Marcos Aurélio Lessa

CrawlEX: uma ferramenta para extração de dados na web configurável através de exemplos

Florianópolis – SC

2022

Marcos Aurélio Lessa

CrawlEX: uma ferramenta para extração de dados na web configurável através de exemplos

Proposta de Trabalho de Conclusão de Curso para obtenção do grau de Bacharel no curso de Ciência da Computação na Universidade Federal de Santa Catarina.

Universidade Federal de Santa Catarina – UFSC

Departamento de Informática e Estatística – INE

Graduação em Ciência da Computação

Orientador: Carina Friedrich Dorneles

Florianópolis – SC

2022

Marcos Aurélio Lessa
CrawlEX: uma ferramenta para extração de dados na web configurável através de exemplos/Marcos Aurélio Lessa. – Florianópolis – SC, 2022
Orientador: Carina Friedrich Dorneles

Trabalho de Conclusão de Curso (Graduação) – Universidade Federal de Santa Catarina - UFSC, 2022.

Marcos Aurélio Lessa

CrawlEX: uma ferramenta para extração de dados na web configurável através de exemplos

Trabalho aprovado.

Carina Friedrich Dorneles
Orientador

Maicon Rafael Zatelli
Banca 1

Ronaldo dos Santos Mello
Banca 2

Florianópolis – SC
2022

Agradecimentos

Àqueles a quem devo tudo: pai e mãe, por todo suporte e amor ao longo de toda minha trajetória. Aos meus irmãos, que sempre serviram de inspiração e exemplo. Ao meu sobrinho, Logan, que chegou há pouco para trazer mais alegria à nossas vidas. Aos verdadeiros amigos que carrego comigo desde a infância e àqueles que conheci na graduação. A todos os professores que tive ao longo da vida, em especial aos da UFSC, por todos ensinamentos, os quais carregarei comigo no meu dia a dia profissional.

"Pois a sabedoria é uma proteção. Assim como o dinheiro é uma proteção. Mas a vantagem do conhecimento é esta: a sabedoria preserva a vida de quem a possui."
(Eclesiastes 7:12)

Resumo

Com o grande avanço da internet ao longo dos anos, é natural que tenhamos uma enorme quantidade de dados disponível na rede. Esses dados, podem nos informar coisas completamente diferentes, como o que foi falado no último discurso do Presidente da República ou, a coordenada geográfica de um local que estamos interessados em visitar. Dependendo do perfil de interesse de um usuário ou até mesmo de uma empresa, é muito importante ter esses dados em mãos para que se possa analisá-los e, eventualmente, tomar algum tipo de ação. Porém, na grande maioria das vezes, é inviável que esses dados sejam coletados manualmente, pois demandam tempo e esforço, logo, faz-se necessário que a coletada seja feita de maneira automática, permitindo ao interessado apenas fazer a análise daquilo que efetivamente já foi coletado. Além disso, para que a configuração de uma coleta de um *website* seja feita de forma automática, é necessário que o usuário tenha habilidade em programação, sendo assim, um empecilho para muitas pessoas. Nesse contexto, o presente trabalho apresenta uma ferramenta para navegação e extração de artigos disponíveis na internet, onde um *web crawler* pode ser configurado por um usuário comum, sem conhecimentos em programação, apenas por fornecer exemplos de artigos das páginas as quais tem interesse. É apresentado os experimentos feitos pelo autor e usuários leigos, e depois analisados os seus resultados.

Palavras-chave: extração de dados, artigos, web crawler

Lista de ilustrações

Lista de tabelas

Lista de códigos

Lista de abreviaturas e siglas

SQL	Structured Query Language
API	Application Programming Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
URL	Uniform Resource Locator
XML	Extensible Markup Language

Sumário

1 Introdução

É notório que ao longo dos últimos anos, o crescimento no uso da internet é extremamente grande. Graças aos *smartphones* e outros dispositivos portáteis, o acesso a uma infinidade de páginas *web* se tornou extremamente acessível, pois agora, as pessoas podem navegar por essas páginas apenas segurando um equipamento eletrônico na palma de sua mão, em praticamente qualquer lugar e a qualquer horário. Essa acessibilidade faz com que cada vez mais o número de dados disponíveis na Internet cresça. Muitos desses dados estão publicados como artigos (sejam eles notícias, artigos científicos, textos publicados em blogs, portais, entre outros) que podem trazer inúmeras informações relevantes para diversas pessoas e empresas com perfis totalmente diferentes umas das outras.

Um jeito prático para poder monitorar esses artigos é através da criação de um web *crawler* capaz de extrair as informações desejadas que se fazem presente nas páginas *web* onde esses artigos estão publicados. Para isso, é necessário que se tenha um conhecimento mínimo em programação e, às vezes, até mesmo em redes de computadores, já que a extração de dados nesse caso se dá por conteúdos publicados na Internet.

É verdade que ao longo dos últimos anos, diversas linguagens de programação de alto nível possuem alguma API ou biblioteca que facilitam que esta extração ocorra, fazendo de forma automática um trabalho que o programador teria que fazer de forma manual. Mas isso, ainda não retira a necessidade de que os usuários dessas APIs necessitem ter um certo conhecimento técnico para fazer alguns ajustes em termos de código e análise do conteúdo a ser extraído.

Nesse contexto, o presente trabalho tem como objetivo a implementação de uma aplicação que permite que a configuração de coleta de informações de páginas com publicações de artigos em geral, seja possível de ser realizada por um usuário comum, sem qualquer tipo de conhecimento em programação, extração de dados ou *crawling*. A proposta é que isso seja feito por meio do fornecimento de exemplos, onde um usuário pode entrar com a URL da página onde o artigo está publicado, título, subtítulo, conteúdo, nome do autor, e a data de publicação.

Ainda, o usuário tem a liberdade de configurar um *crawler* de maneira personalizada, podendo selecionar um ou mais dos quaisquer *websites* que foram previamente mapeados. Também, terá a possibilidade de fornecer palavras e/ou frase de seu interesse, para que sirva como filtro para o *crawler* fazer ou não a coleta e, após isso, enviar um relatório dos artigos extraídos para os *e-mails* que o usuário também desejar.

Este trabalho está estruturado em 7 capítulos com esta introdução. No capítulo 2 são apresentados os conceitos básicos envolvidos no processo de extração de dados de

páginas *web* e da navegação metódica por um robô na internet, além do Algoritmo de Ratcliff-Obershelp, usado para comparação entre dois conteúdos textuais. Em seguida, no Capítulo 3, são apresentados alguns trabalhos relacionados que serviram de inspiração para este, além de apresentar um comparativo entre eles. A descrição das atividades práticas desenvolvidas para a software proposto, sua arquitetura e seu funcionamento estão no capítulo 4. O capítulo 5 traz os experimentos realizados e seus resultados obtidos, sendo realizado análise em cima das informações. No capítulo 6 são feitas algumas considerações finais e sugerido algumas atividades para trabalhos futuros, a fim de melhorar a aplicação. Por fim, as referências bibliográficas usadas aqui são apresentadas no capítulo 7.

1.1 Objetivo geral

O objetivo geral deste trabalho é desenvolver um software onde um usuário sem conhecimento em programação, *web crawler* e extração de dados da *web*, consiga configurar a extração de artigos em sites através de exemplos, bem como a configuração de um *web crawler* para navegar nos sites configurados por ele, sem qualquer tipo de manipulação em código de qualquer linguagem de programação.

1.2 Objetivos específicos

- O1. Desenvolver uma interface amigável para a aplicação, onde permita que o usuário consiga interagir de forma fácil e intuitiva;
- O2. Permitir que o usuário forneça exemplos dos conteúdos das páginas para a configuração da extração dos dados;
- 03. Desenvolver mecanismo de navegação pelas páginas *web* e fazer a extração dos conteúdos;
- O4. Analisar os resultados obtidos.

2 Fundamentação Teórica

Neste capítulo, é discutido o referencial teórico necessário para compreensão da proposta apresentada pelo presente trabalho. São abordados os principais conceitos, definições e características relacionados a extração de dados em páginas *webs* e *Web Crawler*. Também, é apresentado o algoritmo de Ratcliff-Obershelp para comparação textual.

2.1 Extração de dados de páginas web

A extração de informações da *Web* se baseia no problema de extrair itens alvos de informação de páginas *Web*. Existem dois problemas gerais nesta área: extrair informação de textos em linguagem natural e extrair dados estruturados de páginas *Web* (LIU, 2011, p. 363). Este trabalho enfatiza a extração de dados estruturado de páginas *Web*, no caso, dados provenientes da publicação de artigos.

"Uma abordagem tradicional para a extração de dados de páginas *Web* é a escrita de programas especializados, chamados de *wrappers*, que identificam os dados de interesse e mapeiam eles para um formato adequado" (LAENDER et al., 2002, tradução nossa). A seguir são descritos os seis principais grupos de ferramentas para a geração de *wrappers* apresentados em Laender et al. (2002):

1. Linguagens para desenvolvimento de wrappers: Linguagens especificamente criadas para ajudar os usuários a construir wrappers. Estas linguagens foram propostas como alternativas a linguagens de propósito geral como Java. Alguns exemplos de ferramentas que adotam essa abordagem são: Minerva (CRESCENZI; MECCA, 1998) e TSIMMIS (HAMMER; MCHUGH; GARCIA-MOLINA, 1997);

2. Ferramentas conscientes do HTML: Ferramentas que dependem da estrutura inerente dos documentos HTML. Antes de começar o processo de extração, estas ferramentas transformam o documento em uma árvore de analise, uma representação que reflete a hierarquia de tags da página HTML. Em seguida, regras de extração são geradas semiautomaticamente ou automaticamente e são aplicadas a árvore. Algumas ferramentas representativas baseadas nessa abordagem são: W4F (SAHUGUET; AZAVANT, 2001) e XWRAP (LIU; PU; HAN, 2000);

3. Ferramentas baseadas no Processamento de Linguagem Natural: Estas ferramentas geralmente aplicam técnicas como filtragem de dados e marcação léxica e semântica para construir relações entre elementos de frases e sentenças para derivar regras de

extração. Tais regras são baseadas em restrições sintáticas e semânticas que ajudam a identificar informações relevantes dentro de documentos. Algumas ferramentas que utilizam essa abordagem são: RAPIER (CALIFF; MOONEY, 1997) e SRV (FREITAG, 2000);

4. Ferramentas para indução de *wrappers*: Ferramentas neste grupo geram regras de extração baseadas em delimitadores derivados de um determinado conjunto de exemplos de treinamento. A principal distinção entre essas ferramentas e as baseadas em Processamento de Linguagem Natural é que elas não dependem de restrições linguísticas, mas sim em características de formatação que implicitamente delineiam a estrutura dos pedaços de dados encontrados. Ferramentas como WIEN (KUSHMERICK, 2000) e SoftMealy (HSU; DUNG, 1998) são representativos dessa abordagem;

5. Ferramentas baseadas em modelagem: Ferramentas nesta categoria recebem como entrada certas estruturas, como listas e tabelas, e tentam localizar nas páginas *Web* porções de dados que implicitamente se acomodam a essas estruturas. Algumas ferramentas que adotam essa abordagem são: NoDoSE (ADELBERG, 1998) e DEByE (LAENDER; RIBEIRO-NETO; DA SILVA, 2001);

6. Ferramentas baseadas em Ontologias: Diferente das abordagens apresentadas acima, que dependem da estrutura de apresentação dos dados para gerar regras de extração, as ferramentas deste grupo dependem diretamente dos dados das páginas *Web* e utilizam ontologias para localizar constantes nessas páginas e criar objetos a partir dessas constantes. A ferramenta mais representativa dessa abordagem é a ferramenta desenvolvida pelo Grupo de Extração de Dados da Universidade Brigham Young (EMBLEY et al., 1999).

2.2 Web Crawler

"Um *web crawler*, também conhecido como *spider* ou robô, é um programa que baixa automaticamente páginas da *Web*" (LIU, 2011, p. 311, tradução nossa). Ele pode ser utilizado para vários fins, como por exemplo, para indexar páginas *web* para motores de busca, como o da Google, para fazer o arquivamento da *Web*, e para realizar *data mining* (OLSTON; NAJORK, 2010).

O algoritmo básico de um *web crawler* é bem simples: dada uma lista de URLs iniciais, chamadas de sementes, o *crawler* faz o *download* de todas as páginas *web* endereçadas por essas URLs iniciais, extrai os *hyperlinks* contidos nessas páginas, e interativamente faz o download das páginas web apontadas por esses *hyperlinks* (OLSTON; NAJORK, 2010).

Muitas vezes pode-se não querer fazer a navegação de toda a *Web*, focando-se apenas em acessar páginas de certas categorias ou tópicos. Um *web crawler* focado tenta direcionar o *crawler* para páginas de certas categorias interessantes para o usuário (LIU, 2011, p. 327). Este trabalho implementa um *web crawler* focado que busca na *Web* páginas que possuem artigos onlines publicados, e faz a extração dos mesmos.

2.3 O algoritmo de Ratcliff-Obershelp

O algoritmo de Ratcliff-Obershelp, também conhecido como *Gestalt Pattern Matching*, foi desenvolvido John W. Ratcliff e John A. Obershelp em 1983 e publicado em julho de 1988 no Dr. Dobb's Journal. Este, é um dos algoritmos existentes para determinar a similaridade entre duas strings, ou seja, dois conteúdos textuais. O cálculo de similaridade é definido através da seguinte equação:

$$Dro = \frac{2Km}{|S1| + |S2|} \tag{2.1}$$

Na equação, S1 representa o tamanho da primeira string, S2 o tamanho da segunda string e Km é a variável de caracteres correspondente. Esta última, é definida como o tamanho da substring comum mais longa entre os dois textos, somado recursivamente com o número de caracteres correspondente fora desta região, tanto para a esquerda quanto para a direita.

O resultado desta equação, representado por Dro, é o valor de similaridade entre duas cadeias de caracteres, onde assume valores entre 0 e 1:

$$0 \leq Dro \leq 1 \tag{2.2}$$

O valor 1, representa uma correspondência completa, ou seja, 100% de similaridade. Já 0, significa que não há correspondência, nem mesmo de um único carácter.

2.3.1 Exemplo

Supondo S1 = Pennsylvania e S2 = Pencilvaneya, o passo a passo abaixo exemplifica o cálculo que representa a similaridade entre as duas palavras:

- A maior sequência de carácter entre as duas palavras é a substring "lvan", de tamanho 4. Portanto, o valor de Km inicial é 2 x 4 = 8;
- Continuando com os dois pares restantes das palavras, tanto da parte esquerda quanto da parte direita, temos $\{Pennsy, Penci\}$ à esquerda e $\{ia, eya\}$ à direita;

- Partindo do conjunto da esquerda, $\{Pennsy, Penci\}$, a maior sequência de caracteres é a substring "Pen". Então, incrementa-se o valor de Km para 8 + (2 x 3) = 14. As porções restantes aqui, são $\{nsy, ci\}$ que não possui nem um carácter sequer em comum, para esta parte então, finaliza-se.

- Agora para o lado direito das palavras, temos $\{ia, eya\}$. Pode-se perceber que a única sequência correspondente entre as duas substrings é o carácter "a". Então, incrementa-se Km para 14 + (2 x 1) = 16 e ignora-se os carácteres restantes.

- Aplicando na fórmula, temos

$$Dro = \frac{2Km}{|S1| + |S2|} = \frac{2 * (4 + 3 + 1)}{|12| + |12|} = \frac{16}{24} = \frac{2}{3} = 0,66 \tag{2.3}$$

Portanto, o valor de similaridade entre as palavras Pennsylvania e Pencilvaneya é de 0,66, ou seja, 66%.

2.3.2 Complexidade do algoritmo

A complexidade do algoritmo é $O(n^2)$ para o pior caso e de $O(n)$ para o melhor caso.

3 Trabalhos Relacionados

A seguir, são descritos alguns trabalhos muito interessantes que serviram como inspiração e referência para o desenvolvimento deste.

3.1 Extração automática de sites de notícias

A proposta abordada em (REIS et al., 2004) para fazer a extração automática de notícias contidas em sites da *Web* utiliza como base o algoritmo de Distância de Edição entre Árvores. A lógica por trás deste algoritmo é encontrar um conjunto mínimo de operações que seja capaz de realizar a transformação de uma árvore X em uma outra árvore Y. Esta ideia levou os autores a criar o seu próprio algoritmo, chamado de RTDM, que é então usado para identificar passagens de textos relevantes contendo notícias e seus componentes, extraí-los e descartar elementos que não fazem parte da notícia em si, como por exemplo, *banners*, links, etc. A extração das notícias é feita em quatro passos:

i Agrupamento de páginas: é passado como entrada um conjunto de páginas de treinamento e utiliza um algoritmo de agrupamento hierárquico (*hierarchical clustering*) para gerar os *clusters* de páginas que compartilham características comuns de formatação. A medida de distância utilizada pelo algoritmo de *clustering* é o valor gerado pelo algoritmo RTDM;

ii Geração de padrões de extração: é feito uma generalização dos *clusters* de páginas a partir do passo anterior em padrões, que os autores chamam de node *extraction pattern* (ne-pattern), que são expressões regulares para árvores.

iii Correspondência de dados: é feito uma comparação entre os ne-patterns gerados no passo anterior e as árvores das páginas sendo visitadas pelo *crawler* na tentativa de encontrar o ne-pattern mais apropriado para fazer a extração de dados de cada página.

iv Rotulação dos dados: A saída do passo anterior é um conjunto de passagens de texto ordenadas. O objetivo aqui é tentar encontrar nesse conjunto as passagens que correspondem ao título e ao corpo da notícia. Para isso são aplicadas algumas heurísticas sobre as passagens desse conjunto, como: o corpo da notícia é a maior passagem com mais de 100 palavras e o título da notícia é a passagem que possui entre 1 ou 20 palavras e é a mais próxima do corpo da notícia.

3.2 DEByE

DEByE - Data Extraction By Example - (LAENDER; RIBEIRO-NETO; DA SILVA, 2001), apresenta uma abordagem para extrair informações de fontes da *Web*, baseado em um pequeno conjunto de exemplos especificados pelo próprio usuário, onde ele especifica os exemplos de acordo com uma estrutura de sua preferência. Esta estrutura é descrita no tempo de especificação do exemplo, para isso, o usuário interage com uma ferramenta gráfica que adota tabelas aninhadas como seu paradigma visual. As tabelas são simples, intuitivas e permitem proteger o usuário de detalhes técnicos (como tags HTML, operadores e autômatos de aprendizado) relacionados ao problema de extração. Os exemplos fornecidos pelo usuário são então usados para gerar padrões que permitem extrair dados de novos documentos. Para a extração, foi adotado um procedimento *bottom-up* que se mostrou eficaz com várias fontes da *Web*.

No DEByE, os exemplos de objetos especificados pelo usuário são usados para gerar o que é chamado de *Object Extraction Patterns* (oe-patterns). Esses padrões oe-patterns combinam informações estruturais e textuais que são usados para reconhecer e extrair novos objetos.

3.3 qFex

qFex: um *crawler* para busca e extração de questionários de pesquisa em documentos HTML - (Mathias, 2017), teve como objetivo a extração de questionários de pesquisa presentes nas mais variadas páginas *webs*. A aplicação é dividida em dois grandes componentes: o *crawler* e o extrator. Ambos, recebem como entrada um arquivo de configuração que possui as configurações do banco de dados, biblioteca de *crawler*, das URLs sementes para a busca e extração de valores para os parâmetros utilizados.

A função do *crawler* é varrer a *web*, utilizando como ponto de partida as URLs de entrada fornecidas no arquivo de configuração, a fim de buscar questionários que possuem características específicas e salvar os links na base de dados.

Já o extrator, tem como função buscar na base as URLs, e possivelmente também o arquivo de configuração, e extrair os dados dos questionários para um outro banco de dados.

Para realizar a detecção e a extração dos questionários, o autor aplicou 6 heurísticas para diferenciar os questionários de qualquer outro tipo de formulário na *web*. São elas:

i A descrição de uma pergunta normalmente começa com um número ou uma palavra qualquer iniciando com letra maiúscula, possuindo pelo menos quatro caracteres, um espaço e terminando com o carácter ‘:’, ‘?’ ou ‘.’;

ii Perguntas normalmente possuem suas descrições e componentes de formulário próximos uns dos outros;

iii É possível eliminar certos formulários/campos que não pertencem a um questionário, verificando a distância entre os componentes e certas palavras/frases que são normalmente encontradas nos mesmos;

iv Páginas com questionários normalmente possuem certas palavras comuns em seu título e/ou em seu corpo;

v É possível determinar a presença de um questionário verificando a quantidade de clusters (agrupamento de nodos) que possuam pelo menos um componente de formulário ou que possuam uma certa quantidade de componentes de formulário;

vi Todas as formas de perguntas seguem um ou mais padrões diferentes em suas estruturações nas páginas HTML e é possível utilizar tais padrões para fazer a extração das mesmas.

3.4 Comparativo

Os dois primeiros trabalhos citados apresentam a ideia de uma coleta automática para páginas *web*. Mas, o primeiro está focado apenas na extração automática de notícias, seguindo diversas heurísticas, como por exemplo número de caracteres em um determinado campo e a proximidade com outro, algo que pode ser bem limitante na extração de conteúdo caso alguma página por qualquer motivo que seja, esteja em um padrão completamente diferente do comum.

O DEByE, tem como objetivo a extração do conteúdo de de forma genérica, e necessita que o usuário que estiver interessado no mapeamento da fonte, tenha algum conhecimento mínimo da estrutura dá página, já que é necessário a criação de tabelas aninhadas dos objetos a serem coletados e, fazer isso não é trivial para um usuário comum.

Já o qFex, busca a extração de questionários de pesquisa, onde foi adotado heurísticas para a identificação e extração dos mesmos, utilizando-se de uma arquitetura básica com banco de dados, *crawler* e extrator.

A proposta do CrawlEX, é ser o mais básico e simples possível para o usuário fazer extração apenas de artigos. Os exemplos a serem passados são completamente textuais, sem que haja a necessidade de inspecionar qualquer documento HTML que seja. Também, é possível personalizar um *crawler* customizado do agrado do usuário e selecionar palavras ou frases específicas a serem monitoradas pelo mesmo. Tudo isso, deve ser feito através de uma interface gráfica funcional e bem intuitiva, possibilitando qualquer pessoa sem

nenhum conhecimento de programação extrair conteúdos de artigos disponíveis nos mais variados *websites*.

Característica	Extração automática de sites de notícias	DEByE	qFex	crawlEX
Item alvo	Notícias	Conteúdos em geral	Questionários de pesquisa	Artigos em portais
Principais objetivos	Identificação e extração dos dados	Identificação e extração dos dados	Identificação e extração dos dados	Identificação e extração dos dados
Técnica	Tree Edit Distance e Heurísticas	Object Extraction Patterns	Heurísticas e Numeração Dewey	Comparação textual
Algoritmo utilizado	Do autor	Do autor	Do autor	Do autor + Ratcliff-Obershelp
Interface gráfica	Não	Sim	Não	Sim

Figura 1 – Comparação entre o trabalho desenvolvido e os trabalhos relacionados.

4 CrawlEX

Neste capítulo, é apresentada a ferramenta desenvolvida durante o trabalho, que auxiliará usuários leigos a configurarem suas próprias coletas de artigos que estão publicados na internet.

4.1 Interface Gráfica

A fim da aplicação ser de fácil uso aos usuários, fez-se necessário a criação de um projeto para a interface gráfica da aplicação. Para o desenvolvimento foi utilizada a linguagem de programação Python em sua versão 3.6.9, com a conhecida biblioteca Qt, neste caso, o pacote `<pyQt5>` para auxiliar no desenvolvimento. Nas subseções, será listada cada tela e descrito suas respectivas funcionalidades.

4.1.1 Menu CrawlEX

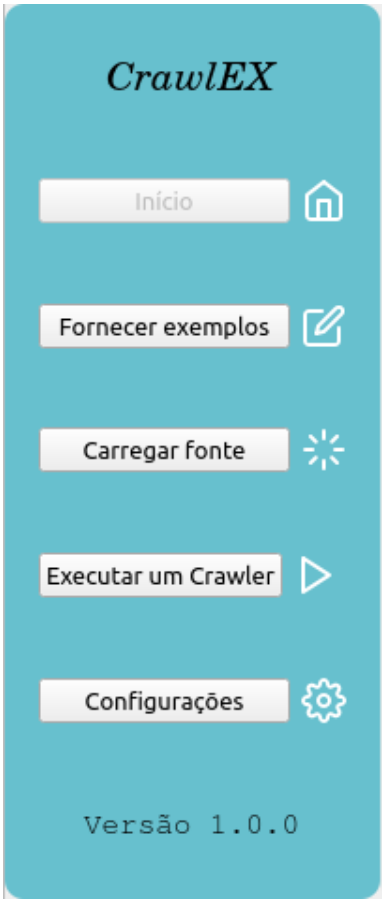

Figura 2 – Captura do menu da aplicação.

A figura 2 apresenta o menu da aplicação CrawlEX. Em todas as telas, este menu sempre estará presente, e sempre na mesma posição: canto esquerdo.

Aqui, temos 5 botões: Início, Fornecer exemplos, Carregar fonte, Executar um Crawler e Configurações. Todos eles são de ação simples (um único clique) e, quando acionados, têm como função alterar a visão disponível no restante da tela. Quando pressionado, o botão ficará ofuscado, para que o usuário saiba qual visão está selecionada, facilitando a compreensão do usuário quanto à aplicação. Nas subseções seguintes, poderá ser visto o botão correspondente à visão apresentada com o ofuscamento citado.

4.1.2 Início

A primeira visão do usuário ao iniciar a aplicação é a tela de início, conforme figura 3. Esta, é uma tela apenas informativa, onde é dado boas vindas ao usuário e descreve de forma objetiva a funcionalidade de cada visão do software.

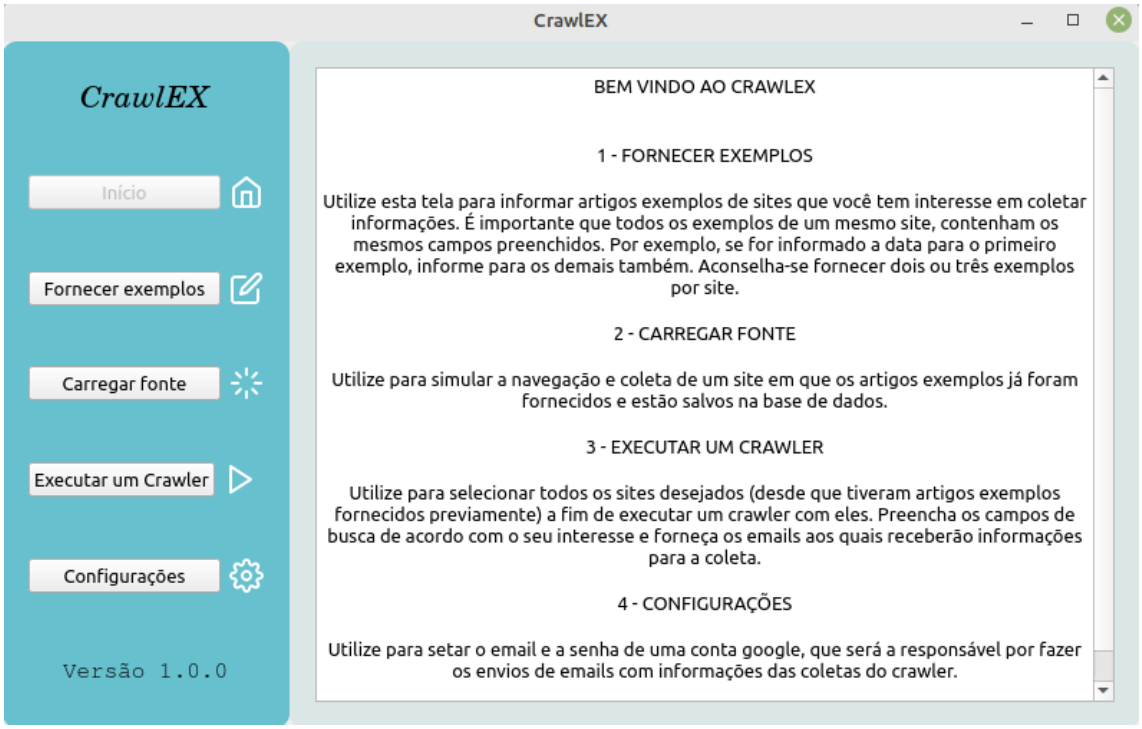

Figura 3 – Captura da tela de início da aplicação, onde instrui de forma básica o funcionamento da aplicação.

4.1.3 Fornecer exemplos

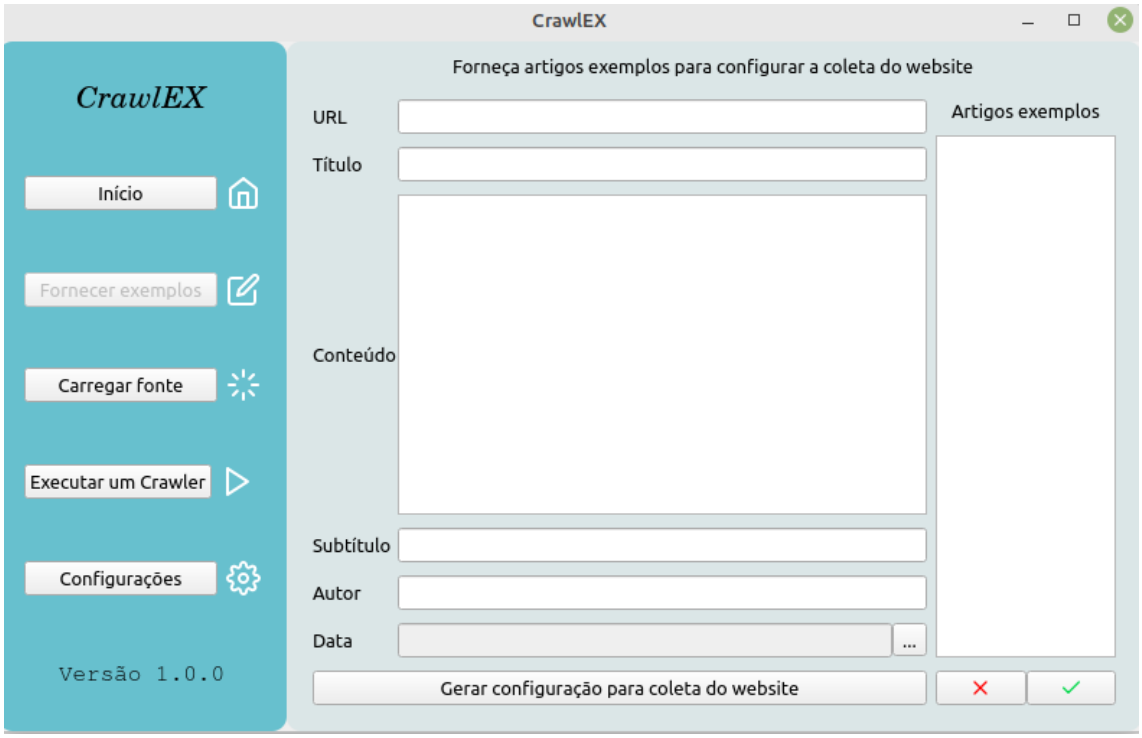

Figura 4 – Captura da tela onde os artigos exemplos serão fornecidos.

Como citado na subseção 4.1.1, pode-se ver o botão “Fornecer exemplos” no menu da figura 4 ofuscado, o que indica que a visão apresentada é aquela em que os exemplos dos artigos serão fornecidos pelo usuário.

Temos seis campos onde é possível a inserção de dados a respeito de um artigo. São eles: URL, título, conteúdo, subtítulo, autor e a data de publicação. Sendo que URL, título e conteúdo são campos obrigatórios, caso contrário, os dados informados não serão validados, e o artigo exemplo, não será adicionado.

Após ao menos os campos obrigatórios estarem preenchidos, o usuário poderá clicar no botão de confirmar, que está localizado no canto inferior direito da tela, onde o ícone está na cor verde. Feito isso, o artigo será adicionado no retângulo branco abaixo da escrita "Artigos exemplos", que contará com uma lista de todas as URLs dos artigos que já foram fornecidos como exemplo.

Para que seja possível visualizar todos os campos de um artigo exemplo que já foi adicionado na lista, basta clicar duas vezes na URL desejada, e uma tela contendo todos os dados será apresentada, como a figura 5 mostra:

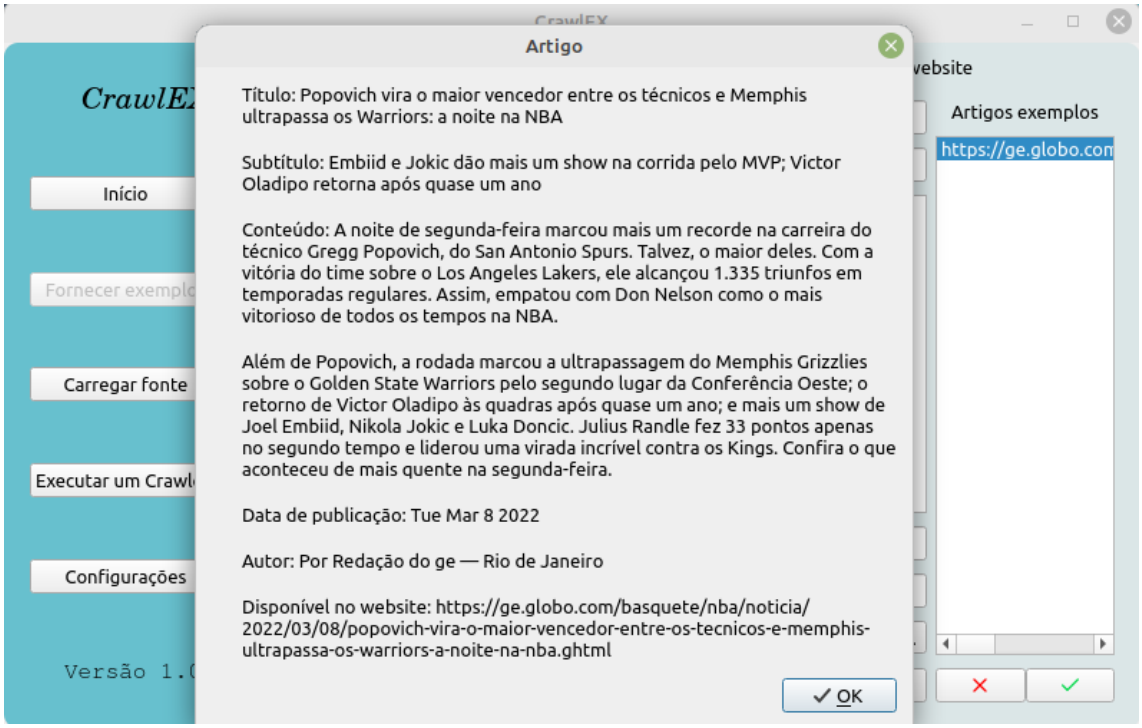

Figura 5 – Tela com todos os dados de um artigo adicionado como exemplo. Lista de "Artigos exemplos"mostrando a URL do único artigo adicionado até então.

Caso o usuário queira remover da lista de exemplos um artigo que adicionou, por qualquer motivo que seja, basta selecionar a URL do artigo em questão e então, clicar no botão que possui o ícone X com a cor vermelha.

Quando todos os artigos exemplos tiverem sido adicionados à lista na direita da tela, o usuário deve clicar no botão "Gerar configuração para coleta do website". Esse botão invocará um algoritmo que receberá como entrada os dados de todos os artigos, e para cada um deles, fará um requisição HTTP para obter o conteúdo HTML da página.

Feito isso, o algoritmo deve trabalhar com os artigos exemplos fornecidos, buscando gerar uma configuração de extração, que é tratada com mais detalhes na seção 4.3.4

4.1.4 Carregar fonte

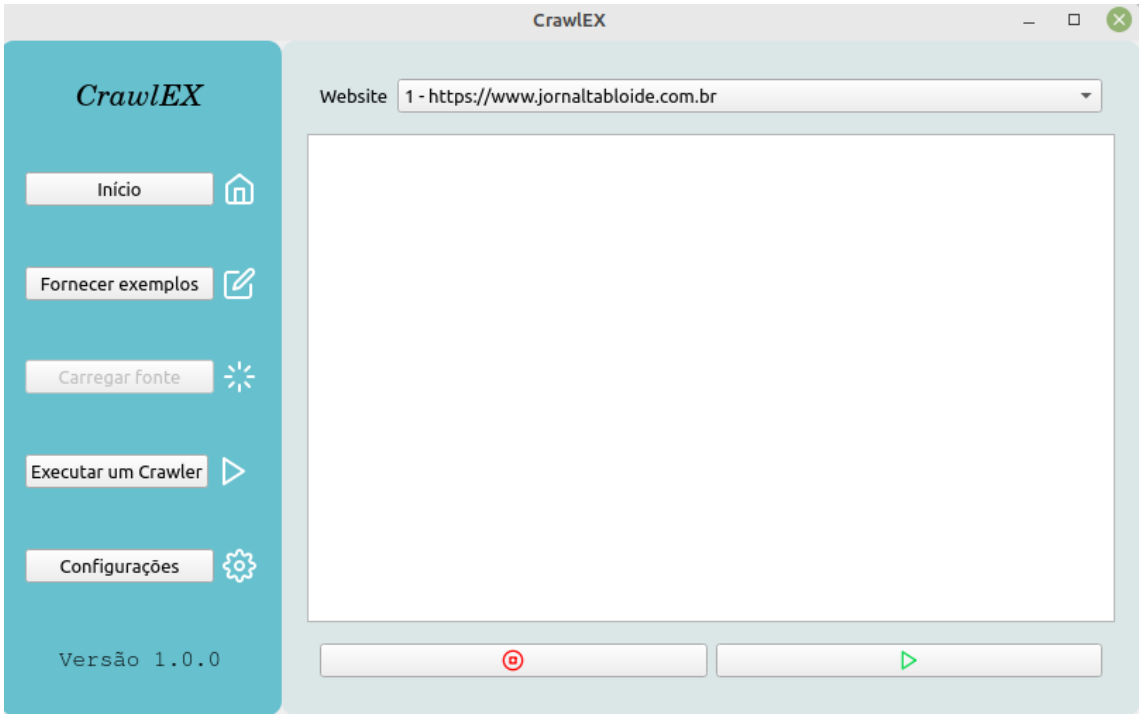

Figura 6 – Captura da tela onde é possível selecionar um website já configurado e verificar os dados coletados.

Os sites que já possuírem configurações de coleta definidas podem ser testados na tela apresentada na figura 6. Todos esses sites, estarão disponíveis para seleção ao lado do *label* "Website". Na captura apresentada, o *website* já configurado que está selecionado para teste é do Jornal Tabloide.

Para inicializar a coleta e fazer a verificação dos dados que estão sendo extraídos, basta clicar no botão de *play*, que está com um ícone na cor verde. Feito isso, a coleta será inicializada e os dados inseridos no grande retângulo de fundo branco.

A navegação pelo site acontecerá pelo período fixo de 5 minutos ou, caso o usuário desejar interromper precocemente, basta fazer isso clicando no botão de *stop*, identificado com o ícone vermelho.

A figura 7 apresenta conteúdos extraídos do teste para o Jornal Tabloide.

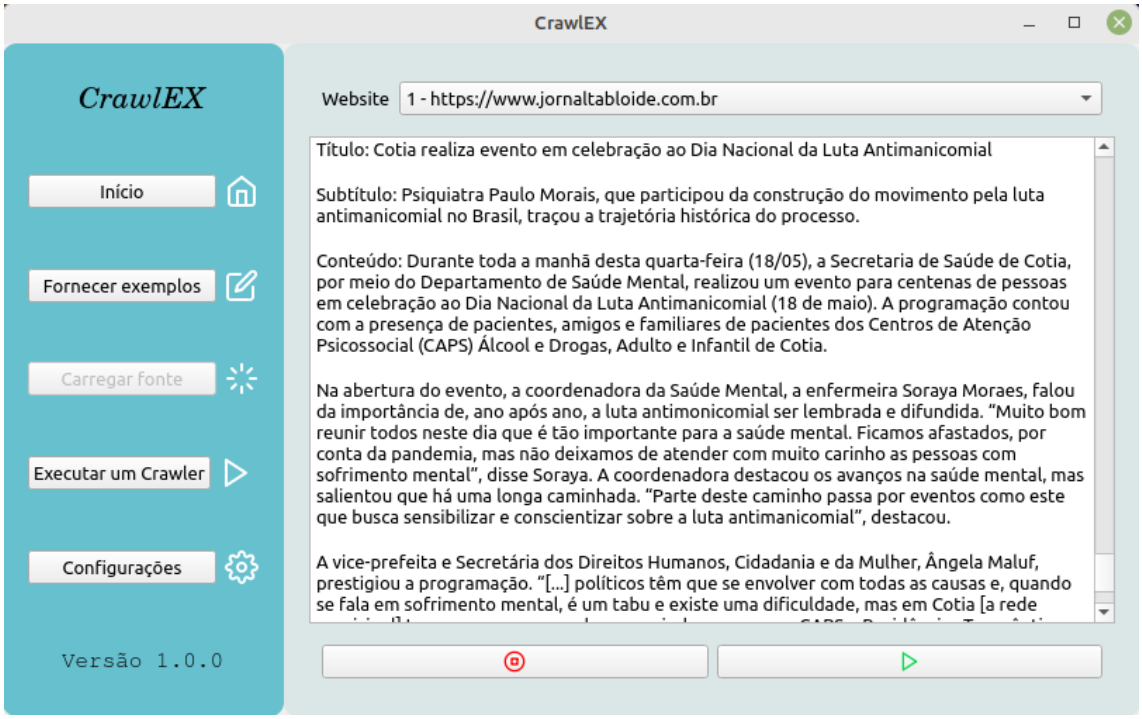

Figura 7 – Exemplo de informações coletadas e mostradas ao usuário após execução do teste da coleta.

4.1.5 Executar um crawler

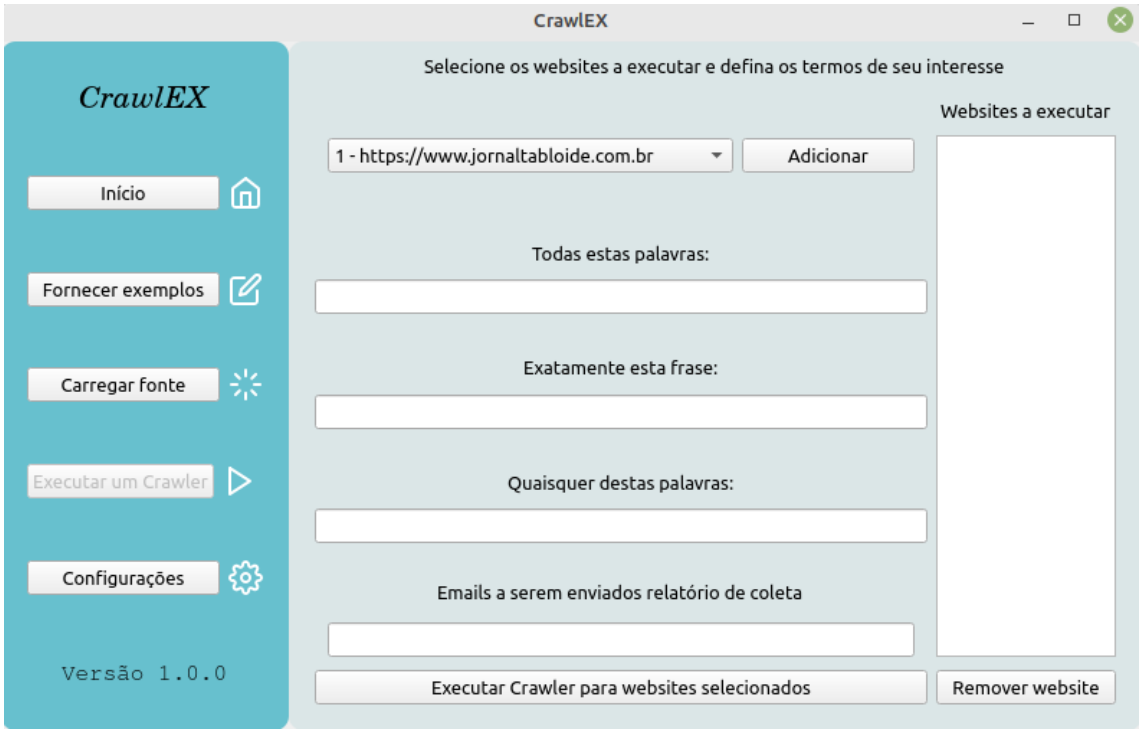

Figura 8 – Captura da tela de configuração para execução de um novo crawler.

A figura 8, apresenta a tela onde é possível que o usuário faça a seleção de todas as páginas *web* que desejar, desde que a mesma já tenha tido sua coleta previamente

configurada através do fornecimento de exemplos, como vimos.

A página que está selecionada é o Jornal Tabloide. Para que esta página seja navegada pelo *crawler* a ser configurado, é necessário que o usuário clique no botão "Adicionar", ao lado. Feito isso, esta página será incluída no grande retângulo branco na direita da tela, logo abaixo do label "Websites a executar". Pode-se adicionar a quantidade de páginas que o usuário quiser. Caso uma página erroneamente for adicionada na lista para ser visitada pelo *crawler*, basta selecionar a mesma e então, clicar no botão "Remover website".

Também, ainda na configuração de um novo *crawler*, o usuário é capaz de digitar palavras e frases do seu interesse que possam se fazer presente nos artigos que serão navegados por esta página, sendo assim, qualquer artigo que não satisfaça as condições expressas pelo usuário, serão completamente ignorados e portanto, não coletados.

No campo de digitação de "Todas estas palavras:", deverão ser digitadas todas as palavras que devem estar presentes no artigo, independente da frase a qual ela está inserida. O artigo somente será coletado, se conter absolutamente todas as palavras que foram digitadas, e não somente algumas. Cada palavra digitada deve estar separada por um simples carácter de espaço entre a próxima.

Já em "Exatamente esta frase", deve ser digitado uma frase específica em que o usuário tenha interesse e que esteja presente no artigo. Também, o artigo só será coletado se possuir exatamente a frase digitada.

Em "Quaisquer destas palavras", assim como em "Todas estas palavras", o usuário deve digitar as palavras de interesse separadas pelo carácter de espaço. A diferença aqui, se dá que o artigo será coletado se conter qualquer uma das palavras, apenas uma já é o suficiente.

Para que o robô faça o envio dos artigos coletados que cumpram com os termos especificados, no último campo a ser preenchido deve-se colocar todos os *e-mails* aos quais se deseja enviar o mesmo, espaçados por um carácter de espaço simples.

Por último, após adicionado todas as páginas *web* de interesse, e configurado todos os termos de busca, basta clicar no botão "Executar Crawler para *website* selecionados"e então, um novo *crawler* totalmente configurado deve estar pronto para a execução.

4.1.6 Configurações

Figura 9 – Captura da tela de configuração, onde um e-mail Google e a senha devem ser preenchidos.

Todos os *crawlers* que estiverem configurados, farão o envio dos relatórios através do *e-mail* que estiver configurado na tela apresentada na figura 9. Para isso, basta selecionar o *checkbox* "Editar", fornecer o *e-mail* e a senha, e confirmar clicando no botão "Salvar email e senha". Feito isso, o *checkbox* deverá ficar sem a seleção, e os campos de *e-mail* e senha deverão ficar desabilitados, o que indica que houve o salvamento das credenciais da maneira correta.

Para que a aplicação funcione corretamente, o *e-mail* informado aqui, deverá obrigatoriamente ser um *e-mail* Google válido.

4.2 Biblioteca BeautifulSoup

BeautifulSoup é uma biblioteca Python de extração de dados de arquivos HTML e XML. Ela funciona com alguns interpretadores (*parsers*) a fim de prover maneiras mais intuitivas de navegar, buscar e modificar uma árvore de análise (*parse tree*).

Parser	Uso Padrão	Vantagens	Desvantagens
html.parser (puro)	`BeautifulSoup(markup, "html.parser")`	• Baterias inclusas • Velocidade Decente • Leniente (Python 2.7.3 e 3.2.)	• Não tão rápido quanto lxml, menos leniente que html5lib.
HTML (lxml)	`BeautifulSoup(markup, "lxml")`	• Muito rápido • Leniente	• Dependencia externa de C
XML (lxml)	`BeautifulSoup(markup, "lxml-xml")` `BeautifulSoup(markup, "xml")`	• Muito rápido • O único parser XML atualmente suportado	• Dependência externa de C
html5lib	`BeautifulSoup(markup, "html5lib")`	• Extremamente leniente • Analisa as páginas da mesma maneira que o navegador o faz • Cria HTML5 válidos	• Muito lento • Dependência externa de Python

Figura 10 – Comparação entre os interpretadores de documentos utilizáveis com BeautifulSoup.

O interpretador escolhido pelo autor para o desenvolvimento da aplicação, foi html5lib (apresentado na figura 10), por fazer a análise da mesma forma que um navegador faz e também for lidar com os documentos como sendo HTML5, garantindo um padrão para todos.

A biblioteca, transforma um documento HTML complexo em uma árvore de objetos Python. Mas, o usuário tem que lidar apenas com quatro tipos de objetos: *BeautifulSoup*, *Tag*, *NavigableString* e *Comment*. Para este trabalho, foi manipulado diretamente apenas dois dos quatro tipos de objetos: *BeautifulSoup* e *Tag*

4.2.1 Objeto BeautifulSoup

```
from bs4 import BeautifulSoup
import html5lib

soup = BeautifulSoup('<p class="tcc">Trabalho de conclusao</p>', "html5lib")
```

Código 1 – Criação do objeto BeautifulSoup
Fonte: Elaborado pelo autor

No código 1, é criado um objeto BeautifulSoup passando uma tag HTML como string, utilizando o interpretador html5lib. Automaticamente, o interpretador criará um conteúdo HTML5 com a seguinte estrutura básica apresentada no código 2:

```
<html>
 <head>
 </head>
 <body>
  <p class="tcc">
   Trabalho de conclusao de curso
  </p>
 </body>
</html>
```

Código 2 – HTML criado a partir do código 1
Fonte: Elaborado pelo autor

Ainda, um objeto BeautifulSoup, possui diversos métodos. Mas dois deles são os principais, e foram usados frequentemente no desenvolvimento da aplicação: find_all e find.

O método find_all aceita diversos tipo de parâmetros e tem como objetivo buscar todas as tags HTML que possui alguma característica de interesse do usuário da biblioteca, desde que ele passe essa informação corretamente como parâmetro. O código 3 exemplifica uma situação onde o usuário busca na árvore de análise todas as tags do tipo *li*, ou seja, todas as tags que listam itens em um documento HTML.

```
soup.find_all('li')
```

Código 3 – Buscando todos os elementos *li* em um documento HTML
Fonte: Elaborado pelo autor

O método find apresentado no código 4 é similar ao método find_all. A diferença é que find_all irá retornar todas as tags HTML com uma determinada característica, já o método find retornará apenas a primeira ocorrência da tag. Este método é extremamente útil e relevante para o contexto do desenvolvimento do trabalho, pois, dado uma vez em que se tem certeza da unicidade de uma tag com determinada característica, podemos buscá-la de forma rápida, certeira e objetiva.

```
soup.find('li')
```

Código 4 – Buscando primeiro elemento *li* em um documento HTML
Fonte: Elaborado pelo autor

4.2.2 Objeto Tag

Uma tag HTML é muito simples. Vejamos por exemplo a tag no código 5. O nome desta tag é div e possui um único atributo, type, onde o valor é article. E o texto desta

tag é "foo!".

```
<div type="article">foo!</div>
```

Código 5 – Tag html de nome div e atributo type com valor article
Fonte: Elaborado pelo autor

Um objeto tag da biblioteca BeautifulSoup, pode ter seu nome, atributos e texto da tag, acessados. O código 6 mostra isso.

```
from bs4 import BeautifulSoup
import html5lib

soup = BeautifulSoup('<div type="article">foo!</div>', "html5lib")
tag = tag.div
tag.name #'div'
tag.attrs #{'type': 'article'}
tag.text #'foo!'
```

Código 6 – Recuperando nome, atributo e texto de uma tag HTML
Fonte: Elaborado pelo autor

Esses atributos, são de grande ajuda para buscar uma tag muito especifica dentro de um documento HTML, pois, podemos usá-los como filtros na busca de uma tag dentro de um objeto BeautifulSoup, conforme o código 7.

```
soup.find('div', {'type': 'article'})
```

Código 7 – Buscando uma tag em um documento HTML, dado seu nome e seus atributos
Fonte: Elaborado pelo autor

Uma tag HTML pode conter diversas outras tags dentro dela, formando uma hierarquia. O BeautifulSoup possui diversas funcionalidades para encontrar uma tag filha dentro de uma tag pai (ou vice versa), encontrar tag irmã, entre outros. Mas uma funcionalidade extremamente importante e usada incessantemente neste trabalho, é a capacidade de desvincular e remover uma tag que esteja dentro de outra. Isso pode ser feito utilizando o metódo *decompose*, apresentado no código 8.

```
from bs4 import BeautifulSoup
import html5lib

example = '<a href="http://example.com/">Site exemplo:  <i>example.com</i></a>'
soup = BeautifulSoup(example, "html5lib")
soup.i.decompose() # Remove <i>example.com</i> do objeto soup.
```

Código 8 – Removendo uma tag que está inserida em outra
Fonte: Elaborado pelo autor

Esta funcionalidade, permite com que seja feito a remoção de tags e seus conteúdos textuais dentro de outras, eliminando conteúdos não relevantes, como por exemplo lixos, anúncios, entre outros. A seção 4.3.4 traz em sua arquitetura o componente TagCleaner, criado no desenvolvimento deste trabalho para ser responsável por esta funcionalidade.

4.3 Back-End

A implementação da aplicação foi toda realizada na linguagem Python, na versão 3.6.9. Para o armazenamento das configurações dos *websites* mapeados, bem como o armazenamento temporário dos artigos coletados, foi usado uma biblioteca que implementa um banco de dados SQL embutido, o SQLite. O pacote Python utilizado que corresponde a implementação desta API foi o `<sqlite3>`. Também, a biblioteca `<urllib.parse>` está sendo usada, para a manipulação das URLs navegadas, auxiliando na geração de expressões regulares automaticamente, que servem para que o *crawler* identifique quais URLs deve-se nevegar, bem como quais são possíveis artigos a serem coletados.

A biblioteca usada para fazer as requisições para os sites, a fim de receber o seu conteúdo HTML, foi a requests. E para fazer a manipulação dos documentos requisitados, a biblioteca BeautifulSoup, como mencionado na subseção anterior. Também, uma última biblioteca foi necessário, a html5lib, esta para fazer uso de um interpretador HTML5, facilitando a manipulação dos documentos obtidos.

4.3.1 Base de dados embutida

Utilizando o SQLite, três tabelas foram criadas: Article, Website, ExampleArticle.

A primeira tabela, apresentada na figura 11, armazena dados dos artigos coletados de forma temporária (até o fim da execução de um *crawler*), e foi criada somente para facilitar a manipulação de expressões de busca booleana (utilizadas de acordo com as informações inseridas na tela apresentada na seção 4.1.5), fazendo uso das *queries* do SQL. Sempre que um novo *crawler* for finalizado, todo o conteúdo desta tabela será deletado. Caso algum erro ocorra na execução do *crawler* e ele não seja finalizado da maneira

esperada, por conseguinte não deletando os dados desta tabela, esses dados serão deletados obrigatoriamente no início da próxima execução do mesmo.

A segunda, armazena as URLs sementes de cada *website* que já tenha exemplos fornecidos e salvos, ou seja, as URLs em que o *crawler* iniciará a sua navegação para o *website* em questão.

Já a terceira tabela, armazena os dados dos artigos exemplos informados pelo usuário. Esses dados são utilizados para gerar a configuração de extração dos campos de um artigo na *web*.

A figura 12 apresenta o relacionamento entre as tabelas ExampleArticle e Website.

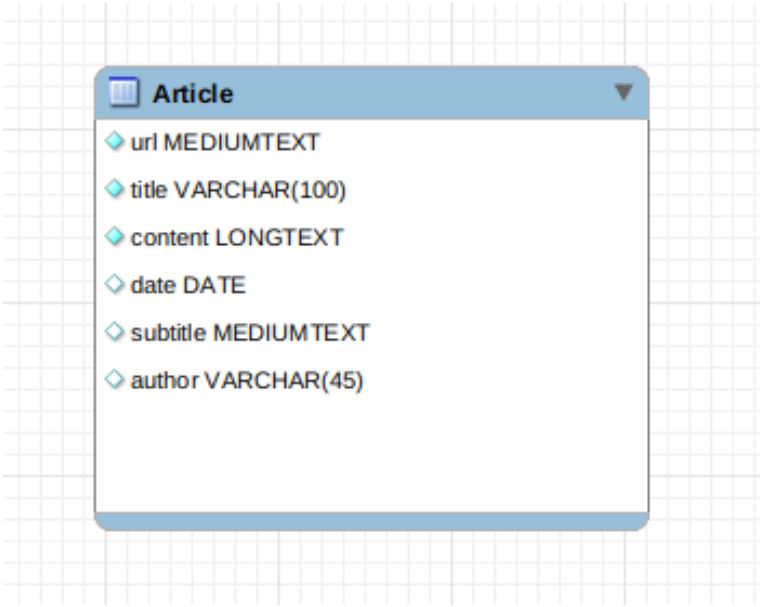

Figura 11 – Tabela Article, responsável por armazenar os artigos coletados temporariamente, e após o fim da execução do crawler, excluí-lo.

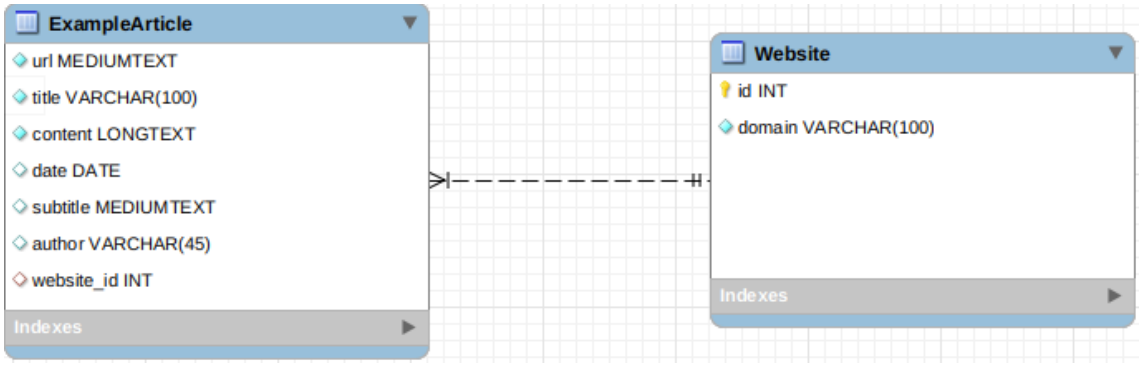

Figura 12 – Tabela ExampleArticle e Website, onde se relacionam através da chave estrangeira website_id, da primeira tabela.

4.3.2 Geração automática de expressões regulares

Após o fornecimento dos artigos exemplos de um *website*, é necessário criar uma expressão regular automaticamente para identificar uma possível URL que contenha um artigo. Para isso, foi desenvolvido um algoritmo, que recebe como entrada todas as URLs dos artigos exemplos fornecidos e retorna uma expressão regular válida. Essa expressão regular, servirá para que o *crawler* decida se deve ou não inspecionar o conteúdo de uma página. Se a expressão regular satisfazer uma determinada URL, então, essa página será analisada.

Em Python, é utilizado algumas sequências especiais de caracteres que identificam um padrão em uma String. A tabela 1, traz aqueles que são usados para a geração automática da expressão regular e seus respectivos significados:

[/]+	\d+	?
quaisquer caracteres até "\"	sequência numérica	anterior

Tabela 1 – Sequências de caracteres que determinam padrões na linguagem Python.

i Para cada URL, separa-se cada sequência de caracteres pelo carácter delimitador "/", e coloca todas sequências em uma lista;

ii Se o tamanho das listas de caracteres de cada URL for igual entre si, seguir para o passo iii, caso contrario, seguir para o passo viii;

iii Para cada posição de cada lista, comparar entre si se a sequência de caracteres é igual;

iv Caso seja igual, adicionar esta sequência na mesma posição da expressão regular;

v Caso seja diferente, mas todas numéricas, adicionar \d+ na mesma posição da expressão regular;

vi Caso não seja igual entre si, nem numéricas, adicionar "[/]+"na mesma posição da expressão regular;

vii adicione "/?"no fim da expressão regular e finalize;

viii Percorra cada lista até o tamanho da menor lista existente;

ix Para cada posição da lista, executar os passos iii, iv, v e vi;

x Abrir parênteses e adicionar a sequência "[/]+/"repetidamente pelo números de vezes da subtração entre a maior lista e a menor;

xi adicionar o carácter "?", fechar parênteses e finalize.

4.3.3 Crawler

O *crawler* foi todo desenvolvido pelo autor do trabalho, utilizando a biblioteca *requests* do Python para fazer as requisições HTTP para os endereços *webs*. A navegação do *crawler* é limitada para páginas *web* estáticas, não sendo eficiente para páginas onde os links internos são gerados dinamicamente pela linguagem de script JavaScript ou algum *framework*.

4.3.3.1 Arquitetura do crawler

A arquitetura e o funcionamento do *crawler* é bem simples, e pode-se abstrair em 3 componentes: Websites, LinkFinder e URLPatternChecker.

Websites: tabela do banco de dados, responsável por armazenar a URL semente de cada site ao qual tenha exemplos fornecidos pelo usuário e salvos na tabela ExampleArticle.

LinkFinder: componente responsável por buscar na base de dados a URL semente de uma fonte e a partir desta, buscar em seu documento HTML apenas URLs que possuem o mesmo domínio. Toda e qualquer URL de domínio diferente será descartada e não será navegada.

URLPatternChecker: componente responsável por identificar se uma URL contém um possível artigo ou não, de acordo com a funcionalidade descrita na seção 4.3.2. Se possuir, a URL é enviada para o extrator, caso contrário, é descartada.

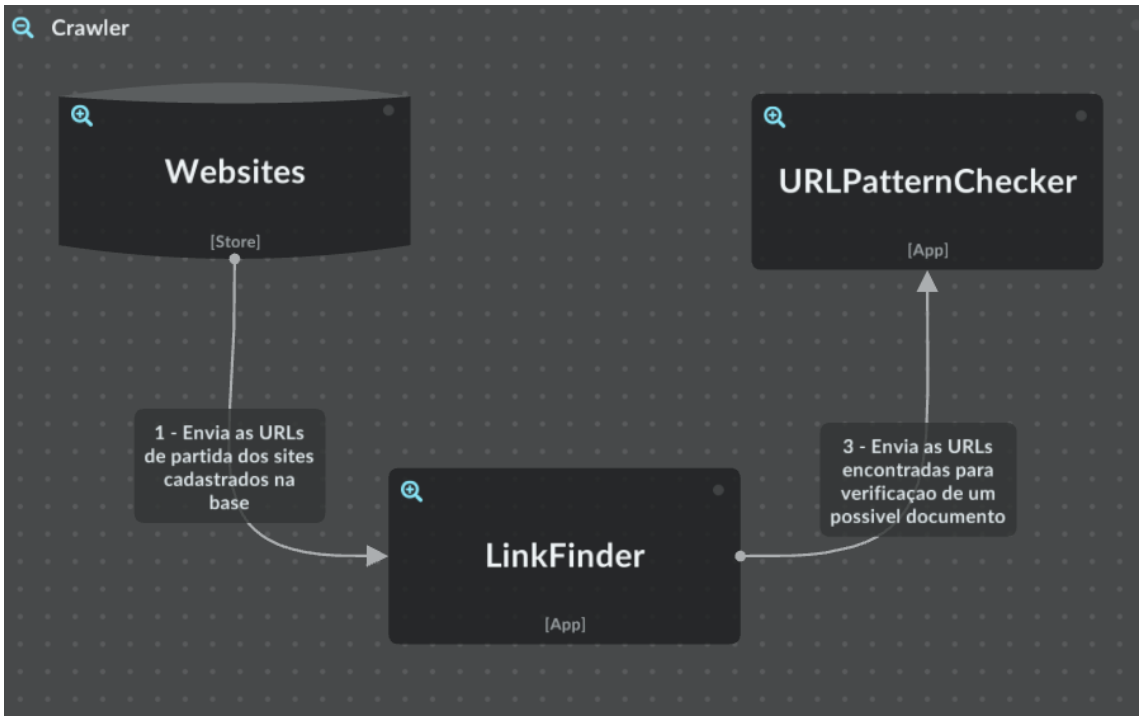

Figura 13 – Abstração da arquitetura do crawler

-

4.3.3.2 Algoritmo de navegação do crawler

O seguinte algoritmo é usado pela aplicação a fim de encontrar novas páginas para navegar. O *crawler* inicia sua navegação através de uma URL de entrada, e a partir desta, busca encontrar outras URLs de mesmo domínio e que ainda não foram visitadas pela navegação.

Algorithm 1 Algoritmo da navegação básica do crawler

```
urlDePartida ← String
urlsAVisitar ← Conjunto
urlsJaVisitadas ← Conjunto
urlsAVisitar ← urlsAVisitar ⋃ urlDePartida
PararNavegacao ← Falso
while PararNavegacao é Falso e urlsAVisitar não é vazia do
    urlsJaVisitadas.adicione(urlDePartida)
    urlsAVisitar ← busqueLinksNoHTMLDaUrlDePartida()
    novaUrl ← urlsAVisitar.pegueUmaUrl()
    if novaUrl ∈ urlsJaVisitadas then
        continue
    else
        Envia a URL para o extrator
    end if
end while
```

4.3.4 Configuração de extração

Para que a extração de informações de um site seja viabilizada, é necessário que se tenha uma configuração de extração única para cada um deles. Essa configuração, é composta por objetos Tags do BeautifulSoup descrito na seção 4.2.2. A tag principal para a extração, será aquela a qual o seu conteúdo textual tenha o maior coeficiente de similaridade com os exemplos fornecidos pelo usuário. Esse coeficiente, é obtido aplicando o algoritmo da seção 2.3.

Durante o desenvolvimento do trabalho, foi notado mudança na estrutura de alguns sites analisados. Embora isso seja raro de acontecer corriqueiramente, vez ou outra um site pode mudar de estrutura por diversos fatores. Por essa razão, foi optado por fazer com que a geração da configuração da extração para um *website* seja feita sempre que o mesmo for executado pelo *crawler*. Essa abordagem faz com que o *crawler* perca um pouco de desempenho inicialmente, pois demanda uma fração do seu tempo gerando a configuração de extração mas, por outro lado, ganha em confiabilidade na extração das informações.

4.3.4.1 Arquitetura do configurador de extração

ExampleArticle: tabela do banco de dados responsável por armazenar os artigos fornecidos como exemplos pelo usuário.

TagFinder: componente responsável por buscar todos as tags HTML dentro do documento de um site.

ContentComparator: componente que tem como responsabilidade receber as tags HTML e seus conteúdos, e aplicar o algoritmo Ratcliff-Obershelp, comparando os conteúdos dessas tags com os conteúdos dos exemplos fornecidos pelo usuário. Por fim, retorna a tag que obteve o maior percentual de similaridade para cada campo de extração.

TagCleaner: após a identificação da melhor tag por parte do ContentComparator para extração de um campo, este componente tem como objetivo identificar possíveis tags filhas desta, que possuem conteúdos que não devem ser extraídos, como por exemplo anúncios no meio de um texto. Esta funcionalidade é aplicada em nível de código de acordo com o código 8 da seção 4.2. Aqui, o algoritmo Ratcliff-Obershelp é aplicado a todo momento, a fim de verificar se a remoção de uma tag resultou em uma maior similaridade ou não. Se resultou, remove-a de fato, caso contrário, mantém.

ExtractionConfigurator: componente central, responsável por designar tarefas aos demais e fazer o controle geral. No fim, retorna uma tag principal para extração e uma lista com tags de limpeza do conteúdo, para cada campo que se deseja extrair.

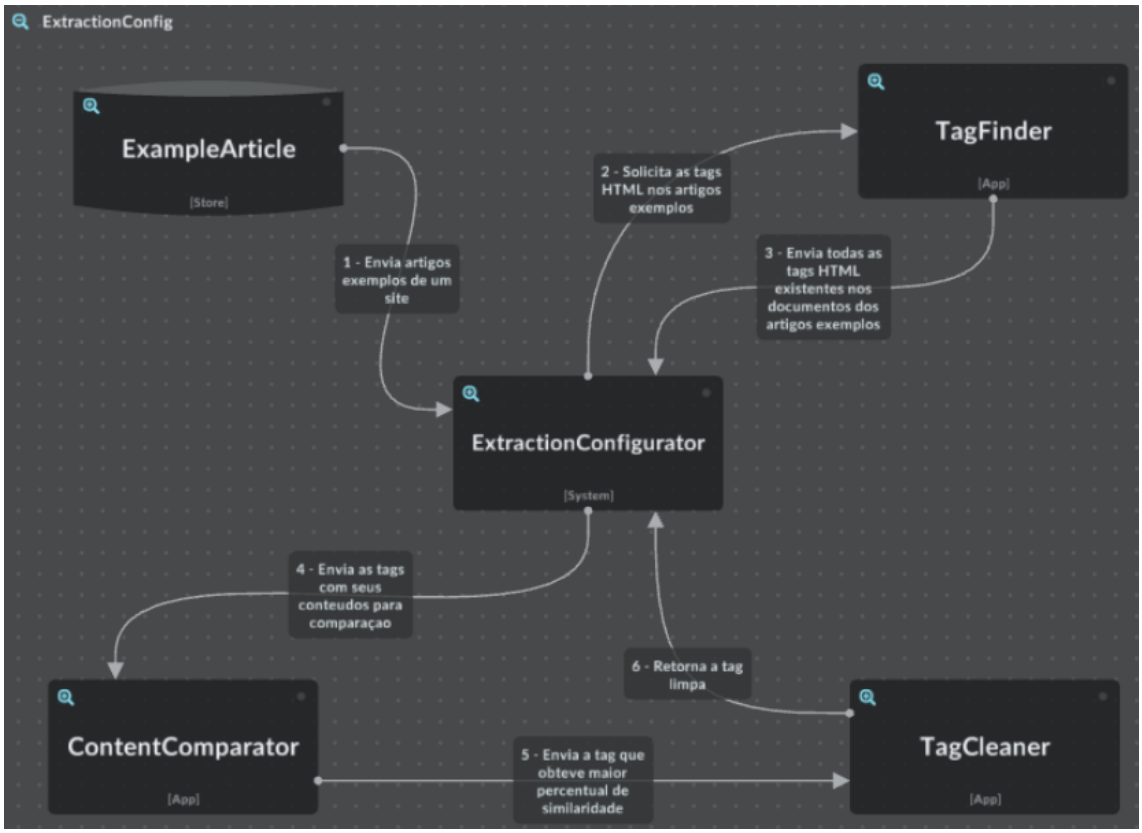

Figura 14 – Abstração da arquitetura do funcionamento para geração da configuração de extração de artigos em um site.

4.3.4.2 Algoritmo do configurador de extração

O algoritmo a seguir é exemplificado para a geração da configuração de extração de título, conforme parâmetro na terceira linha, mas é aplicado para os outros campos também.

Através dos artigos exemplos informados pelo usuário, é buscado dentro do HTML desses artigos a tag que contém o conteúdo esperado. Após encontrar essa tag, é feito a remoção de possíveis tags lixos que estejam dentro desta.

Algorithm 2 Algoritmo que gera a configuração de extração para um website

artigosExemplos ← **busqueArtigosExemplosParaOSite()**
tagsHTML ← *TagFinder*.**busqueTags**(*artigosExemplos*)
titulosArtigos ← **buscarTextoNoCampo**(*"titulo", artigosExemplos*)
melhorTagParaExtracao ← *ContentComparator*.**compare**(*tagsHTML, titulosArtigos*)
tagsLimpadoras ← *TagCleaner*.**busqueTagsLimpadoras**(*melhorTagParaExtracao*)
configuracaoDeExtracao ← *ExtractionConfig(melhorTagParaExtracao, tagsLimpadoras)*
return *configuracaoDeExtracao*

4.3.5 Extrator

O extrator é responsável por fazer a extração propriamente dita para todos os campos possíveis do artigo, e salvá-los na base de dados.

4.3.5.1 Arquitetura do Extrator

ExtractionConfig: resultado do algoritmo descrito em 4.3.4. Onde uma tag principal para extração e uma lista de possíveis tags para remoção de conteúdo lixo, servem como uma das entradas para o extrator, a fim de realizar a extração dos campos de um artigo.

Crawler: componente descrito na seção 4.3.3, que faz o envio das possíveis URLs a terem seu conteúdo extraído pelo extrator.

Extrator: extrator propriamente dito, que recebe uma URL que contém um possível artigo e uma configuração para extração de informações daquele site. Caso a extração ocorra com sucesso, as informações são salvas no banco.

Articles: tabela do banco de dados responsável por armazenar os artigos coletados pelo extrator durante a navegação do *crawler*, de acordo com os temos configurados pelo usuário.

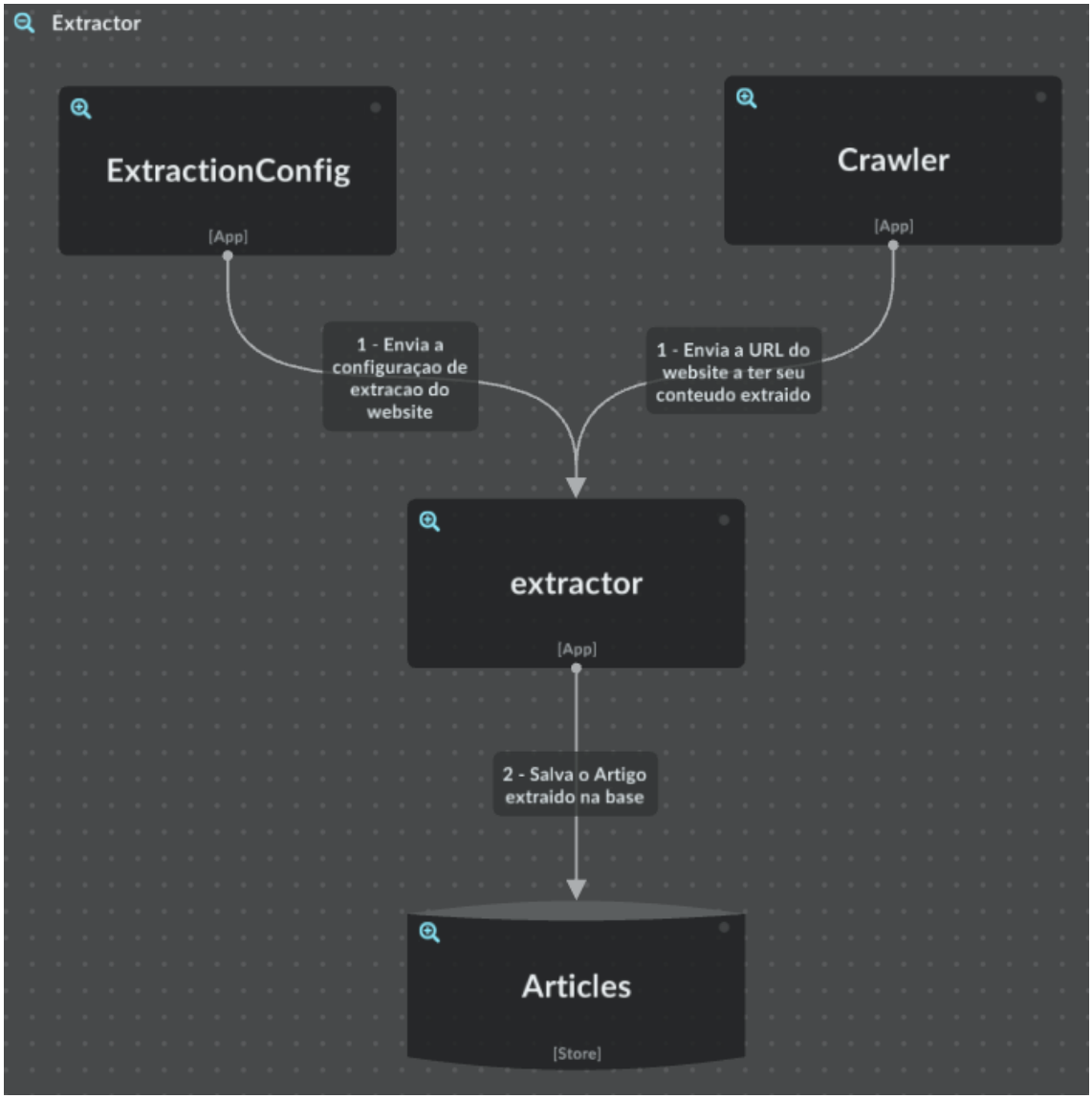

Figura 15 – Abstração da arquitetura do extrator de artigos.

4.3.5.2 Algoritmo de extração das informações

O algoritmo a seguir recebe como entrada uma URL para a extração e a configuração de extração para os conteúdos, e tenta extrair o artigo.

Algorithm 3 Algoritmo para extração dos dados de uma página

artigosExemplos ← **busqueArtigosExemplosParaOSite()**
configuraçãoDeExtracao ← *ExtractionConfigurator*.**gereConfiguração**(*artigosExemplos*)
urlATerConteudoExtraido ← *Crawler*.**fornecaUrl()**
artigoExtraido ← *Extractor*.**extraiaArtigo**(*urlATerConteudoExtraido, configuração-DeExtracao*)
if *artigoExtraido* é **valido then**
    inserirArtigoNaBase(*artigoExtraido*)
end if

5 Experimentos

Neste capítulo são apresentados os experimentos conduzidos com o intuito de avaliar a quantidade e qualidade dos conteúdos extraídos de acordo com os exemplos fornecidos. Primeiramente, descreve o experimento conduzido pelo autor do trabalho e posteriormente, descreve o experimento com a participação de 3 usuários leigos, externos ao desenvolvimento deste trabalho.

5.1 Exemplos fornecidos pelo autor

O autor conduziu 3 experimentos. O primeiro, fornecendo apenas um exemplo de artigo por fonte, o segundo 2 exemplos, e o terceiro 3 exemplos. Os exemplos para os 3 experimentos foram para os mesmos *websites*, onde o *crawler* navegou por 10 minutos em cada um desses sites, totalizando 5 horas de execução para cada um dos experimentos.

O cálculo do percentual de similaridade foi feito de forma manual, uma vez que não temos na base exemplos esperados para todo e qualquer artigo publicado pelo site em questão, e por isso, foi considerado apenas os 10 primeiros artigos coletados para cada um dos *websites*. O cálculo seguiu o algoritmo descrito na seção 2.3. A equação 5.1 é aplicada para cada campo extraído, a fim de obter-se a média de similaridade conforme a equação apresentada em 2.1.

$$\frac{Dro1 + Dro2 + Dro3 + ... + Dro10}{10} * 100 \tag{5.1}$$

5.1.1 1 exemplo

Com 1 único exemplo fornecido por fonte, o *crawler* navegou por 20467 páginas, onde dessas, 13566 sendo possíveis páginas com artigos para ser coletados. Entretanto, ao tentar extrair conteúdo para essas 13566 páginas, a extração ocorreu com sucesso para apenas 588, ou seja, 4,33%. Além disso, a assertividade para os campos foi muito baixa, conforme mostra a tabela 2 e o gráfico da figura 16, demonstrando que com 1 único exemplo é muito difícil encontrar elementos extratores de forma mais generalista. Isso também nos indica que a aplicação identificou como possível URL contendo um artigo muitas daquelas que de fato não o possuíam.

Site	**Título**	**Subtítulo**	**Conteúdo**	**Data**	**Autor**
https://www.jornaltabloide.com.br/	100%	100%	100%	0%	24%
https://extra.globo.com/	100%	-	45%	20%	0%
https://noticias.uol.com.br/	0%	-	40,3%	20%	-
https://www.techtudo.com.br/	100%	10%	30,7%	10%	10%
https://www.terra.com.br/	40%	30%	70,4%	10%	10%
http://www.tribunadonorte.com.br/	80%	-	67,9%	50%	-
https://www.folhape.com.br/	50%	30%	80,7%	40%	20%
https://www.cnnbrasil.com.br/	30%	10%	59,3%	30%	100%
https://www.futebolinterior.com.br/	100%	40%	80,5%	0%	100%
https://www.pmf.sc.gov.br/	80%	-	100%	60%	-
https://www.sc.gov.br/	60%	-	89,8%	40%	-
https://www.gov.br/pt-br/	100%	100%	100%	100%	-
https://noticias.ufsc.br/	100%	-	70,2%	20%	-
https://www.ifsc.edu.br/	30%	-	100%	50%	-
https://www.omelete.com.br/	10%	10%	92,2%	40%	10%
https://www.letras.mus.br/	80%	-	100%	-	20%
https://www.adorocinema.com/	40%	20%	96,6%	0%	10%
https://www.brasil247.com/	100%	100%	82,5%	0%	-
https://www.oeconomista.com.br/	100%	-	99,6%	100%	-
https://www.diariodepernambuco.com.br/	70%	-	67,7%	0%	30%
https://cronicabrasileira.org.br/	100%	-	100%	100%	100%
https://sitedepoesias.com/	100%	-	0%	0%	0%
https://www.torcedores.com/	100%	30%	76,2%	20%	40%
https://cinepop.com.br/	100%	-	96,9%	0%	10%
https://ge.globo.com/	100%	100%	50,2%	0%	0%
https://gamerview.uai.com.br/	100%	30%	65,4%	0%	0%
https://www.ofuxico.com.br/	50%	-	79,3%	70%	20%
https://contobrasileiro.com.br/	100%	-	99,1%	100%	100%
https://www.correiobraziliense.com.br/	10%	10%	44,5%	0%	0%
https://g1.globo.com/	100%	0%	56,3%	0%	0%

Tabela 2 – Percentual de similaridade dos conteúdos extraídos em relação aos textos originais dos sites com 1 único exemplo por fonte.
Fonte: Elaborado pelo autor.

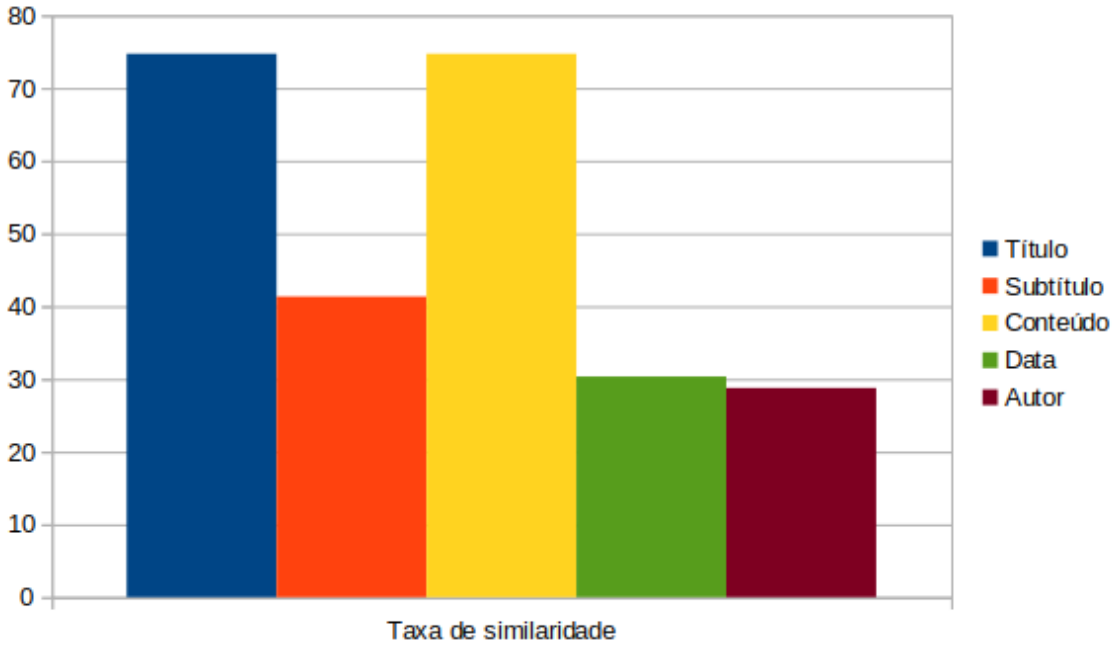

Figura 16 – Gráfico de similaridade entre os conteúdos extraídos e os conteúdos esperados, após 1 exemplo fornecido por fonte.

5.1.2 2 exemplos

Com 2 exemplos fornecidos por fonte, o *crawler* navegou por 18304 páginas, onde dessas, 8939 sendo possíveis páginas com artigos para ser coletados. Ao tentar extrair conteúdo para essas 8939 páginas, a extração ocorreu com sucesso para 6982, o que nos dá 78,10% de efetividade na combinação da identificação de artigos e extração dos mesmos. A assertividade para os campos foi muito melhor comparado ao experimento com 1 único exemplo fornecido, conforme mostra a tabela 3 e o gráfico da figura 17.

Site	Título	Subtítulo	Conteúdo	Data	Autor
https://www.jornaltabloide.com.br/	100%	100%	99,93%	100%	100%
https://extra.globo.com/	100%	-	99,2%	100%	100%
https://noticias.uol.com.br/	100%	-	96,72%	100%	-
https://www.techtudo.com.br/	100%	100%	96,4%	100%	100%
https://www.terra.com.br/	100%	100%	97,5%	100%	100%
http://www.tribunadonorte.com.br/	100%	-	100%	100%	-
https://www.folhape.com.br/	100%	100%	98,7%	100%	100%
https://www.cnnbrasil.com.br/	100%	100%	92,5%	100%	100%
https://www.futebolinterior.com.br/	100%	100%	99,28%	100%	100%
https://www.pmf.sc.gov.br/	100%	-	100%	100%	-
https://www.sc.gov.br/	100%	-	99,80%	100%	-
https://www.gov.br/pt-br/	100%	100%	100%	100%	-
https://noticias.ufsc.br/	100%	-	99,24%	100%	-
https://www.ifsc.edu.br/	100%	-	98,13%	100%	-
https://www.omelete.com.br/	100%	100%	93,2%	100%	100%
https://www.letras.mus.br/	100%	-	100%	-	100%
https://www.adorocinema.com/	100%	100%	95,1%	100%	100%
https://www.brasil247.com/	100%	100%	99%	100%	-
https://www.oeconomista.com.br/	100%	-	99,6%	100%	-
https://www.diariodepernambuco.com.br/	100%	-	100%	100%	100%
https://cronicabrasileira.org.br/	100%	-	100%	100%	100%
https://sitedepoesias.com/	100%	-	0%	0%	0%
https://www.torcedores.com/	100%	100%	100%	100%	100%
https://cinepop.com.br/	100%	-	99,97%	0%	100%
https://ge.globo.com/	100%	100%	91%	100%	0%
https://gamerview.uai.com.br/	100%	100%	99,1%	100%	100%
https://www.ofuxico.com.br/	100%	-	97,9%	100%	100%
https://contobrasileiro.com.br/	100%	-	98,7%	100%	100%
https://www.correiobraziliense.com.br/	100%	100%	80,7%	100%	0%
https://g1.globo.com/	100%	100%	84,62%	100%	72,2%

Tabela 3 – Percentual de similaridade dos conteúdos extraídos em relação aos textos originais dos sites com 2 exemplos por fonte.
Fonte: Elaborado pelo autor.

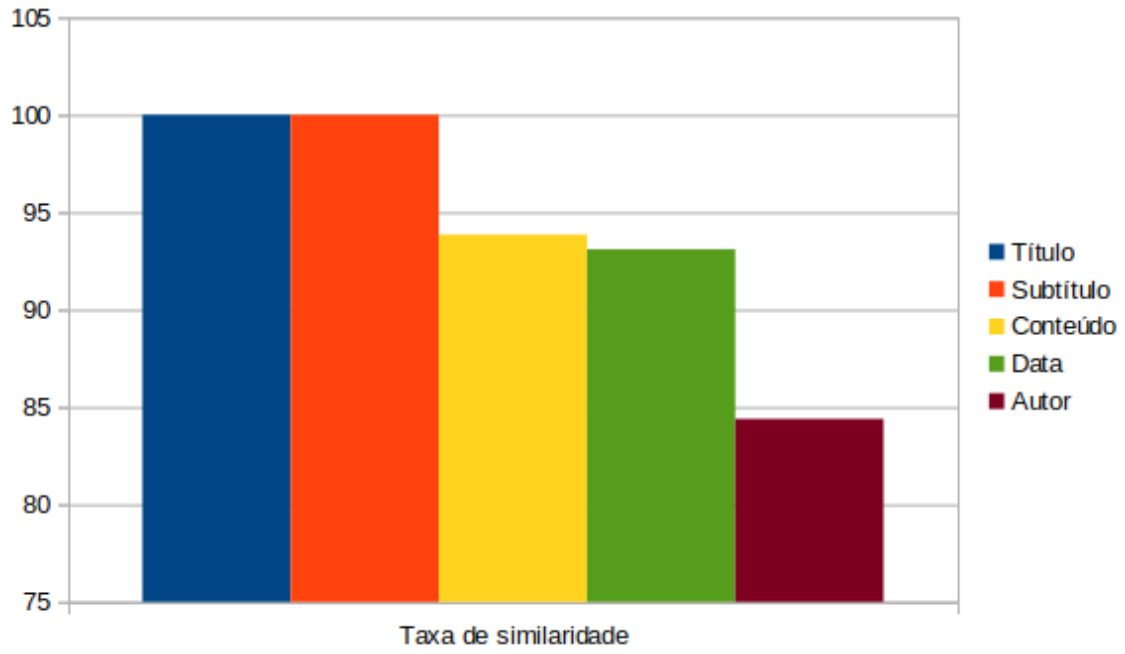

Figura 17 – Gráfico de similaridade entre os conteúdos extraídos e os conteúdos esperados, após 2 exemplos fornecidos por fonte

5.1.3 3 exemplos

Com 3 exemplos fornecidos por fonte, o *crawler* navegou por 18145 páginas, onde dessas, 10453 sendo possíveis páginas com artigos para ser coletados. Ao tentar extrair conteúdo para essas 10453 páginas, a extração ocorreu com sucesso para 7092, o que nos dá 67,84% de efetividade na combinação da identificação de artigos e extração dos mesmos. Isso é menos do que quando comparado com 2 exemplos, o que faz sentido, uma vez que quanto mais URLs exemplos forem fornecidas, mais o padrão da URL para extração será genérico (conforme seção 4.3.2), fazendo com que identifique como possíveis artigos mais páginas que não são. A assertividade na extração das informações se mostrou praticamente igual quando comparado com 2 exemplos, como mostra a tabela 4 e o gráfico da figura 18.

Site	**Título**	**Subtítulo**	**Conteúdo**	**Data**	**Autor**
https://www.jornaltabloide.com.br/	100%	100%	99,98%	100%	100%
https://extra.globo.com/	100%	-	99,87%	100%	100%
https://noticias.uol.com.br/	100%	-	98%	100%	-
https://www.techtudo.com.br/	100%	100%	97,5%	100%	100%
https://www.terra.com.br/	100%	100%	97,2%	100%	100%
http://www.tribunadonorte.com.br/	100%	-	100%	100%	-
https://www.folhape.com.br/	100%	100%	98,87%	100%	100%
https://www.cnnbrasil.com.br/	100%	100%	97%	100%	100%
https://www.futebolinterior.com.br/	100%	100%	99,75%	100%	100%
https://www.pmf.sc.gov.br/	100%	-	100%	100%	-
https://www.sc.gov.br/	100%	-	100%	100%	-
https://www.gov.br/pt-br/	100%	100%	100%	100%	-
https://noticias.ufsc.br/	100%	-	98,05%	100%	-
https://www.ifsc.edu.br/	100%	-	99,92%	100%	-
https://www.omelete.com.br/	100%	100%	92,2%	100%	100%
https://www.letras.mus.br/	100%	-	100%	-	100%
https://www.adorocinema.com/	100%	100%	96,6%	100%	100%
https://www.brasil247.com/	100%	100%	98,88%	100%	-
https://www.oeconomista.com.br/	100%	-	99,6%	100%	-
https://www.diariodepernambuco.com.br/	100%	-	100%	100%	100%
https://cronicabrasileira.org.br/	100%	-	100%	100%	100%
https://sitedepoesias.com/	100%	-	0%	0%	0%
https://www.torcedores.com/	100%	100%	100%	100%	100%
https://cinepop.com.br/	100%	-	100%	0%	100%
https://ge.globo.com/	100%	100%	94,3%	100%	0%
https://gamerview.uai.com.br/	100%	100%	98,90%	100%	100%
https://www.ofuxico.com.br/	100%	-	97,3%	100%	100%
https://contobrasileiro.com.br/	100%	-	99,1%	100%	100%
https://www.correiobraziliense.com.br/	100%	100%	80,5%	100%	0%
https://g1.globo.com/	100%	0%	79,90%	100%	83,5%

Tabela 4 – Percentual de similaridade dos conteúdos extraídos em relação aos textos originais dos sites com 3 exemplos por fonte.
Fonte: Elaborado pelo autor.

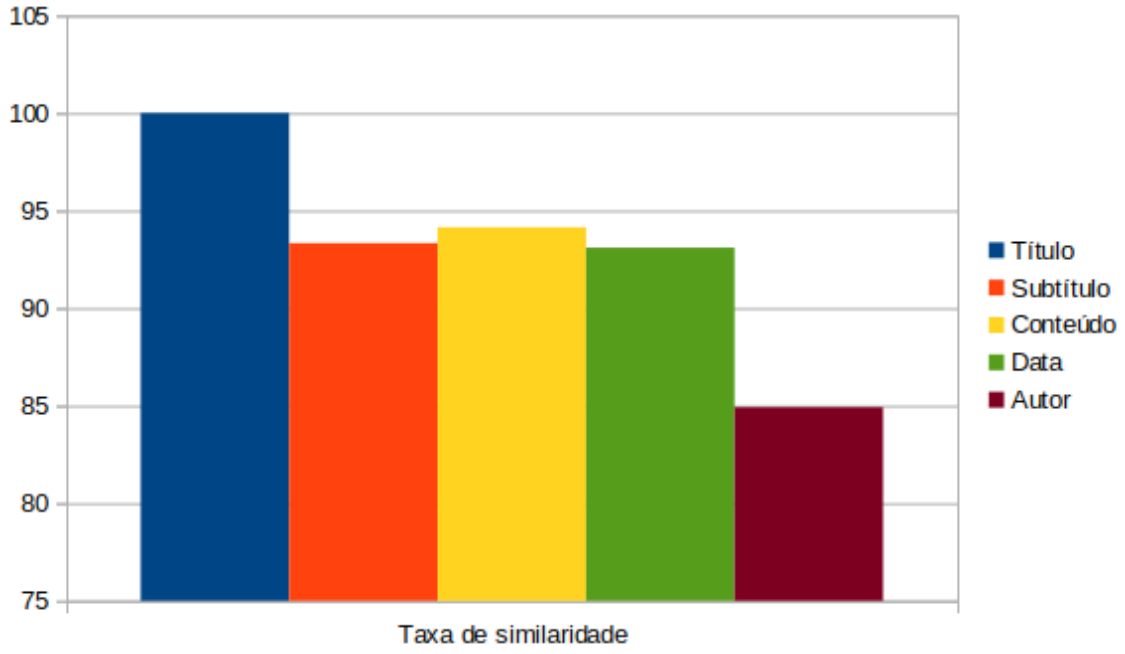

Figura 18 – Gráfico de similaridade entre os conteúdos extraídos e os conteúdos esperados, após 3 exemplos fornecidos por fonte

5.2 Exemplos fornecidos por usuários leigos

De acordo com a seção 5.1, 2 exemplos por *website* se mostrou a melhor abordagem dentre as feitas. Com base nisso, foi solicitado para que 3 usuários leigos fornecessem 2 exemplos para os mesmos *websites*, com a supervisão do autor, mas sem que houvesse interferência. O primeiro usuário com 19 anos de idade, o segundo com 36 e o terceiro com 54. A execução do *crawler* e o cálculo de similaridade seguiu as mesmas regras da seção 5.1

Foi observado que ao fornecerem os exemplos, apenas o usuário de 36 anos obteve cuidado para não incluir conteúdos presentes entre os parágrafos mas que não fazem parte do texto, como propagandas, conteúdos com leia mais e similares. Os resultados obtidos podemos ver nos gráficos 19, 20 e 21.

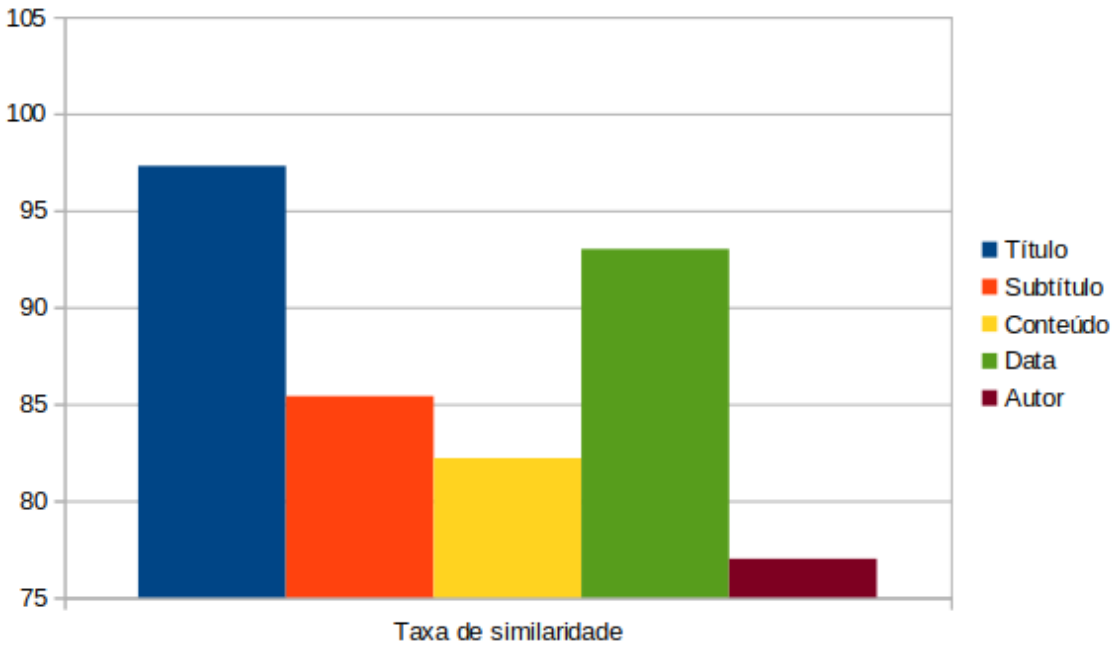

Figura 19 – Gráfico de similaridade entre os conteúdos extraídos e os conteúdos esperados, 2 exemplos fornecidos pelo usuário de 19 anos

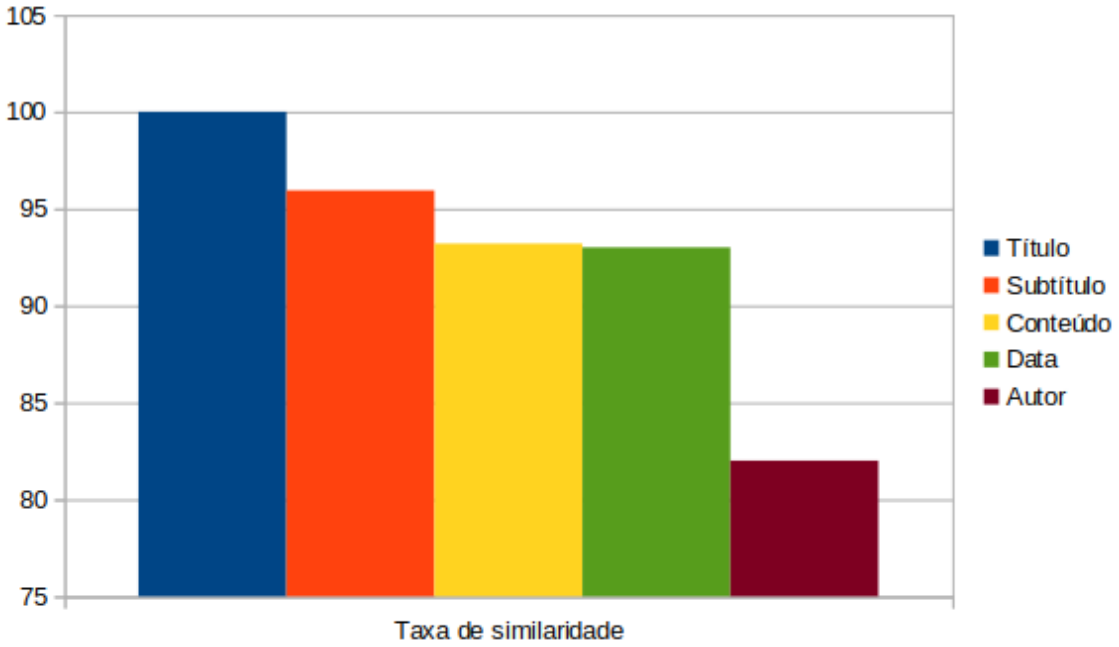

Figura 20 – Gráfico de similaridade entre os conteúdos extraídos e os conteúdos esperados, 2 exemplos fornecidos pelo usuário de 36 anos

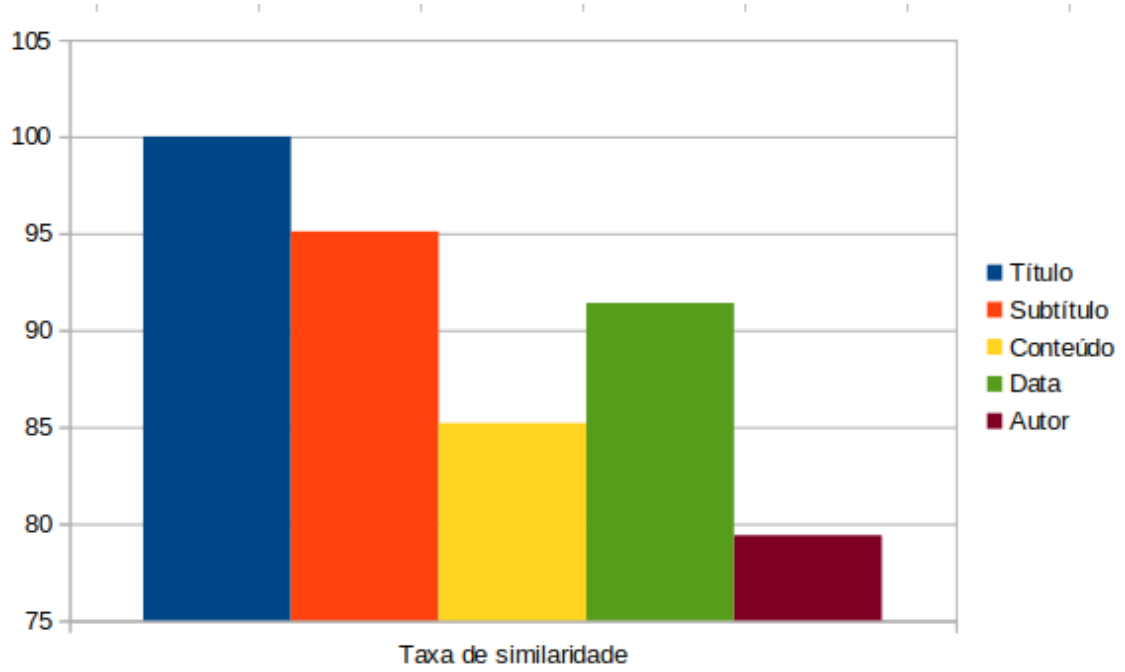

Figura 21 – Gráfico de similaridade entre os conteúdos extraídos e os conteúdos esperados, 2 exemplos fornecidos pelo usuário de 54 anos

Pode-se observar que todos os usuários obtiveram resultados um pouco diferentes um dos outros. Isso é razoável, uma vez que os exemplos fornecidos podem diferir de um usuário para outro. Usuários que fornecem exemplos com mais fidedignidade tendem a conseguir uma taxa de similaridade maior. Aqueles que fornecem exemplos com lixos em seus conteúdos, induzirão a aplicação a gerar uma configuração de extração com tais elementos. Também, usuários que fornecerem exemplos deficitários, com informações que deveriam ser fornecidas e não foram, também induzirão a aplicação a possivelmente não coletar algumas informações.

Ainda, por mais fidedigno que sejam os exemplos, haverão falhas para diversos sites, principalmente àqueles em que têm sua estrutura HTML mais complexa e que possuem muito conteúdo não pertencente ao texto no meio dele. Neste caso, se deve a limitação da aplicação.

6 Considerações finais e trabalhos futuros

Uma enorme quantidade de informação se faz presente em artigos publicados na internet, fazendo com que as pessoas a utilizem a fim de adquirirem conhecimento, se manterem informadas e até mesmo monitorar o que está sendo publicado sobre determinados assuntos. Isso pode ser facilitado com a automatização da navegação e extração destas informações, porém, geralmente, para isso é necessário que se tenha conhecimentos sólidos em programação, extração de dados e até mesmo redes de computadores.

Sendo assim, o objetivo deste trabalho foi viabilizar através de um aplicativo com interface gráfica, a possibilidade de pessoas sem tais conhecimentos de realizarem as tarefas mencionadas de forma automatizada. A proposta foi de um cenário onde o usuário interessado em extrair informações de artigos online em um determinado *website*, pode fazê-lo por fornecer exemplos dos conteúdos em que deseja coletar.

Isso foi possível de ser implementado principalmente com o auxílio da biblioteca BeautifulSoup e da aplicação do algoritmo Ratcliff-Obershelp para a comparação do conteúdo esperado com o conteúdo extraído, e através disto, fazendo lapidação para se ter um conteúdo cada vez mais limpo.

Para trabalhos futuros, pode-se considerar os seguintes pontos para melhoria da aplicação:

1. Permitir a navegação e a extração de informações em sites onde a navegação acontece de forma dinâmica, bem como a apresentação dos conteúdos;
2. Aplicar conceitos de programação concorrente e paralela, a fim de tornar a navegação e extração *multithreading*, otimizando e possibilitando que a extração das informações ocorra de maneira paralela para cada site, e não sequencial (quando possível);
3. Refinar o algoritmo de extração baseado em exemplos, a fim de melhorar a assertividade da coleta das informações.

7 Referências Bibliográficas

ADALBERG, B. NoDoSE: A Tool for Semi-Automatically Extracting Structured and Semi-Structured Data from Text Documents. SIGMOD Record 27, 2 (1998), 283-294.

CALIFF, M. E., MOONEY, R. J. Relational Learning of Pattern-Match Rules for Information Extraction. In Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence (Orlando, Florida, 1999), pp. 328-334.

CRESCENZI, V.; MECCA, G. Grammars Have Exceptions. Information Systems 23, 8 (1998), 539-656.

EMBLEY, D. W.; CAMPBELL, D. M.; JIANG, Y. S.; LIDDLE, S. W.; KAI NG, Y.; QUASS, D.; SMITH, R. D. Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages. Data and Knowledge Engineering 31, 3 (1999), 227-251

FREITAG, D. Machine Learning for Information Extraction in Informal Domains. Machine Learning 39, 2/3 (2000), 169-202.

HAMMER, J.; MCHUGH, J.; GARCIA-MOLINA, H. Semistructured Data: The TSIMMIS Experience. In Proceedings of the First East-European Symposium on Advances in Databases and Information Systems (ADBIS'97)(St. Petersburg, 1997), pp. 2-5.

HSU, C.-N.; Dung, M.-T. Generating Finite-State Transducers for Semi-Structured Data Extraction from the Web. Information Systems 23, 8 (1998), 521-538.

KUSHMERICK, N. WRAPPER Induction: Efficiency and Expressiveness. Artificial Intelligence Journal 118, 1-2 (2000), 15-68.

LAENDER, A. H. F.; RIBEIRO-NETO, B. A.; da SILVA, A. S.; TEIXEIRA, J. S. A Brief Survey of Web Data Extraction Tools. SIGMOD Record, New York, v. 31, n. 2, Junho, 2002.

LAENDER, A. H. F.; RIBEIRO-NETO, B. A.; DA SILVA., A. S. DeByE – Data Extraction by Example. Data and Knowledge Engineering (2001). To appear.

LIU, B. Web Data Mining: Exploring Hyperlinks, Contents and Usage. (Data-Centric Systems and Applications). Chicago, 2011.

LIU, L.; PU, C.; HAN, W. XWRAP: An XML-enable Wrapper Construction System for Web Information Sources. In Proceedings of the 16th IEEE International Conference on Data Engineering (San Diego, California, 2000), pp. 611-621.

OLSTON, C.; NAJORK, M. "Web Crawling", Foundations and Trends® in Information Retrieval: Vol. 4: No. 3, pp 175-246. http://dx.doi.org/10.1561/1500000017.

SAHUGUET, A; AZAVANT, F.F. Building intelligent web applications using lightweight wrappers. Data and Knowledge Engineering 36, 3 (2001), 283-316.

MATHIAS, G. (2017). qFEX - um crawler para busca e extração de questionários de pesquisa em documentos HTML. Repositório Institucional da UFSC.

RATCLIFF, W. J. Pattern Matching: The Gestalt Approach. Dr. Dobb's Journal, page 46, July 1988.

Beautiful Soup Documentation. Crummy, 2022. Disponível em https://www.crummy .com/software/BeautifulSoup/bs4/doc/. Acesso em 20 de abril de 2022.

Helpers for computing deltas. Docs Python, 2022. Disponível em https://docs.python .org/3/library/difflib.html#module-difflib. Acesso em 15 de abril de 2022.

8 Apêndices

8.1 Código fonte

O repositório da aplicação se encontra em https://gitlab.com/marcoslessa.cs/crawlex/

Article.py

```
""" Article model """
class Article:
    def __init__(self, url, title, content, date=None, subtitle=None,
    author=None):
        self.url = url
        self.title = title
        self.content = content
        self.date = date
        self.subtitle = subtitle
        self.author = author

    def get_url(self):
        return self.url

    def set_url(self, url):
        self.url = url

    def get_title(self):
        return self.title

    def set_title(self, title):
        self.title = title

    def get_content(self):
        return self.content

    def set_content(self, content):
        self.content = content

    def get_date(self):
        return self.date

    def set_date(self, date):
        self.date = date

    def get_subtitle(self):
```

```
36          return self.subtitle
37
38      def set_subtitle(self, subtitle):
39          self.subtitle = subtitle
40
41      def get_author(self):
42          return self.author
43
44      def set_author(self, author):
45          self.author = author
46
47      def __str__(self):
48          fullArticle = "Titulo: " + self.title + "\n\n"
49          if self.subtitle is not None and self.subtitle is not "":
50              fullArticle += "Subtitulo: " + self.subtitle + "\n\n"
51          fullArticle += "Conteudo: " + self.content + "\n\n"
52          if self.date is not None and self.date is not "":
53              fullArticle += "Data de publicacao: " + self.date + "\n\n"
54          if self.author is not None and self.author is not "":
55              fullArticle += "Autor: " + self.author + "\n\n"
56          fullArticle += "Disponivel no website: " + self.url
57          return fullArticle
```

ArticleDao.py

```
1  # Article Data Access Object
2  import sqlite3
3  from Article import Article
4
5  conn = sqlite3.connect("Articles.db", check_same_thread=False)
6  cursor = conn.cursor()
7
8
9  def insert_article(article: Article):
10     with conn:
11         cursor.execute("INSERT INTO Articles VALUES (?, ?, ?, ?, ?, ?)", (
       article.get_url(), article.get_title(), article.get_content(), article.
       get_date(), article.get_subtitle(), article.get_author()))
12
13
14 def get_all_articles():
15     with conn:
16         cursor.execute("SELECT * FROM Articles")
17         return cursor.fetchall()
18
19
20 def delete_all_articles():
```

```
    with conn:
        cursor.execute("DELETE FROM Articles")


def get_articles_by_boolean_query(boolean_query):
    with conn:
        cursor.execute("SELECT * FROM Articles WHERE " + boolean_query)
        return cursor.fetchall
```

ArticlePatternGenerator.py

```
from urllib.parse import urlparse as urlParse
import re

""" Regular expression generator for article urls """
class ArticlePatternGenerator:
    def __init__(self, articleUrls=[]):
        self.articleUrls = articleUrls

    def getArticleUrls(self):
        return self.articleUrls

    def setUrl(self, articleUrls):
        self.articleUrls = articleUrls

    def generate(self):
        parseResult = urlParse(self.articleUrls[0])
        regex = parseResult.scheme + "://" + parseResult.netloc + "/"
        tuplePathsList = []
        result_regex = ''
        for url in self.articleUrls:
            urlPath = urlParse(url).path
            url_full_path = urlPath
            if urlParse(url).query is not None:
                url_full_path += '?' + urlParse(url).query
            urlPathList = url_full_path.split("/")
            if urlPathList[-1] == "":
                urlPathList.pop()
            if urlPathList[0] == "":
                urlPathList.pop(0)
            tuplePathsList.append(tuple(urlPathList))
        if self.__isPathsSameSize(tuplePathsList):
            result_regex = regex + self.__generateForSamePathsSize(
    tuplePathsList)
        else:
            result_regex = regex + self.__generateForDifferentPathsSize(
    tuplePathsList)
```

```
35          if result_regex.endswith('?'):
36              return result_regex
37          else:
38              return result_regex + '?'
39
40      def __isPathsSameSize(self, tuplePathsList):
41          firstTuplePathsSize = len(tuplePathsList[0])
42          for tuplePaths in tuplePathsList:
43              if len(tuplePaths) != firstTuplePathsSize:
44                  return False
45          return True
46
47      def __generateForSamePathsSize(self, tuplePathsList):
48          regexForPath = ""
49          tuplesSize = self.__getMinimumSizePath(tuplePathsList)
50          index = 0
51          while index < tuplesSize:
52              indexContent = []
53              for tuple in tuplePathsList:
54                  indexContent.append(tuple[index])
55              regexForPath += self.__generateForIndexContent(indexContent) +
    "/"
56              index += 1
57          return regexForPath + "?"
58
59      def __generateForDifferentPathsSize(self, tuplePathsList):
60          regexForPath = self.__generateForSamePathsSize(tuplePathsList)
61          optionalPath = regexForPath + "("
62          minimumSizePath = self.__getMinimumSizePath(tuplePathsList)
63          maximumSizePath = self.__getMaximumSizePath(tuplePathsList)
64          for _ in range(maximumSizePath - minimumSizePath):
65              optionalPath += '[^/]+/'
66          return optionalPath + "?)"
67
68      def __generateForIndexContent(self, indexContent):
69          if self.__isEquals(indexContent) and self.__isNumber(indexContent)
    is False:
70              return indexContent[0]
71          elif self.__isNumber(indexContent):
72              return '\d+'
73          return '[^/]+'
74
75      def __isEquals(self, indexContent):
76          baseContent = indexContent[0]
77          for content in indexContent:
78              if content != baseContent:
79                  return False
```

```
80          return True
81
82      def __isNumber(self, indexContent):
83          baseContent = indexContent[0]
84          for content in indexContent:
85              if bool(re.fullmatch('\d+', content)) == False:
86                  return False
87          return True
88
89      def __getMinimumSizePath(self, tuplePathsList):
90          minimumSize = len(tuplePathsList[0])
91          for tuple in tuplePathsList:
92              if len(tuple) < minimumSize:
93                  minimumSize = len(tuple)
94          return minimumSize
95
96      def __getMaximumSizePath(self, tuplePathsList):
97          maximumSize = len(tuplePathsList[0])
98          for tuple in tuplePathsList:
99              if len(tuple) > maximumSize:
100                 maximumSize = len(tuple)
101         return maximumSize
```

ContentComparator.py

```
1  import string
2  import difflib
3
4  """Ratcliff-Obershelp algorithm"""
5
6  class ContentComparator():
7      def __init__(self, content_example: string, extracted_content: string):
8          self.content_example = content_example
9          self.extracted_content = extracted_content
10
11     def compare(self) -> bool:
12         sequence_matcher = difflib.SequenceMatcher(None, self.
       content_example, self.extracted_content)
13         return sequence_matcher.ratio()
```

crawlex.py

```
1  import sys
2  from ui import Ui_MainWindow
3  from PyQt5 import QtWidgets
4  from ArticleDao import delete_all_articles
```

```
5
6 #Program entry point
7
8 if __name__ == "__main__":
9     delete_all_articles()
10    app = QtWidgets.QApplication(sys.argv)
11    MainWindow = QtWidgets.QMainWindow()
12    ui = Ui_MainWindow()
13    ui.setupUi(MainWindow)
14    MainWindow.show()
15    sys.exit(app.exec_())
```

CreateTables.py

```
1 # Responsible for creating the database tables
2 import sqlite3
3
4 conn = sqlite3.connect("database.db")
5 cursor = conn.cursor()
6
7 cursor.execute("""CREATE TABLE Article (
8                 url text,
9                 title text,
10                content text,
11                date text,
12                subtitle text,
13                author text
14                )""")
15
16 cursor.execute("""CREATE TABLE Website (
17                id INTEGER PRIMARY KEY,
18                domain TEXT NOT NULL UNIQUE
19                )""")
20
21 cursor.execute("""CREATE TABLE ExampleArticle (
22                website_id INTEGER,
23                url TEXT NOT NULL,
24                title TEXT NOT NULL,
25                content TEXT NOT NULL,
26                date TEXT,
27                subtitle TEXT,
28                author TEXT
29                )""")
30
31 conn.commit()
32 conn.close()
```

DatePatternGenerator.py

```
import locale
from datetime import datetime

#Regular expression generator for date extraction
class DatePatternRegexExtractorGenerator:
    def __init__(self, date: str):
        locale.setlocale(locale.LC_TIME, 'en_US.UTF-8')
        date_obj = datetime.strptime(date, '%a %b %d %Y')
        self.__default_date = str(date_obj.day) + '/' + str(date_obj.month)
    + '/' + str(date_obj.year)

    def generate_patterns_regex_extractor(self) -> set:
        locale.setlocale(locale.LC_TIME, 'pt_BR.UTF-8')
        date_time_obj = datetime.strptime(self.__default_date, '%d/%m/%Y')

        pt_regex = '\\d{1,2} de \\w+ de \\d{2}(\\d{2})?'
        default_barra_regex = '\\d{1,2}/\\d{1,2}/\\d{2}(\\d{2})?'
        default_traco_regex = '\\d{1,2}-\\d{1,2}-\\d{2}(\\d{2})?'
        year_first_barra_regex = '\\d{2}(\\d{2})?/\\d{1,2}/\\d{1,2}'
        year_first_traco_regex = '\\d{2}(\\d{2})?-\\d{1,2}-\\d{1,2}'

        date_patterns_regex_extractor = set()

        date_patterns_regex_extractor.add((date_time_obj.strftime('%d/%m/%Y
    '), '%d/%m/%Y', default_barra_regex))
        date_patterns_regex_extractor.add((date_time_obj.strftime('%d/%m/%y
    '), '%d/%m/%y', default_barra_regex))
        date_patterns_regex_extractor.add((date_time_obj.strftime('%d/%-m/%
    Y'), '%d/%m/%Y', default_barra_regex))
        date_patterns_regex_extractor.add((date_time_obj.strftime('%d/%-m/%
    y'), '%d/%m/%y', default_barra_regex))
        date_patterns_regex_extractor.add((date_time_obj.strftime('%d/%B/%Y
    '), '%d/%B/%Y', default_barra_regex))
        date_patterns_regex_extractor.add((date_time_obj.strftime('%d/%B/%y
    '), '%d/%B/%y', default_barra_regex))
        date_patterns_regex_extractor.add((date_time_obj.strftime('%d/%b/%Y
    '), '%d/%b/%Y', default_barra_regex))
        date_patterns_regex_extractor.add((date_time_obj.strftime('%d/%b/%y
    '), '%d/%b/%y', default_barra_regex))
        date_patterns_regex_extractor.add((date_time_obj.strftime('%-d/%m/%
    Y'), '%d/%m/%Y', default_barra_regex))
        date_patterns_regex_extractor.add((date_time_obj.strftime('%-d/%m/%
    y'), '%d/%m/%y', default_barra_regex))
        date_patterns_regex_extractor.add((date_time_obj.strftime('%-d/%-m
    /%Y'), '%d/%m/%Y', default_barra_regex))
        date_patterns_regex_extractor.add((date_time_obj.strftime('%-d/%-m
```

```
    /%y'), '%d/%m/%y', default_barra_regex))
35          date_patterns_regex_extractor.add((date_time_obj.strftime('%-d/%B/%
    Y'), '%d/%B/%Y', default_barra_regex))
36          date_patterns_regex_extractor.add((date_time_obj.strftime('%-d/%B/%
    y'), '%d/%B/%y', default_barra_regex))
37          date_patterns_regex_extractor.add((date_time_obj.strftime('%-d/%b/%
    Y'), '%d/%b/%Y', default_barra_regex))
38          date_patterns_regex_extractor.add((date_time_obj.strftime('%-d/%b/%
    y'), '%d/%b/%y', default_barra_regex))
39
40          date_patterns_regex_extractor.add((date_time_obj.strftime('%d-%m-%Y
    '), '%d-%m-%Y', default_traco_regex))
41          date_patterns_regex_extractor.add((date_time_obj.strftime('%d-%m-%y
    '), '%d-%m-%y', default_traco_regex))
42          date_patterns_regex_extractor.add((date_time_obj.strftime('%d-%-m-%
    Y'), '%d-%m-%Y', default_traco_regex))
43          date_patterns_regex_extractor.add((date_time_obj.strftime('%d-%-m-%
    y'), '%d-%m-%y', default_traco_regex))
44          date_patterns_regex_extractor.add((date_time_obj.strftime('%d-%B-%Y
    '), '%d-%B-%Y', default_traco_regex))
45          date_patterns_regex_extractor.add((date_time_obj.strftime('%d-%B-%y
    '), '%d-%B-%y', default_traco_regex))
46          date_patterns_regex_extractor.add((date_time_obj.strftime('%d-%b-%Y
    '), '%d-%b-%Y', default_traco_regex))
47          date_patterns_regex_extractor.add((date_time_obj.strftime('%d-%b-%y
    '), '%d-%b-%y', default_traco_regex))
48          date_patterns_regex_extractor.add((date_time_obj.strftime('%-d-%m-%
    Y'), '%d-%m-%Y', default_traco_regex))
49          date_patterns_regex_extractor.add((date_time_obj.strftime('%-d-%m-%
    y'), '%d-%m-%y', default_traco_regex))
50          date_patterns_regex_extractor.add((date_time_obj.strftime('%-d-%-m
    -%Y'), '%d-%m-%Y', default_traco_regex))
51          date_patterns_regex_extractor.add((date_time_obj.strftime('%-d-%-m
    -%y'), '%d-%m-%y', default_traco_regex))
52          date_patterns_regex_extractor.add((date_time_obj.strftime('%-d-%B-%
    Y'), '%d-%B-%Y', default_traco_regex))
53          date_patterns_regex_extractor.add((date_time_obj.strftime('%-d-%B-%
    y'), '%d-%B-%y', default_traco_regex))
54          date_patterns_regex_extractor.add((date_time_obj.strftime('%-d-%b-%
    Y'), '%d-%b-%Y', default_traco_regex))
55          date_patterns_regex_extractor.add((date_time_obj.strftime('%-d-%b-%
    y'), '%d-%b-%y', default_traco_regex))
56
57          date_patterns_regex_extractor.add((date_time_obj.strftime('%d de %B
     de %Y'), '%d de %B de %Y', pt_regex))
58          date_patterns_regex_extractor.add((date_time_obj.strftime('%d de %B
     de %y'), '%d de %B de %y', pt_regex))
```

```
59          date_patterns_regex_extractor.add((date_time_obj.strftime('%d de %b
    de %Y'), '%d de %b de %Y', pt_regex))
60          date_patterns_regex_extractor.add((date_time_obj.strftime('%d de %b
    de %y'), '%d de %B de %y', pt_regex))
61          date_patterns_regex_extractor.add((date_time_obj.strftime('%-d de %
    B de %Y'), '%d de %B de %Y', pt_regex))
62          date_patterns_regex_extractor.add((date_time_obj.strftime('%-d de %
    B de %y'), '%d de %B de %y', pt_regex))
63          date_patterns_regex_extractor.add((date_time_obj.strftime('%-d de %
    b de %Y'), '%d de %b de %Y', pt_regex))
64          date_patterns_regex_extractor.add((date_time_obj.strftime('%-d de %
    b de %y'), '%d de %B de %y', pt_regex))
65
66          date_patterns_regex_extractor.add((date_time_obj.strftime('%m/%d/%Y
    '), '%m/%d/%Y', default_barra_regex))
67          date_patterns_regex_extractor.add((date_time_obj.strftime('%m/%d/%y
    '), '%m/%d/%y', default_barra_regex))
68          date_patterns_regex_extractor.add((date_time_obj.strftime('%m/%-d/%
    Y'), '%m/%d/%Y', default_barra_regex))
69          date_patterns_regex_extractor.add((date_time_obj.strftime('%m/%-d/%
    y'), '%m/%d/%y', default_barra_regex))
70          date_patterns_regex_extractor.add((date_time_obj.strftime('%-m/%d/%
    Y'), '%m/%d/%Y', default_barra_regex))
71          date_patterns_regex_extractor.add((date_time_obj.strftime('%-m/%d/%
    y'), '%m/%d/%y', default_barra_regex))
72          date_patterns_regex_extractor.add((date_time_obj.strftime('%-m/%-d
    /%Y'), '%m/%d/%Y', default_barra_regex))
73          date_patterns_regex_extractor.add((date_time_obj.strftime('%-m/%-d
    /%y'), '%m/%d/%y', default_barra_regex))
74
75          date_patterns_regex_extractor.add((date_time_obj.strftime('%m-%d-%Y
    '), '%m-%d-%Y', default_traco_regex))
76          date_patterns_regex_extractor.add((date_time_obj.strftime('%m-%d-%y
    '), '%m-%d-%y', default_traco_regex))
77          date_patterns_regex_extractor.add((date_time_obj.strftime('%m-%-d-%
    Y'), '%m-%d-%Y', default_traco_regex))
78          date_patterns_regex_extractor.add((date_time_obj.strftime('%m-%-d-%
    y'), '%m-%d-%y', default_traco_regex))
79          date_patterns_regex_extractor.add((date_time_obj.strftime('%-m-%d-%
    Y'), '%m-%d-%Y', default_traco_regex))
80          date_patterns_regex_extractor.add((date_time_obj.strftime('%-m-%d-%
    y'), '%m-%d-%y', default_traco_regex))
81          date_patterns_regex_extractor.add((date_time_obj.strftime('%-m-%-d
    -%Y'), '%m-%d-%Y', default_traco_regex))
82          date_patterns_regex_extractor.add((date_time_obj.strftime('%-m-%-d
    -%y'), '%m-%d-%y', default_traco_regex))
83
```

```
84          date_patterns_regex_extractor.add((date_time_obj.strftime('%Y/%m/%d
    '), '%Y/%m/%d', year_first_barra_regex))
85          date_patterns_regex_extractor.add((date_time_obj.strftime('%y/%m/%d
    '), '%y/%m/%d', year_first_barra_regex))
86          date_patterns_regex_extractor.add((date_time_obj.strftime('%Y/%m/%-
    d'), '%Y/%m/%d', year_first_barra_regex))
87          date_patterns_regex_extractor.add((date_time_obj.strftime('%y/%m/%-
    d'), '%y/%m/%d', year_first_barra_regex))
88          date_patterns_regex_extractor.add((date_time_obj.strftime('%Y/%-m/%
    d'), '%Y/%m/%d', year_first_barra_regex))
89          date_patterns_regex_extractor.add((date_time_obj.strftime('%y/%-m/%
    d'), '%y/%m/%d', year_first_barra_regex))
90          date_patterns_regex_extractor.add((date_time_obj.strftime('%Y/%-m
    /%-d'), '%Y/%m/%d', year_first_barra_regex))
91          date_patterns_regex_extractor.add((date_time_obj.strftime('%y/%-m
    /%-d'), '%y/%m/%d', year_first_barra_regex))
92
93          date_patterns_regex_extractor.add((date_time_obj.strftime('%Y-%m-%d
    '), '%Y-%m-%d', year_first_traco_regex))
94          date_patterns_regex_extractor.add((date_time_obj.strftime('%y-%m-%d
    '), '%y-%m-%d', year_first_traco_regex))
95          date_patterns_regex_extractor.add((date_time_obj.strftime('%Y-%m-%-
    d'), '%Y-%m-%d', year_first_traco_regex))
96          date_patterns_regex_extractor.add((date_time_obj.strftime('%y-%m-%-
    d'), '%y-%m-%d', year_first_traco_regex))
97          date_patterns_regex_extractor.add((date_time_obj.strftime('%Y-%-m-%
    d'), '%Y-%m-%d', year_first_traco_regex))
98          date_patterns_regex_extractor.add((date_time_obj.strftime('%y-%-m-%
    d'), '%y-%m-%d', year_first_traco_regex))
99          date_patterns_regex_extractor.add((date_time_obj.strftime('%Y-%-m
    -%-d'), '%Y-%m-%d', year_first_traco_regex))
100         date_patterns_regex_extractor.add((date_time_obj.strftime('%y-%-m
    -%-d'), '%y-%m-%d', year_first_traco_regex))
101
102         return date_patterns_regex_extractor
```

DedicatedCrawler.py

```
1 import LinkFinder
2 from PyQt5.QtCore import *
3 from WebsiteDao import get_website_id_by_domain
4 from ExampleArticlesDao import get_example_articles_by_website_id
5 import ArticlePatternGenerator
6 import re
7 import ExtractionConfig
8 import Extractor
9
```

```
#Dedicated Crawler for tests
class DedicatedCrawler(QThread):
    update_log = pyqtSignal(str)

    def __init__(self, start_url: str):
        QThread.__init__(self)
        self.__start_url = start_url
        self.__urls_already_visited = set()
        self.__urls_to_visit = set()
        self.__urls_to_visit_after = set()
        self.__stop_crawl = False
        self.__example_articles = self.load_examples_articles_from_website()
        self.__ARTICLE_PATTERN_TO_EXTRACT = self.generate_article_pattern_to_extract()
        self.__extraction_config = self.generate_extraction_config()
        self.__extractor = Extractor.Extractor(self.__extraction_config)
        self.update_log.emit('Padrao da URL de artigo gerada: ' + self.__ARTICLE_PATTERN_TO_EXTRACT)

    def run(self):
        self.__urls_already_visited.add(self.__start_url)
        link_finder = LinkFinder.LinkFinder(self.__start_url)
        self.__urls_to_visit = link_finder.find_links()


        while not self.__stop_crawl and len(self.__urls_to_visit) > 0:
            #select the url head of the list, and remove it
            new_url = self.__urls_to_visit.pop()

            #check if the url has already been visited
            if new_url in self.__urls_already_visited:
                continue

            #if not visited, mark as visited
            self.__urls_already_visited.add(new_url)
            link_finder = LinkFinder.LinkFinder(new_url)

            try:
                self.__urls_to_visit_after = self.__urls_to_visit_after.union(link_finder.find_links())
            except:
                print('Erro ao buscar urls na na pagina: ', new_url)
                print('')
                continue

            #try to extract the content of the new url
```

```
53            print('Nova URL: ', new_url)
54            print('')
55            print('')
56
57            #try to extract if it matches with the article pattern
    expression
58            if self.match_article_pattern(new_url):
59                try:
60                    self.update_log.emit(str(self.__extractor.extract(
    new_url)))
61                    self.update_log.emit('
    ------------------------------------------------------------------------\
    n\n')
62                except Exception as e:
63                    print(e)
64
65        if self.__stop_crawl is False:
66            if len(self.__urls_to_visit_after) == 0:
67                print('sem mais urls, encerrando...')
68                return
69            self.__start_url = self.__urls_to_visit_after.pop()
70            print('ultima url ', self.__start_url)
71            return self.run()
72
73
74    def match_article_pattern(self, url):
75        return bool(re.fullmatch(self.__ARTICLE_PATTERN_TO_EXTRACT, url))
76
77    def stop_crawl(self):
78        self.__stop_crawl = True
79
80    def load_examples_articles_from_website(self):
81        website_id = get_website_id_by_domain(self.__start_url)
82        return get_example_articles_by_website_id(website_id)
83
84    def generate_article_pattern_to_extract(self):
85        example_articles_url = []
86        for example_article in self.__example_articles:
87            example_articles_url.append(example_article.get_url())
88        article_pattern_generator = ArticlePatternGenerator.
    ArticlePatternGenerator(example_articles_url)
89        return article_pattern_generator.generate()
90
91    def generate_extraction_config(self):
92        extraction_config = ExtractionConfig.ExtractionConfig(self.
    __example_articles)
93        print('Gerando configuracao para coleta...\n')
```

```
94          extraction_config.generate_config()
95          return extraction_config
```

DefaultDatePatternConverter.py

```
1  import locale
2  from datetime import datetime
3
4  # Convert the extracted date to a default date pattern (day/month/year)
5  class DefaultDatePatternConverter:
6      def __init__(self, date_and_pattern_extracted: tuple):
7          self.date_and_pattern_extracted = date_and_pattern_extracted
8
9      def convert_date(self):
10         locale.setlocale(locale.LC_TIME, 'pt_BR.UTF-8')
11         date_str = self.date_and_pattern_extracted[0]
12         extracted_date_pattern = self.date_and_pattern_extracted[1]
13         date_time_obj = datetime.strptime(date_str, extracted_date_pattern)
14         date_in_default_pattern = date_time_obj.strftime('%d/%m/%Y')
15         return date_in_default_pattern
```

EmailSender.py

```
1  import smtplib
2  import EmailSenderConfig as config
3
4  # Responsible for sending emails
5  class EmailSender:
6      def __init__(self, subject, content, recipient):
7          self.recipient = recipient
8          self.subject = subject
9          self.content = content
10
11     def send(self):
12         server = self.setup_server()
13         server.sendmail(config.email, self.recipient, self.
       get_subject_and_content())
14         server.quit()
15
16     def setup_server(self):
17         server = smtplib.SMTP_SSL(config.host, config.port)
18         server.login(config.email, config.password)
19         return server
20
21     def get_subject_and_content(self):
22         return 'Subject: {}\n\n{}'.format(self.subject, self.content)
```

EmailSenderConfig.py

```
# username and password for sending emails
host="smtp.gmail.com"
port=465
email="email@gmail.com"
password="senha"
```

ExampleArticlesDao.py

```
# Example articles Data Access Object
import sqlite3
from Article import Article


def insert_example_article(website_id: int, article: Article):
    conn = sqlite3.connect("database.db")
    cursor = conn.cursor()
    with conn:
        cursor.execute("INSERT INTO ExampleArticle VALUES (?, ?, ?, ?, ?, ?, ?)", (website_id, article.get_url(), article.get_title(), article.content, article.get_date(), article.get_subtitle(), article.get_author()))
    conn.close()


def get_all_example_articles():
    conn = sqlite3.connect("database.db")
    cursor = conn.cursor()
    with conn:
        cursor.execute("SELECT * FROM ExampleArticle")
        return cursor.fetchall()


def get_example_articles_by_website_id(website_id: int):
    conn = sqlite3.connect("database.db")
    cursor = conn.cursor()
    with conn:
        cursor.execute("SELECT * FROM ExampleArticle WHERE website_id = ?", (website_id,))
        results = cursor.fetchall()
        articles = []
        for article in results:
            url = article[1]
            title = article[2]
            content = article[3]
            date = article[4]
```

```
            subtitle = article[5]
            author = article[6]
            articles.append(Article(url, title, content, date, subtitle,
    author))
    conn.close()
    return articles
```

ExtractionConfig.py

```
import bs4
import re
import ContentComparator
import requests
from bs4 import BeautifulSoup
from DatePatternGenerator import DatePatternRegexExtractorGenerator

# Generate the extract configuration
class ExtractionConfig:
    def __init__(self, example_articles: []):
        self.__example_articles = example_articles
        self.__title_extractor_element = None
        self.__subtitle_extractor_element = None
        self.__content_extractor_element = None
        self.__author_extractor_element = None
        self.__date_extractor_element = None
        self.__content_cleaner_element = None
        self.__full_html_content = self.load_full_html_content_dict()

    def get_title_extractor(self):
        return self.__title_extractor_element

    def get_subtitle_extractor(self):
        return self.__subtitle_extractor_element

    def get_content_extractor(self):
        return self.__content_extractor_element

    def get_content_cleaner_element(self):
        return self.__content_cleaner_element

    def get_author_extractor(self):
        return self.__author_extractor_element

    def get_date_extractor(self):
        return self.__date_extractor_element

    def generate_config(self):
```

```
39          self.search_title_extractor_element()
40          self.search_subtitle_extractor_element()
41          self.search_content_extractor_element()
42          self.search_author_extractor_element()
43          self.search_date_extractor_element()
44
45      def search_title_extractor_element(self):
46          possible_extractor_elements = set()
47          first_example_article = self.__example_articles[0]
48          soup = BeautifulSoup(self.__full_html_content[first_example_article
    .get_url()], 'html5lib')
49          for tag in soup.find_all():
50              if tag.text.lower().__contains__(first_example_article.
    get_title().lower()):
51                  possible_extractor_elements.add((tag.name, str(tag.attrs)))
52
53          possible_extractor_elements_copy = possible_extractor_elements.copy
    ()
54
55          for possible_extractor_element in possible_extractor_elements_copy:
56              for example_article in self.__example_articles:
57                  soup = BeautifulSoup(self.__full_html_content[
    example_article.get_url()], 'html5lib')
58                  tag = soup.find(possible_extractor_element[0], eval(
    possible_extractor_element[1]))
59                  if tag is not None:
60                      if possible_extractor_elements.__contains__((tag.name,
    str(tag.attrs))):
61                          if tag.text.lower().__contains__(example_article.
    get_title().lower()) is False:
62                              possible_extractor_elements.remove((tag.name,
    str(tag.attrs)))
63                              break
64                  else:
65                      if possible_extractor_elements.__contains__((
    possible_extractor_element[0], possible_extractor_element[1])):
66                          possible_extractor_elements.remove((
    possible_extractor_element[0], possible_extractor_element[1]))
67
68          shortest_tag = None
69          for possible_extractor_element in possible_extractor_elements:
70              soup = BeautifulSoup(self.__full_html_content[
    first_example_article.get_url()], 'html5lib')
71              tag = soup.find(possible_extractor_element[0], eval(
    possible_extractor_element[1]))
72              if tag is not None:
73                  if shortest_tag is None or len(shortest_tag.text) > len(tag
```

```
    .text):
74                  if tag.text.lower().__contains__(first_example_article.
    get_title().lower()):
75                      shortest_tag = tag
76      self.__title_extractor_element = shortest_tag
77
78  def search_subtitle_extractor_element(self):
79      possible_extractor_elements = set()
80      first_example_article = self.__example_articles[0]
81      if first_example_article.get_subtitle() is None or
    first_example_article.get_subtitle() == '':
82          self.__subtitle_extractor_element = None
83          return
84      soup = BeautifulSoup(self.__full_html_content[first_example_article
    .get_url()], 'html5lib')
85      for tag in soup.find_all():
86          if tag.text.__contains__(first_example_article.get_subtitle()):
87              possible_extractor_elements.add((tag.name, str(tag.attrs)))
88
89      possible_extractor_elements_copy = possible_extractor_elements.copy
    ()
90
91      for possible_extractor_element in possible_extractor_elements_copy:
92          for example_article in self.__example_articles:
93              soup = BeautifulSoup(self.__full_html_content[
    example_article.get_url()], 'html5lib')
94              tag = soup.find(possible_extractor_element[0], eval(
    possible_extractor_element[1]))
95              if tag is not None:
96                  if possible_extractor_elements.__contains__((tag.name,
    str(tag.attrs))):
97                      if tag.text.__contains__(example_article.
    get_subtitle()) is False:
98                          possible_extractor_elements.remove((tag.name,
    str(tag.attrs)))
99                          break
100
101     shortest_tag = None
102     for possible_extractor_element in possible_extractor_elements:
103         soup = BeautifulSoup(self.__full_html_content[
    first_example_article.get_url()], 'html5lib')
104         tag = soup.find(possible_extractor_element[0], eval(
    possible_extractor_element[1]))
105         if tag is not None:
106             if shortest_tag is None or len(shortest_tag.text) > len(tag
    .text):
107                 if tag.text.__contains__(first_example_article.
```

```
    get_subtitle()):
108                         shortest_tag = tag
109         self.__subtitle_extractor_element = shortest_tag
110 
111     def search_content_extractor_element(self):
112         ratio = 0
113         extractor = None
114         for example_article in self.__example_articles:
115             soup = BeautifulSoup(self.__full_html_content[example_article.
    get_url()], 'html5lib')
116             for tag in soup.find_all():
117                 content_comparator = ContentComparator.ContentComparator(
    example_article.get_content(), tag.text)
118                 if content_comparator.compare() > ratio:
119                     ratio = content_comparator.compare()
120                     extractor = tag
121         self.__content_extractor_element = extractor
122 
123         base_article_content = None
124         base_ratio = 0
125         for example_article in self.__example_articles:
126             content_comparator = ContentComparator.ContentComparator(
    example_article.get_content(), extractor.text)
127             if content_comparator.compare() > base_ratio:
128                 base_ratio = content_comparator.compare()
129                 base_article_content = example_article.get_content()
130 
131         extractor_copy = extractor
132         cleaner_elements = set()
133 
134         for tag in extractor_copy.find_all_next():
135             if isinstance(tag, bs4.Tag):
136                 if not base_article_content.__contains__(tag.text):
137                     str_tag = (tag.name, str(tag.attrs))
138                     if str_tag[0] == 'p':
139                         continue
140                     tag.decompose()
141                     content_comparator = ContentComparator.
    ContentComparator(base_article_content, extractor.text)
142                     if content_comparator.compare() >= ratio:
143                         ratio = content_comparator.compare()
144                         if tag is not None:
145                             cleaner_elements.add(str_tag)
146         self.__content_cleaner_element = cleaner_elements
147 
148     def search_author_extractor_element(self):
149         possible_extractor_elements = set()
```

```
150         first_example_article = self.__example_articles[0]
151         if first_example_article.get_author() is None or
    first_example_article.get_author() == '':
152             self.__author_extractor_element = None
153             return
154         soup = BeautifulSoup(self.__full_html_content[first_example_article
    .get_url()], 'html5lib')
155         for tag in soup.find_all():
156             if tag.text.__contains__(first_example_article.get_author()):
157                 possible_extractor_elements.add((tag.name, str(tag.attrs)))
158 
159         possible_extractor_elements_copy = possible_extractor_elements.copy
    ()
160 
161         for possible_extractor_element in possible_extractor_elements_copy:
162             for example_article in self.__example_articles:
163                 soup = BeautifulSoup(self.__full_html_content[
    example_article.get_url()], 'html5lib')
164                 tag = soup.find(possible_extractor_element[0], eval(
    possible_extractor_element[1]))
165                 if tag is not None:
166                     if possible_extractor_elements.__contains__((tag.name,
    str(tag.attrs))):
167                         if tag.text.__contains__(example_article.get_author
    ()) is False:
168                             possible_extractor_elements.remove((tag.name,
    str(tag.attrs)))
169                             break
170 
171         shortest_tag = None
172         for possible_extractor_element in possible_extractor_elements:
173             soup = BeautifulSoup(self.__full_html_content[
    first_example_article.get_url()], 'html5lib')
174             tag = soup.find(possible_extractor_element[0], eval(
    possible_extractor_element[1]))
175             if tag is not None:
176                 if shortest_tag is None or len(shortest_tag.text) > len(tag
    .text):
177                     if tag.text.__contains__(first_example_article.
    get_author()):
178                         shortest_tag = tag
179         self.__author_extractor_element = shortest_tag
180 
181     def search_date_extractor_element(self):
182         if self.__example_articles[0].get_date() is None or self.
    __example_articles[0].get_date() == '':
183             self.__date_extractor_element = None
```

```
184             return
185         date_possible_extractor_elements = self.
    search_possible_date_extractor_elements()
186         articles_with_date = self.load_articles_with_date()
187         for date_possible_extractor_element in
    date_possible_extractor_elements.copy():
188             for example_article in articles_with_date:
189                 date_pattern_regex_extractor_generator =
    DatePatternRegexExtractorGenerator(example_article.get_date())
190                 date_elements = date_pattern_regex_extractor_generator.
    generate_patterns_regex_extractor()
191                 soup = BeautifulSoup(self.__full_html_content[
    example_article.get_url()], 'html5lib')
192                 tag = soup.find(date_possible_extractor_element[0].name,
    date_possible_extractor_element[0].attrs)
193                 if tag is not None:
194                     text = tag.text
195                 ok = 0
196                 for date_element in date_elements:
197                     date = date_element[0]
198                     date_pattern = date_element[2]
199                     try:
200                         result = re.search(date_pattern, text)
201                         if result is not None and result.group() == date:
202                             ok = 1
203                             break
204                     except Exception as e:
205                         raise(e)
206                 if ok == 0:
207                     date_possible_extractor_elements.remove(
    date_possible_extractor_element)
208                     break
209
210         short_date_possible_extractor_element = None
211         for date_possible_extractor_element in
    date_possible_extractor_elements:
212             if short_date_possible_extractor_element is None or len(
    date_possible_extractor_element[0]) < \
213                     len(short_date_possible_extractor_element[0]):
214                 short_date_possible_extractor_element =
    date_possible_extractor_element
215
216         self.__date_extractor_element =
    short_date_possible_extractor_element
217
218     def load_articles_with_date(self):
219         articles_with_date = set()
```

```
220         for example_article in self.__example_articles:
221             if example_article.get_date() is not None and example_article.
    get_date != '':
222                 articles_with_date.add(example_article)
223         return articles_with_date
224 
225     def search_possible_date_extractor_elements(self):
226         example_article = self.__example_articles[0]
227         date_possible_extractor_elements = set()
228         possible_extractor_elements = set()
229         date_pattern_regex_extractor_generator =
    DatePatternRegexExtractorGenerator(example_article.get_date())
230         date_elements = date_pattern_regex_extractor_generator.
    generate_patterns_regex_extractor()
231         soup = BeautifulSoup(self.__full_html_content[example_article.
    get_url()], 'html5lib')
232         for tag in soup.find_all():
233             for element in date_elements:
234                 if tag.text.__contains__(element[0]):
235                     possible_extractor_elements.add((tag, element))
236 
237         for possible_extractor_element in possible_extractor_elements:
238             element = soup.find(possible_extractor_element[0].name,
    possible_extractor_element[0].attrs)
239             if isinstance(element, bs4.Tag):
240                 text = element.text
241                 result = re.search(possible_extractor_element[1][2], text)
242                 if result is not None and result.group() ==
    possible_extractor_element[1][0]:
243                     date_possible_extractor_elements.add(
244                         (element, possible_extractor_element[1][1],
    possible_extractor_element[1][2]))
245 
246         return date_possible_extractor_elements
247 
248     def load_full_html_content_dict(self):
249         full_html_content_dict = {}
250         headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv
    :50.0) Gecko/20100101 Firefox/50.0'}
251         for example_article in self.__example_articles:
252             response = requests.get(example_article.get_url(), headers=
    headers)
253             html = response.content
254             full_html_content_dict[example_article.get_url()] = html
255         return full_html_content_dict
```

Extractor.py

```
import bs4
import ExtractionConfig
import requests
from bs4 import BeautifulSoup
import Article
import re
import DefaultDatePatternConverter

# Receives the extraction configuration and extracts the fields
class Extractor:
    def __init__(self, extraction_config: ExtractionConfig):
        self.__extraction_config = extraction_config

    def extract(self, url_to_extract):
        headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:50.0) Gecko/20100101 Firefox/50.0'}
        response = requests.get(url_to_extract, headers=headers)
        html = response.content
        soup = BeautifulSoup(html, 'html5lib')
        try:
            title = self.extract_title(soup)
            subtitle = self.extract_subtitle(soup)
            content = self.extract_content(soup)
            author = self.extract_author(soup)
            date = self.extract_date(soup)
        except Exception as e:
            print(e)
        return Article.Article(url_to_extract, title, content, author=author, subtitle=subtitle, date=date)

    def extract_title(self, soup: BeautifulSoup):
        extract_element = soup.find(self.__extraction_config.get_title_extractor().name,
        self.__extraction_config.get_title_extractor().attrs)
        if extract_element is not None:
            return extract_element.text
        raise Exception("Unable to extract title")

    def extract_subtitle(self, soup: BeautifulSoup):
        extract_element = None
        if self.__extraction_config.get_subtitle_extractor() is not None:
            extract_element = soup.find(self.__extraction_config.get_subtitle_extractor().name,
            self.__extraction_config.get_subtitle_extractor().attrs)
        if extract_element is not None:
            return extract_element.text
```

```
43          return None
44
45      def extract_content(self, soup: BeautifulSoup):
46          possible_elements = soup.find_all(self.__extraction_config.
    get_content_extractor().name, self.__extraction_config.
    get_content_extractor().attrs)
47          extract_element = None
48          for possible_element in possible_elements:
49              if possible_element.attrs == self.__extraction_config.
    get_content_extractor().attrs and \
50                      possible_element.name == self.__extraction_config.
    get_content_extractor().name:
51                  extract_element = possible_element
52                  break
53          cleaner_elements = self.__extraction_config.
    get_content_cleaner_element()
54
55          for tag in extract_element.find_all():
56              for cleaner_element in cleaner_elements:
57                  if tag.name == 'div' and str(tag.attrs) == '{}':
58                      continue
59                  if tag.name == cleaner_element[0] and str(tag.attrs) ==
    cleaner_element[1]:
60                      tag.decompose()
61
62          for tag in extract_element.find_all(['style', 'script']):
63              tag.decompose()
64
65          content = ''
66          if extract_element is not None:
67              for element in extract_element:
68                  if isinstance(element, bs4.Tag):
69                      block = element.text.strip()
70                      if block == '':
71                          continue
72                      content += block + '\n\n'
73              return content
74          raise Exception("Unable to extract content")
75
76      def extract_author(self, soup: BeautifulSoup):
77          extract_element = None
78          if self.__extraction_config.get_author_extractor() is not None:
79              extract_element = soup.find(self.__extraction_config.
    get_author_extractor().name, self.__extraction_config.
    get_author_extractor().attrs)
80          if extract_element is not None:
81              return extract_element.text
```

```
82          return None
83
84      def extract_date(self, soup: BeautifulSoup):
85          if self.__extraction_config.get_date_extractor() is None:
86              return None
87          tag_name = self.__extraction_config.get_date_extractor()[0].name
88          tag_attrs = self.__extraction_config.get_date_extractor()[0].attrs
89          extract_element = soup.find(tag_name, tag_attrs)
90          if extract_element is not None:
91              date_regex = self.__extraction_config.get_date_extractor()[2]
92              date_format = self.__extraction_config.get_date_extractor()[1]
93              date_result = re.search(date_regex, extract_element.text)
94              if date_result is not None:
95                  date = date_result.group()
96                  default_date_pattern_converter =
    DefaultDatePatternConverter.DefaultDatePatternConverter((date,
    date_format))
97                  return default_date_pattern_converter.convert_date()
98          return None
```

LinkFinder.py

```
1  import re
2  import requests
3  import urllib.parse
4  from bs4 import BeautifulSoup
5
6  # Search for urls within a webpage
7  class LinkFinder:
8      def __init__(self, url_to_extract: str):
9          self.__url_to_extract = url_to_extract
10         self.__VALID_LINK_PATTERN = 'https?://.*'
11
12     def find_links(self) -> set:
13         possible_links = set()
14         headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv
    :50.0) Gecko/20100101 Firefox/50.0'}
15         response = requests.get(self.__url_to_extract, headers=headers)
16         html_from_response = response.content
17         soup = BeautifulSoup(html_from_response, 'html5lib')
18         for tag in soup.find_all(href=True):
19             possible_link = tag.get('href')
20             if str(possible_link).startswith('/'):
21                 possible_links.add(self.__url_to_extract + str(
    possible_link))
22             else:
23                 possible_links.add(str(possible_link))
```

```
24          links = self.__remove_no_links(possible_links)
25          links = self.__remove_links_from_another_domain(possible_links)
26          return links
27
28      def __remove_no_links(self, possible_links: set) -> set:
29          interactive_possible_links = possible_links.copy()
30          for possible_link in interactive_possible_links:
31              if not bool(re.fullmatch(self.__VALID_LINK_PATTERN,
    possible_link)):
32                  possible_links.remove(possible_link)
33          return possible_links
34
35      def __remove_links_from_another_domain(self, possible_links: set) ->
    set:
36          interactive_possible_links = possible_links.copy()
37          website_domain = urllib.parse.urlparse(self.__url_to_extract).
    netloc
38          for possible_link in interactive_possible_links:
39              if not urllib.parse.urlparse(possible_link).netloc ==
    website_domain:
40                  possible_links.remove(possible_link)
41          return possible_links
```

SharedCrawler.py

```
1  import threading
2  import WebsiteExecutor
3  from PyQt5.QtWidgets import QMessageBox
4  import TermController
5  from ArticleDao import delete_all_articles, get_all_articles
6  import EmailSender
7
8  """ Crawler itself. Receives the seed URLs, how long each website will be
       browsed and the user's terms of interest """
9  class SharedCrawler(threading.Thread):
10     def __init__(self, name, start_urls, execution_seconds_per_website: int
       , term_controller: TermController, emails_to_be_sent: set()):
11         threading.Thread.__init__(self)
12         self.__name = name
13         self.__start_urls = start_urls
14         self.__execution_seconds_per_website =
       execution_seconds_per_website
15         self.__term_controller = term_controller
16         self.__email_to_be_sent = emails_to_be_sent
17
18     def run(self):
19         for start_url in self.__start_urls:
```

```
20              print("Iniciando coleta para " + start_url + "\n")
21              website_executor = WebsiteExecutor.WebsiteExecutor(start_url,
    self.__execution_seconds_per_website, self.__term_controller)
22              try:
23                  website_executor.execute()
24              except Exception as e:
25                  print(e)
26          try:
27              self.send_email()
28          except Exception as e:
29              print(e)
30          delete_all_articles()
31          QMessageBox.information(None, "INFO", "Execucao do crawler " + self
    .__name + " finalizada!")
32
33      def send_email(self):
34          message = "Relatorio de coleta para o Crawler " + self.__name + ":
    \n\n"
35          message += str(self.__term_controller)
36          message += self.get_articles_url()
37          message = message.encode("ascii", errors="replace")
38          email_sender = EmailSender.EmailSender("Relatorio coleta: Crawler "
     + self.__name, message.decode(), list(self.__email_to_be_sent))
39          email_sender.send()
40
41      def get_articles_url(self):
42          urls = "URLs: \n"
43          for article in get_all_articles():
44              urls += article[0] + "\n"
45          return urls
```

TermController.py

```
1  import Article
2
3  # User terms of interest controller
4  class TermController:
5      def __init__(self, all_this_words: set(), exactly_this_phrase: str,
    any_of_this_words: set()):
6          self.__all_this_words = all_this_words
7          self.__exactly_this_phrase = exactly_this_phrase
8          self.__any_of_this_words = any_of_this_words
9
10     def have_all_this_words(self, article: Article):
11         for word in self.__any_of_this_words:
12             if article.get_title().lower() is not None and word.lower() not
     in article.get_title().lower():
```

```
13                  return False
14              if article.get_subtitle().lower() is not None and word.lower()
    not in article.get_subtitle().lower():
15                  return False
16              if article.get_content().lower() is not None and word.lower()
    not in article.get_content().lower():
17                  return False
18              if article.get_author().lower() is not None and word.lower()
    not in article.get_author().lower():
19                  return False
20              if article.get_date().lower() is not None and word.lower() not
    in article.get_date().lower():
21                  return False
22          return True
23
24      def has_exactly_this_phrase(self, article: Article):
25          if article.get_title().lower() is not None and self.
    __exactly_this_phrase.lower() in article.get_title().lower():
26              return True
27          if article.get_subtitle().lower() is not None and self.
    __exactly_this_phrase.lower() in article.get_subtitle().lower():
28              return True
29          if article.get_content().lower() is not None and self.
    __exactly_this_phrase.lower() in article.get_content().lower():
30              return True
31          if article.get_author().lower() is not None and self.
    __exactly_this_phrase.lower() in article.get_author().lower():
32              return True
33          if article.get_date().lower() is not None and self.
    __exactly_this_phrase.lower() in article.get_date().lower():
34              return True
35
36      def have_any_of_this_words(self, article: Article):
37          for word in self.__any_of_this_words:
38              if article.get_title().lower() is not None and word.lower() in
    article.get_title().lower():
39                  return True
40              if article.get_subtitle().lower() is not None and word.lower()
    in article.get_subtitle().lower():
41                  return True
42              if article.get_content().lower() is not None and word.lower()
    in article.get_content().lower():
43                  return True
44              if article.get_author().lower() is not None and word.lower() in
     article.get_author().lower():
45                  return True
46              if article.get_date().lower() is not None and word.lower() in
```

```
    article.get_date().lower():
47                  return True
48          return False
49
50      def is_valid(self, article: Article):
51          return self.have_any_of_this_words(article) or self.
    has_exactly_this_phrase(article) or \
52                 self.have_all_this_words(article)
53
54      def __str__(self):
55          to_string = "Filtros de interesse: \n"
56          to_string += "Todas estas palavras: " + str(self.__all_this_words)
    + "\n"
57          to_string += "Quaisquer destas palavras: " + str(self.
    __any_of_this_words) + "\n"
58          to_string += "Exatamente esta frase: " + self.__exactly_this_phrase
     + "\n\n"
59          return to_string
```

ui.py

```
1  import re
2  import resources_rc
3  from PyQt5 import QtCore, QtGui, QtWidgets
4  from PyQt5.QtWidgets import QMessageBox, QInputDialog, QLineEdit
5  import EmailSenderConfig as emailFields
6  from Article import Article
7  from Website import WebsiteConfigGenerator
8  from sqlite3 import IntegrityError
9  from WebsiteDao import get_all_websites
10 import DedicatedCrawler
11 import SharedCrawler
12 import TermController
13
14 # Graphical User Interface
15 class Ui_MainWindow(object):
16     def setupUi(self, MainWindow):
17         MainWindow.setObjectName("MainWindow")
18         MainWindow.resize(793, 478)
19         self.examplesArticlesDict = {}
20         self.centralwidget = QtWidgets.QWidget(MainWindow)
21         self.centralwidget.setFocusPolicy(QtCore.Qt.ClickFocus)
22         self.centralwidget.setAutoFillBackground(False)
23         self.centralwidget.setStyleSheet("#leftMenu {\n"
24 "      background-color:#67C0CE;\n"
25 "     border-radius: 10px;\n"
26 "}\n"
```

```
"\n"
"#mainBody {\n"
"     background-color:#DBE6E7;\n"
"     border-radius: 10px\n"
"}")
        self.centralwidget.setObjectName("centralwidget")
        self.horizontalLayout = QtWidgets.QHBoxLayout(self.centralwidget)
        self.horizontalLayout.setContentsMargins(0, 0, 0, 0)
        self.horizontalLayout.setSpacing(0)
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.leftMenu = QtWidgets.QWidget(self.centralwidget)
        self.leftMenu.setEnabled(True)
        self.leftMenu.setMaximumSize(QtCore.QSize(200, 16777215))
        self.leftMenu.setObjectName("leftMenu")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.leftMenu)
        self.verticalLayout.setObjectName("verticalLayout")
        self.crawlExframe = QtWidgets.QFrame(self.leftMenu)
        self.crawlExframe.setMaximumSize(QtCore.QSize(16777215, 60))
        font = QtGui.QFont()
        font.setKerning(True)
        self.crawlExframe.setFont(font)
        self.crawlExframe.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.crawlExframe.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.crawlExframe.setLineWidth(0)
        self.crawlExframe.setObjectName("crawlExframe")
        self.verticalLayout_2 = QtWidgets.QVBoxLayout(self.crawlExframe)
        self.verticalLayout_2.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout_2.setSpacing(0)
        self.verticalLayout_2.setObjectName("verticalLayout_2")
        self.crawlExlabel = QtWidgets.QLabel(self.crawlExframe)
        font = QtGui.QFont()
        font.setFamily("Century Schoolbook L")
        font.setPointSize(16)
        font.setItalic(True)
        self.crawlExlabel.setFont(font)
        self.crawlExlabel.setObjectName("crawlExlabel")
        self.verticalLayout_2.addWidget(self.crawlExlabel, 0, QtCore.Qt.
    AlignHCenter)
        self.verticalLayout.addWidget(self.crawlExframe)
        self.buttonsFrame = QtWidgets.QFrame(self.leftMenu)
        self.buttonsFrame.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.buttonsFrame.setFrameShadow(QtWidgets.QFrame.Plain)
        self.buttonsFrame.setLineWidth(0)
        self.buttonsFrame.setObjectName("buttonsFrame")
        self.verticalLayout_3 = QtWidgets.QVBoxLayout(self.buttonsFrame)
        self.verticalLayout_3.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout_3.setSpacing(6)
```

```
73          self.verticalLayout_3.setObjectName("verticalLayout_3")
74          self.inicioWidget = QtWidgets.QWidget(self.buttonsFrame)
75          self.inicioWidget.setEnabled(True)
76          self.inicioWidget.setObjectName("inicioWidget")
77          self.horizontalLayout_5 = QtWidgets.QHBoxLayout(self.inicioWidget)
78          self.horizontalLayout_5.setObjectName("horizontalLayout_5")
79          self.inicioButton = QtWidgets.QPushButton(self.inicioWidget)
80          self.inicioButton.setMinimumSize(QtCore.QSize(134, 0))
81          font = QtGui.QFont()
82          font.setPointSize(10)
83          self.inicioButton.setFont(font)
84          self.inicioButton.setObjectName("inicioButton")
85          self.horizontalLayout_5.addWidget(self.inicioButton)
86          self.inicioIcon = QtWidgets.QLabel(self.inicioWidget)
87          self.inicioIcon.setText("")
88          self.inicioIcon.setPixmap(QtGui.QPixmap(":/whiteIcons/assets/
    whiteIcons/home.svg"))
89          self.inicioIcon.setObjectName("inicioIcon")
90          self.horizontalLayout_5.addWidget(self.inicioIcon)
91          self.verticalLayout_3.addWidget(self.inicioWidget, 0, QtCore.Qt.
    AlignRight)
92          self.fornecerWidget = QtWidgets.QWidget(self.buttonsFrame)
93          self.fornecerWidget.setObjectName("fornecerWidget")
94          self.horizontalLayout_2 = QtWidgets.QHBoxLayout(self.fornecerWidget
    )
95          self.horizontalLayout_2.setObjectName("horizontalLayout_2")
96          self.fornecerButton = QtWidgets.QPushButton(self.fornecerWidget)
97          self.fornecerButton.setMinimumSize(QtCore.QSize(134, 0))
98          self.fornecerButton.setObjectName("fornecerButton")
99          self.horizontalLayout_2.addWidget(self.fornecerButton)
100         self.fornecerIcon = QtWidgets.QLabel(self.fornecerWidget)
101         self.fornecerIcon.setText("")
102         self.fornecerIcon.setPixmap(QtGui.QPixmap(":/whiteIcons/assets/
    whiteIcons/edit.svg"))
103         self.fornecerIcon.setObjectName("fornecerIcon")
104         self.horizontalLayout_2.addWidget(self.fornecerIcon)
105         self.verticalLayout_3.addWidget(self.fornecerWidget, 0, QtCore.Qt.
    AlignRight)
106         self.carregarWidget = QtWidgets.QWidget(self.buttonsFrame)
107         self.carregarWidget.setObjectName("carregarWidget")
108         self.horizontalLayout_6 = QtWidgets.QHBoxLayout(self.carregarWidget
    )
109         self.horizontalLayout_6.setObjectName("horizontalLayout_6")
110         self.carregarButton = QtWidgets.QPushButton(self.carregarWidget)
111         self.carregarButton.setMinimumSize(QtCore.QSize(134, 0))
112         self.carregarButton.setObjectName("carregarButton")
113         self.horizontalLayout_6.addWidget(self.carregarButton)
```

```
114         self.carregarIcon = QtWidgets.QLabel(self.carregarWidget)
115         self.carregarIcon.setText("")
116         self.carregarIcon.setPixmap(QtGui.QPixmap(":/whiteIcons/assets/
    whiteIcons/loader.svg"))
117         self.carregarIcon.setObjectName("carregarIcon")
118         self.horizontalLayout_6.addWidget(self.carregarIcon)
119         self.verticalLayout_3.addWidget(self.carregarWidget, 0, QtCore.Qt.
    AlignRight)
120         self.inicializarWidget = QtWidgets.QWidget(self.buttonsFrame)
121         self.inicializarWidget.setObjectName("inicializarWidget")
122         self.horizontalLayout_7 = QtWidgets.QHBoxLayout(self.
    inicializarWidget)
123         self.horizontalLayout_7.setObjectName("horizontalLayout_7")
124         self.execute_crawler_button = QtWidgets.QPushButton(self.
    inicializarWidget)
125         self.execute_crawler_button.setObjectName("inicializarButton")
126         self.horizontalLayout_7.addWidget(self.execute_crawler_button)
127         self.inicializarIcon = QtWidgets.QLabel(self.inicializarWidget)
128         self.inicializarIcon.setText("")
129         self.inicializarIcon.setPixmap(QtGui.QPixmap(":/whiteIcons/assets/
    whiteIcons/play.svg"))
130         self.inicializarIcon.setObjectName("inicializarIcon")
131         self.horizontalLayout_7.addWidget(self.inicializarIcon)
132         self.verticalLayout_3.addWidget(self.inicializarWidget)
133         self.configuracoesWidget = QtWidgets.QWidget(self.buttonsFrame)
134         self.configuracoesWidget.setObjectName("configuracoesWidget")
135         self.horizontalLayout_8 = QtWidgets.QHBoxLayout(self.
    configuracoesWidget)
136         self.horizontalLayout_8.setObjectName("horizontalLayout_8")
137         self.settings_button = QtWidgets.QPushButton(self.
    configuracoesWidget)
138         self.settings_button.setMinimumSize(QtCore.QSize(134, 0))
139         self.settings_button.setObjectName("pushButton_5")
140         self.horizontalLayout_8.addWidget(self.settings_button)
141         self.label_3 = QtWidgets.QLabel(self.configuracoesWidget)
142         self.label_3.setText("")
143         self.label_3.setPixmap(QtGui.QPixmap(":/whiteIcons/assets/
    whiteIcons/settings.svg"))
144         self.label_3.setObjectName("label_3")
145         self.horizontalLayout_8.addWidget(self.label_3)
146         self.verticalLayout_3.addWidget(self.configuracoesWidget, 0, QtCore
    .Qt.AlignRight)
147         self.versaoWidget = QtWidgets.QWidget(self.buttonsFrame)
148         self.versaoWidget.setObjectName("versaoWidget")
149         self.verticalLayout_4 = QtWidgets.QVBoxLayout(self.versaoWidget)
150         self.verticalLayout_4.setObjectName("verticalLayout_4")
151         self.versaoLabel = QtWidgets.QLabel(self.versaoWidget)
```

```
152         font = QtGui.QFont()
153         font.setFamily("Nimbus Mono L")
154         font.setPointSize(12)
155         self.versaoLabel.setFont(font)
156         self.versaoLabel.setObjectName("versaoLabel")
157         self.verticalLayout_4.addWidget(self.versaoLabel, 0, QtCore.Qt.
    AlignHCenter)
158         self.verticalLayout_3.addWidget(self.versaoWidget)
159         self.verticalLayout.addWidget(self.buttonsFrame)
160         self.horizontalLayout.addWidget(self.leftMenu)
161         self.mainBody = QtWidgets.QWidget(self.centralwidget)
162         self.mainBody.setFocusPolicy(QtCore.Qt.NoFocus)
163         self.mainBody.setAcceptDrops(False)
164         self.verticalLayout_5 = QtWidgets.QVBoxLayout(self.mainBody)
165         self.verticalLayout_5.setObjectName("verticalLayout_5")
166         self.create_settings_screen()
167         self.createCarregarBody()
168         self.create_article_examples_screen()
169         self.create_configure_crawler_screen()
170         self.create_entry_point_screen()
171         self.horizontalLayout.addWidget(self.mainBody)
172         MainWindow.setCentralWidget(self.centralwidget)
173         self.setupButtons()
174         self.retranslateUi(MainWindow)
175         QtCore.QMetaObject.connectSlotsByName(MainWindow)
176 
177     def create_popup(self, window_title: str, message: str):
178         msg = QMessageBox()
179         msg.setWindowTitle(window_title)
180         msg.setText(message)
181         msg.exec_()
182 
183     # Configuration screen
184     def create_settings_screen(self):
185         self.mainBody.setObjectName("mainBody")
186         self.label_4 = QtWidgets.QLabel(self.mainBody)
187         self.label_4.setMaximumSize(QtCore.QSize(16777215, 50))
188         self.label_4.setObjectName("label_4")
189         self.verticalLayout_5.addWidget(self.label_4, 0, QtCore.Qt.
    AlignHCenter)
190         self.configsWidget = QtWidgets.QWidget(self.mainBody)
191         self.configsWidget.setMaximumSize(QtCore.QSize(16777215, 220))
192         self.configsWidget.setObjectName("configsWidget")
193         self.verticalLayout_6 = QtWidgets.QVBoxLayout(self.configsWidget)
194         self.verticalLayout_6.setObjectName("verticalLayout_6")
195         self.widget_5 = QtWidgets.QWidget(self.configsWidget)
196         self.widget_5.setObjectName("widget_5")
```

```
197         self.horizontalLayout_3 = QtWidgets.QHBoxLayout(self.widget_5)
198         self.horizontalLayout_3.setObjectName("horizontalLayout_3")
199         self.label = QtWidgets.QLabel(self.widget_5)
200         self.label.setObjectName("label")
201         self.horizontalLayout_3.addWidget(self.label)
202         self.lineEdit = QtWidgets.QLineEdit(self.widget_5)
203         self.lineEdit.setObjectName("lineEdit")
204         self.horizontalLayout_3.addWidget(self.lineEdit)
205         self.verticalLayout_6.addWidget(self.widget_5)
206         self.widget_4 = QtWidgets.QWidget(self.configsWidget)
207         self.widget_4.setObjectName("widget_4")
208         self.horizontalLayout_4 = QtWidgets.QHBoxLayout(self.widget_4)
209         self.horizontalLayout_4.setObjectName("horizontalLayout_4")
210         self.labelPassword = QtWidgets.QLabel(self.widget_4)
211         self.labelPassword.setObjectName("labelPassword")
212         self.horizontalLayout_4.addWidget(self.labelPassword)
213         self.lineEdit_2 = QtWidgets.QLineEdit(self.widget_4)
214         self.lineEdit_2.setObjectName("lineEdit_2")
215         self.horizontalLayout_4.addWidget(self.lineEdit_2)
216         self.verticalLayout_6.addWidget(self.widget_4)
217         self.checkBox = QtWidgets.QCheckBox(self.configsWidget)
218         self.checkBox.setObjectName("checkBox")
219         self.verticalLayout_6.addWidget(self.checkBox)
220         self.pushButton = QtWidgets.QPushButton(self.configsWidget)
221         self.pushButton.setMaximumSize(QtCore.QSize(300, 16777215))
222         self.pushButton.setObjectName("pushButton")
223         self.verticalLayout_6.addWidget(self.pushButton, 0, QtCore.Qt.
    AlignHCenter)
224         self.verticalLayout_5.addWidget(self.configsWidget)
225         self.close_settings_screen()
226 
227     def restore_settings_default_value(self):
228         if self.checkBox.isChecked():
229             self.lineEdit.setEnabled(True)
230             self.lineEdit_2.setEnabled(True)
231         else:
232             self.lineEdit.setEnabled(False)
233             self.lineEdit_2.setEnabled(False)
234 
235     def load_settings_value(self):
236         self.lineEdit.setText(emailFields.email)
237         self.lineEdit_2.setText(emailFields.password)
238         self.checkBox.setChecked(False)
239         self.lineEdit.setEnabled(False)
240         self.lineEdit_2.setEnabled(False)
241 
242     def open_settings_screen(self):
```

```
243         self.load_settings_value()
244         self.configsWidget.show()
245         self.label_4.show()
246 
247     def close_settings_screen(self):
248         self.label_4.close()
249         self.configsWidget.close()
250 
251     def update_email_fields(self):
252         emailFields.email = self.lineEdit.text().strip()
253         emailFields.password = self.lineEdit_2.text().strip()
254         with open('EmailSenderConfig.py', 'r+') as file:
255             data = file.readlines()
256             data[2] = 'email=\"' + emailFields.email + '\"\n'
257             data[3] = 'password=\"' + emailFields.password + '\"\n'
258             file.seek(0)
259             file.writelines(data)
260             file.close()
261         self.load_settings_value()
262 
263     # Articles examples screen
264     def create_article_examples_screen(self):
265         self.labelInfoExamples = QtWidgets.QLabel(self.mainBody)
266         self.labelInfoExamples.setMaximumSize(QtCore.QSize(16777215, 25))
267         self.labelInfoExamples.setObjectName("label")
268         self.verticalLayout_5.addWidget(self.labelInfoExamples, 0, QtCore.
    Qt.AlignHCenter|QtCore.Qt.AlignVCenter)
269         self.examplesBody = QtWidgets.QWidget(self.mainBody)
270         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
     QtWidgets.QSizePolicy.Preferred)
271         sizePolicy.setHorizontalStretch(0)
272         sizePolicy.setVerticalStretch(0)
273         sizePolicy.setHeightForWidth(self.examplesBody.sizePolicy().
    hasHeightForWidth())
274         self.examplesBody.setSizePolicy(sizePolicy)
275         self.examplesBody.setObjectName("examplesBody")
276         self.horizontalLayout_3 = QtWidgets.QHBoxLayout(self.examplesBody)
277         self.horizontalLayout_3.setObjectName("horizontalLayout_3")
278         self.exampleFieldsWidget = QtWidgets.QWidget(self.examplesBody)
279         self.exampleFieldsWidget.setObjectName("exampleFieldsWidget")
280         self.verticalLayout_6 = QtWidgets.QVBoxLayout(self.
    exampleFieldsWidget)
281         self.verticalLayout_6.setContentsMargins(0, 0, 0, 0)
282         self.verticalLayout_6.setSpacing(0)
283         self.verticalLayout_6.setObjectName("verticalLayout_6")
284         self.urlWidget = QtWidgets.QWidget(self.exampleFieldsWidget)
285         self.urlWidget.setObjectName("urlWidget")
```

```
286         self.horizontalLayout_4 = QtWidgets.QHBoxLayout(self.urlWidget)
287         self.horizontalLayout_4.setContentsMargins(0, 0, 0, 0)
288         self.horizontalLayout_4.setSpacing(0)
289         self.horizontalLayout_4.setObjectName("horizontalLayout_4")
290         self.urlLabel = QtWidgets.QLabel(self.urlWidget)
291         self.urlLabel.setMinimumSize(QtCore.QSize(59, 0))
292         self.urlLabel.setObjectName("urlLabel")
293         self.horizontalLayout_4.addWidget(self.urlLabel, 0, QtCore.Qt.
    AlignRight)
294         self.urlLineField = QtWidgets.QLineEdit(self.urlWidget)
295         self.urlLineField.setObjectName("urlLineField")
296         self.horizontalLayout_4.addWidget(self.urlLineField)
297         self.verticalLayout_6.addWidget(self.urlWidget)
298         self.titleWidget = QtWidgets.QWidget(self.exampleFieldsWidget)
299         self.titleWidget.setObjectName("titleWidget")
300         self.horizontalLayout_9 = QtWidgets.QHBoxLayout(self.titleWidget)
301         self.horizontalLayout_9.setContentsMargins(0, 9, 0, 0)
302         self.horizontalLayout_9.setSpacing(0)
303         self.horizontalLayout_9.setObjectName("horizontalLayout_9")
304         self.titleLabel = QtWidgets.QLabel(self.titleWidget)
305         self.titleLabel.setEnabled(True)
306         self.titleLabel.setMinimumSize(QtCore.QSize(59, 0))
307         self.titleLabel.setObjectName("titleLabel")
308         self.horizontalLayout_9.addWidget(self.titleLabel)
309         self.titleLineField = QtWidgets.QLineEdit(self.titleWidget)
310         self.titleLineField.setObjectName("titleLineField")
311         self.horizontalLayout_9.addWidget(self.titleLineField)
312         self.verticalLayout_6.addWidget(self.titleWidget)
313         self.contentWidget = QtWidgets.QWidget(self.exampleFieldsWidget)
314         self.contentWidget.setObjectName("contentWidget")
315         self.horizontalLayout_12 = QtWidgets.QHBoxLayout(self.contentWidget
    )
316         self.horizontalLayout_12.setContentsMargins(0, -1, 0, 0)
317         self.horizontalLayout_12.setSpacing(0)
318         self.horizontalLayout_12.setObjectName("horizontalLayout_12")
319         self.contentLabel = QtWidgets.QLabel(self.contentWidget)
320         self.contentLabel.setObjectName("contentLabel")
321         self.horizontalLayout_12.addWidget(self.contentLabel)
322         self.contentPlainText = QtWidgets.QPlainTextEdit(self.contentWidget
    )
323         self.contentPlainText.setObjectName("contentPlainText")
324         self.horizontalLayout_12.addWidget(self.contentPlainText)
325         self.verticalLayout_6.addWidget(self.contentWidget)
326         self.subtitleWidget = QtWidgets.QWidget(self.exampleFieldsWidget)
327         self.subtitleWidget.setObjectName("subtitleWidget")
328         self.horizontalLayout_10 = QtWidgets.QHBoxLayout(self.
    subtitleWidget)
```

```
329         self.horizontalLayout_10.setContentsMargins(0, -1, 0, 0)
330         self.horizontalLayout_10.setSpacing(0)
331         self.horizontalLayout_10.setObjectName("horizontalLayout_10")
332         self.subtitleLabel = QtWidgets.QLabel(self.subtitleWidget)
333         self.subtitleLabel.setMinimumSize(QtCore.QSize(59, 0))
334         self.subtitleLabel.setObjectName("subtitleLabel")
335         self.horizontalLayout_10.addWidget(self.subtitleLabel)
336         self.subtitleLineField = QtWidgets.QLineEdit(self.subtitleWidget)
337         self.subtitleLineField.setObjectName("subtitleLineField")
338         self.horizontalLayout_10.addWidget(self.subtitleLineField)
339         self.verticalLayout_6.addWidget(self.subtitleWidget)
340         self.authorWidget = QtWidgets.QWidget(self.exampleFieldsWidget)
341         self.authorWidget.setObjectName("authorWidget")
342         self.horizontalLayout_11 = QtWidgets.QHBoxLayout(self.authorWidget)
343         self.horizontalLayout_11.setContentsMargins(0, -1, 0, 0)
344         self.horizontalLayout_11.setSpacing(0)
345         self.horizontalLayout_11.setObjectName("horizontalLayout_11")
346         self.authorLabel = QtWidgets.QLabel(self.authorWidget)
347         self.authorLabel.setMinimumSize(QtCore.QSize(59, 0))
348         self.authorLabel.setObjectName("authorLabel")
349         self.horizontalLayout_11.addWidget(self.authorLabel, 0, QtCore.Qt.
    AlignHCenter|QtCore.Qt.AlignVCenter)
350         self.authorLineField = QtWidgets.QLineEdit(self.authorWidget)
351         self.authorLineField.setObjectName("authorLineField")
352         self.horizontalLayout_11.addWidget(self.authorLineField)
353         self.verticalLayout_6.addWidget(self.authorWidget)
354         self.dateWidget = QtWidgets.QWidget(self.exampleFieldsWidget)
355         self.dateWidget.setObjectName("dateWidget")
356         self.horizontalLayout_13 = QtWidgets.QHBoxLayout(self.dateWidget)
357         self.horizontalLayout_13.setContentsMargins(0, 9, 0, 9)
358         self.horizontalLayout_13.setSpacing(0)
359         self.horizontalLayout_13.setObjectName("horizontalLayout_13")
360         self.dateLineLabel = QtWidgets.QLabel(self.dateWidget)
361         self.dateLineLabel.setMinimumSize(QtCore.QSize(59, 0))
362         self.dateLineLabel.setObjectName("dateLineLabel")
363         self.horizontalLayout_13.addWidget(self.dateLineLabel)
364         self.dateLineField = QtWidgets.QLineEdit(self.dateWidget)
365         self.dateLineField.setObjectName("dateLineField")
366         self.dateLineField.setEnabled(False)
367         self.horizontalLayout_13.addWidget(self.dateLineField)
368         self.dateButton = QtWidgets.QToolButton(self.dateWidget)
369         self.dateButton.setObjectName("dateButton")
370         self.horizontalLayout_13.addWidget(self.dateButton)
371         self.verticalLayout_6.addWidget(self.dateWidget)
372         self.generate_web_site_config_button = QtWidgets.QPushButton(self.
    exampleFieldsWidget)
373         self.generate_web_site_config_button.setObjectName("
```

```
    generate_web_site_config_button")
374         self.verticalLayout_6.addWidget(self.
    generate_web_site_config_button)
375         self.horizontalLayout_3.addWidget(self.exampleFieldsWidget)
376         self.articlesExamplesAddedWidget = QtWidgets.QWidget(self.
    examplesBody)
377         self.articlesExamplesAddedWidget.setMinimumSize(QtCore.QSize(100,
    0))
378         self.articlesExamplesAddedWidget.setMaximumSize(QtCore.QSize(125,
    16777215))
379         self.articlesExamplesAddedWidget.setObjectName("
    articlesExamplesAddedWidget")
380         self.verticalLayout_7 = QtWidgets.QVBoxLayout(self.
    articlesExamplesAddedWidget)
381         self.verticalLayout_7.setContentsMargins(0, 0, 0, 0)
382         self.verticalLayout_7.setSpacing(9)
383         self.verticalLayout_7.setObjectName("verticalLayout_7")
384         self.label_2 = QtWidgets.QLabel(self.articlesExamplesAddedWidget)
385         self.label_2.setObjectName("label_2")
386         self.verticalLayout_7.addWidget(self.label_2, 0, QtCore.Qt.
    AlignHCenter)
387         self.articlesListWidget = QtWidgets.QListWidget(self.
    articlesExamplesAddedWidget)
388         self.articlesListWidget.setObjectName("articlesListWidget")
389         self.articlesListWidget.itemDoubleClicked.connect(self.viewExample)
390         self.verticalLayout_7.addWidget(self.articlesListWidget)
391         self.articlesListButtonsWidget = QtWidgets.QWidget(self.
    articlesExamplesAddedWidget)
392         self.articlesListButtonsWidget.setObjectName("
    articlesListButtonsWidget")
393         self.horizontalLayout_14 = QtWidgets.QHBoxLayout(self.
    articlesListButtonsWidget)
394         self.horizontalLayout_14.setSizeConstraint(QtWidgets.QLayout.
    SetDefaultConstraint)
395         self.horizontalLayout_14.setContentsMargins(0, 0, 0, 0)
396         self.horizontalLayout_14.setSpacing(0)
397         self.horizontalLayout_14.setObjectName("horizontalLayout_14")
398         self.removeArticleExampleButton = QtWidgets.QPushButton(self.
    articlesListButtonsWidget)
399         self.removeArticleExampleButton.setObjectName("
    removeArticleExampleButton")
400         self.removeArticleExampleButton.setIcon(QtGui.QIcon(":/whiteIcons/
    assets/redIcons/x.svg"))
401         self.horizontalLayout_14.addWidget(self.removeArticleExampleButton)
402         self.finalizeExampleButton = QtWidgets.QPushButton(self.
    articlesListButtonsWidget)
403         self.finalizeExampleButton.setObjectName("finalizeExampleButton")
```

```
404         self.finalizeExampleButton.setIcon(QtGui.QIcon(":/whiteIcons/assets
    /greenIcons/check.svg"))
405         self.horizontalLayout_14.addWidget(self.finalizeExampleButton)
406         self.verticalLayout_7.addWidget(self.articlesListButtonsWidget)
407         self.horizontalLayout_3.addWidget(self.articlesExamplesAddedWidget)
408         self.verticalLayout_5.addWidget(self.examplesBody)
409         self.close_articles_examples_screen()
410 
411     def close_articles_examples_screen(self):
412         self.labelInfoExamples.close()
413         self.examplesBody.close()
414 
415     def open_articles_examples_screen(self):
416         self.labelInfoExamples.show()
417         self.examplesBody.show()
418 
419     def show_calendar(self):
420         self.calendarWidget = QtWidgets.QMainWindow(self.examplesBody)
421         self.calendarWidget.setObjectName("calendarWidget")
422         self.calendarWidget.resize(378, 194)
423         self.calendarWidget.setFixedSize(QtCore.QSize(378, 194))
424         self.calendarWidget.setWindowFlag(QtCore.Qt.
    WindowMinimizeButtonHint, False)
425         self.calendar = QtWidgets.QCalendarWidget(self.calendarWidget)
426         self.calendar.setMinimumSize(QtCore.QSize(378, 194))
427         self.calendar.setAcceptDrops(False)
428         self.calendar.setLayoutDirection(QtCore.Qt.LeftToRight)
429         self.calendar.setGridVisible(False)
430         self.calendar.setSelectionMode(QtWidgets.QCalendarWidget.
    SingleSelection)
431         self.calendar.setHorizontalHeaderFormat(QtWidgets.QCalendarWidget.
    ShortDayNames)
432         self.calendar.setVerticalHeaderFormat(QtWidgets.QCalendarWidget.
    NoVerticalHeader)
433         self.calendar.setNavigationBarVisible(True)
434         self.calendar.setDateEditEnabled(True)
435         self.calendar.setObjectName("calendar")
436         self.calendar.setLocale(QtCore.QLocale(QtCore.QLocale.Portuguese))
437         self.calendar.activated.connect(self.setCalendarDate)
438         QtCore.QMetaObject.connectSlotsByName(self.calendarWidget)
439         _translate = QtCore.QCoreApplication.translate
440         self.calendarWidget.setWindowTitle(_translate("calendarWidget", "
    Calendario"))
441         self.calendarWidget.show()
442 
443     def setCalendarDate(self):
444         self.dateLineField.setText(self.calendar.selectedDate().toString())
```

```
445         self.calendarWidget.close()
446         self.calendarWidget.destroy()
447 
448     def add_example_article(self):
449         article = self.__getMandatoryFields()
450         if article is not None:
451             if article.get_url() in self.examplesArticlesDict:
452                 self.create_popup("Alerta", "Esta URL ja foi adicionada aos
    exemplos")
453                 raise Exception("URL ja adicionada aos exemplos")
454             self.articlesListWidget.addItem(article.get_url())
455             self.examplesArticlesDict[article.get_url()] = article
456             self.__clearExampleFields()
457 
458     def removeExample(self):
459         for item in self.articlesListWidget.selectedItems():
460             self.articlesListWidget.takeItem(self.articlesListWidget.row(
    item))
461             self.examplesArticlesDict.pop(item.text())
462             self.__clearExampleFields()
463 
464     def viewExample(self):
465         msg = QMessageBox()
466         msg.setWindowTitle("Artigo")
467         article = self.examplesArticlesDict.get(self.articlesListWidget.
    currentItem().text())
468         msg.setText(str(article))
469         msg.exec_()
470 
471     def __getMandatoryFields(self):
472         url = self.urlLineField.text().strip()
473         title = self.titleLineField.text().strip()
474         content = self.contentPlainText.toPlainText().strip()
475         print(self.dateLineField.text().strip())
476         if url == "" or title == "" or content == "":
477             self.create_popup("Alerta", "Os campos: URL, titulo e conteudo
    sao obrigatorios")
478             return None
479         date = self.dateLineField.text().strip()
480         subtitle = self.subtitleLineField.text().strip()
481         author = self.authorLineField.text().strip()
482         return Article(url, title, content, date, subtitle, author)
483 
484     def __clearExampleFields(self):
485         self.urlLineField.setText("")
486         self.titleLineField.setText("")
487         self.contentPlainText.setPlainText("")
```

```
488         self.dateLineField.setText("")
489         self.subtitleLineField.setText("")
490         self.authorLineField.setText("")
491 
492     def generate_website_config_action(self):
493         examples_articles = []
494         for example_article_url in self.examplesArticlesDict:
495             examples_articles.append(self.examplesArticlesDict[
    example_article_url])
496         website_config_generator = WebsiteConfigGenerator(examples_articles
    )
497         try:
498             web_site_start_url = website_config_generator.generate()
499             window_title = 'INFO'
500             window_message = 'Configuracao gerada para o website ' +
    web_site_start_url
501         except IndexError:
502             window_title = 'ERRO'
503             window_message = 'E necessario que ao menos um artigo exemplo
    seja adicionado.'
504         except IntegrityError:
505             window_title = 'ERRO'
506             window_message = 'Nao foi possivel gerar a configuracao, pois o
     website solicitado ja esta na base.'
507         self.examplesArticlesDict.clear()
508         self.articlesListWidget.clear()
509         msg = QMessageBox()
510         msg.setWindowTitle(window_title)
511         msg.setText(window_message)
512         msg.exec_()
513 
514 
515     def create_entry_point_screen(self):
516         self.entry_point_widget = QtWidgets.QWidget(self.mainBody)
517         self.entry_point_widget.setObjectName("entry_point_widget")
518         self.verticalLayout_6 = QtWidgets.QVBoxLayout(self.
    entry_point_widget)
519         self.verticalLayout_6.setObjectName("verticalLayout_6")
520         self.entry_point_text_browser = QtWidgets.QTextBrowser(self.
    entry_point_widget)
521         self.entry_point_text_browser.setObjectName("
    entry_point_text_browser")
522         self.verticalLayout_6.addWidget(self.entry_point_text_browser)
523         self.verticalLayout_5.addWidget(self.entry_point_widget)
524         self.horizontalLayout.addWidget(self.mainBody)
525         self.__highlight_button(self.inicioButton)
526 
```

```
527
528     # Load source screen
529     def createCarregarBody(self):
530         self.carregarBody = QtWidgets.QWidget(self.mainBody)
531         self.carregarBody.setObjectName("carregarBody")
532         self.carregarBodyVerticalLayout = QtWidgets.QVBoxLayout(self.
    carregarBody)
533         self.carregarBodyVerticalLayout.setObjectName("
    carregarBodyVerticalLayout")
534         self.carregarBodyWidget = QtWidgets.QWidget(self.carregarBody)
535         self.carregarBodyWidget.setObjectName("carregarBodyWidget")
536         self.carregarBodyHorizontalLayout = QtWidgets.QHBoxLayout(self.
    carregarBodyWidget)
537         self.carregarBodyHorizontalLayout.setObjectName("
    carregarBodyHorizontalLayout")
538         self.carregarBodyLabel = QtWidgets.QLabel(self.carregarBodyWidget)
539         self.carregarBodyLabel.setMaximumSize(QtCore.QSize(70, 16777215))
540         self.carregarBodyLabel.setObjectName("carregarBodyLabel")
541         self.carregarBodyHorizontalLayout.addWidget(self.carregarBodyLabel,
     0, QtCore.Qt.AlignLeft)
542         self.select_website_combo_box = QtWidgets.QComboBox(self.
    carregarBodyWidget)
543         self.select_website_combo_box.setMinimumSize(QtCore.QSize(500, 0))
544         self.select_website_combo_box.setObjectName("
    selecionarFonteComboBox")
545         self.carregarBodyHorizontalLayout.addWidget(self.
    select_website_combo_box)
546         self.carregarBodyHorizontalLayout.setAlignment(QtCore.Qt.
    AlignCenter)
547         self.carregarBodyVerticalLayout.addWidget(self.carregarBodyWidget)
548         self.carregarBodyScrollArea = QtWidgets.QPlainTextEdit(self.
    carregarBody)
549         self.carregarBodyScrollArea.setObjectName("carregarBodyScrollArea")
550         self.carregarBodyScrollAreaWidgetContents = QtWidgets.QWidget()
551         self.carregarBodyScrollAreaWidgetContents.setGeometry(QtCore.QRect
    (0, 0, 555, 340))
552         self.carregarBodyScrollAreaWidgetContents.setObjectName("
    carregarBodyScrollAreaWidgetContents")
553         self.carregarBodyVerticalLayout.addWidget(self.
    carregarBodyScrollArea)
554         self.carregarBodyWidget_2 = QtWidgets.QWidget(self.carregarBody)
555         self.carregarBodyWidget_2.setObjectName("carregarBodyWidget_2")
556         self.carregarBodyHorizontalLayout2 = QtWidgets.QHBoxLayout(self.
    carregarBodyWidget_2)
557         self.carregarBodyHorizontalLayout2.setObjectName("
    carregarBodyHorizontalLayout2")
558         self.paraColetaButton = QtWidgets.QPushButton(self.
```

```
    carregarBodyWidget_2)
559         self.paraColetaButton.setObjectName("paraColetaButton")
560         self.paraColetaButton.setIcon(QtGui.QIcon(":/whiteIcons/assets/
    redIcons/stop-circle.svg"))
561         self.carregarBodyHorizontalLayout2.addWidget(self.paraColetaButton)
562         self.iniciarColetaButton = QtWidgets.QPushButton(self.
    carregarBodyWidget_2)
563         self.iniciarColetaButton.setObjectName("iniciarColetaButton")
564         self.iniciarColetaButton.setIcon(QtGui.QIcon(":/whiteIcons/assets/
    greenIcons/play.svg"))
565         self.carregarBodyHorizontalLayout2.addWidget(self.
    iniciarColetaButton)
566         self.carregarBodyVerticalLayout.addWidget(self.carregarBodyWidget_2
    )
567         self.verticalLayout_5.addWidget(self.carregarBody)
568         self.carregarBodyScrollArea.setReadOnly(True)
569         self.carregarBody.close()
570 
571     def setupButtons(self):
572         self.inicioButton.clicked.connect(lambda: self.
    open_widget_in_main_body(self.entry_point_widget, self.inicioButton))
573         self.fornecerButton.clicked.connect(lambda: self.
    open_widget_in_main_body(self.examplesBody, self.fornecerButton))
574         self.carregarButton.clicked.connect(lambda: self.
    open_widget_in_main_body(self.carregarBody, self.carregarButton))
575         self.settings_button.clicked.connect(lambda: self.
    open_widget_in_main_body(self.configsWidget, self.settings_button))
576         self.execute_crawler_button.clicked.connect(lambda: self.
    open_widget_in_main_body(self.configure_crawler_widget, self.
    execute_crawler_button))
577         self.checkBox.clicked.connect(self.restore_settings_default_value)
578         self.pushButton.clicked.connect(self.update_email_fields)
579         self.dateButton.clicked.connect(self.show_calendar)
580         self.finalizeExampleButton.clicked.connect(self.add_example_article
    )
581         self.removeArticleExampleButton.clicked.connect(self.removeExample)
582         self.generate_web_site_config_button.clicked.connect(self.
    generate_website_config_action)
583         self.iniciarColetaButton.clicked.connect(self.start_collect_action)
584         self.paraColetaButton.clicked.connect(self.stop_collect_action)
585         self.add_website_to_execute_button.clicked.connect(self.
    add_website_to_execute_action)
586         self.remove_website_button.clicked.connect(self.
    remove_website_to_execute_action)
587         self.configure_crawler_for_websites_button.clicked.connect(self.
    configure_crawler_for_websites_action)
588 
```

```
589     def start_collect_action(self):
590         self.carregarBodyScrollArea.clear()
591         start_url = tuple(self.select_website_combo_box.currentText().split
    (' - '))[1]
592         self.dedicated_crawler = DedicatedCrawler.DedicatedCrawler(
    start_url)
593         print('Starting collect...')
594         self.dedicated_crawler.start()
595         self.dedicated_crawler.finished.connect(self.collect_finished)
596         self.dedicated_crawler.update_log.connect(self.evt_update_log)
597 
598     def collect_finished(self):
599         QMessageBox.information(self.carregarBodyScrollArea, "INFO", "
    Simulacao finalizada!")
600 
601     def evt_update_log(self, val):
602         if self.carregarBodyScrollArea.toPlainText() == "":
603             self.carregarBodyScrollArea.setPlainText(val)
604             return
605         self.carregarBodyScrollArea.setPlainText(self.
    carregarBodyScrollArea.toPlainText() + '\n\n' + val)
606         scroll_position = self.carregarBodyScrollArea.verticalScrollBar().
    maximum()
607         self.carregarBodyScrollArea.verticalScrollBar().setValue(
    scroll_position)
608 
609 
610     def stop_collect_action(self):
611         self.dedicated_crawler.stop_crawl()
612 
613     def __highlight_button(self, button: QtWidgets.QPushButton):
614         elegible_buttons = [self.inicioButton, self.fornecerButton, self.
    carregarButton, self.execute_crawler_button, self.settings_button]
615         button.setEnabled(False)
616         for b in elegible_buttons:
617             if b is not button:
618                 b.setEnabled(True)
619 
620     def get_widgets_in_main_body(self):
621         return [self.configsWidget, self.carregarBody, self.examplesBody,
    self.configure_crawler_widget,
622                 self.entry_point_widget]
623 
624     def open_widget_in_main_body(self, widget: QtWidgets.QWidget, button:
    QtWidgets.QPushButton):
625         self.__highlight_button(button)
626         for screen in self.get_widgets_in_main_body():
```

```
627             screen.close()
628             if screen == self.configsWidget:
629                 self.label_4.close()
630             if screen == self.examplesBody:
631                 self.close_articles_examples_screen()
632             if screen == self.carregarBody:
633                 self.select_website_combo_box.clear()
634             if screen == self.configure_crawler_widget:
635                 self.close_configure_crawler_screen()
636                 self.website_to_execute_combo_box.clear()
637         widget.show()
638         if widget == self.configsWidget:
639             self.open_settings_screen()
640         if widget == self.examplesBody:
641             self.open_articles_examples_screen()
642         if widget == self.configure_crawler_widget:
643             self.open_configure_crawler_screen()
644             websites = get_all_websites()
645             for website in websites:
646                 self.website_to_execute_combo_box.addItem(str(website[0]) +
    ' - ' + website[1])
647         if widget == self.carregarBody:
648             websites = get_all_websites()
649             for website in websites:
650                 self.select_website_combo_box.addItem(str(website[0]) + ' -
    ' + website[1])
651
652     #configure crawler screen
653     def create_configure_crawler_screen(self):
654         self.select_websites_title_label = QtWidgets.QLabel(self.mainBody)
655         self.select_websites_title_label.setMaximumSize(QtCore.QSize
    (16777215, 25))
656         self.select_websites_title_label.setObjectName("
    select_websites_title_label")
657         self.verticalLayout_5.addWidget(self.select_websites_title_label,
    0,QtCore.Qt.AlignHCenter | QtCore.Qt.AlignVCenter)
658         self.configure_crawler_widget = QtWidgets.QWidget(self.mainBody)
659         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
     QtWidgets.QSizePolicy.Preferred)
660         sizePolicy.setHorizontalStretch(0)
661         sizePolicy.setVerticalStretch(0)
662         sizePolicy.setHeightForWidth(self.configure_crawler_widget.
    sizePolicy().hasHeightForWidth())
663         self.configure_crawler_widget.setSizePolicy(sizePolicy)
664         self.configure_crawler_widget.setObjectName("
    configure_crawler_widget")
665         self.horizontalLayout_3 = QtWidgets.QHBoxLayout(self.
```

```
    configure_crawler_widget)
666         self.horizontalLayout_3.setObjectName("horizontalLayout_3")
667         self.exampleFieldsWidget = QtWidgets.QWidget(self.
    configure_crawler_widget)
668         self.exampleFieldsWidget.setObjectName("exampleFieldsWidget")
669         self.verticalLayout_6 = QtWidgets.QVBoxLayout(self.
    exampleFieldsWidget)
670         self.verticalLayout_6.setContentsMargins(0, 0, 0, 0)
671         self.verticalLayout_6.setSpacing(0)
672         self.verticalLayout_6.setObjectName("verticalLayout_6")
673         self.add_and_select_website_to_execute_widget = QtWidgets.QWidget(
    self.exampleFieldsWidget)
674         self.add_and_select_website_to_execute_widget.setObjectName("
    add_and_select_website_to_execute_widget")
675         self.horizontalLayout_4 = QtWidgets.QHBoxLayout(self.
    add_and_select_website_to_execute_widget)
676         self.horizontalLayout_4.setObjectName("horizontalLayout_4")
677         self.website_to_execute_combo_box = QtWidgets.QComboBox(self.
    add_and_select_website_to_execute_widget)
678         self.website_to_execute_combo_box.setEnabled(True)
679         sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
     QtWidgets.QSizePolicy.Preferred)
680         sizePolicy.setHorizontalStretch(0)
681         sizePolicy.setVerticalStretch(0)
682         sizePolicy.setHeightForWidth(self.website_to_execute_combo_box.
    sizePolicy().hasHeightForWidth())
683         self.website_to_execute_combo_box.setSizePolicy(sizePolicy)
684         self.website_to_execute_combo_box.setMinimumSize(QtCore.QSize(270,
    0))
685         self.website_to_execute_combo_box.setMaximumSize(QtCore.QSize
    (16777215, 16777215))
686         self.website_to_execute_combo_box.setSizeIncrement(QtCore.QSize(0,
    0))
687         self.website_to_execute_combo_box.setBaseSize(QtCore.QSize(0, 0))
688         self.website_to_execute_combo_box.setObjectName("
    website_to_execute_combo_box")
689         self.website_to_execute_combo_box.addItem("")
690         self.horizontalLayout_4.addWidget(self.website_to_execute_combo_box
    , 0, QtCore.Qt.AlignVCenter)
691         self.add_website_to_execute_button = QtWidgets.QPushButton(self.
    add_and_select_website_to_execute_widget)
692         self.add_website_to_execute_button.setMaximumSize(QtCore.QSize(120,
     16777215))
693         self.add_website_to_execute_button.setObjectName("
    add_website_to_execute_button")
694         self.horizontalLayout_4.addWidget(self.
    add_website_to_execute_button)
```

```
695         self.verticalLayout_6.addWidget(self.
    add_and_select_website_to_execute_widget)
696         self.all_this_words_widget = QtWidgets.QWidget(self.
    exampleFieldsWidget)
697         self.all_this_words_widget.setObjectName("all_this_words_widget")
698         self.verticalLayout_8 = QtWidgets.QVBoxLayout(self.
    all_this_words_widget)
699         self.verticalLayout_8.setContentsMargins(0, 9, 0, 9)
700         self.verticalLayout_8.setSpacing(0)
701         self.verticalLayout_8.setObjectName("verticalLayout_8")
702         self.all_this_words_label = QtWidgets.QLabel(self.
    all_this_words_widget)
703         self.all_this_words_label.setObjectName("all_this_words_label")
704         self.verticalLayout_8.addWidget(self.all_this_words_label, 0,
    QtCore.Qt.AlignHCenter)
705         self.all_this_words_line_edit = QtWidgets.QLineEdit(self.
    all_this_words_widget)
706         self.all_this_words_line_edit.setObjectName("
    all_this_words_line_edit")
707         self.verticalLayout_8.addWidget(self.all_this_words_line_edit)
708         self.verticalLayout_6.addWidget(self.all_this_words_widget)
709         self.exactly_this_phrase_widget = QtWidgets.QWidget(self.
    exampleFieldsWidget)
710         self.exactly_this_phrase_widget.setObjectName("
    exactly_this_phrase_widget")
711         self.verticalLayout_10 = QtWidgets.QVBoxLayout(self.
    exactly_this_phrase_widget)
712         self.verticalLayout_10.setContentsMargins(0, -1, 0, 9)
713         self.verticalLayout_10.setSpacing(0)
714         self.verticalLayout_10.setObjectName("verticalLayout_10")
715         self.exactly_this_phrase_label = QtWidgets.QLabel(self.
    exactly_this_phrase_widget)
716         self.exactly_this_phrase_label.setObjectName("
    exactly_this_phrase_label")
717         self.verticalLayout_10.addWidget(self.exactly_this_phrase_label, 0,
     QtCore.Qt.AlignHCenter)
718         self.exactly_this_phrase_line_edit = QtWidgets.QLineEdit(self.
    exactly_this_phrase_widget)
719         self.exactly_this_phrase_line_edit.setObjectName("
    exactly_this_phrase_line_edit")
720         self.verticalLayout_10.addWidget(self.exactly_this_phrase_line_edit
    )
721         self.verticalLayout_6.addWidget(self.exactly_this_phrase_widget)
722         self.any_of_this_words_widget = QtWidgets.QWidget(self.
    exampleFieldsWidget)
723         self.any_of_this_words_widget.setObjectName("
    any_of_this_words_widget")
```

```
724         self.verticalLayout_9 = QtWidgets.QVBoxLayout(self.
    any_of_this_words_widget)
725         self.verticalLayout_9.setContentsMargins(0, -1, 0, 9)
726         self.verticalLayout_9.setSpacing(0)
727         self.verticalLayout_9.setObjectName("verticalLayout_9")
728         self.any_of_this_words_label = QtWidgets.QLabel(self.
    any_of_this_words_widget)
729         self.any_of_this_words_label.setObjectName("any_of_this_words_label
    ")
730         self.verticalLayout_9.addWidget(self.any_of_this_words_label, 0,
    QtCore.Qt.AlignHCenter)
731         self.any_of_this_words_line_edit = QtWidgets.QLineEdit(self.
    any_of_this_words_widget)
732         self.any_of_this_words_line_edit.setObjectName("
    any_of_this_words_line_edit")
733         self.verticalLayout_9.addWidget(self.any_of_this_words_line_edit)
734         self.verticalLayout_6.addWidget(self.any_of_this_words_widget)
735         self.emails_to_be_sent_widget = QtWidgets.QWidget(self.
    exampleFieldsWidget)
736         self.emails_to_be_sent_widget.setObjectName("
    emails_to_be_sent_widget")
737         self.verticalLayout_11 = QtWidgets.QVBoxLayout(self.
    emails_to_be_sent_widget)
738         self.verticalLayout_11.setObjectName("verticalLayout_11")
739         self.emails_to_be_sent_label = QtWidgets.QLabel(self.
    emails_to_be_sent_widget)
740         self.emails_to_be_sent_label.setObjectName("emails_to_be_sent_label
    ")
741         self.verticalLayout_11.addWidget(self.emails_to_be_sent_label, 0,
    QtCore.Qt.AlignHCenter)
742         self.emails_to_be_sent_line_edit = QtWidgets.QLineEdit(self.
    emails_to_be_sent_widget)
743         self.emails_to_be_sent_line_edit.setObjectName("
    emails_to_be_sent_line_edit")
744         self.verticalLayout_11.addWidget(self.emails_to_be_sent_line_edit)
745         self.verticalLayout_6.addWidget(self.emails_to_be_sent_widget)
746         self.configure_crawler_for_websites_button = QtWidgets.QPushButton(
    self.exampleFieldsWidget)
747         self.configure_crawler_for_websites_button.setObjectName("
    configure_crawler_for_websites_button")
748         self.verticalLayout_6.addWidget(self.
    configure_crawler_for_websites_button)
749         self.horizontalLayout_3.addWidget(self.exampleFieldsWidget)
750         self.websites_to_execute_widget = QtWidgets.QWidget(self.
    configure_crawler_widget)
751         self.websites_to_execute_widget.setMinimumSize(QtCore.QSize(100, 0)
    )
```

```
752         self.websites_to_execute_widget.setMaximumSize(QtCore.QSize(125,
    16777215))
753         self.websites_to_execute_widget.setObjectName("
    websites_to_execute_widget")
754         self.verticalLayout_7 = QtWidgets.QVBoxLayout(self.
    websites_to_execute_widget)
755         self.verticalLayout_7.setContentsMargins(0, 0, 0, 0)
756         self.verticalLayout_7.setSpacing(9)
757         self.verticalLayout_7.setObjectName("verticalLayout_7")
758         self.websites_to_execute_label = QtWidgets.QLabel(self.
    websites_to_execute_widget)
759         self.websites_to_execute_label.setMinimumSize(QtCore.QSize(119, 0))
760         self.websites_to_execute_label.setObjectName("
    websites_to_execute_label")
761         self.verticalLayout_7.addWidget(self.websites_to_execute_label, 0,
    QtCore.Qt.AlignHCenter)
762         self.website_to_execute_list_widget = QtWidgets.QListWidget(self.
    websites_to_execute_widget)
763         self.website_to_execute_list_widget.setObjectName("
    website_to_execute_list_widget")
764         self.verticalLayout_7.addWidget(self.website_to_execute_list_widget
    )
765         self.articlesListButtonsWidget = QtWidgets.QWidget(self.
    websites_to_execute_widget)
766         self.articlesListButtonsWidget.setObjectName("
    articlesListButtonsWidget")
767         self.horizontalLayout_14 = QtWidgets.QHBoxLayout(self.
    articlesListButtonsWidget)
768         self.horizontalLayout_14.setSizeConstraint(QtWidgets.QLayout.
    SetDefaultConstraint)
769         self.horizontalLayout_14.setContentsMargins(0, 0, 0, 0)
770         self.horizontalLayout_14.setSpacing(0)
771         self.horizontalLayout_14.setObjectName("horizontalLayout_14")
772         self.remove_website_button = QtWidgets.QPushButton(self.
    articlesListButtonsWidget)
773         self.remove_website_button.setObjectName("remove_website_button")
774         self.horizontalLayout_14.addWidget(self.remove_website_button)
775         self.verticalLayout_7.addWidget(self.articlesListButtonsWidget)
776         self.horizontalLayout_3.addWidget(self.websites_to_execute_widget)
777         self.verticalLayout_5.addWidget(self.configure_crawler_widget)
778         self.close_configure_crawler_screen()
779
780     def close_configure_crawler_screen(self):
781         self.select_websites_title_label.close()
782         self.configure_crawler_widget.close()
783
784     def open_configure_crawler_screen(self):
```

```
785         self.select_websites_title_label.show()
786         self.configure_crawler_widget.show()
787 
788     def add_website_to_execute_action(self):
789         website_to_execute = self.website_to_execute_combo_box.currentText
    ()
790         for i in range(self.website_to_execute_list_widget.count()):
791             if self.website_to_execute_list_widget.item(i).text() ==
    website_to_execute:
792                 return
793         self.website_to_execute_list_widget.addItem(website_to_execute)
794 
795     def remove_website_to_execute_action(self):
796         for item in self.website_to_execute_list_widget.selectedItems():
797             self.website_to_execute_list_widget.takeItem(self.
    website_to_execute_list_widget.row(item))
798 
799     def configure_crawler_for_websites_action(self):
800         if self.website_to_execute_list_widget.count() == 0:
801             QMessageBox.information(self.configure_crawler_widget, "ALERTA"
    , "E necessario adicionar pelo menos um website para executar!")
802             return
803         all_this_words = set(word for word in self.all_this_words_line_edit
    .text().split())
804         exactly_this_phrase = self.exactly_this_phrase_line_edit.text().
    strip()
805         any_of_this_words = set(word for word in self.
    any_of_this_words_line_edit.text().split())
806         emails_to_be_sent = set(email for email in self.
    emails_to_be_sent_line_edit.text().split())
807         crawler_name, crawler_name_ok = QInputDialog().getText(self.
    configure_crawler_widget, "Identificacao do crawler", "De um nome para a
     identificacao do crawler", QLineEdit.Normal)
808         if not crawler_name_ok:
809             return
810 
811         text, ok = QInputDialog().getText(self.configure_crawler_widget,
    crawler_name, "Quantos segundos cada fonte devera executar?", QLineEdit.
    Normal)
812 
813         term_controller = TermController.TermController(all_this_words,
    exactly_this_phrase, any_of_this_words)
814 
815         start_urls = []
816         for i in range(self.website_to_execute_list_widget.count()):
817             id_and_site = self.website_to_execute_list_widget.item(i).text
    ()
```

```
818             start_url = re.sub('\\d+ - ', '', id_and_site)
819             start_urls.append(start_url)
820 
821         if text.strip() and ok:
822             shared_crawler = SharedCrawler.SharedCrawler(crawler_name,
    start_urls, int(text.strip()), term_controller, emails_to_be_sent)
823             shared_crawler.start()
824             QMessageBox.information(self.configure_crawler_widget, "INFO",
    "Execucao do crawler " + crawler_name + " iniciada")
825 
826 
827     def retranslateUi(self, MainWindow):
828         _translate = QtCore.QCoreApplication.translate
829         MainWindow.setWindowTitle(_translate("MainWindow", "CrawlEX"))
830         self.crawlExlabel.setText(_translate("MainWindow", "CrawlEX"))
831         self.inicioButton.setText(_translate("MainWindow", "Inicio"))
832         self.fornecerButton.setText(_translate("MainWindow", "Fornecer
    exemplos"))
833         self.carregarButton.setText(_translate("MainWindow", "Carregar
    fonte"))
834         self.execute_crawler_button.setText(_translate("MainWindow", "
    Executar um Crawler"))
835         self.settings_button.setText(_translate("MainWindow", "
    Configuracoes"))
836         self.versaoLabel.setText(_translate("MainWindow", "Versao 1.0.0"))
837         self.labelInfoExamples.setText(_translate("MainWindow", "Forneca
    artigos exemplos para configurar a coleta do website"))
838         self.label_4.setText(_translate("MainWindow", "Informe o endereco
    de e-mail e senha de uma conta Google que sera o robo de envio de
    relatorios"))
839         self.label.setText(_translate("MainWindow", "Email"))
840         self.labelPassword.setText(_translate("MainWindow", "Senha"))
841         self.label_2.setText(_translate("MainWindow", "Artigos exemplos"))
842         self.checkBox.setText(_translate("MainWindow", "Editar"))
843         self.pushButton.setText(_translate("MainWindow", "Salvar email e
    senha"))
844         self.carregarBodyLabel.setText(_translate("MainWindow", "Website"))
845         self.urlLabel.setText(_translate("MainWindow", "URL"))
846         self.titleLabel.setText(_translate("MainWindow", "Titulo"))
847         self.contentLabel.setText(_translate("MainWindow", "Conteudo"))
848         self.subtitleLabel.setText(_translate("MainWindow", "Subtitulo"))
849         self.authorLabel.setText(_translate("MainWindow", "Autor"))
850         self.dateLineLabel.setText(_translate("MainWindow", "Data"))
851         self.dateButton.setText(_translate("MainWindow", "..."))
852         self.generate_web_site_config_button.setText(_translate("MainWindow
    ", "Gerar configuracao para coleta do website"))
853         self.select_websites_title_label.setText(
```

```
854             _translate("MainWindow", "Selecione os websites a executar e
    defina os termos de seu interesse"))
855         self.add_website_to_execute_button.setText(_translate("MainWindow",
     "Adicionar"))
856         self.all_this_words_label.setText(_translate("MainWindow", "Todas
    estas palavras:"))
857         self.exactly_this_phrase_label.setText(_translate("MainWindow", "
    Exatamente esta frase:"))
858         self.any_of_this_words_label.setText(_translate("MainWindow", "
    Quaisquer destas palavras:"))
859         self.emails_to_be_sent_label.setText(_translate("MainWindow", "
    Emails a serem enviados relatorio de coleta"))
860         self.configure_crawler_for_websites_button.setText(
861             _translate("MainWindow", "Executar Crawler para websites
    selecionados"))
862         self.websites_to_execute_label.setText(_translate("MainWindow", "
    Websites a executar"))
863         self.remove_website_button.setText(_translate("MainWindow", "
    Remover website"))
864         self.entry_point_text_browser.setHtml(_translate("MainWindow",
865                                                          "<!DOCTYPE HTML
    PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.org/TR/REC-html40/
    strict.dtd\">\n"
866                                                          "<html><head><meta
     name=\"qrichtext\" content=\"1\" /><style type=\"text/css\">\n"
867                                                          "p, li { white-
    space: pre-wrap; }\n"
868                                                          "</style></head><
    body style=\" font-family:\'Ubuntu\'; font-size:10pt; font-weight:400;
    font-style:normal;\">\n"
869                                                          "<p align=\"center
    \" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-
    right:0px; -qt-block-indent:0; text-indent:0px;\">BEM VINDO AO CRAWLEX</
    p>\n"
870                                                          "<p align=\"center
    \" style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px;
    margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px
    ;\"><br /></p>\n"
871                                                          "<p align=\"center
    \" style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px;
    margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px
    ;\"><br /></p>\n"
872                                                          "<p align=\"center
    \" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-
    right:0px; -qt-block-indent:0; text-indent:0px;\">1 - FORNECER EXEMPLOS
    </p>\n"
873                                                          "<p align=\"center
```

```
\" style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px;
margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px
;\"><br /></p>\n"
874                                                 "<p align=\"center
\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-
right:0px; -qt-block-indent:0; text-indent:0px;\">Utilize esta tela para
 informar artigos exemplos de sites que voce tem interesse em coletar
informacoes. E importante que todos os exemplos de um mesmo site,
contenham os mesmos campos preenchidos. Por exemplo, se for informado a
data para o primeiro exemplo, informe para os demais tambem. Aconselha-
se fornecer dois ou tres exemplos por site.</p>\n"
875                                                 "<p align=\"center
\" style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px;
margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px
;\"><br /></p>\n"
876                                                 "<p align=\"center
\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-
right:0px; -qt-block-indent:0; text-indent:0px;\">2 - CARREGAR FONTE</p
>\n"
877                                                 "<p align=\"center
\" style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px;
margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px
;\"><br /></p>\n"
878                                                 "<p align=\"center
\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-
right:0px; -qt-block-indent:0; text-indent:0px;\">Utilize para simular a
 navegacao e coleta de um site em que os artigos exemplos ja foram
fornecidos e estao salvos na base de dados.</p>\n"
879                                                 "<p align=\"center
\" style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px;
margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px
;\"><br /></p>\n"
880                                                 "<p align=\"center
\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-
right:0px; -qt-block-indent:0; text-indent:0px;\">3 - EXECUTAR UM
CRAWLER</p>\n"
881                                                 "<p align=\"center
\" style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px;
margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px
;\"><br /></p>\n"
882                                                 "<p align=\"center
\" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-
right:0px; -qt-block-indent:0; text-indent:0px;\">Utilize para
selecionar todos os sites desejados (desde que tiveram artigos exemplos
fornecidos previamente) a fim de executar um crawler com eles. Preencha
os campos de busca de acordo com o seu interesse e forneca os emails aos
 quais receberao informacoes para a coleta.</p>\n"
```

```
883                                                 "<p align=\"center
    \" style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px;
    margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px
    ;\"><br /></p>\n"
884                                                 "<p align=\"center
    \" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-
    right:0px; -qt-block-indent:0; text-indent:0px;\">4 - CONFIGURACOES</p>\
    n"
885                                                 "<p align=\"center
    \" style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px;
    margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px
    ;\"><br /></p>\n"
886                                                 "<p align=\"center
    \" style=\" margin-top:0px; margin-bottom:0px; margin-left:0px; margin-
    right:0px; -qt-block-indent:0; text-indent:0px;\">Utilize para setar o
    email e a senha de uma conta google, que sera a responsavel por fazer os
     envios de emails com informacoes das coletas do crawler.</p>\n"
887                                                 "<p align=\"center
    \" style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px;
    margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px
    ;\"><br /></p>\n"
888                                                 "<p align=\"center
    \" style=\"-qt-paragraph-type:empty; margin-top:0px; margin-bottom:0px;
    margin-left:0px; margin-right:0px; -qt-block-indent:0; text-indent:0px
    ;\"><br /></p></body></html>"))
```

Website.py

```
1  from WebsiteDao import insert_web_site, get_website_id_by_domain
2  from ExampleArticlesDao import insert_example_article
3  from urllib.parse import urlparse
4
5  # Website configuration
6  class WebSiteConfig:
7      def __init__(self, id: int, name: str, example_articles: [],
8      text_similarity_coefficient: float, minutes_to_crawl: int):
9          self.__id = id
10         self.__name = name
11         self.__example_articles = example_articles
12         self.__text_similarity_coefficient = text_similarity_coefficient
13         self.__minutes_to_crawl = minutes_to_crawl
14         self.__start_url = None
15         self.__articles_url_pattern = None
16         self.__title_extractor = None
17         self.__subtitle_extractor = None
18         self.__content_extractor = None
19         self.__author_extractor = None
```

```
20          self.__date_extractor = None
21
22      def get_id(self):
23          return self.__id
24
25      def get_name(self):
26          return self.__name
27
28      def get_star_url(self):
29          return self.__start_url
30
31      def get_example_articles(self):
32          return self.__example_articles
33
34      def get_text_similarity_coefficient(self):
35          return self.__text_similarity_coefficient
36
37      def get_minutes_to_crawl(self):
38          return self.__minutes_to_crawl
39
40      def get_articles_url_pattern(self):
41          return self.__articles_url_pattern
42
43      def get_title_extractor(self):
44          return self.__title_extractor
45
46      def get_subtitle_extractor(self):
47          return self.__subtitle_extractor
48
49      def get_content(self):
50          return self.__content_extractor
51
52      def get_author(self):
53          return self.__author_extractor
54
55      def get_date(self):
56          return self.__date_extractor
57
58  # Website configuration generator
59  class WebsiteConfigGenerator:
60      def __init__(self, example_articles: []):
61          self.__example_articles = example_articles
62
63      def generate(self) -> str:
64          examples_copy = self.__example_articles.copy()
65          random_article = examples_copy.pop()
66          website_protocol = urlparse(random_article.get_url()).scheme
```

```
67          website_domain = urlparse(random_article.get_url()).netloc
68          website_start_url = website_protocol + '://' + website_domain
69          insert_web_site(website_start_url)
70          website_id = get_website_id_by_domain(website_start_url)
71          for example_article in self.__example_articles:
72              insert_example_article(website_id, example_article)
73          return website_start_url
```

WebsiteDao.py

```
1  # Website Data Access Object
2  import sqlite3
3
4
5  def insert_web_site(domain: str):
6      conn = sqlite3.connect("database.db")
7      cursor = conn.cursor()
8      with conn:
9          cursor.execute("INSERT INTO Website VALUES (?, ?)", (None, domain))
10     conn.close()
11
12
13 def get_all_websites() -> tuple:
14     conn = sqlite3.connect("database.db")
15     cursor = conn.cursor()
16     with conn:
17         cursor.execute("SELECT * FROM Website")
18         result = cursor.fetchall()
19     conn.close()
20     return result
21
22
23 def get_all_websites_ids() -> []:
24     conn = sqlite3.connect("database.db")
25     cursor = conn.cursor()
26     with conn:
27         cursor.execute("SELECT id FROM Website")
28         results = cursor.fetchall()
29         ids = []
30         for result in results:
31             ids.append(result[0])
32         ids.sort()
33         return ids
34
35
36 def get_website_id_by_domain(domain: str):
37     conn = sqlite3.connect("database.db")
```

```
38      cursor = conn.cursor()
39      with conn:
40          result = cursor.execute("SELECT id FROM Website WHERE domain = ?", (
        domain,)).fetchone()[0]
41      conn.close()
42      return result
43
44
45  def delete_website_by_domain(domain: str):
46      conn = sqlite3.connect("database.db")
47      cursor = conn.cursor()
48      with conn:
49          cursor.execute("DELETE FROM Website WHERE domain = ?", (domain,))
50      conn.close()
51
52
53  def delete_all_websites():
54      conn = sqlite3.connect("database.db")
55      cursor = conn.cursor()
56      with conn:
57          cursor.execute("DELETE FROM Website")
58      conn.close()
59
60
61  def get_examples_articles(id: str):
62      conn = sqlite3.connect("database.db")
63      cursor = conn.cursor()
64      with conn:
65          cursor.execute("DELETE FROM Website")
66          return cursor.fetchall
67
68  def delete_website_by_id(id: int):
69      conn = sqlite3.connect("database.db")
70      cursor = conn.cursor()
71      with conn:
72          cursor.execute("DELETE FROM Website WHERE id = ?", (id,))
73      conn.close()
```

WebsiteDao.py

```
1  import LinkFinder
2  from WebsiteDao import get_website_id_by_domain
3  from ExampleArticlesDao import get_example_articles_by_website_id
4  import ArticlePatternGenerator
5  import re
6  import ExtractionConfig
7  import Extractor
```

```
8  import time
9  import TermController
10 from ArticleDao import insert_article
11
12 # Website navigation executor
13 class WebsiteExecutor:
14     def __init__(self, start_url: str, seconds_to_execute: int,
       term_controller: TermController):
15         self.__start_url = start_url
16         self.__start_time = None
17         self.__seconds_to_execute = seconds_to_execute
18         self.__term_controller = term_controller
19         self.__urls_already_visited = set()
20         self.__urls_to_visit = set()
21         self.__urls_to_visit_after = set()
22         self.__example_articles = self.load_examples_articles_from_website
       ()
23         self.__ARTICLE_PATTERN_TO_EXTRACT = self.
       generate_article_pattern_to_extract()
24         self.__extraction_config = self.generate_extraction_config()
25         self.__extractor = Extractor.Extractor(self.__extraction_config)
26
27     def execute(self):
28         self.__start_time = time.time()
29         self.__urls_already_visited.add(self.__start_url)
30         link_finder = LinkFinder.LinkFinder(self.__start_url)
31         self.__urls_to_visit = link_finder.find_links()
32
33         while len(self.__urls_to_visit) > 0:
34             if self.stop_crawl():
35                 print('Tempo de execucao esgotado para ' + self.__start_url
       + "\n")
36                 return
37
38             # selects the url head of the list, and remove it
39             new_url = self.__urls_to_visit.pop()
40
41             # checks if the url has already been visited
42             if new_url in self.__urls_already_visited:
43                 continue
44
45             # if not visited, mark as visited
46             self.__urls_already_visited.add(new_url)
47             link_finder = LinkFinder.LinkFinder(new_url)
48
49             try:
50                 self.__urls_to_visit_after = self.__urls_to_visit_after.
```

```
    union(link_finder.find_links())
51              except:
52                  print('Erro ao buscar urls na na pagina: ', new_url)
53                  print('')
54                  continue
55
56              print('Nova URL: ', new_url)
57              print('')
58              print('')
59
60              # tries to extract if it passes the article pattern regular
    expression
61              if self.match_article_pattern(new_url):
62                  try:
63                      print("Tentando extrair possivel artigo em: " + new_url
    )
64                      extracted_article = self.__extractor.extract(new_url)
65                      if self.__term_controller.is_valid(extracted_article):
66                          insert_article(extracted_article)
67                  except Exception as e:
68                      print(e)
69
70          if len(self.__urls_to_visit_after) == 0:
71              return
72          self.__start_url = self.__urls_to_visit_after.pop()
73          return self.run()
74
75      def match_article_pattern(self, url):
76          return bool(re.fullmatch(self.__ARTICLE_PATTERN_TO_EXTRACT, url))
77
78      def stop_crawl(self):
79          time_now = time.time()
80          return (time_now - self.__start_time) >= self.__seconds_to_execute
81
82      def load_examples_articles_from_website(self):
83          website_id = get_website_id_by_domain(self.__start_url)
84          return get_example_articles_by_website_id(website_id)
85
86      def generate_article_pattern_to_extract(self):
87          example_articles_url = []
88          for example_article in self.__example_articles:
89              example_articles_url.append(example_article.get_url())
90          article_pattern_generator = ArticlePatternGenerator.
    ArticlePatternGenerator(example_articles_url)
91          return article_pattern_generator.generate()
92
93      def generate_extraction_config(self):
```

```
94          extraction_config = ExtractionConfig.ExtractionConfig(self.
    __example_articles)
95          print('Gerando configuracao para coleta do web site ' + self.
    __start_url)
96          extraction_config.generate_config()
97          return extraction_config
```

8.2 Artigo

CrawlEX: uma ferramenta para extração de dados na web configurável através de exemplos

Marcos Aurélio Lessa[1], **Carina Friedrich Dorneles**[1]

[1]Departamento de Informática e Estatística - INE
Universidade Federal de Santa Catarina - UFSC/Florianópolis

marcoslessa.cs@gmail.com, carina.dorneles@ufsc.br

Abstract. *With the great advancement of the internet over the years, it is natural that we have a huge amount of data available on the network. These data can tell us things completely different, like what was said in the last speech by the President of the Republic or, the geographic coordinate of a place that we are interested in visiting. Depending on the interest profile of a user or even a company, it is very important to have these data in hand so that they can be analyzed and, eventually, taken some kind of action. However, in the vast majority of cases, it is unfeasible that these data collected manually, as they require time and effort, therefore, it is necessary to that the collection is done automatically, allowing the interested party only to make the analysis of what has actually been collected. In addition, for the configuration collection of a website is done automatically, it is necessary that the user have programming skills, thus, a hindrance for many people. In that context, the present work presents a tool for navigation and extraction of articles available on the internet, where a web crawler can be configured by a user common, without programming knowledge, just for providing examples of articles from the pages that are interesting to the user. The experiments performed by the author and lay users are presented, and then their results are analyzed.*

Resumo. *Com o grande avanço da internet ao longo dos anos, é natural que tenhamos uma enorme quantidade de dados disponível na rede. Esses dados, podem nos informar coisas completamente diferentes, como o que foi falado no último discurso do Presidente da República ou, a coordenada geográfica de um local que estamos interessados em visitar. Dependendo do perfil de interesse de um usuário ou até mesmo de uma empresa, é muito importante ter esses dados em mãos para que se possa analisá-los e, eventualmente, tomar algum tipo de ação. Porém, na grande maioria das vezes, é inviável que esses dados sejam coletados manualmente, pois demandam tempo e esforço, logo, faz-se necessário que a coletada seja feita de maneira automática, permitindo ao interessado apenas fazer a análise daquilo que efetivamente já foi coletado. Além disso, para que a configuração de uma coleta de um website seja feita de forma automática, é necessário que o usuário tenha habilidade em programação, sendo assim, um empecilho para muitas pessoas. Nesse contexto, o presente trabalho apresenta uma ferramenta para navegação e extração de artigos disponíveis na internet, onde um web crawler pode ser configurado por um usuário comum, sem conhecimentos em programação, apenas por fornecer exemplos de artigos das páginas as quais tem interesse. É apresentado os experimentos feitos pelo autor e usuários leigos, e depois analisados os seus resultados.*

1. Introdução

É notório que ao longo dos últimos anos, o crescimento no uso da internet é extremamente grande. Graças aos smartphones e outros dispositivos portáteis, o acesso a uma infinidade de páginas web se tornou extremamente acessível, pois agora, as pessoas podem navegar por essas páginas apenas segurando um equipamento eletrônico na palma de sua mão, em praticamente qualquer lugar e a qualquer horário. Essa acessibilidade faz com que cada vez mais o número de dados disponíveis na Internet cresça. Muitos desses dados estão publicados como artigos (sejam eles notícias, artigos científicos, textos publicados em blogs, portais, entre outros) que podem trazer inúmeras informações relevantes para diversas pessoas e empresas com perfis totalmente diferentes umas das outras.

Um jeito prático para poder monitorar esses artigos é através da criação de um web crawler capaz de extrair as informações desejadas que se fazem presente nas páginas web onde esses artigos estão publicados. Para isso, é necessário que se tenha um conhecimento mínimo em programação e, às vezes, até mesmo em redes de computadores, já que a extração de dados nesse caso se dá por conteúdos publicados na Internet.

É verdade que ao longo dos últimos anos, diversas linguagens de programação de alto nível possuem alguma API ou biblioteca que facilitam que esta extração ocorra, fazendo de forma automática um trabalho que o programador teria que fazer de forma manual. Mas isso, ainda não retira a necessidade de que os usuários dessas APIs necessitem ter um certo conhecimento técnico para fazer alguns ajustes em termos de código e análise do conteúdo a ser extraído.

Nesse contexto, o presente trabalho tem como objetivo a implementação de uma aplicação que permite que a configuração de coleta de informações de páginas com publicações de artigos em geral, seja possível de ser realizada por um usuário comum, sem qualquer tipo de conhecimento em programação, extração de dados ou crawling. A proposta é que isso seja feito por meio do fornecimento de exemplos, onde um usuário pode entrar com a URL da página onde o artigo está publicado, título, subtítulo, conteúdo, nome do autor, e a data de publicação.

Ainda, o usuário tem a liberdade de configurar um crawler de maneira personalizada, podendo selecionar um ou mais dos quaisquer websites que foram previamente mapeados. Também, terá a possibilidade de fornecer palavras e/ou frase de seu interesse, para que sirva como filtro para o crawler fazer ou não a coleta e, após isso, enviar um relatório dos artigos extraídos para os e-mails que o usuário também desejar.

2. Fundamentação teórica

Nesta seção é discutido o referencial teórico necessário para compreensão da proposta apresentada pelo presente trabalho. São abordados os principais conceitos, definições e características relacionados a extração de dados em páginas webs e Web Crawler. Também, é apresentado o algoritmo de Ratcliff-Obershelp para comparação textual.

2.1. Web Crawler

"Um web crawler, também conhecido como spider ou robô, é um programa que baixa automaticamente páginas da Web" [Liu 2011]. Ele pode ser utilizado para vários fins, como por exemplo, para indexar páginas web para motores de busca, como o da Google, para fazer o arquivamento da Web, e para realizar data mining [Olston and Najork 2010].

O algoritmo básico de um web crawler é bem simples: dada uma lista de URLs iniciais, chamadas de sementes, o crawler faz o download de todas as páginas web endereçadas por essas URLs iniciais, extrai os hyperlinks contidos nessas páginas, e interativamente faz o download das páginas web apontadas por esses hyperlinks [Olston and Najork 2010].

Muitas vezes pode-se não querer fazer a navegação de toda a Web, focando-se apenas em acessar páginas de certas categorias ou tópicos. Um web crawler focado tenta direcionar o crawler para páginas de certas categorias interessantes para o usuário [Liu 2011]. Este trabalho implementa um web crawler focado que busca na Web páginas que possuem artigos onlines publicados, e faz a extração dos mesmos.

2.2. O algoritmo de Ratcliff-Obershelp

O algoritmo de Ratcliff-Obershelp [Ratcliff 1988], também conhecido como *Gestalt Pattern Matching*, foi desenvolvido John W. Ratcliff e John A. Obershelp em 1983 e publicado em julho de 1988 no Dr. Dobb's Journal. Este, é um dos algoritmos existentes para determinar a similaridade entre duas strings, ou seja, dois conteúdos textuais. O cálculo de similaridade é definido através da seguinte equação:

$$Dro = \frac{2Km}{|S1| + |S2|} \tag{1}$$

Na equação, S1 representa o tamanho da primeira string, S2 o tamanho da segunda string e Km é a variável de caracteres correspondente. Esta última, é definida como o tamanho da substring comum mais longa entre os dois textos, somado recursivamente com o número de caracteres correspondente fora desta região, tanto para a esquerda quanto para a direita.

O resultado desta equação, representado por Dro, é o valor de similaridade entre duas cadeias de caracteres, onde assume valores entre 0 e 1:

$$0 \leq Dro \leq 1 \tag{2}$$

O valor 1, representa uma correspondência completa, ou seja, 100% de similaridade. Já 0, significa que não há correspondência, nem mesmo de um único carácter.

Supondo S1 = Pennsylvania e S2 = Pencilvaneya, o passo a passo abaixo exemplifica o cálculo que representa a similaridade entre as duas palavras:

- A maior sequência de carácter entre as duas palavras é a substring "lvan", de tamanho 4. Portanto, o valor de Km inicial é 2 x 4 = 8;
- Continuando com os dois pares restantes das palavras, tanto da parte esquerda quanto da parte direita, temos $\{Pennsy, Penci\}$ à esquerda e $\{ia, eya\}$ à direita;
- Partindo do conjunto da esquerda, $\{Pennsy, Penci\}$, a maior sequência de caracteres é a substring "Pen". Então, incrementa-se o valor de Km para 8 + (2 x 3) = 14. As porções restantes aqui, são $\{nsy, ci\}$ que não possui nem um carácter sequer em comum, para esta parte então, finaliza-se.
- Agora para o lado direito das palavras, temos $\{ia, eya\}$. Pode-se perceber que a única sequência correspondente entre as duas substrings é o carácter "a". Então, incrementa-se Km para 14 + (2 x 1) = 16 e ignora-se os carácteres restantes.

- Aplicando na fórmula, temos

$$Dro = \frac{2Km}{|S1| + |S2|} = \frac{2 * (4 + 3 + 1)}{|12| + |12|} = \frac{16}{24} = \frac{2}{3} = 0,66 \quad (3)$$

Portanto, o valor de similaridade entre as palavras Pennsylvania e Pencilvaneya é de 0,66, ou seja, 66%.

3. Trabalhos relacionados

A seguir, são descritos alguns trabalhos muito interessantes que serviram como inspiração e referência para o desenvolvimento deste.

3.1. Extração automática de sites de notícias

A proposta abordada em [Reis et al. 2004] para fazer a extração automática de notícias contidas em sites da Web utiliza como base o algoritmo de Distância de Edição entre Árvores. A lógica por trás deste algoritmo é encontrar um conjunto mínimo de operações que seja capaz de realizar a transformação de uma árvore X em uma outra árvore Y. Esta ideia levou os autores a criar o seu próprio algoritmo, chamado de RTDM, que é então usado para identificar passagens de textos relevantes contendo notícias e seus componentes, extraí-los e descartar elementos que não fazem parte da notícia em si, como por exemplo, banners, links, etc.

3.2. DEByE

DEByE - Data Extraction By Example - [Laender et al. 2001], apresenta uma abordagem para extrair informações de fontes da Web, baseado em um pequeno conjunto de exemplos especificados pelo próprio usuário, onde ele especifica os exemplos de acordo com uma estrutura de sua preferência. Esta estrutura é descrita no tempo de especificação do exemplo, para isso, o usuário interage com uma ferramenta gráfica que adota tabelas aninhadas como seu paradigma visual. As tabelas são simples, intuitivas e permitem proteger o usuário de detalhes técnicos (como tags HTML, operadores e autômatos de aprendizado) relacionados ao problema de extração. Os exemplos fornecidos pelo usuário são então usados para gerar padrões que permitem extrair dados de novos documentos. Para a extração, foi adotado um procedimento bottom-up que se mostrou eficaz com várias fontes da Web.

3.3. qFex

qFex: um crawler para busca e extração de questionários de pesquisa em documentos HTML - [Mathias 2017], teve como objetivo a extração de questionários de pesquisa presentes nas mais variadas páginas webs. A aplicação é dividida em dois grandes componentes: o crawler e o extrator. Ambos, recebem como entrada um arquivo de configuração que possui as configurações do banco de dados, biblioteca de crawler, das URLs sementes para a busca e extração de valores para os parâmetros utilizados.

A função do crawler é varrer a web, utilizando como ponto de partida as URLs de entrada fornecidas no arquivo de configuração, a fim de buscar questionários que possuem características específicas e salvar os links na base de dados.

Já o extrator, tem como função buscar na base as URLs, e possivelmente também o arquivo de configuração, e extrair os dados dos questionários para um outro banco de dados.

Para realizar a detecção e a extração dos questionários, o autor aplicou heurísticas para diferenciar os questionários de qualquer outro tipo de formulário na web.

4. CrawlEX

Nesta seção, é apresentada a ferramenta desenvolvida durante o trabalho, que auxiliará usuários leigos a configurarem suas próprias coletas de artigos que estão publicados na internet. O repositório onde o projeto está publicado é acessado em https://gitlab.com/marcoslessa.cs/crawlex/.

4.1. Interface Gráfica

A fim da aplicação ser de fácil uso aos usuários, fez-se necessário a criação de um projeto para a interface gráfica da aplicação, onde o usuário pode fornecer exemplos de artigos e configurar um crawler para execução.

4.1.1. Fornecer exemplos

A figura 1 mostra seis campos onde é possível a inserção de dados a respeito de um artigo. São eles: URL, título, conteúdo, subtítulo, autor e a data de publicação. Sendo que URL, título e conteúdo são campos obrigatórios, caso contrário, os dados informados não serão validados, e o artigo exemplo, não será adicionado.

Após ao menos os campos obrigatórios estarem preenchidos, o usuário poderá clicar no botão de confirmar, que está localizado no canto inferior direito da tela, onde o ícone está na cor verde. Feito isso, o artigo será adicionado no retângulo branco abaixo da escrita “Artigos exemplos”, que contará com uma lista de todas as URLs dos artigos que já foram fornecidos como exemplo.

Figure 1. Captura da tela onde os artigos exemplos serão fornecidos

4.1.2. Executar um crawler

A figura 2, apresenta a tela onde é possível que o usuário faça a seleção de todas as páginas web que desejar, desde que a mesma já tenha tido sua coleta previamente configurada através do fornecimento de exemplos, como vimos.

A página que está selecionada é o Jornal Tabloide. Para que esta página seja navegada pelo crawler a ser configurado, é necessário que o usuário clique no botão ”Adicionar”, ao lado. Feito isso, esta página será incluída no grande retângulo branco na direita da tela, logo abaixo do label ”Websites a executar”. Pode-se adicionar a quantidade de páginas que o usuário quiser. Caso uma página erroneamente for adicionada na lista para ser visitada pelo crawler, basta selecionar a mesma e então, clicar no botão ”Remover website”.

Também, ainda na configuração de um novo crawler, o usuário é capaz de digitar palavras e frases do seu interesse que possam se fazer presente nos artigos que serão navegados por esta página, sendo assim, qualquer artigo que não satisfaça as condições expressas pelo usuário, serão completamente ignorados e portanto, não coletados.

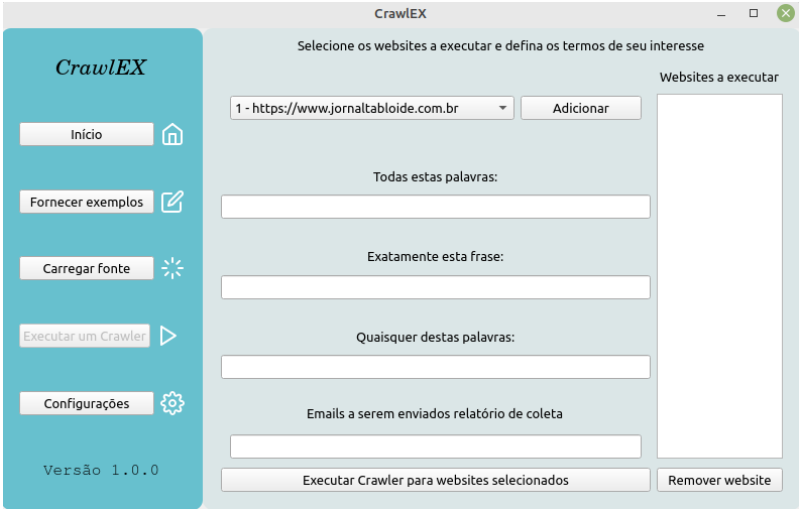

Figure 2. Captura da tela para configuração de um crawler

4.2. Crawler

A arquitetura e o funcionamento do crawler é bem simples, e pode-se abstrair em 3 componentes: Websites, LinkFinder e URLPatternChecker. A figura 3 traz a arquitetura básica do crawler.

Websites: tabela do banco de dados, responsável por armazenar a URL semente de cada site ao qual tenha exemplos fornecidos pelo usuário e salvos na tabela ExampleArticle.

LinkFinder: componente responsável por buscar na base de dados a URL semente de uma fonte e a partir desta, buscar em seu documento HTML apenas URLs que possuem o mesmo domínio. Toda e qualquer URL de domínio diferente será descartada e não será navegada.

URLPatternChecker: componente responsável por identificar se uma URL contém um possível artigo ou não. Se possuir, a URL é enviada para o extrator, caso contrário, é descartada.

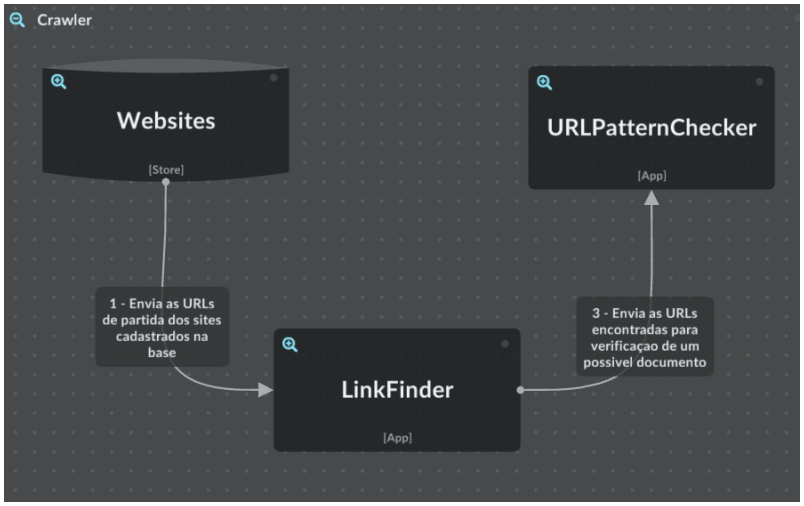

Figure 3. Abstração da arquitetura do crawler

4.3. Configuração de extração

Para que a extração de informações de um site seja viabilizada, é necessário que se tenha uma configuração de extração única para cada um deles. Essa configuração é formada por tags Tags HTML. A tag principal para a extração, será aquela a qual o seu conteúdo textual tenha o maior coeficiente de similaridade com os exemplos fornecidos pelo usuário. Esse coeficiente, é obtido aplicando o algoritmo Ratcliff-Obershelp. A figura 4 traz o fluxograma para a geração da configuração.

ExampleArticle: tabela do banco de dados responsável por armazenar os artigos fornecidos como exemplos pelo usuário.

TagFinder: componente responsável por buscar todos as tags HTML dentro do documento de um site.

ContentComparator: componente que tem como responsabilidade receber as tags HTML e seus conteúdos, e aplicar o algoritmo Ratcliff-Obershelp, comparando os conteúdos dessas tags com os conteúdos dos exemplos fornecidos pelo usuário. Por fim, retorna a tag que obteve o maior percentual de similaridade para cada campo de extração.

TagCleaner: após a identificação da melhor tag por parte do ContentComparator para extração de um campo, este componente tem como objetivo identificar possíveis tags filhas desta, que possuem conteúdos que não devem ser extraídos, como por exemplo anúncios no meio de um texto. Aqui, o algoritmo Ratcliff-Obershelp é aplicado a todo momento, a fim de verificar se a remoção de uma tag resultou em uma maior similaridade ou não. Se resultou, remove-a de fato, caso contrário, mantém.

ExtractionConfigurator: componente central, responsável por designar tarefas aos demais e fazer o controle geral. No fim, retorna uma tag principal para extração e uma lista com tags de limpeza do conteúdo, para cada campo que se deseja extrair.

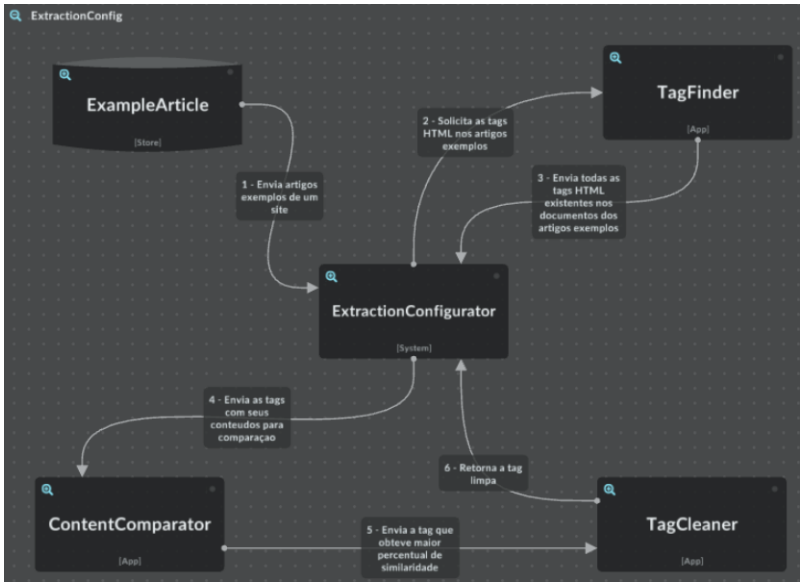

Figure 4. Abstração da arquitetura do funcionamento para geração da configuração de extração de artigos em um site

4.4. Extrator

Responsável por receber a configuração de extração e a URL de onde extrair o conteúdo pelo crawler, o extrator simplesmente busca as tags HTML referente a cada campo do artigo, extrai as informações e armazena no banco de dados. A figura 5 mostra o fluxo básico.

ExtractionConfig: resultado do procedimento descrito na seção 4.3. Onde uma tag principal para extração e uma lista de possíveis tags para remoção de conteúdo lixo, servem como uma das entradas para o extrator, a fim de realizar a extração dos campos de um artigo.

Crawler: componente descrito na seção 4.2, que faz o envio das possíveis URLs a terem seu conteúdo extraído pelo extrator.

ContentComparator: componente que tem como responsabilidade receber as tags HTML e seus conteúdos, e aplicar o algoritmo Ratcliff-Obershelp, comparando os conteúdos dessas tags com os conteúdos dos exemplos fornecidos pelo usuário. Por fim, retorna a tag que obteve o maior percentual de similaridade para cada campo de extração.

Articles: tabela do banco de dados responsável por armazenar os artigos coletados pelo extrator durante a navegação do crawler, de acordo com os temos configurados pelo usuário.

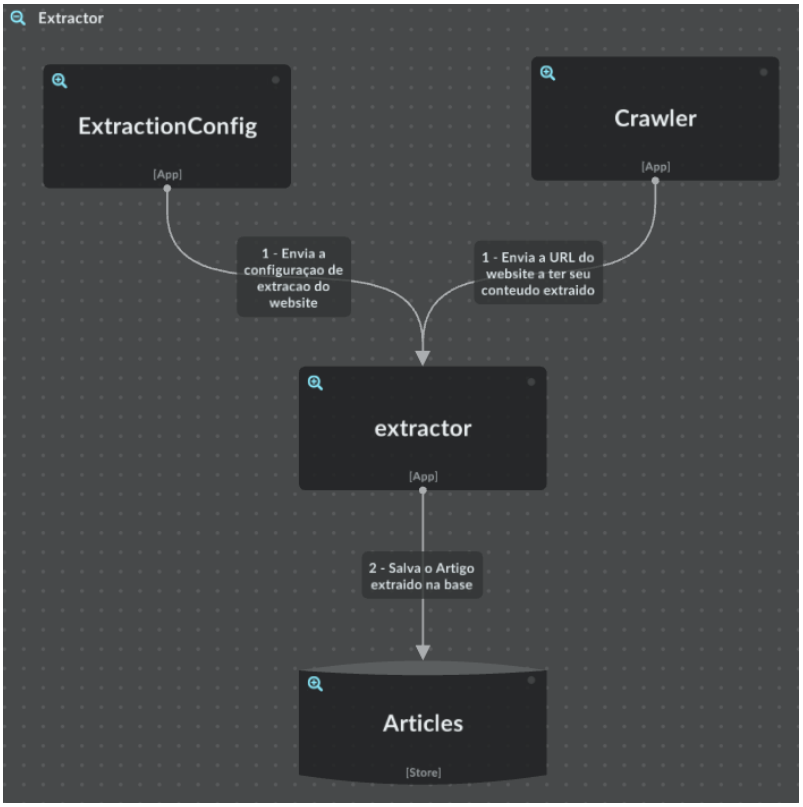

Figure 5. Captura da tela onde os artigos exemplos serão fornecidos

5. Experimentos e resultados

Nesta seção são apresentados os experimentos conduzidos com o intuito de avaliar a quantidade e qualidade dos conteúdos extraídos de acordo com os exemplos fornecidos. Primeiramente, descreve o experimento conduzido pelo autor do trabalho e posteriormente, descreve o experimento com a participação de 3 usuários leigos, externos ao desenvolvimento deste trabalho.

30 websites foram testados, entre eles sites de notícias, crônicas, contos, letras de música, blogs, entre outros. O crawler por 5 horas. Os experimentos trazem as médias de similaridade obtidas para cada um dos campos entre todos os artigos que foram coletados.

5.1. Exemplos fornecidos pelo autor

Com 1 único exemplo fornecido por fonte, o crawler navegou por 20467 páginas, onde dessas, 13566 sendo possíveis páginas com artigos para ser coletados. Entretanto, ao tentar extrair conteúdo para essas 13566 páginas, a extração ocorreu com sucesso para apenas 588, ou seja, 4,33%. A figura 6 mostra a similaridade de cada campo extraído.

Com 2 exemplos fornecidos por fonte, o crawler navegou por 18304 páginas, onde dessas, 8939 sendo possíveis páginas com artigos para ser coletados. Ao tentar extrair conteúdo para essas 8939 páginas, a extração ocorreu com sucesso para 6982, o que nos dá 78,10% de efetividade na combinação da identificação de artigos e extração dos mesmos. A assertividade para os campos foi muito melhor comparado ao experimento com 1 único exemplo fornecido. A figura 7 mostra a similaridade de cada campo extraído.

Com 3 exemplos fornecidos por fonte, o crawler navegou por 18145 páginas, onde dessas, 10453 sendo possíveis páginas com artigos para ser coletados. Ao tentar extrair conteúdo para essas 10453 páginas, a extração ocorreu com sucesso para 7092, o que nos dá 67,84% de efetividade na combinação da identificação de artigos e extração dos mesmos. A figura 8 mostra a similaridade de cada campo extraído.

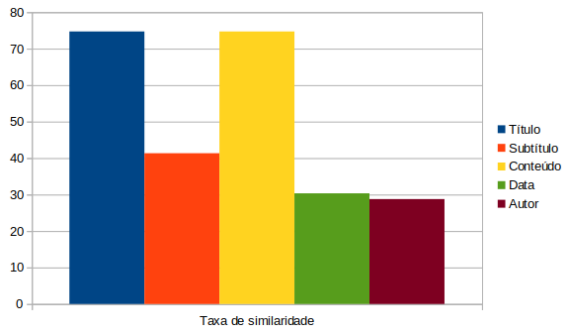

Figure 6. Taxa de similaridade dos conteúdos com 1 exemplo fornecidos

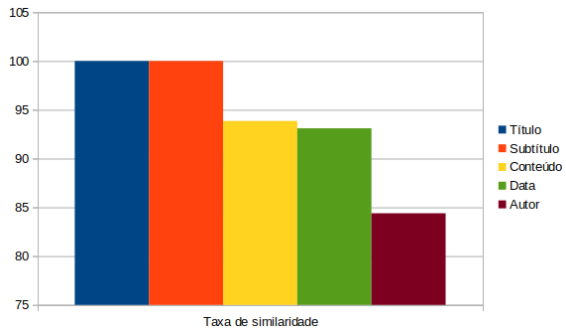

Figure 7. Taxa de similaridade dos conteúdos com 2 exemplos fornecidos

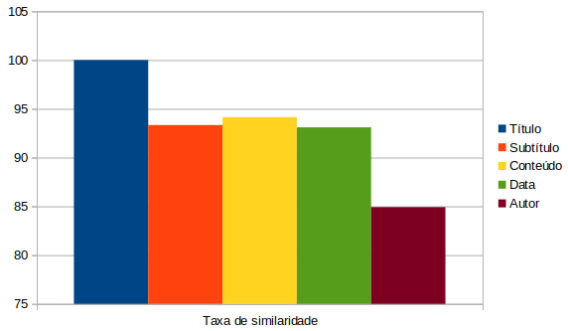

Figure 8. Taxa de similaridade dos conteúdos com 3 exemplos fornecidos

5.2. Exemplos por usuários leigos

De acordo com a seção anterior, 2 exemplos por website se mostrou a melhor abordagem dentre as feitas. Com base nisso, foi solicitado para que 3 usuários leigos fornecessem 2 exemplos para os mesmos websites, com a supervisão do autor, mas sem que houvesse interferência. O primeiro usuário com 19 anos de idade, o segundo com 36 e o terceiro com 54. As figuras 9, 10 e 11 trazem os resultados, onde podemos observar que são bem diferentes entre si, indicando que a qualidade da extração depende da qualidade do exemplo fornecido pelo usuário.

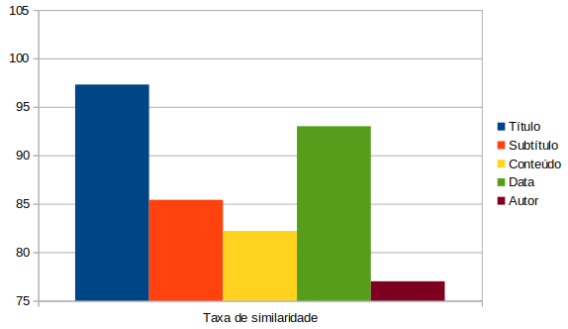

Figure 9. Gráfico de similaridade entre os conteúdos extraídos e os conteúdos esperados, 2 exemplos fornecidos pelo usuário de 19 anos

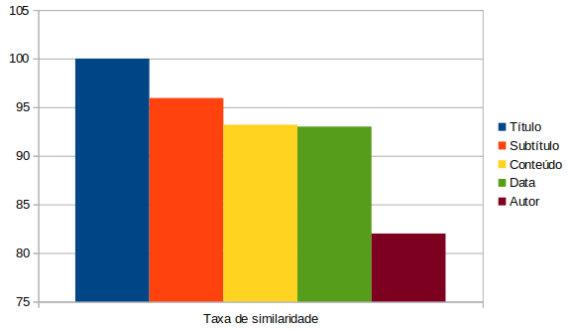

Figure 10. Gráfico de similaridade entre os conteúdos extraídos e os conteúdos esperados, 2 exemplos fornecidos pelo usuário de 36 anos

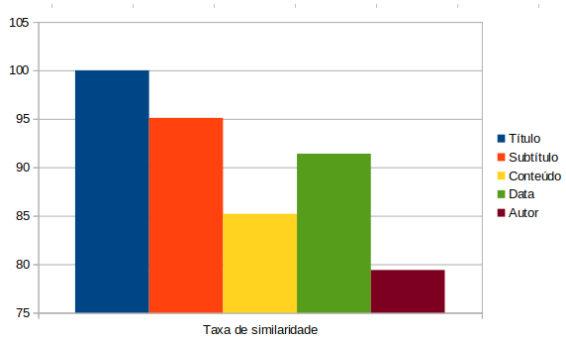

Figure 11. Gráfico de similaridade entre os conteúdos extraídos e os conteúdos esperados, 2 exemplos fornecidos pelo usuário de 54 anos

6. Conclusões e trabalhos futuros

Uma enorme quantidade de informação se faz presente em artigos publicados na internet, fazendo com que as pessoas a utilizem a fim de adquirirem conhecimento, se manterem informadas e até mesmo monitorar o que está sendo publicado sobre determinados assuntos. Isso pode ser facilitado com a automatização da navegação e extração destas informações, porém, geralmente, para isso é necessário que se tenha conhecimentos sólidos em programação, extração de dados e até mesmo redes de computadores.

Sendo assim, o objetivo deste trabalho foi viabilizar através de um aplicativo com interface gráfica, a possibilidade de pessoas sem tais conhecimentos de realizarem as tarefas mencionadas de forma automatizada. A proposta foi de um cenário onde o usuário interessado em extrair informações de artigos online em um determinado website, pode fazê-lo por fornecer exemplos dos conteúdos em que deseja coletar.

Isso foi possível de ser implementado com o auxílio do algoritmo Ratcliff-Obershelp para a comparação do conteúdo esperado com o conteúdo extraído, e através disto, fazendo lapidação para se ter um conteúdo cada vez mais limpo.

Para trabalhos futuros, pode-se considerar os seguintes pontos para melhoria da aplicação:

1. Permitir a navegação e a extração de informações em sites onde a navegação acontece de forma dinâmica, bem como a apresentação dos conteúdos;
2. Aplicar conceitos de programação concorrente e paralela, a fim de tornar a navegação e extração *multithreading*, otimizando e possibilitando que a extração das informações ocorra de maneira paralela para cada site, e não sequencial (quando possível);
3. Refinar o algoritmo de extração baseado em exemplos, a fim de melhorar a assertividade da coleta das informações.

7. Referências

Olston, C.; Najork, M. ”Web Crawling”, Foundations and Trends® in Information Retrieval: Vol. 4: No. 3, pp 175-246. http://dx.doi.org/10.1561/1500000017.

Leander, A. H. F.; Ribeiro-Neto, B. A.; Da Silva., A. S. DeByE – Data Extraction by Example. Data and Knowledge Engineering (2001). To appear.

Mathias, G. (2017). qFEX - um crawler para busca e extração de questionários de pesquisa em documentos HTML. Repositório Institucional da UFSC.

Liu, B. Web Data Mining: Exploring Hyperlinks, Contents and Usage. (Data-Centric Systems and Applications). Chicago, 2011.

Ratcliff, W. J. Pattern Matching: The Gestalt Approach. Dr. Dobb's Journal, page 46, July 1988.

Helpers for computing deltas. Docs Python, 2022. Disponível em https://docs.python.org/3/library/difflib.html#module-difflib. Acesso em 15 de abril de 2022.