantes de tempo estão embutidos nos dados, sendo possível empregar ferramentas de sistemas dinâmicos e séries temporais, a fim de mapear as relações entre as observações, permitindo a modelagem e a predição de futuras realizações.

2.3 O *Reddit*

O *Reddit* foi fundado por Steve Huffman e Alexis Ohanian em 2005 (DEMÉTRIO et al., 2023). A ideia surgiu quando eles eram estudantes da Universidade de Virgínia, nos Estados Unidos. Eles tiveram a inspiração para criar uma plataforma que permitisse às pessoas discutir e compartilhar *links* e conteúdos de maneira fácil e anônima na Internet.

O nome *Reddit* é uma combinação das palavras *read* (ler) e *edit* (editar), refletindo sua natureza: os usuários podem ler e editar (ou seja, votar e comentar sobre) os conteúdos compartilhados por outros. O *Reddit* foi lançado oficialmente em junho de 2005 e desde então, cresceu significativamente em popularidade, tornando-se uma das maiores redes sociais e fóruns de discussão online do mundo.

A plataforma é conhecida por suas *subreddits*, que são comunidades dedicadas a tópicos específicos, onde os usuários podem postar e discutir conteúdos relacionados a esses temas. Ao longo dos anos passou por várias mudanças, atualizações e aprimoramentos, mas a essência de ser um lugar para discussões e compartilhamento de informações entre os usuários permaneceu a mesma. O *Reddit* desempenhou um papel importante na cultura da Internet e tem sido um local onde as pessoas podem se reunir para compartilhar interesses, aprender, debater e interagir com outros usuários de todo o mundo.

2.4 O que é o Reddit

O *Reddit* é uma plataforma de mídia social e fórum de discussão online onde os usuários podem compartilhar *links*, postar conteúdo, fazer perguntas, participar de discussões e interagir com outros membros (JR; COELI, 2000). É conhecido por sua estrutura de fóruns, onde os tópicos de discussão são organizados em comunidades temáticas dedicadas a interesses específicos. Essas comunidades são subfóruns dentro da plataforma, cada um dedicado a um tópico específico.

Por exemplo, há fóruns para discutir notícias, ciência, tecnologia, *hobbies*, entretenimento e praticamente qualquer outro tópico imaginável. Cada um deles é moderado por voluntários, que ajudam a manter as discussões dentro das diretrizes da comunidade. Além do mais, os usuários podem votar positivamente (*upvote*) ou negativamente (*downvote*) nas postagens e comentários de outras pessoas. Essa votação determina a visibilidade do conteúdo. Postagens populares, com muitos *upvotes* são exibidos de forma proeminente na página inicial do site, enquanto aqueles com *downvotes* tendem a ser menos visíveis.

No aplicativo, os usuários ganham *karma* com base nas votações que recebem em suas postagens e comentários. O *karma* é uma medida da reputação de um usuário na plataforma. Os usuários também podem deixar comentários nas postagens de outrosusuários e participar de discussões. Os comentários também são votados pelos usuários, afetando sua visibilidade. Além disso, é possível criar postagens que podem ser *links* para conteúdo externo, como artigos, imagens, vídeos ou postagens de texto diretas.

O *Reddit* abriga uma grande variedade de conteúdo, desde discussões sérias sobre notícias e eventos atuais até "memes"engraçados, conselhos práticos, perguntas e respostas e muito mais. Ou seja, é uma plataforma popular para compartilhamento de informações, aprendizado, entretenimento e discussões, envolvendo uma ampla gama de tópicos. Sua estrutura descentralizada e foco na comunidade o tornam único no mundo das redes sociais online.

2.5 O que é um *Bot* de Rede Social

Um *bot* de redes sociais é um programa de computador (ou software) projetado para simular ações e interações humanas em plataformas de mídia social. Esses programas são desenvolvidos para realizar tarefas automatizadas, como publicar conteúdo, interagir com outros usuários, seguir ou deixar de seguir contas e responder à mensagens.

Existem dois tipos principais de *bots* de redes sociais: os legítimos e os maliciosos. Os legítimos são criados por desenvolvedores e empresas para auxiliar na gestão de contas e automatizar tarefas específicas, geralmente com finalidades comerciais ou de *marketing*. Por outro lado, os *bots* maliciosos são projetados para espalhar *spam*[1,] disseminar informações falsas, gerar seguidores falsos ou realizar atividades fraudulentas, como golpes e *phishing*[2.] Segundo o *site Cloudflare* (cloudflare, 2023):

> "De um modo geral, os *bots* de redes sociais são programas automatizados usados para se envolver em redes sociais. Esses *bots* se comportam de maneira parcial ou totalmente autônoma e costumam ser projetados para imitar usuários humanos. Embora existam *bots* de redes sociais benevolentes, muitos são usados de maneiras desonestas e nefastas. Algumas estimativas sugerem que esses *bots* maliciosos constituem uma porcentagem considerável de todas as contas nas redes sociais."

Em resumo, os *bots* de redes sociais são programas de computador projetados para simular ações e interações humanas nas plataformas de mídia social. Eles podem ser usados tanto para automatizar tarefas, aumentar a visibilidade e o engajamento ou espalhar *spam* e desinformação. É importante que as plataformas de mídia social adotem medidas para combater os *bots* maliciosos e que os usuários façam uso ético e transparente dessas ferramentas automatizadas. É por essa razão que no desenvolvimento do robô foram considerados os recursos e acessos fornecidos pela API do *Reddit*, de forma a deixar todo o processo seguro e confiável.[1]

---

1 Envio de mensagens para uma grande quantidade de pessoas; [2]Consiste em tentativas de fraude para obter ilegalmente informações como número da identidade, senhas bancárias, número de cartão de crédito, entre outras, por meio de e-mail com conteúdo duvidoso; [3]Endpoints são URIs (Uniform Resource Identifiers) em uma API que um aplicativo pode acessar; [4]É um formato de dados usado para transmitir e armazenar dados de forma organizada e legível por máquinas. Ele é baseado na sintaxe de objetos JavaScript e é amplamente utilizado em APIs para enviar e receber dados entre diferentes sistemas; [5]Uma aplicação ou processo stateless são recursos isolados. Nenhuma referência ou informação sobretransações antigas são armazenadas, e cada uma delas é feita do zero; [6]Conter links ou informações que informam o cliente sobre quais ações podem ser executadas a partir de um determinado ponto; [7]É a execução em massa de várias threads em um mesmo processo. Threads, por sua vez, é uma unidade originada de maneira independente na execução de um processo de software ou aplicativo; [8]É um protocolo de segurança que cria um link criptografado entre um servidor Web e um navegador Web, [9]É uma função de dispersão criptográfica (ou função hash criptográfica); [10]É uma função de dispersão criptográfica de 128 bits unidirecional; [11]O *Spring Data JPA*, parte da família *Spring Data*, facilita a implementação de repositórios baseados em JPA (*Java Persistence API*); [12]*Hibernate* é o *framework* para persistência de dados mais utilizado em projetos Java.

2.6 O que é uma API

As APIs (*Applications Programming Interface*) desempenham um papel fundamental no desenvolvimento de aplicativos modernos, permitindo a integração e a interoperabilidade entre diferentes sistemas e serviços. Elas possibilitam a criação de ecossistemas de aplicativos mais amplos, estimulando a inovação e o compartilhamento de recursos. As APIs são amplamente utilizadas em áreas como serviços web, aplicativos móveis, Internet das Coisas e integração de sistemas empresariais (aws, 2023):

> "APIs são mecanismos que permitem que dois componentes de software se comuniquem usando um conjunto de definições e protocolos. No contexto de APIs, a palavra Aplicação refere-se a qualquer software com uma função distinta. A Interface pode ser pensada como um contrato de serviço entre duas aplicações, que define como as duas se comunicam usando solicitações e respostas. A documentação de suas respectivas APIs contém informações sobre como os desenvolvedores devem estruturar essas solicitações e respostas."

A documentação de uma API contém informações detalhadas sobre como os desenvolvedores devem estruturar as solicitações e respostas para interagir com a aplicação. Ela descreve os *endpoints*[3] disponíveis, os parâmetros necessários, o formato dos dados esperados e como interpretar as respostas. Essa documentação serve como um guia para os desenvolvedores utilizarem corretamente a API e aproveitarem suas funcionalidades.

Em resumo, as APIs são interfaces de programação que permitem a comunicação entre diferentes componentes de software. Elas estabelecem um contrato de serviço entre as aplicações, definindo como elas devem se comunicar por meio de solicitações e respostas. As APIs são essenciais para a integração e interoperabilidade de sistemas, facilitando a troca de informações e o compartilhamento de recursos. Com o uso adequado das APIs, os desenvolvedores podem criar aplicativos mais poderosos, conectar diferentes plataformas e impulsionar a inovação tecnológica. As APIs desempenham um papel fundamental na construção de um ecossistema de software interconectado e dinâmico.

2.6.1 Como as APIs funcionam

A arquitetura da API geralmente é explicada em termos de cliente e servidor. A aplicação que envia a solicitação é chamada de cliente e a aplicação que envia a resposta é chamada de servidor(aws, 2023). Existem quatro maneiras diferentes pelas quais as APIs podem funcionar, dependendo de quando e por que elas foram criadas.

1. **APIs SOAP**: Essas APIs usam o *Simple Object Access Protocol* (Protocolo de Acesso a Objetos Simples). Cliente e servidor trocam mensagens usando XML. Esta é uma API menos flexível que era mais popular no passado.

2. **APIs RPC**: Essas APIs são conhecidas como *Remote Procedure Calls* (Chamadas de Procedimento Remoto). O cliente conclui uma função (ou um procedimento) no servidor e o servidor envia a saída de volta ao cliente.

3. **APIs *WebSocket***: A API de *WebSocket* é um desenvolvimento de API da web moderno que usa objetos JSON[4] para transmitir dados. Oferece suporte à comunicação bidirecional entre aplicativos cliente e servidor. O servidor pode enviar mensagens de retorno de chamada a clientes conectados, tornando-o mais eficiente que a API REST.

4. **APIs REST**: Essas são as APIs mais populares e flexíveis encontradas na web atualmente. O cliente envia solicitações ao servidor como dados. O servidor usa essa entrada do cliente para iniciar funções internas e retorna os dados de saída ao cliente. Por se tratar do método mais comumente utilizado pela API do *Reddit*, veremos as APIs REST em mais detalhes a seguir..

2.6.2 APIs REST

Inicialmente concebida como uma diretriz para o gerenciamento e comunicação em uma rede complexa, como a Internet, a arquitetura REST (*Representational State Transfer*) estabelece condições para o funcionamento de uma API. De acordo com Salvadori (SALVADORI et al., 2015), esses princípios englobam o comportamento 'stateless'[5] do servidor e a adoção de uma interface uniforme, com ênfase na utilização de controles hiper-mídia[6.] Web Services REST, também conhecidos como Web APIs, são frequentemente empregados na implementação de sistemas distribuídos que seguem esses princípios arquiteturais.

Essas Web APIs utilizam o protocolo HTTP como meio de comunicação e obedecem à semântica desse protocolo, incluindo o uso apropriado dos verbos HTTP (GET, POST, PUT e DELETE) e dos códigos de status. Salvadori ressalta a importância de utilizar a Web como infraestrutura de desenvolvimento ao implementar essas APIs, o que implica o uso de tecnologias padronizadas e protocolos adequados.

As implementações REST tratam as informações no formato de recursos, os quais encapsulam os dados manipulados pelas Web APIs em um conjunto coeso e significativo de informações. A integração de dados é uma das razões para o uso de Web APIs, pois possibilita que diferentes aplicações compartilhem informações com baixo acoplamento, o que é mais flexível do que compartilhar bancos de dados.

É importante destacar também que, no contexto das Web APIs, as informações são transmitidas em formatos apropriados para serem consumidos por aplicações, como XML e JSON. Além disso, ele ressalta a importância de descrever semanticamente os dados, transformando a Web de recursos isolados na Web de Dados.

2.7 Banco de Dados

De acordo com William(ALVES, 2014), um banco de dados pode ser descrito e simplificado como um "Universo de Discurso"que representa uma parte do mundo real. Qualquer alteração ocorrida nesse universo deve ser refletida no banco de dados. Além disso, um banco de dados deve possuir um conjunto lógico e ordenado de dados, em vez de ser uma coleção aleatória sem um propósito específico. Por fim, o banco de dados deve ser construído e preenchido com dados que tenham um objetivo definido, e deve ser acessado por usuários e aplicativos desenvolvidos para manipular esses dados. Aprofundando um pouco mais no conceito de banco de dados, podemos definilo como uma coleção organizada e estruturada de informações ou dados que são armazenados eletronicamente em um sistema de computador. Esses dados podem ser acessados, gerenciados, atualizados e consultados de maneira eficiente.

Os bancos de dados são projetados para armazenar informações relacionadas em tabelas ou estruturas de dados semelhantes, com o objetivo de facilitar a recuperação de informações específicas quando necessário. Eles desempenham um papel fundamental em organizações, empresas e sistemas de informação, permitindo o armazenamento e a recuperação de dados de forma rápida e precisa.

Os bancos de dados desempenham um papel crítico em uma ampla gama de aplicações, desde sistemas de gerenciamento de empresas até redes sociais e sistemas de análise de dados. Eles são essenciais para o armazenamento e recuperação eficiente de informações em um mundo cada vez mais orientado por dados.

Os principais componentes de um banco de dados incluem:

- Dados: São as informações armazenadas no banco de dados, como números, textos, datas e muito mais.
- Sistema de Gerenciamento de Banco de Dados (SGBD): É um software responsável por gerenciar e controlar o acesso aos dados no banco de dados. Ele fornece uma interface para interagir com o banco de dados, executar consultas e atualizar informações.
- Tabelas: São estruturas organizadas de dados que armazenam informações específicas. Cada tabela é composta por colunas (campos) que representam os tipos de informações a serem armazenados e linhas (registros) que contêm os dados reais.
- Consultas: São comandos ou instruções usados para recuperar informações específicas de um banco de dados. As consultas permitem buscar, filtrar e combinar dados de várias tabelas.
- Relações: Em bancos de dados relacionais, as tabelas podem ter relacionamentos entre si. Isso permite que os dados sejam vinculados e recuperados de maneira eficiente, mantendo a integridade dos dados.
- Índices: São estruturas que aceleram a recuperação de dados em grandes bancos de dados. Eles atuam como um guia para localizar informações rapidamente.
- Integridade dos Dados: Refere-se à precisão e consistência dos dados no banco de dados. Os bancos de dados geralmente têm regras e restrições para garantir a integridade dos dados.

A seguir, discutiremos com mais detalhes o banco de dados escolhido para atender aos objetivos deste projeto.

2.7.1 PostgreSQL

Nas palavras de André(MILANI, 2008), o *PostgreSQL* é um servidor de SGBD de grande potencial e confiabilidade, contendo todas as características dos principais bancos de dados utilizados no mercado. Além disso, ele oferece suas licenças para uso de forma gratuita, seja para fins educacionais, seja para a realização de negócios.

O *PostgreSQL* possui as seguintes características notáveis:

- SGBD Relacional com Suporte a ACID (Transações): O *PostgreSQL* é um sistema de gerenciamento de banco de dados relacional completo, que adere aos princípios ACID (Atomicidade, Consistência, Isolamento e Durabilidade), garantindo alta qualidade e confiabilidade nos serviços de banco de dados.

- Replicação: Oferece recursos de replicação entre servidores. A vantagem é que sua licença permite o uso gratuito, mesmo para aplicações comerciais, diferentemente de outras licenças de *software* livre.

- *Cluster* (Alta Disponibilidade): É possível configurar o *PostgreSQL* para atuar como um *cluster* de informações, expandindo sua capacidade para mais de um servidor. Isso envolve interconectar e sincronizar vários computadores para atender às demandas dos usuários, aumentando a capacidade de utilização.

- *Multithreading*: O *PostgreSQL* gerencia várias conexões simultaneamente, permitindo que várias pessoas acessem a mesma informação sem atrasos ou filas de acesso, graças ao recurso de *multithreading*[7] dos sistemas operacionais.

- Segurança SSL e Criptografia: Suporta nativamente SSL[8], criando conexões seguras para informações de login e dados sigilosos. Além disso, oferece extensibilidade para algoritmos de criptografia, como SHA1[9] e MD5[10].

- *SQL*: O *PostgreSQL* segue os padrões estabelecidos pelo *ANSI SQL* em suas funcionalidades. Implementa melhorias com base na versão *ANSI SQL* 2003 e continua desenvolvendo novos recursos. Muitas de suas funcionalidades estão em conformidade com as versões 92 e 99 do *ANSI SQL*.

- Incorporável em Aplicações Gratuitamente: Devido à licença BSD, o *PostgreSQL* pode ser incorporado gratuitamente em aplicações pessoais e comerciais, sem custos para desenvolvedores ou fornecedores de software.

- Capacidade de Armazenamento Eficiente: O *PostgreSQL* é capaz de suportar grandes volumes de informações em suas tabelas de forma eficiente e confiável, incluindotabelas maiores do que a capacidade de tamanho de arquivo fornecida por alguns sistemas operacionais.

O *PostgreSQL* foi escolhido para este projeto devido à sua flexibilidade, desempenho robusto e licença de uso gratuita. Ele segue padrões SQL rigorosos, oferece replicação gratuita e possui recursos de segurança integrados. Sua capacidade de armazenamento eficiente e comunidade ativa de desenvolvedores também foram fatores decisivos.

2.8 Spring Boot

O *Spring Boot* é um *framework* amplamente utilizado para o desenvolvimento de aplicações *Java*, especialmente em arquiteturas baseadas em microsserviços e APIs RESTful. Sua popularidade se deve a várias características que facilitam o desenvolvimento e a manutenção de aplicações, tornando-o uma escolha preferencial para muitos desenvolve- dores.

Um dos principais motivos que levaram Alice Fernandes Silva (SILVA et al., 2019) a optar pelo *framework* em sua pesquisa sobre geração automática de código é a facilidade de configuração e a rapidez na criação de serviços. Ele adota uma filosofia de "convenção sobre configuração", o que significa que ele oferece configurações padrão que permitem que os desenvolvedores comecem rapidamente a construir suas aplicações, sem a necessidade de uma configuração extensa e complexa. Essa característica não apenas acelera o processo de desenvolvimento, mas também reduz a carga de trabalho relacionada à configuração, permitindo que os desenvolvedores se concentrem na lógica de negócios da aplicação.

Além disso, o suporte robusto a testes, integração com diversas bibliotecas e todo seu ecossistema contribuem para a escolha do *Spring Boot* em projetos que requerem agilidade e eficiência. Para esse projeto do robô coletor de dados, essas características são especialmente vantajosas, pois permitiram uma implementação rápida e eficaz de funcionalidades, possibilitando o gerenciamento de tempo de forma eficaz, até a geração do produto final.

O *framework* segue padrões arquiteturais bem definidos, o que favorece uma organização clara do código em camadas: *Model*, *Controller*, *Service* e *Repository*. Essa estrutura facilita a manutenção e escalabilidade do projeto, permitindo que diferentes equipes trabalhem em paralelo em diferentes componentes sem causar conflitos. Além disso, a integração com o *Spring Data JPA*[11] e *Hibernate*[12] simplifica as operações de acesso a dados, permitindo que os desenvolvedores se concentrem na lógica de negócios em vez de se preocuparem com a implementação dos detalhes de interação com o banco de dados.

Outro ponto positivo é a forma como o *Spring Boot* facilita o trabalho com requisições HTTP. A criação de controladores para gerenciar as rotas de forma clara e concisa, aliada a uma abordagem *RESTful*, permite que desenvolvedores construam APIs de forma rápida e intuitiva. Isso é especialmente relevante para esse projeto, que requer a manipulação de dados por meio de múltiplas requisições de forma eficiente. O suporte à validação de dados, serialização e tratamento de exceções também contribui para a criação de uma experiência de usuário mais robusta e confiável.

Assim, a combinação de rapidez, facilidade de configuração, uma arquitetura bem definida e um ecossistema rico torna o *Spring Boot* uma opção atraente para desenvolvedores e equipes que buscam eficiência no desenvolvimento de aplicações Java modernas.

Capítulo 3

Desenvolvimento

3.1 Primeiro contato com o *Reddit*

Como primeiro passo para a realização deste projeto e para testar a conexão com a API do *Reddit*, foi realizada uma pesquisa na Internet em busca de tutoriais e documentação oficial sobre como estabelecer uma conexão com a mesma. O resultado mais comum envolve o uso da linguagem *Python*, a qual foi definida para estabelecer o primeiro contato com a API. Além disso, foi necessária a criação de uma aplicação no *Reddit* para obter as chaves de acesso (*tokens*), que são elementos fundamentais para a realização da conexão, independentemente da linguagem escolhida.

Nesse primeiro momento, a escolha da linguagem *Python* se deu não apenas pela sua popularidade, mas também pela disponibilidade de bibliotecas e módulos que facilitam a integração com APIs de forma geral. A linguagem oferece suporte sólido para realizar solicitações HTTP, o que é essencial para interagir com o *Reddit* de forma eficiente.

Após acessar a página oficial do *Reddit*[1] e obter os *tokens* necessários, com as orientações da documentação do PRAW (*Python Reddit API Wrapper*)[2], foi instalado o módulo, por meio do comando: `pip install praw`

O PRAW é uma biblioteca bem estabelecida, que simplifica muito a interação com a API do *Reddit*, permitindo que desenvolvedores se concentrem na lógica específica de seus aplicativos. Com o PRAW devidamente instalado e as chaves de acesso em mãos, foi possível criar *scripts* em *Python* para testar a conexão com a API do *Reddit*.

Esses *scripts* permitem a exploração dos diversos recursos da API, como a realização de consultas e obtenção de dados, possibilitando uma melhor compreensão da forma como serão incorporados no projeto de desenvolvimento do robô coletor de dados. A Figura 3.1 ilustra um dos *scripts* gerados na fase inicial, cujo propósito principal é obter informações sobre as postagens mais recentes em uma comunidade específica (*subreddits*) do *Reddit*, analisar essas informações e armazená-las para finsde pesquisas futuras. O *script* inicia importando várias bibliotecas *Python* essenciais para executar as operações necessárias (linhas 1 a 7):

2

[1] Disponível em: <https://www.reddit.com/prefs/apps>.
[2] Disponível em: <https://praw.readthedocs.io/en/stable/getting_started/installation.html>.

Figura 3.1: *Script* em *Python* que ilustra o primeiro contato com a API.

```python
from pprint import pprint
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import praw

user_agent = "scraper 1.1 /u/Hiroshi0619"
reddit = praw.Reddit (
    client_id = "8IrYJxOhwvwL6yBKQspX5g",
    client_secret = "oI-D2sP4vheeQ0W1CmB2Rs8x7bIZqg",
    user_agent = user_agent
)

# hot || new || rising || top

headlines = set()
for submission in reddit.subreddit('programacao').new(limit=1):
    print("_____")
    print("\nTitulo: ", submission.title)
    print("\nTexto: ", submission.selftext)
    '''
    print(submission.over_18)
    '''
    print("\nId da postagem: ", submission.id)
    print("\nTotal de comentários do submit:", submission.num_comments)
    #.author retorna uma classe representando o usuario da postagem
    print("\nNome de usúario: ", submission.author.name)
    #convertendo o unixtime para um formato legivel
    post_data = datetime.datetime.utcfromtimestamp(submission.created_utc)
    formato = "%d-%m-%Y %H:%M:%S"
    print("\nData da criação: ", post_data.strftime(formato))
    print("\nNúmero de upvotes: ", submission.score)
    print("\nPorcetagem de upvotes: ", (submission.upvote_ratio * 100), "%")
    print("\nURL da postagem ", submission.url)
    #break
    headlines.add(submission.title)
print("\n**********************************************")
print("Total de submits buscados: ",len(headlines))
```

Fonte: Do autor (2024).

- *pprint (pretty-print)*: usada para imprimir informações de forma mais organizada no console.
- *pandas* e *numpy*: amplamente utilizadas para análise de dados e manipulação de estruturas de dados.
- *matplotlib* e *seaborn*: utilizadas para criar gráficos e visualizar dados.
- *datetime*: para manipulação de datas e horas.
- *praw (Python Reddit API Wrapper)*: permite interagir com a API do *Reddit* programaticamente.

Antes de realizar qualquer operação no *Reddit*, o *script* configura a API com as informações necessárias, como *client_id*, *client_secret* e um *user_agent* (linhas 9 a 13).

Esses detalhes são essenciais para autenticar a aplicação junto ao *Reddit* e obter acesso aos dados.

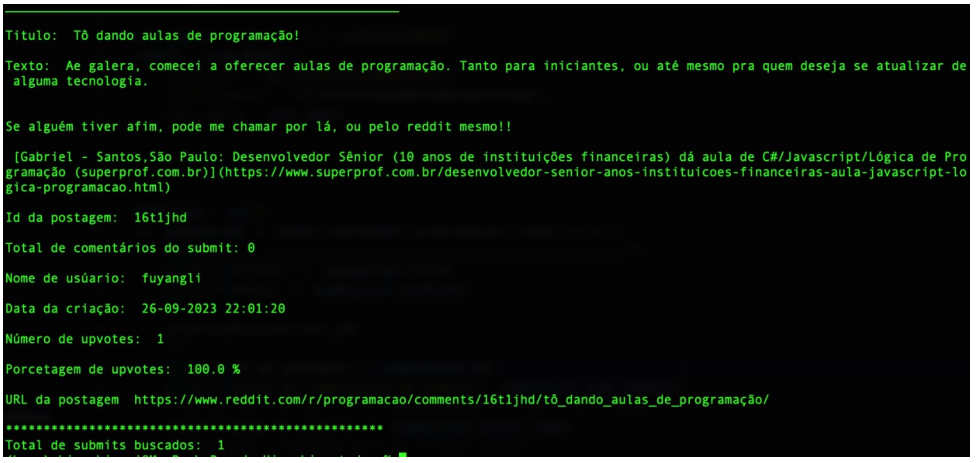
A próxima parte do *script* concentra-se na coleta de dados do *Reddit* (linhas 18 a 26). Aqui, é feito um acesso ao *subreddit* específico, chamado 'programacao', para posterior busca das postagens mais recentes:

- O laço *for* percorre as postagens mais recentes no *subreddit*.
- Cada postagem é analisada e informações relevantes são extraídas, tais como: título, texto, identificador da postagem, número de comentários, nome de usuário do autor, data de criação, número de *upvotes*, porcentagem de *upvotes* e URL da postagem.

Embora não esteja visível no *script* mostrado na Figura 3.1, as informações extraídas podem ser armazenadas em uma estrutura de dados, como um *DataFrame*[3] da biblioteca *pandas*, para análise e processamento posterior. Isso permite que os dados sejam manipulados de várias maneiras, como geração de estatísticas, criação de visualizações ou até mesmo exportação para um banco de dados.

Embora o *script* seja simples, ele proporcionou um primeiro contato bem sucedido com a API do *Reddit*. Isso ajudou a avançar no desenvolvimento das próximas etapas do projeto final, além de servir como base para futuras conexões com a API em outras linguagens. A Figura 3.2, mostra o resultado da execução do *script*.

Figura 3.2: Resultado da execução do *script* usado para teste.



Fonte: Do autor (2024).

3.2 Preparando o projeto

Após o primeiro contato de sucesso com a API do *Reddit*, utilizando a linguagem *Python*, teve início a preparação do projeto com a linguagem definida, o *Java*. Antes de iniciar o desenvolvimento do projeto, vale a pena mencionar algumas limitações impostas ao desenvolvedor sobre o uso da API.[3]

3.2.1 Entendendo algumas limitações

Um fator importante para construir um robô que fará múltiplas requisições HTTP com o *Reddit* é entender as limitações impostas pelo próprio *Reddit* sobre sua API para desenvolvedores. Entre elas, duas devem, em especial, ser seguidas à risca para evitar problemas com o *token* obtido anteriormente ou até o banimento da conta. As duas principais são:

- Não utilizar o *bot* para responder a postagens e comentários em *subreddits* dos quais o desenvolvedor não for administrador. Isso ocorre porque os *endpoints* relacionados à postagem e suas interações foram feitos para que o desenvolvedor modere seu próprio *subreddit*. Caso fosse possível interagir com *bots* em *subreddits* de terceiros, isso aumentaria o risco de segurança, permitindo que desenvolvedores mal-intencionados insultem, persigam ou humilhem usuários, ou ainda disseminassem notícias falsas.
- Intervalo entre requisições. Essa segunda limitação foi o principal impedimento para o desenvolvimento de um robô coletor em tempo real. Devido ao grande avanço de modelos de inteligência artificial e às polêmicas envolvendo o uso indevido de dados públicos por empresas como a *OpenAI*, *Meta* e *Google* para treinar grandes modelos de *LLM* [4], redes sociais como o *Reddit* e o *X* tornaram mais rígidas as regras de acesso contínuo a dados via API. Embora essa limitação não seja divulgada oficialmente pelo *Reddit*, há inúmeros relatos de desenvolvedores relatando o banimento temporário de seus *tokens* após longos períodos de requisições contínuas.

Outra limitação encontrada durante o desenvolvimento foi a restrição temporal que o *Reddit* impõe às leituras de dados, permitindo a leitura de, no máximo, as últimas mil postagens de um *subreddit*, com algumas ressalvas. Mais à frente, em outra seção, será feita uma explicação de como essas limitações foram contornadas para desenvolver o robô coletor.

3*DataFrame* é uma estrutura de dados bidimensional, com colunas de tipos potencialmente diferentes (como uma planilha ou tabela SQL). Geralmente é o objeto *pandas* mais comumente usado.

3.2.2 Criando o projeto

Agora que foram esclarecidas as limitações impostas pelo próprio *Reddit*, serão detalhados os passos para o desenvolvimento do robô na linguagem proposta, o *Java*.

Como o projeto faria uso intensivo de requisições HTTP para a API, o desenvolvimento do robô foi iniciado com o *framework Spring Boot*. Esse *framework* é amplamente conhecido e famoso por permitir a construção de aplicações robustas que fazem uso intensivo de requisições HTTP, seja para consumo, integração ou criação de APIs próprias. Foi utilizado o *Spring Initializr* [5], fazendo uso da versão 3.3.3 do *Spring Boot*, com o gerenciador *Maven* e *JDK*, na versão 17. Com o ambiente de desenvolvimento preparado e com os *tokens* da API em mãos, foi iniciada a codificação do robô.

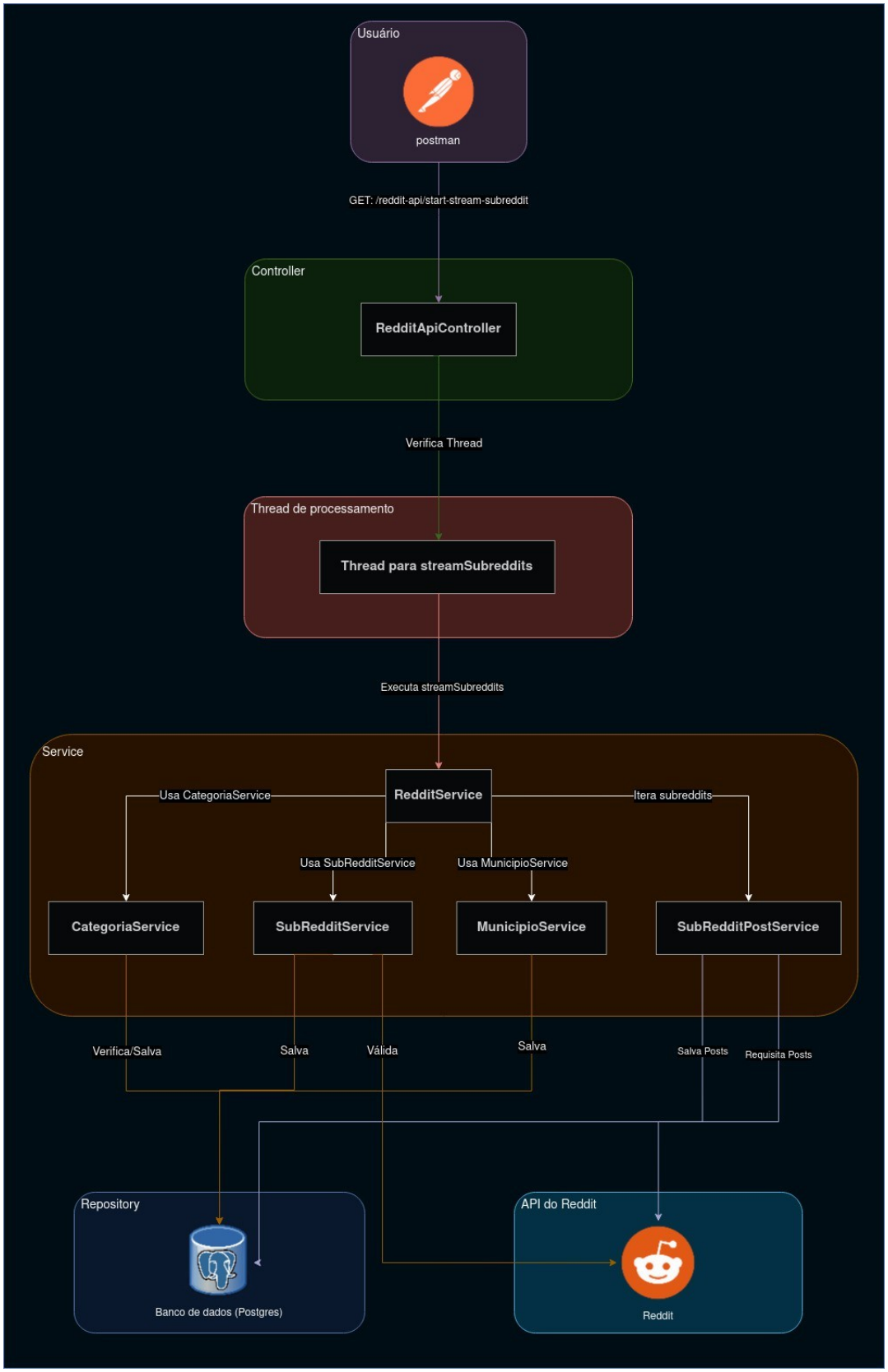
3.2.3 Arquitetura de desenvolvimento

Com a utilização do *framework Spring Boot*, a aplicação foi estruturada em camadas que seguem os padrões de *Controller*, *Service*, *Repository* e *Model*. Cada camada desempenha funções específicas, como lidar com a lógica da API do *Reddit*, gerenciar categorias e salvar postagens no banco de dados, entre outras funcionalidades.

O projeto, por lidar com grandes volumes de dados oriundos da API do *Reddit* em formato *JSON*, exigiu a criação de classes DTO (*Data Transfer Object*) para desserializar os dados e integrá-los às camadas mencionadas. Além disso, para aumentar a segurança do projeto, dados sensíveis, como credenciais de conexão com o banco de dados, *client ID*, *secret ID* e informações de login da conta do *Reddit*, foram armazenados no arquivo application.yml, um recurso padrão do *Spring Boot* que possibilita o armazenamento seguro e a exclusão desses dados do controle de versão por meio do .gitignore.

A Figura 3.3 apresenta o diagrama de arquitetura do sistema, que ilustra o fluxo de dados e a interação entre os diferentes componentes da aplicação. No diagrama, é possível observar a interação inicial do usuário, que utiliza o *Postman* para enviar requisições ao *Controller*, que por sua vez inicia uma *Thread* de processamento. Essa *Thread* delega as operações de verificação, processamento e armazenamento de dados ao *Service*, que interage com a camada *Repository* para persistir os dados no banco de dados. O diagrama também destaca a comunicação com a API do *Reddit* para realizar as leituras de dados e a utilização de um arquivo externo para o registro dos municípios brasileiros.

Figura 3.3: Diagrama de arquitetura do sistema coletor de dados.



Fonte: Do autor (2024).

3.3 Gerenciando a credencial

Após finalizar todas as etapas de criação das classes para cada camada da arquitetura definida e configurar o arquivo application.yml, foi realizada a codificação de algumas classes. A primeira classe desenvolvida foi a Credentials.java. Essa classe é injetada como dependência nas demais classes que foram implementadas e que farão uso de requisições HTTP para a API, pois ela é responsável pela recuperação e gerenciamento do *access token* da aplicação.

3.3.1 O *access token*

Para que se possa autenticar a aplicação com a API do *Reddit*, é preciso obter o *access token*. Para isso, é necessário realizar uma requisição HTTP para o *endpoint*[6], enviando alguns dados no cabeçalho e no corpo da requisição. No cabeçalho foi incluído o *client ID* e o *secret ID*, juntamente com o *user-agent* da aplicação que, no caso do projeto, foi definido como "TCC Universitário". No corpo, são enviados os dados de *login* do *Reddit*, como o *username* e a senha.

Após realizar a requisição, a API do *Reddit* retorna um *JSON* contendo o *access token*, além de outras informações relevantes, como o tempo de expiração desse *token*, indicando quando será necessário solicitar um novo. Esse *token* é fundamental, pois todos os outros *endpoints* do *Reddit* exigem que ele seja incluído no cabeçalho das requisições para realizar a autenticação, utilizando o protocolo *OAuth 2.0* [7].

3.3.2 Codificando a classe de credencial

Com a importância da classe Credentials e sua responsabilidade no gerenciamento do *access token* esclarecidas, é possível iniciar a sua codificação. Primeiro, como essa classe é injetada como dependência nas demais classes que trabalharão com requisições para a API do Reddit, é preciso definir alguns atributos para ela. Os atributos privados incluem: client_id, secret_id, username, e password, todos do tipo String. Além disso, há um atributo estático do tipo AccessTokenDTO (a classe DTO responsável por armazenar o *JSON* recebido da API), chamado accessToken.

O próximo passo, pensando nos métodos da classe, se faz necessário um método privado que realize a requisição HTTP ao *endpoint* da API do *Reddit* e armazene o objeto DTO no atributo accessToken. Outro método privado importante deve verificar se o *access token* ainda é válido ou se

já expirou, retornando o *token* válido, caso não tenha expirado. Dessa forma, evita-se a realização de requisições HTTP desnecessárias para cada tentativa de interação com outros *endpoints* da API, prevenindo o banimento temporário por múltiplas requisições em um curto intervalo de tempo.

Por fim, é preciso definir um método público que retornará o *token* ainda válido ou, se expirado, um novo *access token* renovado através de uma nova requisição HTTP.[4]

3.4 Otimizando os *endpoints* da API

O próximo passo foi criar uma classe que utiliza a injeção de dependência da classe Credentials e também de serviços para fornecer, de maneira organizada, os *endpoints* corretos para cada requisição à API do *Reddit*. Durante o desenvolvimento, foi identificado que ela funcionaria melhor como uma classe abstrata, sendo herdada por um controlador que gerenciaria as requisições de forma inteligente.

3.4.1 A classe BaseReddit

A classe BaseReddit possui três atributos principais. O primeiro é o atributo privado e final do tipo String, chamado baseUrl. Ele é responsável por armazenar a URL base comum entre todos os *endpoints* do *Reddit*, para que seja posteriormente concatenado com o restante do *endpoint*, conforme a necessidade. Esse atributo é `final` porque não mudará durante todo o ciclo de vida da classe.

O segundo atributo é o userAgent, também privado e final, responsável por informar o *User Agent* da aplicação no cabeçalho das requisições, um parâmetro fundamental para que as requisições sejam aceitas pela API. O último atributo é o accessToken, um atributo privado e estático que armazenará o *token* de acesso, fornecido via injeção de dependência da classe Credentials criada anteriormente.

O construtor da classe BaseReddit é simples: ele recebe a classe Credentials para recuperar o accessToken e também define os outros dois atributos (baseUrl e userAgent). Outro elemento fundamental da classe é o enumerador RedditEndpoint, que é declarado internamente e contém os diferentes caminhos dos *endpoints* da API. Esses caminhos são concatenados com o atributo privado baseUrl. O enumerador inclui *endpoints* que não são essenciais para o desenvolvimento do robô, mas foram inseridos com o propósito de testes e experimentações.

Por fim, a classe tem a função de retornar os *endpoints* completos, já com os parâmetros

---

4[6]<https://www.reddit.com/api/v1/access_token>

[7]O OAuth 2.0 é um protocolo de autorização que permite que aplicativos acessem recursos em nome de um usuário sem expor suas credenciais, como senha. É amplamente utilizado para autenticação em APIs e permite fluxos seguros de autorização.

inseridos corretamente, utilizando o enumerador quando necessário.

3.5 Modelando as entidades

Com a base para as requisições à API do *Reddit* criada, o próximo passo foi modelar o banco de dados para, então, começar a desenvolver as entidades JPA, que serão mapeadas para o banco de dados usando o *Hibernate*[8]. A modelagem das entidades foi fortemente influenciada pela maneira como a API retorna os dados em objetos JSON.

A primeira entidade criada foi a tabela subreddit_subreddit, onde são armazenadas as informações dos *subreddits* na aplicação. Ela também possui três propriedades importantes para o desenvolvimento: o valor de after, before e um atributo lógico chamado acabou_after. A importância e aplicabilidade dessas propriedades será descrita mais adiante, na abordagem do *fetch* de postagens de cada *subreddit*.

Outra entidade criada foi a tabela municipios, que se relaciona com a entidadeposts, armazenando a localidade de cada postagem com latitude, longitude, geocódigo e o nome do município. Para popular a tabela municipios, foi utilizado um *script* SQL contendo os dados de todos os 5.570 municípios registrados pelo IBGE. Esse *script* pode ser obtido através de um repositório no GitHub[9], necessitando apenas de pequenas modificações para atender ao uso no projeto.

Também foi criada a tabela categoria, responsável por armazenar todas as categorias nas quais uma postagem pode ser classificada, usando um algoritmo que será discutido nas próximas seções. Para o projeto, foram definidas categorias relacionadas à área de Tecnologia da Informação. A tabela tem duas colunas: descrição, um enumerador entre Linguagem de Programação, IDE, Termo de Tecnologia, Sistema Operacional, Banco de Dados e Metodologia Ágil; e nome, que representa o nome da tecnologia ou termo que o algoritmo tenta atribuir à postagem.

5[8]Hibernate é uma ferramenta ORM (Object-Relational Mapping) que facilita a interação entre objetos Java e bancos de dados relacionais.

[9]Repositório GitHub com os dados dos municípios brasileiros: <https://github.com/kelvins/ municipios-brasileiros>

Tabela 3.1: Categorias utilizadas no projeto.

Categoria	Nome
Linguagem de Programação	C#
Linguagem de Programação	Python
Linguagem de Programação	Rust
Linguagem de Programação	TypeScript
Linguagem de Programação	Lisp
Linguagem de Programação	Prolog
Linguagem de Programação	Java
Linguagem de Programação	C++
Linguagem de Programação	PHP
Linguagem de Programação	JavaScript
Linguagem de Programação	Swift
Linguagem de Programação	Kotlin
Linguagem de Programação	Perl
Linguagem de Programação	Scala
Linguagem de Programação	Dart
Linguagem de Programação	Ruby
Framework	CakePHP
Framework	React
Framework	Vue
Framework	Unity
Framework	Spring
Framework	Next.js
Framework	Node.js
Framework	Laravel
Framework	.NET
Framework	Django
Framework	Flask
Framework	Angular
Framework	Svelte
Framework	Ember

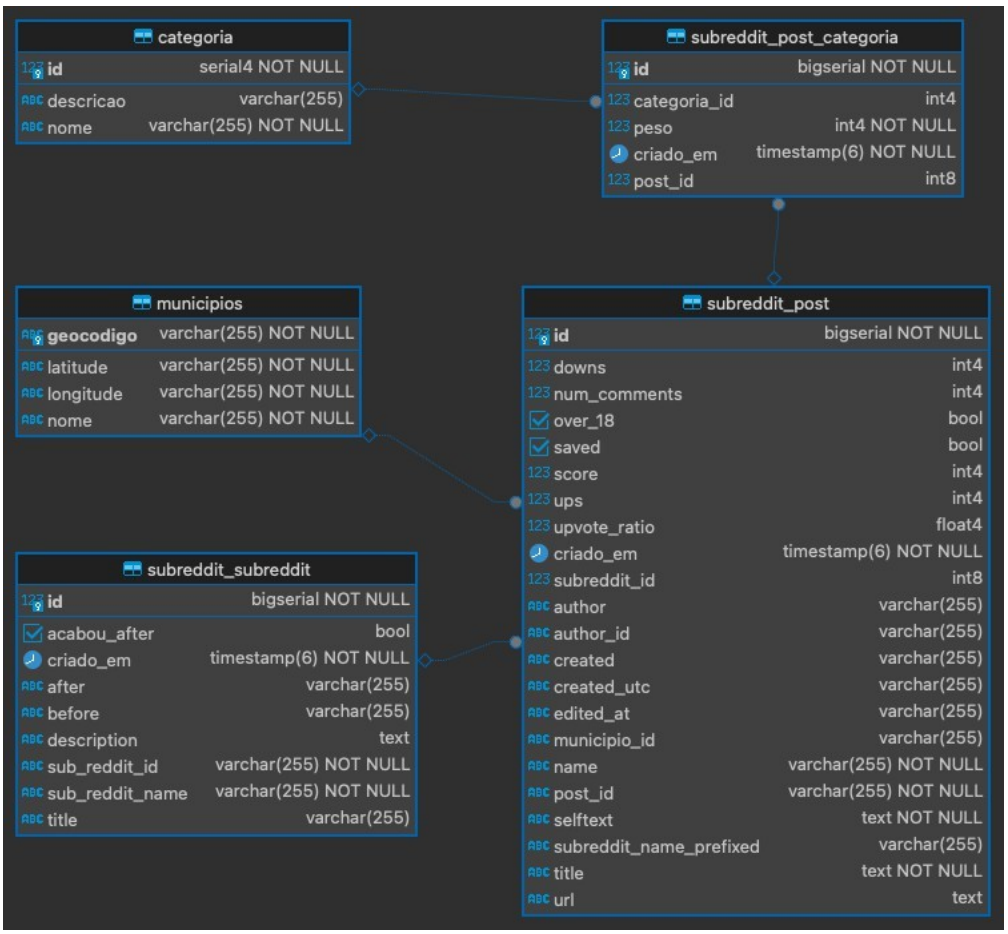
Framework	Symfony
Framework	Express
Framework	Flutter
IDE	VSCode
IDE	IntelliJ
IDE	PHPStorm
IDE	PyCharm
IDE	Eclipse
IDE	NetBeans
IDE	RubyMine
IDE	Android Studio
IDE	Xcode
Termo de Tecnologia	AJAX
Termo de Tecnologia	GameDev
Termo de Tecnologia	Desktop
Termo de Tecnologia	Clean Code
Termo de Tecnologia	SOLID
Termo de Tecnologia	API
Termo de Tecnologia	Microservices
Termo de Tecnologia	DevOps
Termo de Tecnologia	Cloud
Termo de Tecnologia	Docker
Termo de Tecnologia	Kubernetes
Termo de Tecnologia	Blockchain
Termo de Tecnologia	Cybersecurity
Sistema Operacional	Windows
Sistema Operacional	macOS
Sistema Operacional	Linux
Sistema Operacional	Ubuntu
Sistema Operacional	Fedora
Sistema Operacional	Debian
Sistema Operacional	Manjaro
Banco de Dados	MySQL
Banco de Dados	PostgreSQL
Banco de Dados	MongoDB
Banco de Dados	Oracle

Banco de Dados	SQLite
Banco de Dados	Redis
Banco de Dados	Cassandra

Fonte: Do Autor (2024).

Por fim, foi definida a entidade para as postagens que seriam lidas durante o carregamento, chamada subreddit_posts. Os nomes das colunas seguem os mesmos nomes das chaves presentes no objeto JSON retornado pela API do *Reddit*. Ainda assim, foram selecionadas apenas as informações mais relevantes para armazenamento no banco de dados. Entre elas, destacam-se: title e selftext, que são importantes para o algoritmo analisar e classificá-las em alguma categoria; created_utc, que é fundamental para a criação de um fluxo temporal. Além disso, foi definido um relacionamento de 1:N entre esta entidade e a entidade de categorias, visto que uma mesma postagem pode se enquadrar em mais de uma categoria definida. A Figura 3.4 ilustra o Diagrama de Entidades-Relacionamentos com todas as entidades.

Figura 3.4: Diagrama de Entidades-Relacionamentos (DER).



Fonte: Do autor (2024)

3.6 Iniciando a leitura das postagens

Com as entidades mapeadas para o banco de dados, iniciou-se o desenvolvimento do *controller* e *service*, responsáveis por realizar a leitura contínua das postagens, utilizando requisições HTTP por meio da classe BaseReddit, criada anteriormente.

3.6.1 Controlador

O *controller* foi criado com o objetivo de gerenciar o fluxo de dados entre o serviço de leitura das postagens e a API. Para isso, alguns atributos importantes foram definidos:

- **Atributo Thread**: Foi utilizada a classe Thread do *Java* para possibilitar o processamento paralelo, garantindo que a aplicação possa realizar a leitura das postagens de forma assíncrona, sem bloquear a execução de outras tarefas.
- **Atributo streamActive**: Foi criado um atributo privado do tipo volatile boolean, chamado streamActive. Ele tem como função controlar o estado do robô de coleta de dados, ou seja, indica se o processo de leitura está ativo ou inativo.
- **Injeção de dependência do service**: Outro atributo importante do controlador é o service, injetado via *Spring*. Ele permite que o controlador acione os métodos necessários do serviço responsável pela lógica de coleta, classificação e armazenamento das postagens.

Método streamSubreddit

O primeiro método implementado no *controller* foi o streamSubreddit, responsável por iniciar o processo de leitura das postagens. Esse método pode ser chamado por meio de uma requisição HTTP, recebendo parâmetros no corpo da requisição, que são utilizados para configurar a coleta de dados. Entre os principais parâmetros, destacam-se:

- **intervalo**: Define o intervalo de tempo (em segundos) entre as requisições feitas à API do *Reddit*. Esse valor é importante para evitar restrições ou banimentos, devido ao número excessivo de requisições. Foi estabelecido um valor seguro de 10 segundos por requisição.
- **limite**: Define o número máximo de postagens a serem recuperadas por requisição. O valor padrão é 50, garantindo que até 50 postagens sejam lidas em cada chamada à API.
- **sort**: Define a ordem em que as postagens serão recuperadas. Aqui, optou-se por usar o valor 'new', para garantir que as postagens sejam retornadas na ordem em que foram criadas.[6]

6[10]<https://www.reddit.com/r/learnpython/>

- **peso**: Parâmetro que define o peso mínimo que uma postagem deve alcançar para ser classificada em uma categoria pelo algoritmo de categorização.
- **subreddits**: Um *array* contendo os nomes dos *subreddits* a serem monitorados pelo robô de leitura. A escolha desses seguiu dois critérios:
  - *Subreddits* focados em debates, dúvidas e discussões sobre tecnologia da informação e programação.
  - *Subreddits* que tratam esses temas de maneira ampla, evitando os que são dedicados a tecnologias específicas, como o *learnpython*[10], dedicado exclusivamente à linguagem *Python*. Isso visa garantir que os resultados obtidos futuramente não favoreçam uma tecnologia específica.

O método streamSubreddit verifica se o processo de leitura já está ativo. Caso contrário, a Thread é iniciada, e o serviço é chamado para realizar a lógica de leitura, classificação e salvamento dos dados no banco de dados.

Método stopStreamSubreddit

Também foi implementado o método stopStreamSubreddit, responsável por parar o processo de leitura. Ele verifica se a Thread está ativa e em caso positivo, interrompe a execução, desativando o fluxo de dados.

3.6.2 Services

Agora, falaremos sobre a classe *service* RedditService. Ela é responsável por controlar a leitura de dados e mostrar o progresso no terminal. Como essa classe contém muita lógica, algumas dependências são injetadas nela. Entre essas dependências, estão:

- Um atributo privado do tipo SubRedditService, que gerencia a lógica relacionada aos *subreddits*;
- Um atributo privado do tipo SubRedditPostService, que cuida da lógica das postagens de um *subreddit*;

- Um atributo privado do tipo CategoriaService, responsável pela lógica de classificação das postagens;
- Um atributo privado do tipo MunicipioService, que atribui a localização de cada postagem lida.

O método público streamSubreddits é chamado pelo controlador ao iniciar a *thread*. Ele recebe os parâmetros recuperados do corpo da requisição HTTP que invoca o controlador, iniciando a leitura dos dados. Primeiramente, um objeto HashMap<>() é criado para armazenar os resultados das leituras. Em seguida, o método verifica se todos os *subreddits* passados como parâmetros existem e se já estão salvos no banco de dados. Após essa verificação, ele recupera uma lista de todos os que serão usados para busca em entidades JPA, permitindo o uso de métodos úteis que o JPA fornece.

Duas verificações adicionais são realizadas: a primeira, se existem categorias pré- definidas no banco de dados para atribuir a cada postagem; caso contrário, o método de CategoriaService é chamado para salvar todas as categorias. A segunda verificação assegura que existem municípios salvos no banco de dados. Se não existirem, o método MunicipiosService é invocado, realizando um *script* pré-definido, salvando todos os municípios com suas respectivas latitudes e longitudes no banco de dados.

Após essas verificações, cada *subreddit* da lista chama o método streamSubredditPosts. Ao final da leitura de dados, o total de postagens lidos daquele é salvo no HashMap. Quando o laço termina, o resultado das leituras de dados é escrito no terminal.

O método privado streamSubredditPosts controla a leitura de dados de um *subreddit* específico passado como parâmetro dentro do laço do método streamSubreddits.

Utiliza-se uma variável lógica para verificar se ainda existem mais postagens a serem extraídas, além de um contador inteiro que registra quantas postagens foram lidas. Após a criação dessas duas variáveis, um outro laço chama o método fetchAndSavePosts, armazenando o retorno do mesmo para contabilizar o total de postagens lidas e determinar se ainda existem algumas a serem lidas. Ao final do laço, uma mensagem no terminal acompanha o progresso da leitura.

No método privado fetchAndSavePosts, é realizada a busca das postagens, verificando alguns parâmetros para a paginação da API do *Reddit*. Aqui, é válido detalhar as colunas after, before e acabou_after criadas na entidade subreddit_subreddit, pois são essenciais para a paginação de postagens:

- O valor de after refere-se a um ID de postagem, que é um identificador da própria API do *Reddit*. Esse valor pode ser recuperado no objeto JSON de cada postagem retornado pela API. Quando passado na requisição da API, ele busca todos as postagens criadas antes da que contém o ID correspondente. Embora possa parecer confuso, o valor de after busca informações do passado em relação à postagem de ID igual ao after.

- O valor de before funciona de maneira semelhante, sendo também um ID de uma postagem do *Reddit*. Quando passado na requisição da API, retorna todos as postagens criadas após o ID de before. Novamente, a nomenclatura escolhida pelos programadores da API pode ser confusa, pois a busca ocorre no futuro, em relação ao ID da postagem com o valor de before.

- A variável lógica acabou_after indica que foi atingido o limite de leituras que podem ser feitas no passado daquele *subreddit*.

Seguindo com os detalhes do método fetchAndSavePosts, o objeto HashMap<>() é o retorno do método e serve para o controle do laço dentro do método streamSubredditPosts. Verifica-se se ainda é possível realizar mais carregamentos de postagens no passado com uma condicional. Se sim, o método fetchPosts da classe SubRedditPostService é invocado, realizando a requisição com a API. Mais detalhes serão dados na sequência.

Após a condicional, recuperam-se todos as postagens retornadas pelo fetchPosts para chamar o método savePosts da classe SubRedditPostService, que as salvará no banco de dados (esse método será discutido mais a frente). Então, recuperam-se os resultados do salvamento das postagens no banco de dados, para fazer duas verificações, descritas a seguir.

A primeira, verifica se ainda existem postagens a serem lidas no passado (caso a requisição tenha sido feita com o valor de after); se ainda existirem, o objeto HashMap recebe apenas a contagem de postagens lidas, juntamente com uma variável lógica de controle; se não houver mais postagens a serem lidas, além de configurar os valores do objeto HashMap, acabou_after se torna verdadeiro, para que na próxima chamada do método sejam realizadas requisições com o valor de before, que buscará postagens criadas após a postagem de ID salvo no valor de before.

A segunda verificação atribui o novo valor de after ou before, dependendo do fluxo que foi seguido. Assim, a próxima requisição buscará postagens mais próximas do presente ou irá mais além no passado, com postagens mais antigas.

Após essas verificações, os novos valores de after, before e acabou_after são atualizados no banco de dados e são escritas no terminal algumas informações importantes para verificação do progresso, como o requests remaining (restante de requisições) que ainda é possível fazer. Esse valor é controlado pelo próprio *Reddit* e pode ser verificado no cabeçalho da resposta da requisição. Se esse valor chegar a 0, é preciso esperar até o próximo intervalo para realizar mais requisições. Além disso, a tela do terminal exibe algumas informações de uso de memória *heap* do sistema. Por fim, após imprimir as informações no terminal, o comando Thread.sleep(intervalo * 1000L) é executado para pausar a *thread* por um determinado tempo, evitando problemas e bloqueios na conta de desenvolvedor da plataforma, caso haja tentativas de múltiplas requisições em um curto espaço de tempo. Por fim, o objeto HashMap é retornado.

3.7 Salvando e classificando as postagens

Por último, como o núcleo central da lógica por trás do robô, os métodos fetchPosts e savePosts, citados anteriormente, serão detalhados. Ambos fazem parte de outra classe de serviço chamada SubRedditPostService, que é responsável, entre outras coisas, por realizar as requisições, classificar e salvar as postagens no banco de dados. Para sua construção, foram utilizados cinco atributos privados, todos injetados como dependências, seguindo a estrutura do *Spring Boot*:

- O primeiro é o repository, cuja função é realizar transações com o banco de dados relacionadas à entidade de postagens, além de oferecer outros métodos úteis.
- O segundo atributo é o serviço SubRedditService, que gerencia a lógica associada aos *subreddits*.
- O terceiro é o CategoriaService, responsável por atribuir categorias às postagens, conforme necessário.
- O quarto atributo é o SubredditPostCategoriaService, que chama os métodos para salvar as categorias relacionadas a cada postagem no banco de dados.
- Por fim, o quinto atributo é o MunicipioService, encarregado de associar o município correto a cada postagem.

A classe SubRedditPostService também estende a classe BaseReddit, citada anteriormente, para facilitar e otimizar as requisições com a API do *Reddit*.

3.7.1 O método fetchPosts

O método fetchPosts realiza o carregamento de postagens de acordo com os parâmetros recebidos, considerando o *subreddit*, os valores de after ou before, além da ordem e do limite de postagens que a requisição deve retornar. Com esses atributos passados como parâmetros, o primeiro passo do método é construir a URL com o *endpoint* correto e com os parâmetros configurados. Isso é facilmente feito utilizando o método getEndpointPathWithParam, do enumerador criado internamente na classe BaseReddit.

Em seguida, criase a URI da requisição, usando o UriComponentsBuilder oferecido pelo *Spring*, que permite adicionar cada parâmetro à URI e gerar a URL atualizada. O próximo passo é preparar o cabeçalho da requisição, o que também é feito com facilidade, com o acesso ao User-Agent correto da aplicação, além de um accessToken válido. Feito isso, é possível realizar a requisição, utilizando a classe RestTemplate do *Spring*, armazenando a resposta da requisição em uma classe DTO, a RedditListingDTO. Se a requisição retornar um código 200, o método devolve um objeto HashMap<>(), contendo o objeto DTO (a resposta), que inclui todas as postagens obtidas, além de informações relevantes para monitorar o progresso da requisição, como o x-ratelimit-remaining, que indica o número de requisições restantes dentro de um intervalo de tempo.

3.7.2 O método savePosts

O método savePosts é responsável por processar e armazenar as postagens obtidas pela requisição, utilizando os serviços de categoria, município e *subreddit*, além do repositório para salvar os dados no banco. Ele recebe como parâmetros o objeto RedditListingDTO, que contém a lista de postagens, e o pesoMinimo, que está diretamente relacionado ao algoritmo classificador de categorias, o qual será abordado na próxima seção. Esse parâmetro será utilizado para definir as categorias associadas a cada postagem, com base em seu título e conteúdo textual.

Inicialmente, o método itera sobre as postagens através da lista children. Para cada postagem, verifica-se se a entidade já foi armazenada previamente, utilizando o repository.findFirstByPostId, ignorando aquelas que já estão salvas. Caso a postagem não exista no banco, um novo objeto SubRedditPost é criado e preenchido com os dados necessários, como título, autor, número de comentários, pontuação e URL da postagem. Além disso, cada postagem é vinculada a um Municipio aleatório, obtido através do MunicipioService, e ao SubReddit correspondente, que é buscado pelo SubRedditService.

Após a persistência da postagem no banco de dados, o método utiliza o CategoriaService para determinar as categorias relevantes à mesma. O método definirCategorias faz uso do pesoMinimo para aplicar o algoritmo classificador e identificar as categorias, com base no título e no conteúdo da postagem. Para cada categoria identificada, é criada e armazenada uma entidade SubRedditPostCategoria, contendo o respectivo peso calculado. Esse relacionamento entre a postagem e suas categorias é salvo utilizando o SubRedditPostCategoriaService.

Por fim, o método retorna um HashMap contendo o lastPostId, que identifica a última postagem processada, e o número total de postagens salvas. O lastPostId será utilizado para continuar o processo de carregamento em requisições subsequentes, garantindo que não haja duplicidade no armazenamento das postagens.

3.7.3 Definindo as categorias

Por fim, o algoritmo utilizado para classificar cada postagem de acordo com as categorias pré-definidas no banco de dados será detalhado. Este processo é fundamental para associar as postagens extraídas da API do *Reddit* a temas relevantes ao conteúdo, facilitando a organização e a busca por informações dentro da aplicação.

O método definirCategorias recebe como parâmetros o título da postagem, o corpo do texto, e um pesoMinimo, que determina o valor mínimo necessário para que uma postagem seja atribuída a uma categoria específica. O algoritmo faz uso de um conjunto de categorias predefinidas, que é carregado no início do método, através da função categoriasPredefinidasRefresh. A lógica central do algoritmo baseia-se em atribuir pesos a cada categoria com base em sua presença no título ou no corpo da postagem. A seguir, a estrutura do algoritmo:

- Para cada categoria pré-definida, o método começa atribuindo um peso inicial de zero;
- Se a categoria for mencionada no título da postagem, são somados 9 pontos ao peso daquela categoria;
- Em seguida, o método verifica a quantidade de vezes que a categoria aparece no corpo do texto, somando 3 pontos para cada ocorrência;
- Se o peso total de uma categoria alcançar ou exceder o valor de pesoMinimo, a postagem é classificada naquela categoria.

Após a atribuição de pesos, o método busca as categorias correspondentes no banco de dados e cria um mapa contendo o id da categoria, seu nome e o peso atribuído àquela postagem. A lista resultante desse processo é então retornada, permitindo que o relacionamento entre a postagem e suas respectivas categorias seja salvo.

O método getOcorrenciaDeCategoria auxilia o algoritmo ao contar quantas vezes uma categoria aparece no corpo da postagem. Ele percorre o texto verificando as ocorrências da categoria e retorna o número de vezes que a mesma é encontrada.

A lógica do algoritmo é fundamental para garantir que cada postagem seja devidamente classificada com base em seu conteúdo textual, proporcionando uma categorização eficiente e precisa.

3.8 Executando o robô

Para executar o robô e começar a realizar o carregamento de postagens, invoca-se a classe RedditApiController, através de uma requisição HTTP. Durante o desenvolvimento, foi utilizado o *Postman*[11] para realizar as chamadas, testar e depurar o projeto. O *Postman* se mostrou uma ótima solução graças à sua gratuidade, velocidade e capacidade de salvar múltiplos estados de requisição com seus parâmetros, verbo HTTP, cabeçalhos, corpo, entre outros.

O robô é iniciado através do *endpoint* <localhost:8080/reddit-api/ start-stream-subreddit> e o processo de carregamento das postagens pode ser observado, como mostrado na Figura 3.5.[7]

7O *Postman* é uma ferramenta popular para testar APIs, permitindo que desenvolvedores realizem chamadas HTTP de forma fácil e organizada.

Figura 3.5: Progresso exibido no terminal durante a atividade do robô.

```
---inicio do stream para o subreddit  : devBR---

Último Post - Title: Rede Onion, Tor Browser, ORBOT, Created UTC: 02/09/2024 21:15:40

-----------------------------------------
API: requests remaing            -> 957.0
API: requests used               -> 43
API: requests to reset           -> 416
Nome do Subreddit                -> devBR
Heap Memory Usage: Used = 47MB, Max = 2048MB, Committed = 87MB
Total de posts lidos nessa seção -> 65
Último Post - Title: Dúvida de carreira , Created UTC: 16/09/2024 17:05:55

-----------------------------------------
API: requests remaing            -> 956.0
API: requests used               -> 44
API: requests to reset           -> 408
Nome do Subreddit                -> devBR
Heap Memory Usage: Used = 61MB, Max = 2048MB, Committed = 87MB
Total de posts lidos nessa seção -> 99
Último Post - Title: Me ajude no meu TCC , Created UTC: 22/09/2024 11:44:55

-----------------------------------------
API: requests remaing            -> 955.0
API: requests used               -> 45
API: requests to reset           -> 400
Nome do Subreddit                -> devBR
Heap Memory Usage: Used = 36MB, Max = 2048MB, Committed = 87MB
Total de posts lidos nessa seção -> 128

---Fim do stream para o subreddit     : devBR---
```

Fonte:Do autor (2024).

O terminal exibe uma mensagem indicando o *subreddit* do qual o robô irá iniciar o carregamento de postagens. Após isso, o sistema entra em um ciclo, realizando requisições e atualizando os valores de *after* ou *before* para garantir a consistência da paginação, possibilitando a próxima requisição de forma segura.

Cada requisição é separada por caracteres ’-’ para facilitar a visualização no terminal. Também é possível verificar que, a cada requisição, o terminal atualiza o número de requisições restantes até a próxima renovação, o número de requisições realizadas no período e o tempo em segundos até a renovação. Essas linhas podem ser identificadas pelos prefixos nas strings ’API: requests remaining’, ’API: requests used’ e ’API: requests to reset’, respectivamente.

Além disso, são exibidas outras informações, como dados sobre a última postagem lida naquele ciclo de requisição, o *status* da memória da aplicação e o total de postagens lidas daquele *subreddit* até o momento. Vale ressaltar que cada requisição é realizada com uma pausa de alguns segundos entre uma requisição e outra, justamente para evitar maiores penalidades por parte da API do *Reddit*, conforme já mencionado.

Na Figura 3.6, é possível verificar o resultado exibido no terminal ao encerrar a thread em que o robô está realizando os carregamentos. Observa-se uma mensagem descrevendo o encerramento dos carregamentos, bem como um resumo rápido dos *subreddits* processados e a quantidade de postagens salvas de cada um deles. Ao final, a soma total de todos as postagens salvas, somando todos os *subreddits*.

Figura 3.6: Resultado exibido no terminal ao encerrar o robô.

```
*****Terminou o FETCH*****


Total de posts lidos por subreddit:
CodingHelp => 265
programmer => 26
programmation => 52
developersIndia => 0
develop => 0
program => 0
learnprogramming => 0
compsci => 0
coding => 163
computing => 0
Learn_Coding => 0
dev => 0
programacao => 0
devBR => 128
developer => 0
programacion => 0
AskProgramming => 556
ProgramadoresBrasil => 0
programming => 0
brdev => 0

Total de posts lidos: 1190
```

Fonte: Do autor (2024).[8]

8[12]A biblioteca pqxx é a biblioteca C++ oficial para conectar e interagir com o banco de dados *Post- greSQL*. Ela oferece uma interface robusta para realizar consultas e acessar os resultados de forma segura e eficaz, facilitando a integração de aplicações com bancos de dados *PostgreSQL*.

3.9 Integração com o THSC

Após a coleta e armazenamento dos dados no banco de dados, o próximo passo envolveu preparar esses dados para integração com o *Time Hierarchy Stream Cube* (THSC). Para tanto, foi desenvolvido um *script* em *C++* que utiliza a biblioteca pqxx[12] para acessar o *PostgreSQL*, extrair os dados e organizá-los no formato exigido pelo THSC.

O formato necessário é uma coleção do tipo vector em *C++*, organizada com os campos: [categoria, categoria_id, 'latitude', 'longitude', 'created_at'].
Este formato foi selecionado para facilitar a análise e visualização dos dados multidimensionais no THSC, permitindo uma representação eficaz das dimensões geoespaciais e temporais.

3.9.1 Implementação em C++

A Figura 3.7 ilustra o código em *C++* essencial para a extração dos dados do banco de dados, preparando-os para a integração com o THSC. Este *script* utiliza a biblioteca pqxx, facilitando a interação com o *PostgreSQL* de forma eficiente e segura. O código executa uma consulta SQL complexa que agrega informações de várias tabelas para compilar dados que incluem localização geográfica, categorias das postagens, e carimbos de tempo de criação.

Detalhes da Consulta SQL

- **WITH _____posts AS:** Esta subconsulta forma a base da consulta principal, agregando dados dos posts atuais e do arquivo bruto.
- **SELECT Statements:** Seleciona detalhes cruciais como latitude, longitude, identificador e nome da categoria, e a data de criação formatada.
- **UNION:** Combina resultados de duas fontes principais de dados, assegurando uma cobertura completa de postagens.
- **WHERE spc.peso IS NOT NULL:** Filtra postagens para incluir apenas aquelas com categorias designadas.
- **ORDER BY categoria_id:** Ordena os resultados para facilitar o processamento subsequente.
- **LIMIT:** Limita os resultados da consulta, opcionalmente, para facilitar testes e desenvolvimento inicial.

Os objetos Posts gerados são adicionados a um vetor, que é crucial para a transferência dos dados ao THSC. Este processo garante que os dados sejam preparados e formatados adequadamente para análise e visualização, destacando a importância da integração de tecnologias variadas para análises complexas e multidimensionais.

Como descrito por Muller (MüLLER, 2020), a implementação de THSC considera a arquitetura cliente-servidor, com um *browser* (desenvolvido em *JavaScript*) no lado cliente, utilizado para gerar as visualizações, a partir dos dados armazenados na estrutura de cubo de fluxos, no lado do servidor (desenvolvido em *C++*). A integração com os dados gerados pelo robô foi feita com a inserção do código mostrado na Figura 3.7 como um método da classe DataSource, responsável pela definição da fonte de dados a ser utilizada.

Vale ressaltar que os testes realizados para a validação de THSC envolveram dados coletados em tempo real e dados sintéticos, ambos armazenados em arquivos com formato *.csv*. Com a implementação do robô e o armazenamento dos dados no banco de dados, o processo de carregamento para o cubo de fluxos se torna mais dinâmica, pois pode ser realizada de forma contínua, consideradas as restrições de acesso e segurança entre o salvamento dos dados, sua transferência para o vetor e o armazenamento na estrutura de dados.

Figura 3.7: Código que realiza a extração de dados do PostgreSQL para carregamento e análise no THSC.

RedditsPostsRepository.cpp

```cpp
std::vector<Posts> RedditsPostsRepository::fetchPosts(int total) {
    std::vector<Posts> posts;

    try {
        pqxx::work txn(dbConnection.getConnection());

        std::string query =
            "WITH __posts AS ("
            "    SELECT "
            "        m.latitude AS latitude, "
            "        m.longitude AS longitude, "
            "        c.id AS categoria_id, "
            "        c.nome AS categoria_nome, "
            "        TO_CHAR(TO_TIMESTAMP(sp.created_utc, 'DD/MM/YYYY HH24:MI:SS'), 'DD/MM/YYYY HH24:MI:SS') AS criado_em "
            "    FROM subreddit_post sp "
            "    LEFT JOIN municipios m ON sp.municipio_id = m.geocodigo "
            "    LEFT JOIN subreddit_post_categoria spc ON sp.id = spc.post_id "
            "    LEFT JOIN categoria c ON c.id = spc.categoria_id "
            "    WHERE spc.peso IS NOT NULL "
            "    UNION "
            "    SELECT "
            "        m.latitude AS latitude, "
            "        m.longitude AS longitude, "
            "        c.id AS categoria_id, "
            "        c.nome AS categoria_nome, "
            "        TO_CHAR(TO_TIMESTAMP(spab.created_utc, 'DD/MM/YYYY HH24:MI:SS'), 'DD/MM/YYYY HH24:MI:SS') AS criado_em "
            "    FROM subreddit_post_arquivo_bruto spab "
            "    LEFT JOIN municipios m ON spab.municipio_id = m.geocodigo "
            "    LEFT JOIN subreddit_post_categoria_arquivo_bruto spcab ON spab.id = spcab.post_id "
            "    LEFT JOIN categoria c ON c.id = spcab.categoria_id "
            "    WHERE spcab.peso IS NOT NULL "
            ") "
            "SELECT * FROM __posts "
            "ORDER BY categoria_id ";

        if (total > 0) {
            query += "LIMIT " + std::to_string(total) + ";";
        }

        pqxx::result res = txn.exec(query);

        for (auto row : res) {
            Posts post(
                row["latitude"].as<double>(),
                row["longitude"].as<double>(),
                row["categoria_id"].as<int>(),
                row["categoria_nome"].as<std::string>(),
                row["criado_em"].as<std::string>()
            );
            posts.push_back(post);
        }

        txn.commit();
    } catch (const std::exception &e) {
        std::cerr << "Erro ao buscar posts: " << e.what() << std::endl;
    }

    return posts;
}
```

Fonte: Do autor (2024).

Capítulo 4

Resultados Obtidos

Este capítulo inicia-se com a discussão dos desafios enfrentados durante o desenvolvimento do projeto, os quais tiveram um impacto significativo nos resultados finais. A compreensão desses desafios é crucial para avaliar adequadamente os dados obtidos e as soluções implementadas.

4.1 Desafios Encontrados

Durante o desenvolvimento do projeto, diversos desafios foram enfrentados, dificultando o cumprimento de alguns dos objetivos estabelecidos. Para contorná-los, foram adotadas certas alternativas, detalhadas a seguir.

4.1.1 Penalidades da API

Como mencionado anteriormente, medidas precisaram ser implementadas para evitar penalidades associadas à conta de desenvolvedor utilizada para acessar os *client id* e *secret id* da API do *Reddit*. Com relatos frequentes de penalidades para desenvolvedores que fazem múltiplas requisições em um curto espaço de tempo, surgiram banimentos de usuários que utilizavam a API para realizar postagens ou responder a comentários em *subreddits* dos quais não são moderadores. Em fevereiro de 2024, Ivan Mehta(MEHTA, 2024), em seu artigo publicado no *TechCrunch*, abordou como as redes sociais começaram a restringir e monetizar suas APIs, impactando significativamente a forma como desenvolvedores interagem com essas plataformas. Para evitar bloqueios e penalidades, foi necessário impor um intervalo de 5 segundos entre cada requisição à API do *Reddit*, o que inviabilizou a proposta inicial de desenvolver um sistema de coleta em "tempo real".

4.1.2 Desafios na Paginação de Postagens

A implementação da funcionalidade de paginação de postagens também apresentou desafios significativos. O número de postagens salvas no banco de dados nunca ultrapas- sava aproximadamente 800, apesar de as mensagens de *log* indicarem a leitura de mais postagens. Após investigações detalhadas, foi descoberto que a API impôs um limite de 1000 postagens que poderiam ser retornadas por *subreddit*. Este limite inclui postagens.

deletadas por usuários ou moderadores, o que explicava o número inusitadamente baixo de postagens

capturadas. O problema era agravado por uma peculiaridade na API que, ao tentar paginar além desse limite, retornava postagens recentes repetidamente, sem erros.

Para resolver estes problemas, foi necessária uma refatoração do código que incluiu duas alterações principais:

- **Chaves Primárias Autoincrementáveis:** Inicialmente, o sistema utilizava o *ID* das postagens obtidas da API como chave primária na base de dados. Isso levava a conflitos quando a API retornava postagens repetidas devido ao problema de paginação. A solução adotada foi modificar a entidade de postagem no banco de dados para usar uma chave primária inteira autoincrementável. Isso garantiu que cada postagem nova, independentemente de ter um *ID* já existente ou não, recebesse um identificador único no banco de dados.
- **Validação antes do Salvamento:** Implementou-se uma camada de validação antes de salvar qualquer postagem no banco de dados. Este processo verifica se a postagem já existe no banco de dados utilizando os campos de identificação únicos fornecidos pela API. Apenas postagens novas ou modificadas são efetivamente salvas, o que elimina duplicidades e mantém a integridade dos dados coletados.

Estas mudanças não apenas resolveram os problemas de duplicação de dados mas também otimizaram o desempenho da coleta de dados, permitindo que o sistema tratasse de maneira eficiente as limitações impostas pela API do *Reddit*.

4.2 Análise dos Resultados

Após a conclusão do projeto e superação dos desafios mencionados, foi possível analisar alguns elementos com base nos resultados das postagens armazenadas no banco de dados. As análises desses dados visam atingir conclusões relevantes.

4.2.1 Resultados do Robô

O robô desenvolvido operou durante um período de 66 dias, de 28 de julho de 2024 a 2 de outubro de 2024. Este não permaneceu ativo de forma contínua, sendo acionado periodicamente, a cada semana, para realizar a coleta de dados. Esta abordagem não prejudicou a captura de dados, graças ao recurso de paginação, que permitiu recuperar postagens realizadas no intervalo entre uma coleta e outra. Durante esse período, um total de 18.440 postagens foram analisadas e salvas no banco de dados. Embora o número seja relativamente baixo, atribui-se essa limitação às novas restrições impostas pelo *Reddit*. Ainda assim, mesmo com a quantidade reduzida de dados, foi possível extrair algumas conclusões significativas sobre as áreas de tecnologia da informação e programação.

Conforme detalhado anteriormente, os esforços concentraram-se na leitura e análise de *subreddits* específicos, ajustando as técnicas de coleta de dados às limitações encontradas, o que reforça a importância das soluções implementadas para contornar os desafios apresentados neste capítulo.

4.2.2 Linguagem de Programação

Na Tabela 4.1, pode-se observar o total de postagens que se enquadraram em cada linguagem de programação. *Python* e *Java* aparecem como as linguagens mais mencionadas, com *Python* liderando as discussões durante o período de coleta. Esse resultado reflete a popularidade de *Python*, amplamente utilizada em diversas áreas como ciência de dados, automação e desenvolvimento web. A alta quantidade de postagens relacionados a *Java* também é compreensível, dado seu uso contínuo em grandes sistemas e aplicações empresariais. Linguagens como *C++*, *JavaScript* e *PHP* aparecem logo em seguida, embora com uma presença bem menor, o que pode indicar que, durante o período de coleta, as conversas se concentraram mais em linguagens de uso geral como *Python*.

Tabela 4.1: Total de postagens por categoria - Linguagens de Programação.

Linguagem de Programação	Total de Postagens
Python	364
Java	316
C++	116
JavaScript	98
PHP	71
C#	66
Rust	38
Kotlin	22
Swift	14
Ruby	12
Scala	10
Dart	9
Perl	8
TypeScript	6
Lisp	1

Fonte: Do autor (2024)

4.2.3 Banco de Dados

Na Tabela 4.2, é possível ver o total de postagens sobre bancos de dados. *MySQL* se destaca como o banco de dados mais mencionado, o que faz sentido considerando sua ampla adoção em aplicações web. Apesar de serem populares, bancos como *MongoDB*, *Redis*, *Postgres* e *Oracle* aparecem com bem menos menções, o que sugere que, durante o período, as discussões envolvendo bancos de dados foram mais escassas em comparação com as linguagens de programação.

Tabela 4.2: Total de postagens por categoria - Bancos de Dados.

Banco de Dados	Total de Postagens
MySQL	19
MongoDB	9
Redis	8
Postgres	8
Oracle	4
SQLite	4

Fonte: Do autor (2024)

4.2.4 Frameworks

Na Tabela 4.3, verifica-se os resultados para *frameworks*. *React* e *Node*, ambos relacionados ao ecossistema JavaScript, são os mais comentados, o que revela uma alta popularidade dessas plataformas, especialmente para desenvolvimento web. Curiosamente, *frameworks* como *Spring* e *.NET*, mais voltados para *back-end*, também aparecem em posições de destaque, mas com menos discussões. A popularidade de *frameworks JavaScript* no *Reddit* pode ser vista como um reflexo da demanda por soluções modernas e rápidas no desenvolvimento web.

Tabela 4.3: Total de postagens por categoria - *Frameworks*.

Framework	Total de Postagens
React	115
Node	85
Spring	56
Next	55
.NET	50
Unity	44
Laravel	37
Flutter	36
Angular	26
Django	22
Ember	20
Vue	20
Express	19
Flask	13
Symfony	12

Fonte: Do autor (2024)

4.2.5 Ambiente de Desenvolvimento Integrado

Na Tabela 4.4, é possível observar o total de postagens referenciando ambientes de desenvolvimento integrado (IDEs). O *VSCode* lidera com folga, o que reflete sua popularidade por ser gratuito, versátil e amplamente adotado para diversas linguagens de programação. *Android Studio*, voltado ao desenvolvimento *mobile*, aparece como a segunda IDE mais mencionada, seguido por ferramentas mais tradicionais como *Eclipse* e *PyCharm*.

Tabela 4.4: Total de postagens por categorias - IDE.

IDE	Total de Postagens
VSCode	21
Android Studio	9
Xcode	6
Eclipse	5
PyCharm	4
IntelliJ	4
NetBeans	4

Fonte: Do autor (2024)

4.2.6 Termos de Tecnologia

A Tabela 4.5, analisa termos de tecnologia amplamente utilizados. API foi o termo mais mencionado, o que não surpreende, considerando o foco atual em integrações e desenvolvimento de serviços. Outros termos como *cloud*, *DevOps* e *Docker* mostram a importância crescente dessas áreas nas discussões técnicas.

Tabela 4.5: Total de postagens por categorias -Termos de Tecnologia.

Termo de Tecnologia	Total de Postagens
API	263
Cloud	67
Desktop	33
DevOps	27
Docker	20
Cybersecurity	13
Blockchain	13
SOLID	11
Microservices	6
GameDev	2
Kubernetes	2
Clean Code	1

Fonte: Do autor (2024)

4.2.7 Sistema Operacional

A Tabela 4.6, traz o total de ocorrências de postagens por sistema operacional. O *MacOS* surpreende ao liderar as discussões, superando o *Windows*, o que pode refletir a crescente adoção de sistemas *Apple* por desenvolvedores, pelo menos para aqueles que utilizam o *Reddit*.

Tabela 4.6: Total de postagens por categoria - Sistemas Operacionais.

Sistema Operacional	Total de Postagens
MacOS	194
Windows	79
Linux	61
Ubuntu	9
Fedora	1
Debian	1

Fonte: Do autor (2024)

Como pode ser observado nos resultados apresentados em todas as tabelas, com os valores obtidos nas postagens categorizadas, é possível identificar algumas tendências claras nas discussões dentro da comunidade do *Reddit* sobre tecnologia e programação. Vale ressaltar que a análise aqui apresentada se baseia apenas na percepção do desenvolvedor e na sua experiência com algumas das categorias e serve apenas como uma ilustração de uma das possibilidades para se considerar os dados coletados. De forma geral, os resultados apenas evidenciam as áreas de foco e interesse dos desenvolvedores no *Reddit*.

4.3 Resultados do THSC

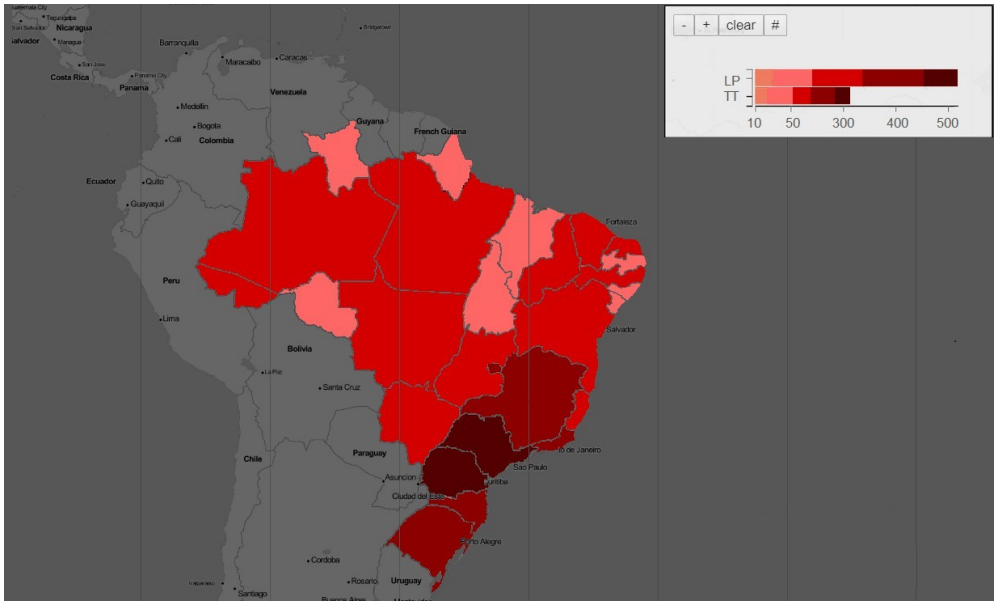
Com a integração do robô coletor de dados e a aplicação THSC, descrita no Capítulo 3, foi possível gerar algumas metáforas de visualização, no formato de mapa de cores.

É importante salientar que, em função das limitações na coleta de dados em tempo real imposta pelo *Reddit* e discutidas nos Capítulos 3 e 4.1, o volume alcançado, de 18.440 registros, é muito inferior ao que se esperava e à capacidade do cubo de fluxos utilizado em THSC. Nos testes realizados para validação da estrutura, foram utilizados mais de 50 milhões de registros, cada um contendo de cinco a seis atributos. Tal fato prejudica as metáforas de visualização e a hierarquia de tempo definidas originalmente na aplicação. Ainda assim, foi possível gerar algumas visões, que são apresentadas a seguir, apenas com os dados coletados em tempo real.

A Figura 4.1 apresenta um mapa de cores, considerando dados com as categorias "Linguagens de Programação"(LP) e "Termos de Tecnologia"(TT), conforme relatado nas Tabelas 4.1 e 4.5, respectivamente. O histograma no canto superior direito mostra os valores de total de postagens obtidos, por categoria, refletidos em cada um dos estados do Brasil.

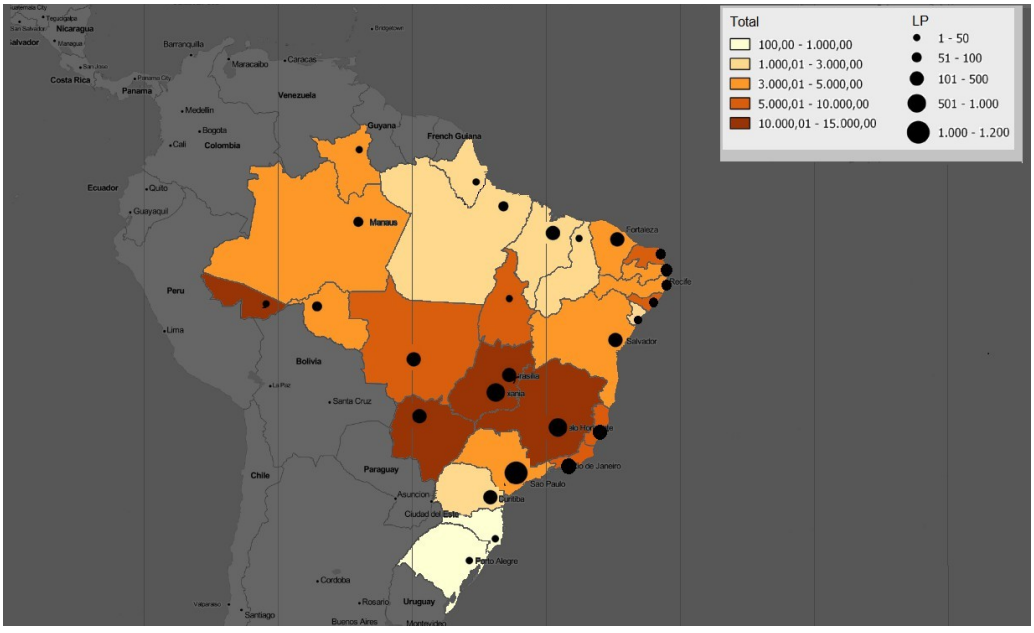
A Figura 4.2 apresenta uma visualização contendo um mapa de cores e plotagem de pontos (no formato de círculos). Para o mapa de cores foram considerados os valores envolvendo todas as categorias listadas nas tabelas da Seção 4.2.1, identificados pelo campo Total do histograma. Os círculos em preto representam apenas a categoria "Linguagens de Programação"(LP) (Tabela 4.1), onde os dados de cada estado aparecem concentrados nas localizações das respectivas capitais.

Figura 4.1: Mapa de cores das categorias "Linguagens de Programação"e "Termos de Tecnologia".



Fonte: Do autor (2024).

Figura 4.2: Mapa de cores e pontos representando valores totais e de categoria individual.



Fonte: Do autor (2024).

Capítulo 5

Conclusão Geral

Este Capítulo apresenta uma reflexão geral sobre o desenvolvimento do projeto, comparando os objetivos estabelecidos inicialmente com os resultados alcançados, discutindo os desafios encontrados, as soluções adotadas e os conhecimentos adquiridos durante todas as fases.

5.1 Avaliação dos Objetivos e Resultados

O projeto propôs o desenvolvimento de um sistema para coleta de dados em tempo real a partir da API do *Reddit*, com integração ao *Time Hierarchy Stream Cube* (THSC). Embora o objetivo de coleta em tempo real não tenha sido plenamente alcançado, devido às restrições impostas pelas mudanças na API e suas limitações, o projeto ainda conseguiu implementar um robô eficiente para a coleta de dados. As medidas adotadas para evitar penalidades e bloqueios foram essenciais para a continuidade da coleta de dados sem interrupções, conforme detalhado no Capítulo 4.1.

5.2 Desafios e Adaptações

Como descrito no Capítulo 4, particularmente na subseção de penalidades da API, foi preciso adaptar o projeto para contornar as limitações impostas. A introdução de um intervalo de 5 segundos entre cada requisição e a mudança do escopo de coleta para dados menos frequentes mas ainda relevantes, mostraram-se estratégias eficazes. Ainda assim, foi possível coletar uma quantidade substancial de dados, que proporcionaram uma análise baseada na experiência do desenvolvedor, como discutido no Capítulo 4.

5.3 Contribuições e Aplicações Práticas

O robô desenvolvido utilizou a API do *Reddit* de maneira inteligente, garantindo a coleta de dados sem infringir as normas da plataforma. Os dados coletados foram integrados com sucesso ao THSC, permitindo a realização de análises multidimensionais e a visualização de dados através de metáforas de visualização como mapas de calor.

Vale ressaltar que a aplicação THSC foi desenvolvida para lidar com grandes volumes de dados, o que não foi possível de se atingir com a coleta feita com o robô. Ainda assim, as implementações demonstram o potencial do projeto para aplicações em ambientes acadêmicos, fornecendo uma ferramenta útil para a análise de tendências e comportamentos em mídias sociais.

5.4 Reflexão Pessoal e Crescimento Profissional

Este projeto foi uma excelente oportunidade para aprofundar os conhecimentos em diversas tecnologias, incluindo *Spring Boot*, *Java* e a integração com APIs. O desafio de adaptar o projeto diante das mudanças na disponibilidade da API do *Reddit* e as restrições impostas contribuiu significativamente para o desenvolvimento pessoal do desenvolvedor e pesquisador. A capacidade de resolver problemas complexos e adaptar-se a mudanças rápidas no ambiente tecnológico foram habilidades essenciais que foram desenvolvidas e refinadas ao longo do desenvolvimento do projeto.

5.5 Código Fonte

O código fonte completo deste projeto pode ser acessado no *GitHub* pelo seguinte link: <https://github.com/MoriHiroshi0619/TCC_Reddit_Bot>. A documentação detalhada e os *scripts* utilizados estão disponíveis neste repositório, permitindo a revisão, utilização ou continuação do trabalho por outros interessados.

5.6 Conclusões Finais

Embora nem todos os objetivos iniciais tenham sido alcançados conforme planejado, o projeto conseguiu adaptar-se e superar vários desafios significativos. As soluções encontradas não apenas permitiram a continuidade da coleta de dados, mas também garantiram a integridade e a utilidade dos dados coletados. Os resultados obtidos e a experiência adquirida demonstram o sucesso do projeto em termos de contribuição para o campo de coleta de dados de mídias sociais e análise de grandes conjuntos de dados. As futuras direções deste trabalho podem incluir a exploração de outras fontes de dados e a expansão das capacidades de análise para incluir novas dimensões e métricas.

Este trabalho não apenas atendeu às exigências acadêmicas estabelecidas para o TCC, mas também proporcionou uma base sólida para futuras investigações e desenvolvimentos na área de análise de dados de redes sociais.

REFERÊNCIAS BIBLIOGRÁFICAS

AGGARWAL, C. C. (Ed.). *Data Streams: Models and Algorithms*. [S.l.]: Springer, 2007.

AIGNER, W. et al. *Visualization of Time-Oriented Data*. [S.l.]: Springer, 2011. ISBN 9780857290786.

ALVES, W. P. *Banco de dados*. [S.l.]: Saraiva Educação SA, 2014.

aws. *O que é uma API?* 2023. Disponível em: <https://aws.amazon.com/pt/what-is/ api/#:~:text=API%20significa%20Application%20Programming%20Interface,de% 20servi%C3%A7o%20entre%20duas%20aplica%C3%A7%C3%B5es.>. Acesso em: 3 de julho 2023.

cloudflare. *O que é um bot de redes sociais? | Definição de bot de redes so- ciais*. 2023. Disponível em: <https://www.cloudflare.com/pt-br/learning/bots/ what-is-a-social-media-bot/#:~:text=Os%20bots%20de%20redes%20sociais,para% 20fins%20maliciosos%20e%20manipuladores.>. Acesso em: 3 de julho 2023.

DEMÉTRIO, M. et al. Coleta de dados do reddit para análise das linguagens de programação mais populares do mercado. Florianópolis, SC., 2023.

HAN, J.; KAMBER, M.; PEI, J. *Data Mining: Concepts and Techniques*. 3rd. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. ISBN 0123814790, 9780123814791.

JR, K. R. d. C.; COELI, C. M. Reclink: aplicativo para o relacionamento de bases de dados, implementando o método probabilistic record linkage. *Cadernos de Saúde Pública*, SciELO Public Health, v. 16, p. 439–447, 2000.

MEHTA, I. *Social networks are getting stingy with their data, leaving third- party developers in the lurch*. 2024. <https://techcrunch.com/2024/02/09/ social-network-api-apps-twitter-reddit-threads-mastodon-bluesky/>. Acesso em: 2 out. 2024.

MILANI, A. *PostgreSQL-Guia do Programador*. [S.l.]: Novatec Editora, 2008.

MüLLER, R. M. *Cubo de Fluxos para Exploração Visual de Hierarquia de Dados Espaço-Temporais*. Tese (Doutorado) — Faculdade de Computação da Universidade Federal de Mato Grosso do Sul – Facom-UFMS, Campo Grande-MS, 2020.

SALVADORI, I. L. et al. Desenvolvimento de web apis restful semânticas baseadas em json. 2015.

SILVA, A. F. et al. Gerador de código para uma api rest com base no framework spring boot.

Universidade Federal de Campina Grande, 2019.