UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA
CAMPUS DE VITÓRIA DA CONQUISTA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RODRIGO SANTOS DO CARMO

NOTIFICA-DIÁRIO: WEB CRAWLER PARA NOTIFICAÇÕES DE OCORRÊNCIAS NO DIÁRIO OFICIAL DA BAHIA

VITÓRIA DA CONQUISTA
2022

RODRIGO SANTOS DO CARMO

NOTIFICA-DIÁRIO: WEB CRAWLER PARA NOTIFICAÇÕES DE OCORRÊNCIAS NO DIÁRIO OFICIAL DA BAHIA

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação da Universidade Estadual Do Sudoeste Da Bahia (UESB), como requisito para obtenção do título de Bacharel em Ciência Da Computação.

Orientador: Prof. Dr. Fábio Moura Pereira

VITÓRIA DA CONQUISTA

2022

RODRIGO SANTOS DO CARMO

NOTIFICA-DIÁRIO: WEB CRAWLER PARA NOTIFICAÇÕES DE OCORRÊNCIAS NO DIÁRIO OFICIAL DA BAHIA

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado Em Ciência Da Computação da Universidade Estadual Do Sudoeste Da Bahia (UESB), como requisito para obtenção do título de Bacharel em Ciência Da Computação.

Este trabalho foi defendido e aprovado pela banca em 20/03/2023.

BANCA EXAMINADORA

_____

Prof. Dr. Fábio Moura Pereira – UESB
Orientador

_____

Prof. Maísa Soares Dos Santos Lopes – UESB
Avaliador

_____

Prof. Francisco Dos Santos Carvalho – UESB
Avaliador

AGRADECIMENTOS

Agradeço a minha mãe e a toda minha família pela dedicação.
Aos meus amigos e colegas, meu muito obrigado.

Por muito tempo eu não entendi como algo tão caro pudesse ser tão inútil. Aí eu percebi que os computadores são máquinas estúpidas capazes de fazer coisas incrivelmente inteligentes; enquanto que os programadores são pessoas inteligentes capazes de fazer coisas incrivelmente estúpidas. Em resumo, nasceram um para o outro. - Bill Bryson

RESUMO

O trabalho realizado trata do desenvolvimento de um sistema para notificações automáticas do Diário Oficial da Bahia, fazendo uso de web crawlers e métodos especializados para extração de textos. Foi realizado utilizando uma metodologia prática de Design Science Research (DSR), além da utilização de conceitos relacionados a modelagem de sistemas. Os desafios práticos giram em torno de três pontos chaves, web crawlers (recuperação das páginas), extração de texto e busca de texto (comparação). O método de crawler focado utilizado, além dos métodos escolhidos para extração de texto baseado no tipo de conteúdo da página tiveram resultados satisfatórios, mas a técnica de comparação escolhida deixou a desejar. O sistema pode se acessado pelo link: https://yrodrigo2219.github.io/notifica-diario

Palavras-chave: Web Crawler; Diário Oficial da Bahia; Automação; Desenvolvimento.

ABSTRACT

The work carried out deals with the development of a system for automatic notifications of the Diário Oficial da Bahia, making use of web crawlers and specialized methods for extracting texts. It was carried out using a practical methodology of Design Science Research (DSR), in addition to the use of concepts related to systems modeling. The practical challenges revolve around three key points, web crawlers (page retrieval), text extraction and text search (comparison). The focused crawler method used, in addition to the methods chosen for text extraction based on the type of page content, had satisfactory results, but the chosen comparison technique left something to be desired. The system can be accessed through the link: https://yrodrigo2219.github.io/notifica-diario

Keywords: Web Crawler; Diário Oficial da Bahia; Automation; Development.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

UESB	Universidade Estadual Do Sudoeste Da Bahia
DOdB	Diário Oficial da Bahia
HTML	HyperText Markup Language
URL	Uniform Resource Locator
CRUD	Create, Read, Update and Delete
WWW	World Wide Web
IoT	Internet of Things
UML	Unified Modeling Language
SPA	Single Page Application
REST	Representational State Transfer
API	Application Programming Interface
VPS	Virtual Private Server

SUMÁRIO

1 INTRODUÇÃO

Dada a quantidade crescente de páginas disponíveis para acesso, a tarefa de buscar informações pertinentes em uma dessas páginas, apesar de parecer simples ao usuário final, acaba exigindo muito dos motores de busca. Então, é importante desenvolver, pelo menos os conceitos gerais, de como um motor de busca funciona para relacioná-lo com o trabalho aqui produzido.

Um motor de busca pode ser compreendido como um programa desenhado para procurar palavras-chaves fornecidas pelo utilizador em documentos e bases de dados. Porém, no contexto da internet, um motor de busca permite a pesquisa de documentos armazenados ao redor da *Web*, como aqueles que se encontram em *web sites*. Um ponto importante desses motores de busca da *Web* é que nem sempre existe um conhecimento prévio das páginas que podem ser pesquisadas, tornando a atividade de reconhecimento das páginas disponíveis uma parte fundamental do processo.

Um motor de busca funciona em três etapas: *Web crawling,* indexação e busca. A parte referente ao *Web crawling* é a parte de reconhecimento/extração dos dados, onde um *Web crawler,* que funciona como um *Web browser* automatizado que vai seguindo cada link que vê, extrai todas as informações da página para que seja realizada uma análise. Os dados extraídos são analisados e é então decidido o que/como a página será indexada. A última etapa é a busca realizada pelo usuário, que é executada quando palavras-chaves são digitadas, o sistema procura os índices e fornece uma lista das páginas que melhor combinaram aos critérios selecionados.

Esse trabalho faz uso de uma das técnicas utilizadas pelos motores de busca, os *Web crawlers*, mas ao invés de varrer por toda a *Web*, esse é um *crawler* focado e tem o seu limite de busca previamente definido para apenas o domínio do Diário Oficial da Bahia (DOdB). O *crawler* desenvolvido é executado diariamente, varrendo todas as informações da página do dia do DOdB e salvando-as para posterior análise. Essa análise leva em conta os dados registrados previamente pelos usuários interessados no sistema e compara se algum dos termos salvos foi mencionado, notificando o usuário interessado sobre o resultado positivo no dia.

1.1 METODOLOGIA

Para pesquisa e desenvolvimento do sistema de notificações para o Diário Oficial da Bahia foi utilizada a metodologia DSR (Design Science Research), que segundo (MANSON, 2006) consiste de cinco fases:

- **Conscientização**: Consiste em evidenciar a situação problemática, além de explicitar o ambiente externo e seus principais pontos de interação com o artefato. Sendo necessário explicitar as métricas e os critérios para a aceitação da solução do artefato.
- **Sugestão**: Consiste em explicitar as premissas e requisitos para a construção do artefato e registrar todas as tentativas de, tanto desenvolvimento do artefato quanto às razões que fundamentaram a exclusão da tentativa de artefato do desenvolvimento. Além de verificar possíveis implicações éticas da aplicação do artefato.
- **Desenvolvimento**: A tentativa de solução do problema gerada na fase anterior deve começar a ser desenvolvida, nessa fase poderá ser escolhida uma metodologia adequada para o desenvolvimento da tentativa, no caso desse projeto, uma metodologia para o desenvolvimento de um software. Porém, ainda nessa fase, também é importante justificar a escolha das ferramentas para o desenvolvimento do artefato e explicitar tanto os componentes do artefato, além das relações causais que geram o efeito desejado para que o artefato realize seus objetivos.
- **Avaliação**: Deve ser explicitado, em detalhes, os mecanismos de avaliação do artefato evidenciando os resultados do artefato em relação às métricas inicialmente projetadas. Sendo importante evidenciar o que funcionou como o previsto e os ajustes necessários no artefato.
- **Conclusão**: Serão sintetizadas as principais aprendizagens em todas as fases do projeto, justificando a contribuição do trabalho. Ou seja, uma apresentação dos resultados obtidos.

1.2 PROBLEMA DE PESQUISA

Quando se olha para os arquivos publicados no site do Diário Oficial da Bahia, é fácil notar, que existe uma grande quantidade de arquivos sendo postados diariamente, podendo chegar até mesmo mais de 800 arquivos distintos em certos dias.

Esses arquivos normalmente são divididos em diversas categorias, algumas delas sendo: Executivo, Licitações, Municípios, Diversos e Especial. Cada uma dessas categorias tendem a ter subcategorias que dividem os documentos, por exemplo, a categoria chamada Executivo, pode conter uma subcategoria contendo decretos numerados e decretos financeiros. Porém, apesar de existir uma certa divisão dos arquivos, a não presença de um filtro e nem uma barra de pesquisa dificulta a pesquisa de um arquivo específico.

Outro ponto a ser considerado é, uma vez que um documento aparece no Diário Oficial da Bahia, a não ser que seja postado um lembrete/cópia alguns dias depois, ele nunca mais será postado novamente. Logo, acaba criando uma pressão para a consulta diária do portal caso seja necessária a consulta de alguma informação.

1.3 OBJETIVO GERAL

Sabendo que um dos problemas do Diário Oficial da Bahia é ter que acessá-lo, foi pensado em desenvolver uma ferramenta de registro de interesses, onde a ferramenta, de maneira automática, realizará essa consulta diária ao DOdB pesquisando apenas pelos termos que são interessantes para o usuário.

Logo, a ideia é que o usuário tenha um sistema onde ele possa registrar seu contato (para ser avisado caso a ferramenta encontre algum documento no dia) e seus interesses (que serão usados para filtrar os resultados). E a partir dai, esse usuário não terá mais que passar a olhar o Diário Oficial da Bahia, apenas esperar a ferramenta achar o documento que lhe é relevante.

A partir desse objetivo de reduzir a necessidade do acesso ao Diário Oficial da Bahia, alguns critérios devem ser discutidos primeiro. No que se diz respeito a

aceitação da solução para esse problema, três métricas devem ser consideradas tanto no desenvolvimento do sistema, quanto posteriormente durante a análise do mesmo:

1. Pelo menos 95% dos documentos postados devem ser analisados
2. A análise desses documentos não pode deixar falhas em mais do que 10% das vezes
3. As falhas deixadas pela análise devem ser corrigidas em pelo menos 50% dos casos

Tendo essas métricas em mente, pode se imaginar que uma solução aceitável para esse problema deve contemplar, no pior dos casos, 90% dos documentos sem falhas ou com as falhas corrigidas.

1.4 OBJETIVOS ESPECÍFICOS

Definir as estrategias necessárias para o desenvolvimento do Web Crawler focado.

Avaliar os impactos dos possíveis métodos de extração de texto.

Definir as arquiteturas que devem ser seguidas durante o desenvolvimento da ferramenta.

Desenvolver um Web Crawler focado para buscar as informações diárias do Diário Oficial da Bahia.

Desenvolver os métodos de extração de texto.

Desenvolver um método de busca de texto.

Desenvolver ferramenta que permita o acesso do usuário ao sistema.

Hospedar a ferramente em nuvem.

2 WEB CRAWLERS E EXTRAÇÃO TEXTUAL

Este capítulo aborda os conceitos que são necessários para o entendimento da ideia, concepção e desenvolvimento da ferramenta resultado do trabalho.

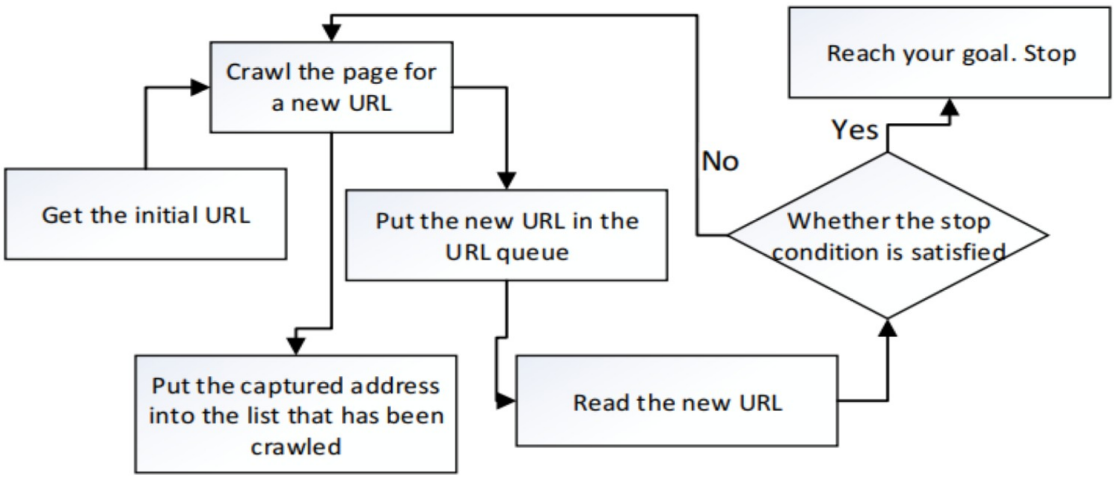
2.1 WEB CRAWLERS

Um *web crawler* é um *script* automatizado, também podendo ser chamado de rastreador, que é usado para extrair dados da WWW de forma periódica. Os *crawlers* consomem recursos nos servidores que rastreiam e muitas vezes visitam sem a aprovação do site. É importante notar que um *crawler* precisa levar em consideração questões de agendamento e carregamento, além de definir um atraso quando grandes coleções de páginas são indexadas (*politeness delay*). Sites públicos têm mecanismos para informar os rastreadores que o conteúdo não pode ser rastreado (PRATIBA et al, 2017).

Os *web crawlers* podem explorar todas as informações na página da *web*, e os motores de busca não podem ser separados dos rastreadores da web. A principal tarefa do *crawler* é rastrear todos os dados na Internet, armazenar dados necessários em bancos de dados locais e obter informações úteis (PRISMANA et al, 2020).

A teoria da realização e o processo do rastreador pode ser visto na figura 1

Figura 1 – Funcionamento *web crawler*



Fonte: Prismana; Prehanto; Nuryana; 2020

Diferentes tipos de *crawlers* estão disponíveis dependendo de como as páginas da web são rastreadas e como as futuras páginas da web são recuperadas e acessadas. Podendo ser dividido em *crawler*: genérico, incremental, focado em formulário, focado, paralelo ou distribuído (XHUMARI et al, 2021; LINXUAN, 2020).

- Os ***crawlers* genéricos**, também conhecidos como *crawlers* tradicionais, são rastreadores que pegam todos os documentos e links relacionados ao tema. Os *crawlers* genéricos recuperam um grande número de páginas de vários campos da *Web*. Para localizar e armazenar essas páginas da Web, um *web crawler* genérico deve ser executado por um longo tempo e consome muito espaço no disco rígido. Por exemplo, o algoritmo PageRank do Google retorna páginas em conformidade para pesquisar critérios de 25 bilhões de documentos na rede (LINXUAN, 2020).
- Uma necessidade importante do repositório web é manter seu conteúdo atualizado e limpar as páginas antigas ou desatualizadas assim mantendo o espaço para novas páginas. Os ***crawlers* incrementais** atualizam as páginas já disponíveis. Para realizar a tarefa, este tipo de *crawler* não inicia o rastreamento do zero, mas acompanha as páginas já disponíveis que foram alteradas desde a última vez que foram rastreadas. Frequentemente visita a web para manter o conteúdo do repositório atualizado e a sua próxima visita à web é decidida pela frequência com que a página mudou (SHRIVASTAVA, 2018).
- ***Crawlers* focados em formulários** lidam com uma dispersa distribuição de formulários na Web. *Crawlers* de formulários evitam rastrear links improdutivos restringindo a pesquisa a um tópico específico, aprendendo as características de links e páginas que levam aos formulários pesquisáveis além de usar critérios de parada. Logo, esse tipo de rastreador usa duas classificações: sites e links para guiar suas buscas. Porém, mais tarde, é utilizado um terceiro classificador, o classificador de formato, que é usado para filtrar formulários inúteis (XHUMARI et al, 2021).
- O ***crawler* focado** também é chamado de rastreador de tópico. Este tipo de *web crawler* minimiza os downloads baixando apenas as páginas apropriadas e significativas que estão relacionadas entre si. Evita rastrear caminhos sem importância limitando a área de interesse de uma pesquisa a um tópico

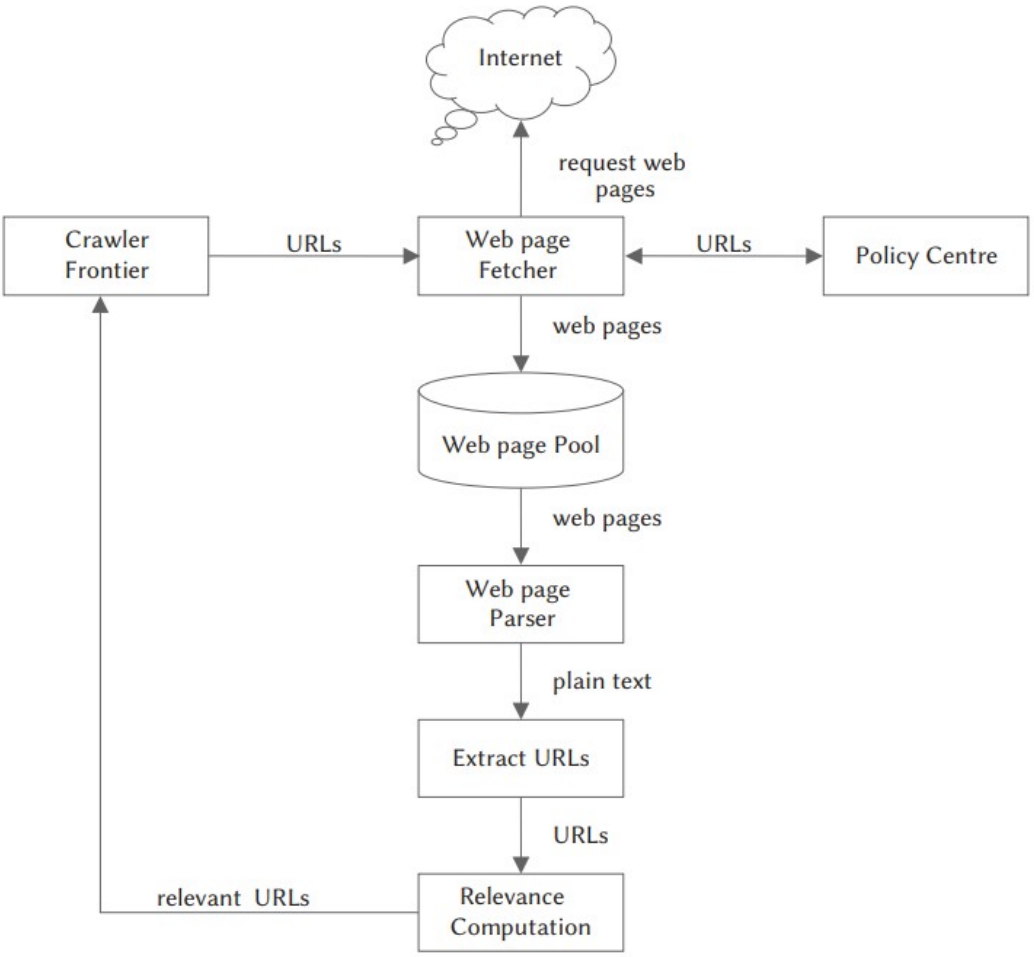
específico. Esse tipo de rastreamento é economicamente vantajoso em relação ao hardware e recurso de rede já que existe um tráfego reduzido (PRATIBA et al, 2017).

- A *Web* é gigantesca, contendo milhões de páginas da web, hiperlinks e outras informações sendo quase impossível acessar e recuperar toda ou parte notável da web em um único esforço. Portanto, muitos mecanismos de pesquisa geralmente executam vários procedimentos de maneira paralela com o objetivo de explorar cada vez mais páginas, links e partes da *web.* Esse tipo de rastreadores são conhecidos como ***crawlers* paralelos** (SHRIVASTAVA, 2018).
- Para o ***crawler* distribuído**, é importante que eles se comuniquem um com os outros, no momento, existem duas soluções, uma é o modo Senhor-Escravo, a outra é o modo Peer to Peer. A arquitetura Senhor-Escravo é amplamente utilizada em cenários distribuídos onde um nó de controle é responsável pela comunicação com todos os *web crawlers* autônomos. Os escravos só precisam adquirir os pedidos do senhor. Após o download das solicitações, a nova solicitação extraída é devolvida ao senhor (FAN, 2018).

Além dos tipos de *crawlers* citados acimas, alguns outros autores (LINXUAN, 2020; SHRIVASTAVA, 2018) chegam a mencionar a existência ainda mais tipos de rastreadores, como por exemplo: *crawlers* da *deep web*, *UML crawler e crawlers de IoT*. Porém, se tratam de tipos bem específicos de rastreadores.

Na figura 2 apresentamos um exemplo de uma arquitetura utilizada para a construção de *crawlers* focados, que foi o tipo de rastreador utilizado para a construção do trabalho.

Figura 2 – Arquitetura de um *web crawler* focado



Fonte: Dhanith; Surendiran; Raja; 2020

Como é mostrado na figura 2, uma arquitetura de um *web crawler* focado tem seus maiores componentes sendo:

- *Crawler Frontier:* podendo ser traduzido como “fronteira do rastreador”, nada mais é do que uma fila prioritária que armazena uma lista de URLs em uma ordem priorizada, com base na pontuação de relevância;
- *Web page Fetcher:* tem como função receber uma URL e, a partir dessa URL, fazer o download da página;
- *Policy Centre*: podendo ser chamada de “central de políticas”, verifica se a página da web está disponível para o download, com base em uma profundidade definida inicialmente;

- *Web page Pool*: como o nome sugere, é uma *pool* (um “conjunto”) de páginas web que já foram feitas o download. O seu objetivo é armazenar essa páginas baixadas para que futuramente possa ser feita a análise;
- *Web page Parser*: apesar de poder ser traduzido como “analisador de página”, sua função não é analisar as páginas da web e sim transformar uma página web, possivelmente complexa, em texto simples;
- *Extract URLs*: tem como objetivo encontrar e extrair todas as URLs da página;
- *Relevance Computation*: traduzida como “cálculo de relevância”, calcula a relevância de páginas da web ainda não visitadas.

Sabendo disso, o funcionamento passo a passo do *crawler* focado é o seguinte:

1. A fronteira do rastreador é inicializada com as URLs de semente e o centro de políticas com a profundidade a ser explorada.
2. O buscador de páginas(*Web page Fetcher)* baixa as páginas da web da fronteira do rastreador uma por uma. Depois que uma página da Web é baixada, o buscador de páginas extrai as URLs presentes nela e as envia para o centro de políticas.
3. A central de políticas verifica a capacidade de download da URL recebida. Um URL para download é enviado de volta ao buscador de página para download, caso contrário, ele será encerrado. As etapas (2) e (3) são repetidas até que a profundidade definida no passo (1) seja alcançada.
4. Depois que o buscador de páginas recebe a URL do centro de políticas, ele baixa a página da Web e a armazena no pool de páginas da Web como um documento HTML.
5. As páginas da Web armazenadas são enviadas para o analisador de páginas, com os URLs, para recuperar informações significativas.
6. Os fragmentos de informação extraídos são enviados para o módulo de cálculo de relevância para determinar a relevância da página web para o tópico em questão. Se a página da Web for relevante, a URL extraída será enviada para a fronteira do rastreador ou será encerrada.

Essa é a arquitetura de *crawlers* focados que é descrita por (DHANITH et al, 2020) como sendo uma das mais comuns. É importante de ter a sua ideia

compreendida, já que algumas modificações vão ser necessárias para o correto funcionamento na realização deste trabalho.

2.2 EXTRAÇÃO DE INFORMAÇÃO

A Extração de Informação (EI) é uma tarefa que visa extrair informações de interesse em documentos diversos e representar as informações extraídas de forma estruturada. As tarefas tradicionais da EI incluem reconhecimento de entidades nomeadas, que reconhece as entidades e seus tipos. Como os resultados do EI são estruturados, eles podem ser facilmente utilizados por sistemas computacionais em diferentes aplicações, como mineração de texto (WU et al, 2022).

2.2.1 Extração de Informação: Documentos HTML

Documentos na *web* são geralmente codificados em uma linguagem de marcação chamada *Hypertext Markup Language* (HTML). HTML é usado para descrever a estrutura de um documento e links em um documento de hipertexto. HTML adiciona *tags* que identificam elementos em um documento, como títulos, subtítulos, parágrafos e apêndices. Os códigos HTML incorporados no texto do documento descrevem explicitamente o texto, fornecendo informações ao cliente da w*eb* sobre como interpretá-lo. O foco principal do HTML é o conteúdo do documento, não sua aparência. É uma linguagem para descrever documentos estruturados (DUCKETT, 2011).

Importante notar que, já que o HTML é uma linguagem que descreve documentos estruturados não há a necessidade de aplicar técnicas de EI para estruturação do texto, mas ainda pode existir a necessidade de considerar as *tags* do HTML como entidades a serem reconhecidas dependendo do objetivo. Por exemplo, uma aplicação pode querer encontrar todos os links presentes em uma página, podendo assim optar por buscar por todas as *tags* de âncora (*“a” tags*).

2.2.2 Extração de Informação: Documentos PDF

Com o desenvolvimento da tecnologia da informação e a ampla disseminação da Internet, uma grande quantidade de documentos eletrônicos é armazenada em arquivos PDF. Além disso, PDF é um dos formatos de documentos mais usados para armazenar dados baseados em texto.

Este formato de arquivo foi desenvolvido pela Adobe em 1993 com o objetivo de representar um documento, independentemente da plataforma utilizada, preservando o layout na tela e na impressão. Embora esta seja uma maneira eficiente de armazenar a representação visual de um documento, a estrutura resultante é difícil de trabalhar se o objetivo for extrair partes específicas do texto de maneira estruturada (WIECHORK et al, 2021).

Na pesquisa de (BAST et al, 2017), os autores fornecem uma avaliação de 14 ferramentas de extração de PDF para determinar a qualidade e o escopo de sua funcionalidade, com base em um *benchmark* que eles construíram a partir de dados paralelos de TeX e PDF. Eles usaram 12.098 artigos científicos e para cada artigo, o *benchmark* contém um “arquivo de verdade” (para comparar e saber se foi um resultado positivo ou não), além do arquivo PDF relacionado. O estudo comparativo é um guia para desenvolvedores que desejam integrar a ferramenta de extração de metadados mais adequada e eficaz em seu software.

2.3 PESQUISA DE TEXTO

Os métodos de pesquisa de texto são amplamente necessários em aplicações modernos baseados em texto. Observemos, por exemplo, o processamento de currículos em PDF, extrair uma fórmula de alguma patente, filtragem de e-mails e assim por diante. O problema é declarado geralmente como respondendo a perguntas de algum texto.

Para encontrar uma resposta, diferentes técnicas de busca de texto podem ser aplicadas (NASR et al, 2020). Claramente, a eficiência de uma técnica de busca de texto depende essencialmente do tamanho do texto, sua estrutura e presença de erros de sintaxe. Deste ponto de vista, os métodos baseados em marcação de texto

não são adequados se o texto for mal organizado e não estruturado. Os métodos de busca baseados em palavras-chave não são eficientes em grandes bancos de dados de texto, por exemplo, em servidores web que propõem ofertas de trabalho com bilhões de arquivos. Os métodos baseados em expressões regulares podem não ser eficientes se o texto contiver erros, pois é difícil (se não possível) prever os erros concretos no texto de diferentes tópicos.

Porém, algoritmos que utilizam à distância de Levenshtein podem ajudar a evitar uma classificação errada no caso de textos que contem erros já que essa distância mede quantas operações seriam necessárias para transformar uma palavra em outra. Ou seja, à distância de Levenshtein “k” para duas strings é um número mínimo de operações (inserção, exclusão e substituição) necessárias para converter um termo (string) no outro (NIEWIAROWSKI, 2019).

Sendo assim, ao ter conhecimento do número mínimo de operações que seria necessário para transformar uma string em outra, é possível definir um *breakpoint* para que tal seja ignorada da lista de possibilidades.

Esse número mínimo de operações pode ser melhor visualizado no exemplo da Figura 3, contendo uma matriz com todas as distâncias mínimas possíveis para cada iteração das duas palavras, por exemplo: três operações de substutuição são necessárias para transformar Sim → Não, mas apenas duas para transformar Sim → Sol (i → o, m → l).

Figura 3 – Matriz da distância Levenshtein (*yesterday → tomorrow*)

		y	e	s	t	e	r	d	a	y
	0	1	2	3	4	5	6	7	8	9
t	1	**1**	2	3	3	4	5	6	7	8
o	2	2	2	3	4	4	5	6	7	8
m	3	3	3	3	4	5	5	6	7	8
o	4	4	4	4	4	5	6	6	7	8
r	5	5	5	5	5	5	5	6	7	8
r	6	6	6	6	6	**6**	5	6	7	8
o	7	7	7	7	7	7	6	6	7	8
w	8	8	8	8	8	8	7	7	7	**8**

Fonte: Niewiarowski, 2020

3 MODELAGEM DO SISTEMA

O projeto de sistemas, ou modelagem, é um tópico muito extenso dentro da engenharia de softwares. Durante esse projeto, métodos como a análise de requisitos e diagrama de casos de uso foram utilizados, junto a alguns outros diagramas que ajudaram na produção de soluções técnicas para problemas inicialmente abstratos.

3.1 REQUISITOS

Nessa etapa, foram levantados os requisitos funcionais, não funcionais e também os requisitos estendidos.

3.1.1 Requisitos Funcionais

Esses são os requisitos que podem ser considerados como funcionalidades básicas para o sistema:

- Dado usuário, deve ser enviada uma notificação sempre que algum de seus interesses forem mencionados no Diário Oficial da Bahia (RF01)
- Deve ser disponibilizado um método para o registro de usuários (RF02)
- Deve ser disponibilizado um método para deletar o seu usuário (RF03)

3.1.2 Requisitos Não-Funcionais

Já esses requisitos não-funcionais são as restrições de qualidade que o sistema deve satisfazer:

- As notificações devem ser atuais, apenas documentos publicados no dia podem ser utilizados (RN01)
- Tempo de busca reduzido para o usuário. As notificações devem conter links diretos para os documentos (RN02)

- A execução da busca deve ser rápida, o sistema deve analisar no mínimo de 1000 páginas por hora (RN03)
- As notificações do sistema não devem ser “spam”, máximo de 1 notificação por dia por usuário (RN04)
- O cadastro deve ser acessível, não limitar o registro a um tipo de sistema operacional (RN05)

3.1.3 Requisitos Estendidos

Por fim, estes são basicamente requisitos “bons de se terem” que podem estar fora do escopo do sistema

- O sistema deve salvar um *log* de execução para análise de *bugs* (RE01)
- Os *tokens* identificadores de cada usuário devem ser únicos e não sequenciais (RE02)

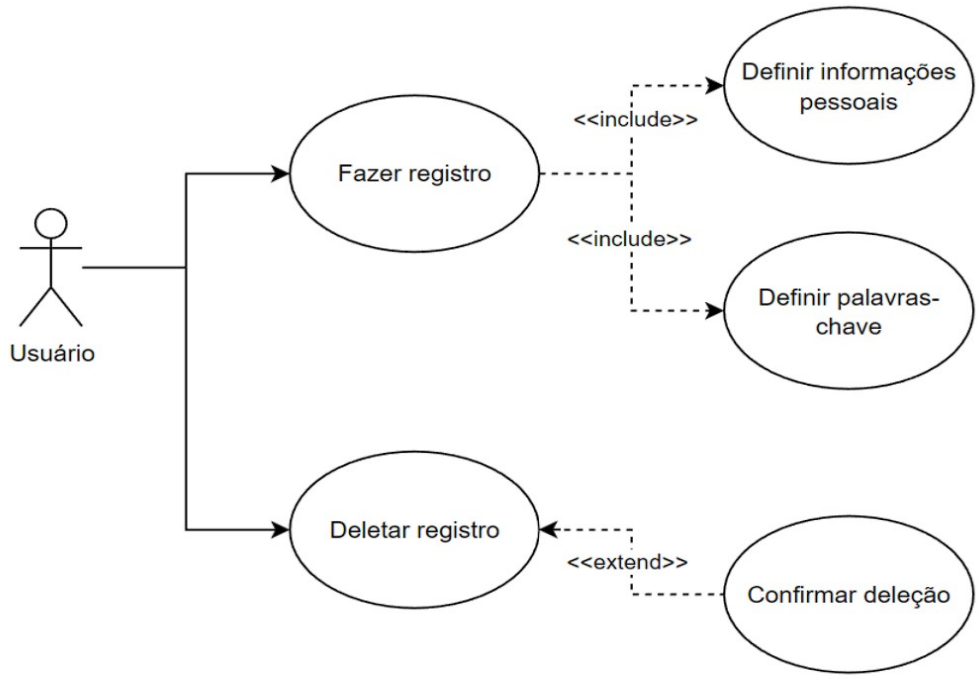
3.2 CASOS DE USO

O diagrama de caso de uso resume os detalhes dos usuários do seu sistema (também conhecidos como atores) e as interações deles com o sistema. Porém, o diagrama de caso de uso não oferece muitos detalhes.

O usuário tem duas ações dentro do sistema, entrar na lista de notificações (Fazer registro) e sair da lista de notificações (Deletar registro).

Sabendo disso, o diagrama mostrado na Figura 4 representa principalmente o modelo de fluxo básico dos eventos do usuário em cada uma dos casos de uso especificados no parágrafo anterior.

Figura 4 – Diagrama de casos de uso do sistema de registro



Ferramenta: *draw.io*

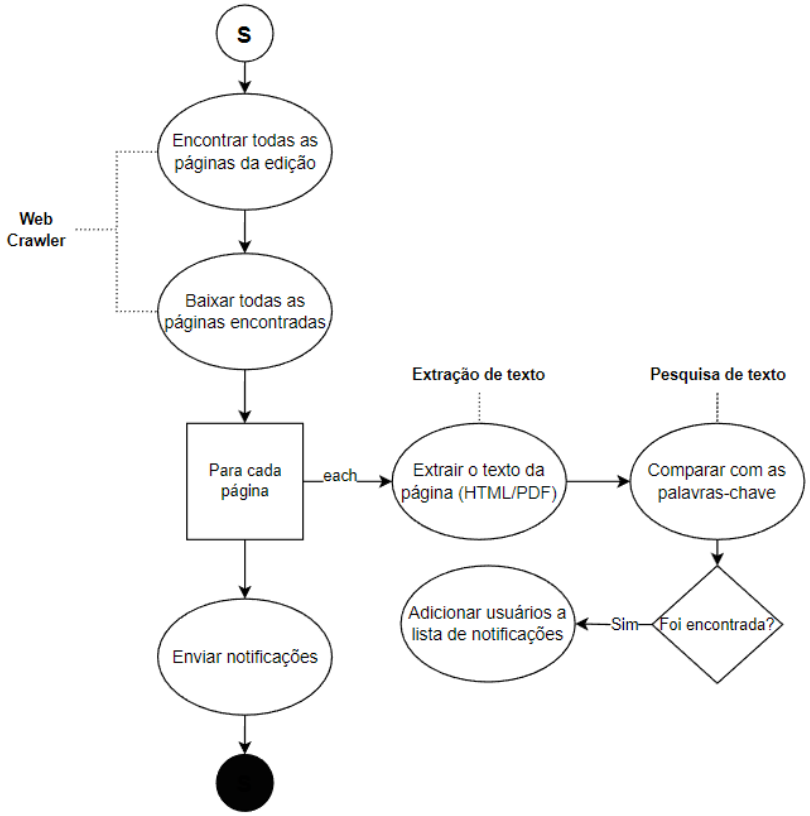
3.3 FLUXOGRAMA

Um dos pontos relacionados ao desenvolvimento da ferramenta que não estava sendo bem representado em nenhum dos outros diagramas é a parte relacionada ao funcionamento interno do sistema de busca.

A partir dessa dificuldade, foi criado um fluxograma que representa a execução, simplificada, dessa parte do sistema. E aproveitando esse fluxograma, identificada as partes mais importantes dentro dessa execução. O *Web Crawler* é o responsável por encontrar e baixar todas as páginas, sendo que cada um dessas páginas devem passar por um sistema de extração de texto para que uma busca de múltiplas palavras-chave possa ser realizada no mesmo.

A criação de versões inciais desse fluxograma permitiu perceber possíveis gargalos de performance relacionados a ordem de execução desse sistema. A versão apresentada na Figura 5 é uma versão mais aperfeiçoada e mais enxuta dessas versões iniciais do mesmo diagrama, sendo assim o fluxograma utilizado como base na codificação do serviço de busca.

Figura 5 – Fluxograma simplificado do serviço de busca



Ferramenta: *draw.io*

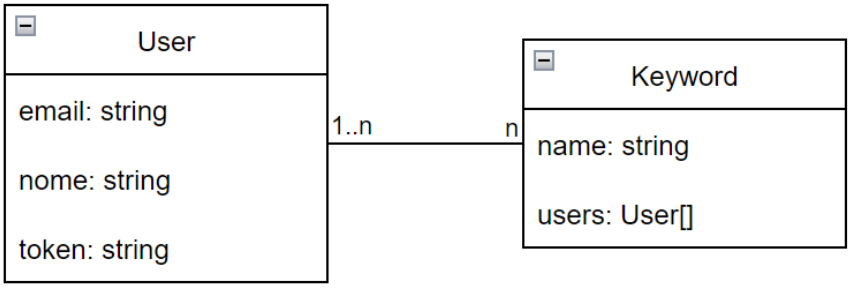
3.4 MODELAGEM DE DADOS

Nessa etapa, foram definidas quais as informações deveriam ser persistidas em banco de dados com um simples diagrama, mostrado na Figura 6.

Importante notar que, apesar de já definido quais os dados deveriam ser salvos, pelos dados presentes não formarem um sistema muito relacional torna possível a utilização de sistemas de armazenamento SQL ou NoSQL caso necessário.

Essa decisão de que tipo de banco de dados utilizar foi adiada para uma análise de opções em nuvem. A implementação final que tivesse um melhor valor seria utilizada.

Figura 6 – Modelagem de dados



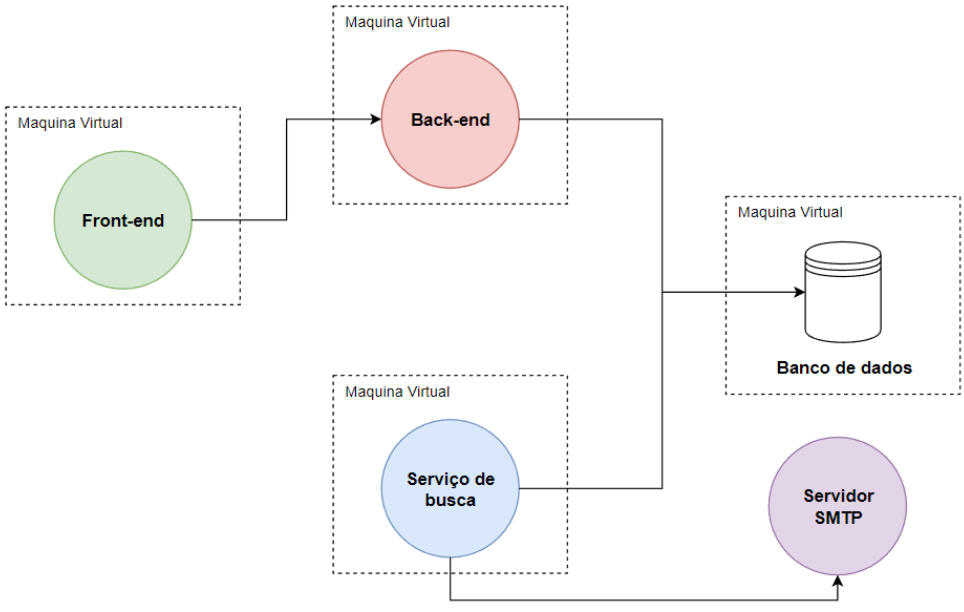
Ferramenta: *draw.io*

3.5 ARQUITETURA DO SISTEMA

O sistema foi arquitetado com base em um modelo voltado a microsserviços, que consiste em uma coleção de serviços pequenos e autônomos em que cada serviço é autocontido e deve implementar um único recurso de negócios dentro de um contexto limitado.

A modelagem inicial do sistema pode ser vista na Figura 7, porém a junção do *back-end* e do serviço de busca foi feita com o objetivo de cortar custos.

Figura 7 – Modelagem do sistema



Ferramenta: draw.io

4 DESENVOLVIMENTO DO NOTIFICA-DIÁRIO

Este capítulo descreve o desenvolvimento do projeto. São apresentados o estudo do escopo e o problema, as ferramentas e tecnologias utilizadas, além dos passos necessário para colocar tornar o sistema disponível na internet, para que qualquer pessoa possa acessar.

4.1 ESTUDO DE FERRAMENTAS E TECNOLOGIAS

Nesta etapa foram definidas as ferramentas e as tecnologias para a implementação do sistema. Mas primeiro é importante relembrar que o sistema foi construído em duas partes, um sistema para cadastro dos usuários e um *script* que pega as informações do Diário Oficial da Bahia (chamado de sistema de vigia ou serviço de busca).

4.1.1 Sistema de vigia

Este sistema, nada mais é do que um *script* que será executado diariamente por uma máquina, e seu único objetivo é pegar as informações do Diário Oficial da Bahia, comparar com a base de interesses registrados, e caso algum interesse tenha sido mencionado em um desses documentos, enviá-lo para os usuários inscritos.

4.1.1.1 Cron job

Por se tratar de um diário que pode, ou não, ter documentos publicados todos os dias, o sistema responsável por fazer essa checagem tem de ser executado também diariamente. Com isso em mente, se torna clara a necessidade de utilizar *cron jobs.*

O Cron é um utilitário de linha de comando que funciona como um agendador de tarefas, os usuários que configuram e mantêm ambientes de software usam o

cron para agendar *jobs*, também conhecidos como cron jobs, para serem executados periodicamente em horários, datas ou intervalos fixos. Ele normalmente automatiza a manutenção ou administração do sistema, embora sua natureza de uso geral também o torna útil para outras aplicações.

Existe também a implementação web da popular ferramenta, que em vez de executar um programa/script no próprio sistema unix é manda uma requisição HTTP para determinado link. Essa ferramenta online segue todos os princípios da ferramenta crontab, mas permite uma modelagem de sistemas mais orientada a microsserviços e não como um monólito (uma única máquina). Essa implementação web da ferramenta que será utilizada.

4.1.1.2 Python

Python é uma linguagem de programação interpretada, orientada a objetos e de alto nível com semântica dinâmica. Suas estruturas de dados integradas de alto nível, combinadas com tipagem dinâmica e vinculação dinâmica, o tornam muito atraente para o Desenvolvimento Rápido de Aplicativos, bem como para uso como uma linguagem de script. A sintaxe simples e fácil de aprender do Python enfatiza a legibilidade e, portanto, reduz o custo de manutenção do programa.

Por esses, e outros motivos (como a vasta quantidade de bibliotecas), Python foi a linguagem escolhida para ser a base de execução do sistema de vigia.

4.1.1.3 FuzzyWuzzy

Foi escolhida por se tratar de uma biblioteca especializada em "*string matching*", que pode ser entendido como uma procura por coincidências em *strings*, utiliza do método de distância de Levenshtein, explicado no capítulo anterior, para calcular um número que representa a similaridade entre duas determinadas *strings* disponibilizadas pelo usuário, sendo o número 0 o valor mínimo e 100 o valor máximo do resultado.

Vale notar que por padrão a ferramenta faz uso do algoritmo de Ratcliff-Obershelp para fazer essa procura, então é necessário também disponibilizar o algorítimo de Levenshtein para que o mesmo seja utilizado.

4.1.1.4 BeautifulSoup e PyPDF2

As bibliotecas BeautifulSoup e PyPDF2 são bibliotecas que permitem fazer a extração de texto de HTML (no caso do BeautifulSoup) e PDF (no caso da PyPDF2). Foram escolhidas por facilitarem a implementação do sistema, já que possuem métodos bem optimizados para a extração de texto.

Outro ponto, é que a biblioteca BeautifulSoup também permite a pesquisa por tags HTML, funcionando como um *parser* na criação do Web Crawler focado.

4.1.2 Sistema de cadastro

Esta é uma parte simples, apesar de fundamental, para o funcionamento da ferramenta. A sua necessidade vem da simples pergunta de “Como eu (usuário do sistema) vou informar o sistema de vigia que tenho interesses nos documentos relacionados ao tema gastos públicos?”.

Dado que esse usuário pode estar acessando de qualquer tipo de plataforma, como Android, iOS, Windows ou até mesmo qualquer tipo de dispositivo com outros sistemas operacionais, foi pensado que um sistema *web* seria o de mais fácil desenvolvimento. Já que uma das características dos sistemas *web* é exatamente a extrema portabilidade, escreva um sistema, seja utilizado em diversos dispositivos.

4.1.2.1 Babel

Sistemas *web* não são perfeitos, um dos seus problemas é a vasta quantidade, tanto de *browsers* disponíveis, quanto versões dos mesmo. Cada versão de cada navegador pode suportar, ou não, diferentes funções (tanto de JavaScript quanto de CSS). É ai que entra a necessidade de utilizar um transpilador como o babel.

Babel é um transpilador de JavaScript para JavaScript, o que significa que a ferramenta vai ler um código fonte em JavaScript e produzir um código equivalente em JavaScript. Mas existe um detalhe importante nesse processo, a implementação do babel permite que ele tenha acesso as funções mais recentes do padrão ECMA, o que faz com que o babel consiga ler código JavaScript recente e produzir um código JavaScript equivalente que pode ser lido por qualquer navegador que o usuário pode vir a utilizar.

4.1.2.2 Webpack

Webpack se torna uma ferramenta interessante quando se pensa em criar sistemas *web*, já que cada navegador define o suporte a certos formatos, por exemplo, o navegador Chrome em sua versão 106.0 pode suportar formatos de imagem como o formato “APNG” (*Animated Portable Network Graphics*), porém o navegador Internet Explorer não da suporte a esse tipo de formato.

Apesar dele facilitar o desenvolvedor a lidar com seus recursos de imagem, a principal vantagem na utilização do *webpack* é permitir escrever o código JavaScript em módulos (CommonJS) e “juntar” toda essa aplicação em alguns códigos estáticos de JavaScript que o navegador pode interpretar.

4.1.2.3 ReactJS

ReactJS, também conhecido como React, é uma biblioteca JavaScript de front-end gratuita e de código aberto. Se trata de uma ferramenta versátil, já que o

React se preocupa apenas com o gerenciamento de estado e renderização desse estado para o DOM (Objeto Modelo do Documento), portanto, a criação de aplicativos React geralmente requer o uso de bibliotecas adicionais para roteamento, bem como certas funcionalidades do lado do cliente.

Foi escolhida pela facilidade em escrever códigos React em contraste com o uso direto de HTML/JavaScript (no fim do processo, códigos React serão compilados para HTML/JavaScript). Além disso, as bibliotecas adicionalmente utilizadas foram:

- React Router: biblioteca mais popular para roteamento de páginas web feitas utilizando React
- Mantire: biblioteca com componentes previamente estilizados, para facilitar a montagem
- Tabler Icons: coleção de ícones grátis em SVG
- Axios: facilitar a execução de chamadas API

4.1.2.4 Flask

Flask é uma biblioteca em python que permite a criação de *backends* com facilidade. Foi escolhida principalmente por utilizar a linguagem python, que já tinha sido decidida como a linguagem do script que roda todos os dias para a coleta de informações do Diário Oficial da Bahia. Por utilizar a mesma linguagem, tanto no *backend* quanto no script, facilita a execução do mesmo, já que ele pode ser simplesmente importado como uma biblioteca em vez de executado como um programa (diminuindo o consumo geral de memória no sistema, que é algo importante ao se pensar em custo).

4.1.3 Banco de dados

Um banco de dados é uma coleção organizada de informações estruturadas, ou dados, normalmente armazenados eletronicamente em um sistema de computador. Um banco de dados geralmente é controlado por um sistema de gerenciamento de banco de dados (SGBD). Juntos, os dados e o SGBD, com os aplicativos que estão associados a eles, são chamados de sistema de banco de dados, muitas vezes abreviado para apenas banco de dados.

E mais especificamente sobre o banco de dados utilizado, é importante relembrar que o mesmo banco de dados é utilizado tanto pelo Sistema Web, quanto pelo Sistema de vigia.

4.1.3.1 NoSQL

No mundo dos bancos de dados, existem dois tipos principais de soluções, bancos de dados SQL (relacionais) e NoSQL (não relacionais). Ambos diferem na forma como foram construídos, no tipo de informação que armazenam e como a armazenam. Os bancos de dados relacionais são estruturados e possuem esquemas predefinidos, enquanto os bancos de dados não relacionais são não estruturados, distribuídos e possuem um esquema dinâmico.

Tendo essa diferenciação básica em mente, o tipo de banco de dados não relacional (NoSQL) foi escolhido como a solução utilizada por familiaridade com a plataforma de hospedagem oferecida por uma das opções de NoSQL chamada de MongoDB. Como os dados armazenados pela aplicação são bem simples, as vantagens de ambos não parecem fazer tanta diferença, já que a indexação não vai ser de grande importância (SQL) e a carga de trabalho da aplicação também não faz um uso intensivo de dados do banco (que seria um dos principais benefícios da utilização de sistemas NoSQL).

4.1.3.2 MongoDB

MongoDB é uma das mais populares implementações de banco de dados de documentos, um banco de dados de documentos é um banco de dados que armazena informações em documentos. Eles são bancos de dados de uso geral que atendem a uma variedade de casos de uso para aplicativos transacionais e analíticos. Tem como características a facilidade de uso, flexibilidade e escala horizontal, porém, pode deixar a desejar caso se faça necessário um uso de esquemas mais complexos.

4.2 IMPLEMENTAÇÃO

Tendo discutido sobre as principais implementações tecnológicas que foram utilizadas, agora será compartilhado um pouco dos desafios que apareceram durante a codificação do sistema, além de claro, as soluções encontradas.

4.2.1 Engenharia Reversa

Por não existir uma API dedicada para o consumo das notícias do diário, foi necessário um pouco de engenharia reversa para identificar qual a página do dia, já que acessar essa informação utilizando o *crawler* era inviável pela complexidade das tags HTML (se trata de um calendário) e da necessidade de executar JavaScript. O que, apesar de possível, traria uma complexidade desnecessária para o desenvolvimento.

O processo de engenharia reversa foi iniciado acessando uma página de um dia qualquer do diário, como por exemplo a página do dia 10/04/2021, e acessar uma página de um outro dia qualquer, como o dia 11/04/2021 por exemplo, para assim verificar se o *id* utilizado na URL era incremental. Porém, foi percebido que não é, o dia 10/04/2021 tem id=11954 e o dia 11/04/2021 tem id=11962, com um pouco mais de pesquisa foi percebido que isso se dá pois o portal também

contempla outros “diários” (como de prefeituras, câmaras e etc) que não são publicados diariamente.

Mas, ao inspecionar as requisições que eram feitas ao selecionar uma nova data ficou fácil de identificar como era obtido esse *id* que representa o dia do diário. Existe um *endpoint* utilizado que ao ser enviado a data do dia escolhido, no formato YYYY-MM-DD (Ano-Mês-Dia), responde com um json contendo o *id* daquela data, caso existente.

A partir daí, a parte de identificar quais são as notícias que podem ser clicadas e pegar o conteúdo textual delas vai ser trabalho do *crawler*.

4.2.2 Código

Durante a codificação do *front-end web* da aplicação não muitos desafios foram encontrados, já que se trata de um *layout* simples, um pequeno formulário ao fim da página. Ao preencher esse formulário, uma requisição contendo os dados inseridos é feita ao *back-end* da aplicação, que por sua vez realiza o cadastro.

Figura 8 – Visão geral da página (*Desktop*)

Link: https://yrodrigo2219.github.io/notifica-diario

E quanto aos recursos da página, o usuário pode esperar um tempo de resposta de aproximadamente 0ms com relação ao carregamento das páginas (sem considerar imagens) já que um dos benefícios da utilização do React junto ao React-Router-Dom é a possibilidade de tornar todo o site em um SPA (Single Page Application), onde todo o conteúdo é carregado inicialmente tornando o uso do site similar ao uso de um aplicativo *desktop* em que não é necessário esperar o carregamento de cada uma das páginas a cada clique.

Esta implementação de SPA pode causar problemas em aplicativos com maior quantidade de conteúdos, sendo necessário a utilização de métodos de *lazy loading*, mas como não é o caso desse projeto, foge do escopo de discussão.

A implementação do *back-end* do sistema *web* também foi bem simples, já que a escolha do Flask disponibiliza tudo o que é necessário para implementações de REST APIs. Uma vez que já estavam definidos quais eram os requisitos funcionais do sistema, foram implementadas as rotas que correspondem aos mesmos. Portanto, foram desenvolvidas as seguintes rotas (Tabela 1) para a aplicação:

Tabela 1 – Rotas da aplicação

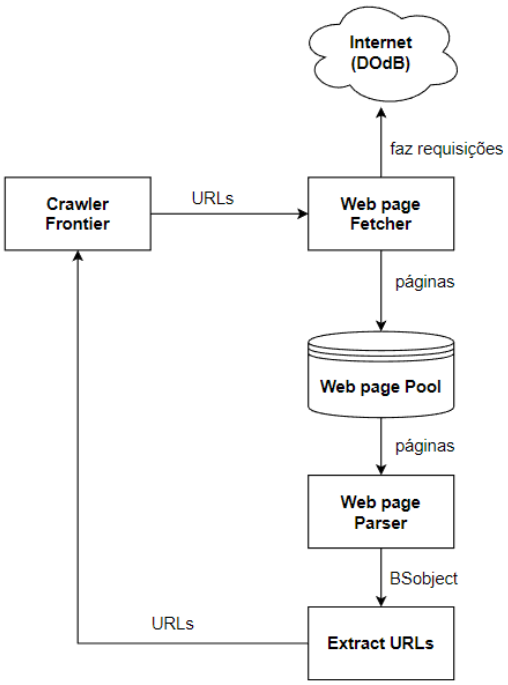
Requisitos	**Método**	**Rotas**	**Objetivo**
RF01	GET	/start_cron_job	Rota protegida, acessada apenas pelo serviço de Cron
RF02	POST	/users	Rota de registro
RF03	POST	/unsubscribe/:id	Rota de deleção
RE01	GET	/log	Rota para acompanhar execução

Agora, no que se diz respeito ao serviço de busca, a codificação teve como base a utilização do fluxograma discutido no capítulo de modelagem do sistema. Porém, esse fluxograma só se tornou útil depois de implementada a parte mais essencial de todo o sistema, o *crawler*.

O *crawler* focado que foi implementado segue a arquitetura discutida durante o segundo capítulo do trabalho, uma melhor visão da arquitetura pode ser encontrada ao se visualizar a Figura 2 presente no segundo capítulo.

Porém, como discutido durante a apresentação da arquitetura, a versão implementada contem algumas diferenças já que tem um objetivo diferente do proposto pelos idealizadores. A Figura 9 representa a arquitetura utilizada para o desenvolvimento do *crawler.*

Figura 9 – Arquitetura utilizada para o desenvolvimento do *crawler*



Como é possível perceber ao comparar a Figura 9 com a Figura 2, quatro modificações foram realizadas na arquitetura do *crawler*:

- O acesso não é feito a Internet de modo geral e sim ao site do Diário Oficial da Bahia
- Não há a presença do *Policy Center*, já que o seu objetivo é avaliar qual a profundidade em que o *crawler* deve percorrer e nessa implementação ele deve sempre percorrer o máximo possível (todos as páginas postadas no dia devem ser baixadas)
- Também não há a presença do *Relevance Computation*, pelo mesmo motivo citado anteriormente, todos os links pertencentes ao Diário Oficial da Bahia devem ser seguidos
- Por fim, o *Web page Parser* não tem como output um texto e sim um objeto, o objetivo é tornar mais fácil a programação do extrator de URLs uma vez que ele pode pesquisar por tags/propriedades.

O *Web page Fetcher* utiliza da biblioteca *requests* do *python* para realização das requisições, já o *Web page Pool* e o *Crawler Frontier* se tratam de objetos que funcionam essencialmente como listas, uma responsável por armazenar as páginas e o outro responsável por armazenar as URLs que ainda precisam ser acessadas. Por fim, o *Web page Parser* utiliza do BeautifulSoup4 para representar o texto obtido em um objeto que é utilizado pelo extrator de URLs pela pesquisa de tags e assim encontrar os links.

Uma vez que o *crawler* estava pronto e buscando todas as páginas assim como o esperado, foi implementado a segunda parte do serviço de busca, a extração de texto.

Um dos primeiros problemas da extração de texto foi relacionado ao formato do conteúdo baixado pelo *crawler*, a depender do formato (HTML ou PDF) uma ferramenta de extração diferente deve ser utilizada. Para isso, são analisados os metadados do conteúdo com o objetivo de determinar se ele se identifica como um conteúdo em PDF ou HTML e a partir dai utilizar a ferramenta mais apropriada para o caso.

Como já foi discutido, os documentos em PDFs tem seus textos extraídos pela ferramenta PyPDF2 e os documentos HTML tem seus textos extraídos pelo BeautifulSoup4, que trata da remoção de tags. Além disso, mais um processamento é realizado, esse se tratado da remoção de pontuações em geral (como vírgulas, pontos e etc) além da remoção de quebras de linhas. O motivo dessas remoções é tornar mais simples a comparação de strings, partindo do princípio que as strings “SUDEC” e “SUDEC;” devem ser consideradas como uma combinação.

Outro ponto é a divisão de palavras adicionalmente com a quebra de linha, por exemplo “SUDEC” e “SU-\nDEC” (considerando \n como uma quebra de linha) devem também serem consideradas como sendo uma combinação e por isso a remoção adicional da quebra de linhas.

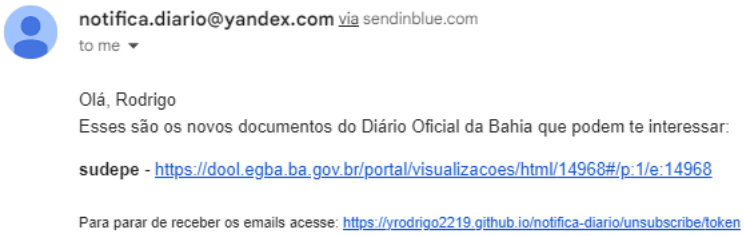
O método de comparação escolhido foi o mais simples possível, para manter o sistema performático mesmo quando muitas palavras-chave devem ser buscadas em determinado texto. A ideia é: ao receber um texto e uma palavra-chave, esse algoritmo divide o texto em palavras e calcula à distância de Levenshtein dessa palavra-chave para cada uma dessas outras palavras extraídas do texto gerando

assim um número que representa a relação entre essas duas palavras (100 sendo a mesma palavra e 0 caso sejam palavras completamente diferentes), sendo assim, caso esse resultado esteja acima de determinado limite, esse texto é marcado como contendo determinada palavra-chave e novos textos/palavras podem ser analisados.

O motivo da utilização de um limite, 85 no caso, é a presença de artefatos ao fazer a extração de texto dos documentos, principalmente os documentos em PDF. Causando de algumas letras se tornarem símbolos irreconhecíveis ou até mesmo ignorando alguma delas.

Por fim, resta apenas o envio dos e-mails com base nos algorítimos anteriores, mas a ideia é criar um simples e-mail em HTML que contenha todas as informações que podem ser relevantes para o usuário, como pode ser visto na Figura 10.

Figura 10 – E-mail enviado ao usuário



4.2.3 Hospedagem na Nuvem

Já que o objetivo do trabalho era disponibilizar a ferramenta para que qualquer pessoa pudesse acessar, foi necessário fazer o *deploy* do código produzido para servidores na internet seguindo a arquitetura de microsserviços discutida no capítulo de modelagem do sistema.

Como foi mostrado na imagem da Figura 7, o sistema terá seu deploy dividido em 4 partes: um para o front-end, outro para o back-end e o serviço de busca (que são independentes, mas dessa forma tem o custo reduzido), outro para o banco de dados e por último, a utilização de um servidor SMTP de um terceiro.

Ao avaliar as opções disponíveis, uma constante utilizada em todas as avaliações foi o preço. Foi importante que as plataformas utilizadas oferecessem

algum *tier* inicial gratuito para que esse projeto pudesse ser mantido por tempo indefinido.

Durante a avaliação de onde disponibilizar o front-end, as principais opções eram o sistema da Vercel e o Github Pages, sendo que para essa aplicação o sistema Vercel não tinha muitas vantagens e tinha a grande desvantagem de utilizar funções lambda para hospedagem (tendo como características a presença de cold starts). Então, por não comprometer em praticamente nada, foi escolhido fazer a hospedagem do front-end com o Github Pages.

Já na avaliação de VPSs (Virtual Private Servers), que é responsável por hospedar tanto o back-end quanto rodar o serviço de busca, a principal alternativa (agora paga) é o sistema da Heroku. Outro serviço que também poderia ser utilizado é o serviço da Vercel, mas por se tratar de funções lambda o tempo de execução máximo de cada uma não pode exceder 10 segundos, tornando assim inviável a execução do serviço de busca que pode chegar a diversos minutos.

A alternativa utilizada foi o serviço do fly.io que tem as mesmas características de escala horizontal do Heroku e também implementa a filosofia de Vms (Virtual Machines) descartáveis. A principal diferença sendo o preço inicial, o Heroku tem um preço de 7 dólares mensais enquanto o fly.io possui um pacote gratuito.

A hospedagem do banco de dados foi facilmente decidida, já que o Mongo Atlas é um dos pouco que oferece um pacote gratuito para teste. E por fim, o serviço do cron-job.org é utilizado uma vez por dia para informar ao servidor que é hora de rodar o serviço de busca, além de periodicamente checar se o back-end está respondendo ou se caiu por alguma razão.

Importante notar que essa arquitetura voltada a microsserviços traz também alguns problemas, como a latência de conexão entre os mesmos, o que pode prejudicar a experiência do usuário em casos onde muitos dados precisam ser trocados a todo instante. Mas tem como principal vantagem a tolerância a falhas, se uma parte do sistema parar de funcionar, nem tudo vai parar.

Por fim, o serviço utilizado para o envio de e-mails é o serviço do Sendinblue, permite o envio de e-mails via código (conectando ao servidor SMTP) e ainda oferece uma *dashboard* que mostra todas as informações relacionadas ao ciclo de vida dos e-mails, se chegaram ao destinatário, se foram abertos, se foi reportado e etc.

5 AVALIAÇÃO

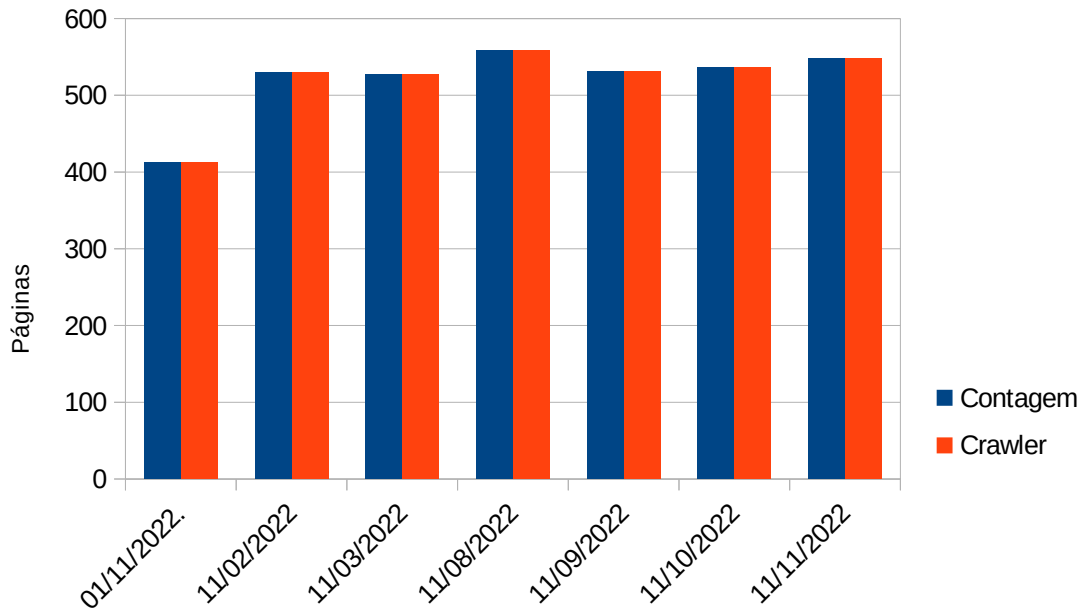
Com o objetivo de validar a eficacia na notificação dos interesses dos usuários, forma realizados *backtests* com arquivos antigos do Diário Oficial da Bahia. A metodologia de teste adotada busca avaliar a eficiência de diferentes partes do sistema.

5.1 CRAWLER

O *crawler* foi avaliado fazendo a comparação da contagem de páginas disponíveis no menu lateral do Diário Oficial da Bahia (que contem todas as páginas da edição). A contagem de páginas será realizada baixando o HTML apenas desse menu lateral, retirando as *tags* relacionadas as pastas de cada categoria e depois realizada a contagem. O resultado ideal é a identificação de 100% dos documentos publicados, mas um resultado maior que 95% também poderia ser considerado um resultado de sucesso.

Ao realizar a análise proposta com páginas do Diário Oficial da Bahia entre os dias 01/11/2022 até o dia 11/11/2022 os resultados obtidos podem ser observados no Gráfico 1, comparando o crawler com a quantidade de páginas disponíveis.

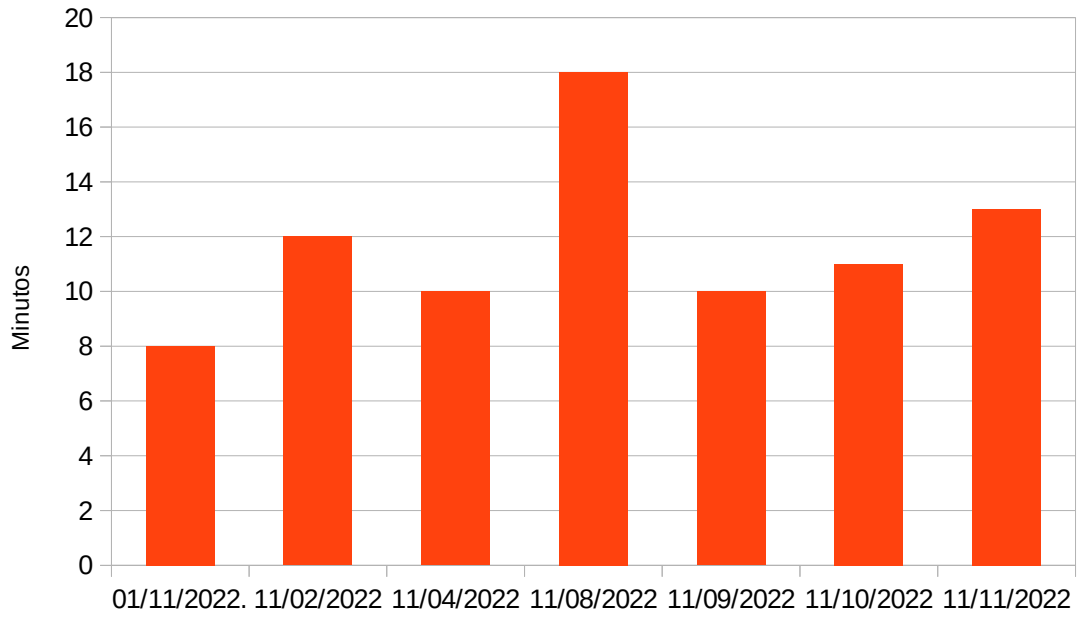
Gráfico 1 – Gráfico de análise do Crawler



Primeiro é importante clarificar que os dias não presentes no gráfico representam dias em que nada foi publicado no Diário Oficial da Bahia. Com isso, é possível perceber que ambos algorítimos chegaram a mesma quantidade de páginas disponíveis em todos os dias testados, registrando assim um sucesso no que se diz respeito a quantidade de páginas que o sistema está encontrando. Todos os documentos postados nesses dias foram baixados.

Outro ponto de avaliação é o tempo de execução (tempo do fim – tempo do início) dessas buscas, já que um dos requisitos de desenvolvimento do sistema é a mesma não deve ser lenta (mais de 1000 arquivos por hora). Observando os tempos de busca no Gráfico 2, junto com as quantidades obtidas no Gráfico 1, é possível chegar ao número de 1860 arquivos por hora (Quantidade / Tempo), nos piores dos casos observados (dia 08/11/2022).

Gráfico 2 – Tempo de busca do Crawler

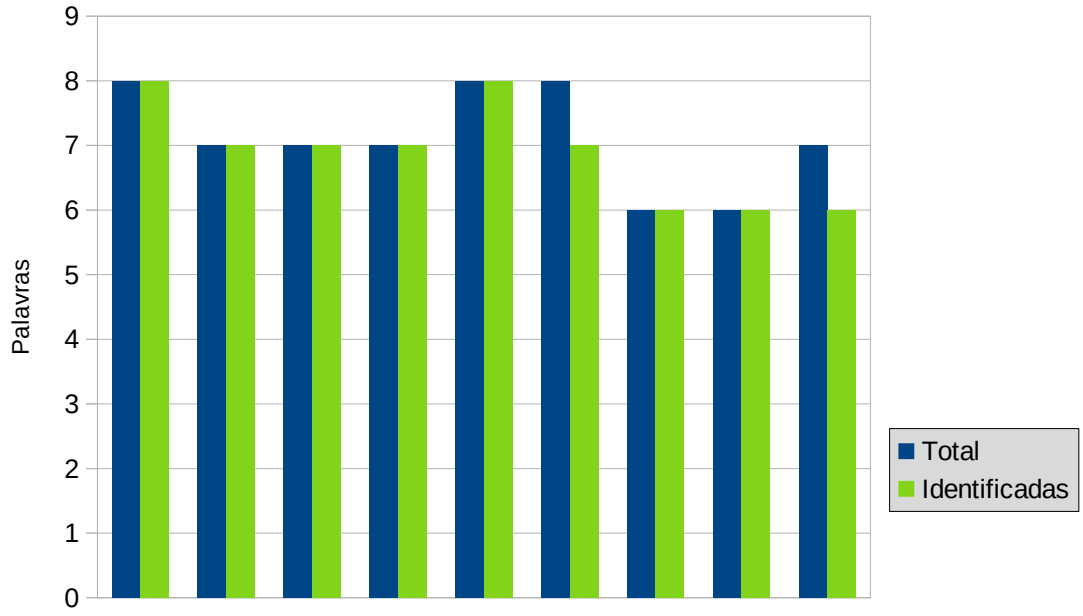


5.2 MÉTODOS DE EXTRAÇÃO TEXTUAL

Os métodos de extração textual serão avaliados com documentos previamente publicados no Diário Oficial da Bahia (PDF e HTML) junto de uma lista de palavras identificadas por um humano em cada um desses documentos, cada uma dessas palavras serão buscadas com um algorítimo de força bruta no resultado da extração, caso a palavra possa ser encontrada (nenhum caractere foi separado ou corrompido) é considerado uma extração de sucesso.

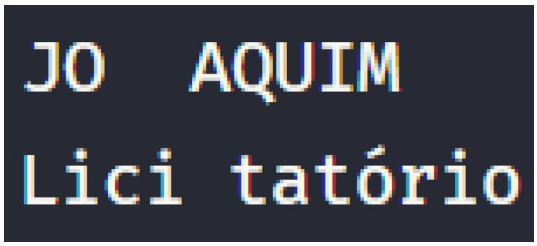
No Gráfico 3 é mostrado a quantidade de palavras que foram buscadas no documento (Total) comparada com a quantidade de palavras que puderam ser identificadas sem nenhum tipo de deformação (Identificadas).

Gráfico 3 – Método de extração textual



Com a análise de 9 documentos aleatoriamente selecionados pelo *crawler* e um total de 64 palavras escolhidas previamente por um ser humano, os métodos de extração de texto foram capazes de produzir 62 palavras perfeitamente escritas e as 2 palavras corrompidas são apresentadas na Figura 11, tendo um aproveitamento de aproximadamente 97%.

Figura 11 – Palavras corrompidas pelo algoritmo



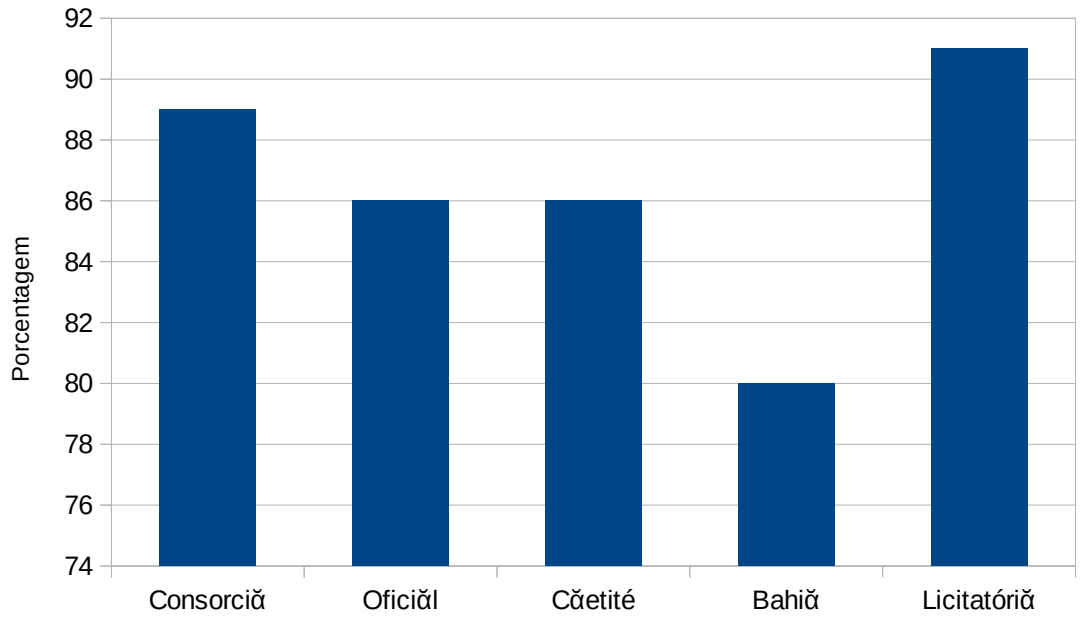
5.3 MÉTODO DE BUSCA

O método de busca de palavras-chave será avaliado com uma lista de palavras levemente corrompidas (até 1 caractere).

Como as palavras corrompidas pelo método extrator adicionaram espaços e não substituíram caracteres, a utilização das mesmas para análise do método de busca é inútil, já que ele utiliza dos espaços para dividir as palavras do texto e compará-las uma a uma com uma dada palavra-chave (com base na distância de Levenshtein).

Mas, considerando que uma das possibilidades levantadas era essa corrupção de palavras e foi o motivo da utilização do algoritmo de distância de Levenshtein, uma lista de palavras levemente corrompidas será utilizada para análise de eficiência do método. O Gráfico 4 demonstra o resultado de quanto a palavra corrompida combina com a palavra original.

Gráfico 4 – Método de extração textual



É notável que todas as palavras, exceto a palavra Bahia, passaram dos 85% necessários para que seja considerado como uma combinação. Porém, dado os resultados da eficiência dos métodos de extração textual, é mais provável que esse método de busca traga mais pontos negativos do que positivos, mais testes seriam necessários para confirmar essa ideia.

6 CONSIDERAÇÕES FINAIS

Este trabalho foi desenvolvido com o objetivo de criar um sistema de notificações para o portal do Diário Oficial da Bahia, partindo da ideia de que acessar o portal frequentemente é uma experiência desagradável.

O sistema produzido tinha como objetivo contemplar pelo menos 90% dos documentos publicados e a julgar dos testes realizados teve um resultado mais do que satisfatório na área de reconhecimento do site (crawler) e no que diz respeito a extração dessas informações (métodos de extração textual), já que contando apenas com esses dois pontos, aproximadamente 97% dos documentos são analisados corretamente.

Porém, um dos pontos projetados, que foi a correção dos erros de extração, não teve um resultado satisfatório, a ideia da utilização do método de distância de Levenshtein era considerar palavras extremamente similares como sendo uma combinação, para que dessa forma erros de poucos caracteres pudessem ainda serem reconhecidos. O método funciona nesse caso, como foi apresentado nos testes, mas, o mesmo é praticamente inútil no que se diz respeito aos erros apresentados dos métodos de extração textual (que contam com espaços extras e não caracteres trocados).

Ainda sobre o método de busca utilizado, o mesmo utiliza uma separação por espaços para realizar a comparação de palavras do texto com palavras-chave dos usuários cadastrados o que impossibilita a utilização de “frases” chave por meio dos usuários. Impossibilitando a pesquisa por nomes completos, por exemplo.

Outro ponto projetado que acabou não sendo cumprido foi a separação em múltiplos serviços independentes, já que o serviço de busca está hospedado na mesma máquina que o back-end do sistema web. Mas este pode ser resolvido com um investimento de capital direcionado ao aluguel de uma outra máquina. Ainda sobre a estrutura utilizada, essa divisão em serviços independentes torna a possibilidade de escalar a aplicação horizontalmente, o que é ainda mais facilitado pela plataforma fly.io atualmente utilizada para hospedagem.

Com esses pontos em mente, os objetivos futuros do trabalho podem ser reduzidos a:

- Remoção do método de distância de Levenshtein

- Pesquisa/desenvolvimento de um método de reconhecimento resistente a espaços
- Separação do back-end web e serviço de busca
- Escala horizontal (dependente da quantidade de usuários)
- Buscar por frases-chave
- Divulgação da ferramenta para uso público

Por fim, a ferramenta pode ser acessada por esse link:

➔ https://yrodrigo2219.github.io/notifica-diario

REFERÊNCIAS

DHANITH, P. R. J.; SURENDIRAN, B.; RAJA, S. P. A Word Embedding Based Approach for Focused Web Crawling Using the Recurrent Neural Network. **International Journal of Interactive Multimedia and Artificial Intelligence**, v. InPress, n. InPress, p. 1, 2020.

FAN, Y. Design and Implementation of Distributed Crawler System Based on Scrapy. **IOP Conference Series: Earth and Environmental Science**, v. 108, jan. 2018.

YU, L. et al. Summary of web crawler technology research. **Journal of Physics: Conference Series**, v. 1449, jan. 2020.

PRATIBA, D; SHOBHA, G; LALITHKUMAR, H; SAMRUDH, J. Distributed Web Crawlers using Hadoop. **International Journal of Applied Engineering Research**, v. 12, 2017.

PRISMANA, G; PREHANTO, D; NURYANA, K. The Design and Implementation of Web Crawler Distributed News Domain Detection System. **International Joint Conference on Science and Engineering**, v. 196, 2020.

SHRIVASTAVA, V. A Methodical Study of Web Crawler. **Journal of Engineering Research and Applicatio**, v. 8, nov. 2018.

XHUMARI, E.; XHUMARI, I. **A review of web crawling approaches**. [s.l: s.n.]. Disponível em: <https://ceur-ws.org/Vol-2872/short01.pdf>. Acesso em: 10 set. 2022.

BAST, H.; KORZEN, C. **A Benchmark and Evaluation for Text Extraction from PDF**. Disponível em: <https://ieeexplore.ieee.org/document/7991564>. Acesso em: 11 nov. 2022.

DUCKETT, J. **HTML & CSS design: design and build websites**. Indianapolis: John Wiley & Sons, Inc, 2011.

WIECHORK, K.; CHARÃO, A. Automated Data Extraction from PDF Documents: Application to Large Sets of Educational Tests. **Proceedings of the 23rd International Conference on Enterprise Information Systems**, 2021.

WU, X.; ZHANG, J.; LI, H. Text-to-Table: A New Way of Information Extraction. **Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics** v. 1, p. 2518–2533, 2022.

NASR, S.; GERMAN, O. V. A SEARCHING ALGORITHM FOR TEXT WITH MISTAKES. **Doklady BGUIR**, n. 1, p. 29–34, 6 mar. 2020.

NIEWIAROWSKI, A. Similarity detection based on document matrix model and edit distance algorithm. **Computer Assisted Methods in Engineering and Science**, v. 26, n. 3–4, p. 163–175, 31 dez. 2019.