Journal of Information and Data Management, 2024, 13:1 doi: 10.5753/jidm.2024.XXXXXX

Desenvolvimento de uma aplicação para a extração de dados abertos

Fernando Aguiar [**Universidade de Caxias do Sul** | ***faguiar@ucs.br***]

Universidade de Caxias do Sul, Rua Francisco Getúlio Vargas, 1130, 95070-560, Caxias do Sul, RS, Brasil.

Abstract The access to and availability of public data in Brazil has gradually increased in recent years, driven by initiatives such as the enactment of the Access to Information Law and the creation of the Brazilian Open Data Portal. However, the platform for making this information available is at the discretion of each public institution, making uniform and centralized access to data difficult. This scenario highlighted the challenges faced by the City Living Lab, which obtains open data in an automated manner from various public platforms. Several projects were developed to assist the Lab in meeting multiple demands, including open data extraction. However, it was not possible to resolve this challenge effectively. One potential solution is developing a proposed web application capable of extracting data from multiple sources in an independent, automated, and user-friendly manner, allowing users to select which data sources to use.

Keywords: Open Data, Data Extraction, Web Scraping, Web Application, ChatGPT

1 Introdução

O acesso a informação pública no Brasil tem aumentado gradualmente desde o início do século XXI. Grande parte desse aumento se deve diversas iniciativas tomadas pela administração pública. Muitas das iniciativas foram direcionadas a temas de transparência, austeridade fiscal e, também, a ampliação da participação social [Neves, 2013].

Através da promulgação da Lei de Acesso à Informação[1], os artigos 5º e 37º da Constituição Federal foram implementados, assegurando o comprometimento do Estado no fornecimento de informações públicas de forma passiva, sem necessidade de solicitação [Neves, 2013]. Além disso, a lei exige também que os dados, tratados ou não, sejam disponibilizados de maneira transparente, clara e acessível, utilizando-se de tecnologias da informação.

Embora a Lei de Acesso à informação tenha representado um avanço significativo na transparência, ainda havia desafios no que se refere à publicação de dados abertos. Dados abertos são informações que podem ser livremente utilizadas, modificadas e compartilhadas por qualquer cidadão para diversos fins [Knowledge, 2024]. Em resposta a essa necessidade, no final de 2011, o Ministério do Planejamento lançou o Portal Brasileiro de Dados Abertos[2], com o objetivo de centralizar a disponibilização das bases de dados governamentais e facilitar o acesso a elas.

Embora o lançamento do Portal Brasileiro de Dados Abertos tenha facilitado o acesso a diversas bases de dados governamentais, sua crescente utilização também evidenciou algumas limitações. Muitos desses conjuntos de dados carecem de detalhamento e estão concentrados nas esferas federais e estaduais, deixando lacunas em outras áreas. Além disso, a inconsistência nos formatos das bases de dados dificulta o acesso, exigindo conhecimento técnico para extração e análise das informações. A falta de participação de outras instituições públicas na publicação de dados na plataforma também agrava esse problema, tornando mais complexa a utilização de múltiplas bases de dados em projetos [Silva *et al*., 2018].

Ainda que o Portal Brasileiro de Dados Abertos tenha representado um avanço, a ausência de dados de algumas instituições, especialmente de esferas municipais, continua sendo um obstáculo significativo. Essa limitação dificulta a integração de dados de diferentes fontes, impactando diretamente iniciativas como o City Living Lab, um laboratório colaborativo que reúne empresas, cidadãos, governos e instituições de ensino para desenvolver soluções voltadas à melhoria da vida urbana. Apesar dos avanços, o City Living Lab ainda enfrenta dificuldades na extração automatizada de dados provenientes de múltiplas fontes. Embora existam ferramentas que buscam automatizar esse processo, muitas carecem de intuitividade e autonomia, exigindo intervenções manuais. Exemplificando, em 2021, foram necessários aproximadamente seis meses de trabalho manual para obter e organizar os dados abertos para uso nos projetos Perfil Socioeconômico de Caxias do Sul[3] e Observatório de Cidades Corede Serra[4] do laboratório.

Diante desse contexto, este estudo busca responder à seguinte questão: É possível automatizar o processo de extração de dados abertos, reduzindo a necessidade de intervenção humana? Portanto, o objetivo principal é desenvolver um protótipo para a automatização de extração de bases dados, visando reduzir a dependência de interação do usuário nesse processo.

Este trabalho está estruturado da seguinte forma: a sessão 2 apresenta os conceitos de fundamentais para o projeto, além de analisar estudos relacionados ao City Living Lab e à extração de dados em contextos governamentais. A sessão 3 detalha as características do projeto desenvolvido. Nos Capí-

[1] https://www.planalto.gov.br/ccivil_03/_ato2011-2014/2011/lei/l12527.htm

[2] https://dados.gov.br

[3] https://www.citylivinglab.com/report-cxs-socioecono

[4] https://www.citylivinglab.com/corede-serra

tulos 4 e 5, são abordados o processo de desenvolvimento e os estudos de caso da aplicação. Por fim, a sessão 6 apresenta as conclusões finais.

2 Referencial Teórico

Nesta sessão, serão discutidos os principais termos, definições e projetos fundamentais para a compreensão e desenvolvimento do tema abordado neste trabalho, incluindo conceitos como dados abertos, *web scraping* e ETL (*Extract, Transform and Load*). A Seção 2.4 explora os projetos desenvolvidos que são essenciais para a realização desta pesquisa. Além disso, os itens subsequentes desta seção proporcionam uma visão sobre o processo de extração de dados por meio de técnicas como *web scraping*.

2.1 Dados Abertos

Dados abertos referem-se a informações disponibilizadas publicamente para que qualquer cidadão possa acessá-las, utilizá-las, modificá-las e compartilhá-las livremente. No entanto, para que esses dados sejam verdadeiramente abertos, é necessário que cumpram certos requisitos que assegurem sua autenticidade, mantenham sua procedência e garantam que permaneçam acessíveis e reutilizáveis ao longo do tempo [Knowledge, 2024]. Esses requisitos incluem:

- Os dados devem estar sob domínio público ou serem disponibilizados sob uma licença aberta, que permita seu uso sem restrições.
- Os dados precisam ser acessíveis via internet sem qualquer custo para o usuário.
- Os dados devem ser disponibilizados em formatos que possam ser lidos e processados por computadores, permitindo que elementos individuais sejam facilmente acessados e modificados. Exemplos de formatos incluem CSV (*Comma-separated values*), XML (*Extensible Markup Language*) e JSON (*JavaScript Object Notation*).
- É essencial que os dados estejam em formatos abertos, sem restrições financeiras ou de uso, e que possam ser processados por ferramentas de *software* livre.

Por sua vez, dados conectados referem-se a um conjunto de boas práticas destinadas à publicação e interligação de dados na *web* de maneira estruturada. Essas práticas envolvem o uso de tecnologias padronizadas para representar os dados, permitindo que diferentes bases de dados sejam conectadas, facilitando o acesso e a reutilização, tanto por pessoas quanto por sistemas automatizados [Bandeira *et al*., 2015]. A adoção dessas práticas contribui para a criação de um ecossistema de dados mais acessível, interconectado e integrado. Essas práticas envolvem:

- Utilizar URI (*Uniform Resource Identifier*) como identificadores para entidades e conceitos.
- Empregar URI baseados em HTTP (*Hypertext Transfer Protocol*), permitindo que entidades e/ou conceitos sejam facilmente localizáveis.
- Seguir os padrões estabelecidos pelo W3C (*World Wide Web Consortium*), garantindo a interoperabilidade dos dados.
- Incluir *links* para outros URI relacionados, ampliando as conexões e o conhecimento disponível sobre os temas abordados.

Por fim, dados abertos conectados são uma combinação entre os princípios de dados abertos e conectados. A implementação dessas práticas, tanto na publicação quanto no consumo dessas informações, simplifica a exploração dos dados abertos governamentais. Além de aumentar a transparência nas operações públicas, essa abordagem assegura que os dados sejam acessíveis e interpretáveis por máquinas, facilitando sua integração e automação no processamento de informações [Bandeira *et al*., 2015].

2.2 ETL

ETL é um processo utilizado para extrair dados de uma ou mais fontes, transformá-los de acordo com as necessidades do negócio e, em seguida, carregá-los em um sistema de armazenamento adequado [Hamoud *et al*., 2018]. O processo ETL é composto por três etapas principais:

1. Extração: Nessa etapa, os dados são coletados de diversas fontes, que podem ser de diferentes formatos e localizações. O objetivo é reunir todas as informações necessárias para o processo.
2. Transformação: Após a extração, os dados passam por modificações conforme as regras do negócio. Isso pode incluir a aplicação de filtros, conversão de formatos, cálculos de valores derivados e outras operações que garantam a consistência e utilidade dos dados.
3. Carregamento: Na última etapa, o destino ou os destinos onde os dados serão armazenados são definidos. Em seguida, os atributos das fontes são mapeados para seus correspondentes no destino, garantindo uma correta associação e organização. Por fim, os dados transformados são carregados no destino, tornando-os prontos para uso e análise.

A etapa de transformação, também, pode realizar a limpeza dos dados. Essa limpeza consiste na detecção de erros e inconsistências nos dados que podem impactar na qualidade dos dados [Trujillo and Luján-Mora, 2003].

2.3 Web Scraping

Web Scraping é uma técnica utilizada para extrair dados da WWW (World Wide Web) e posteriormente salvá-los em arquivos ou bancos de dados para usos futuros [Zhao, 2017]. Essa extração pode ocorrer de diversas formas, sendo elas: manualmente por um usuário, automatizada ou semi-automatizada por *bots* ou rastreadores *web*. Além disso, os dados são coletados da internet utilizando o protocolo HTTP ou através de um navegador *web*.

O *web scraping* pode ser utilizado, por exemplo, para o monitoramento em tempo real dos preços de imóveis, tanto comerciais quanto residenciais, no Reino Unido durante a

crise da Covid-19. Para extrair esses dados, os pesquisadores utilizaram diversas fontes, como Zoopla[5], Rightmove[6] e OnTheMarket[7] [Bricongne *et al*., 2023]. Dessas plataformas, foram extraídas informações como cidade, bairro, preço, quantidade de quartos, entre outros dados relevantes. Após a coleta, os dados foram analisados para compreender o comportamento do mercado imobiliário durante a crise [Bricongne *et al*., 2023].

Outro exemplo de utilização, foi a criação da plataforma *Preventable Death Tracker*, desenvolvido como uma plataforma aberta com o intuito de catalogar os tipos de mortes que podem ser prevenidas na Inglaterra e em Gales. [Zhang and Richards, 2023]. A extração é realizada por meio da aplicação *web Courts and Tribunals Judiciary* [8] e através da fonte *Office of National Statistics*.

2.4 Cidades do Conhecimento

Esta seção engloba três trabalhos desenvolvidos que buscavam atender as demandas do laboratório CityLivingLab, sendo eles: ”Coleta dos Indicadores da Plataforma de Cidades do Conhecimento” [Bregalda, 2021], ”Desenvolvimento da Automatização da Coleta de Dados na Plataforma de Cidades do Conhecimento” [Silva, 2022] e ”Virtualização de Ferramentas de Carga de Dados da Plataforma de Cidades do Conhecimento” [Rodrigues, 2023].

O primeiro trabalho, baseado na pesquisa de Battistelo [Battistelo, 2018], detalha o desenvolvimento de uma aplicação *web* para armazenar, calcular e consultar dimensões do Sistema de Capitais, as dimensões envolvem Identidade, Inteligência, Relacional, Financeiro, Investimento, Humano Individual, Humano Coletivo, Instrumental-material e Instrumental-intangível, eram inseridos e gerenciados por planilhas eletrônicas. No entanto, a aplicação ainda dependia de uma planilha eletrônica manipulada manualmente para a inserção de dados, já que não existe uma regra capaz de categorizar automaticamente os indicadores de capital. Isso força o usuário a buscar, copiar e organizar os dados manualmente, repetindo o processo a cada atualização, o que impede a automação completa da integração de dados [Notari *et al*., 2019].

Afim de automatizar e remover a dependência de utilizar planilhas eletrônicas manipuladas manualmente, foi proposto uma aplicação para realizar o mapeamento das bases de dados e com isso, realizar o processo de ETL para inserir os dados no banco de dados da plataforma de Cidades do Conhecimento [Bregalda, 2021].

O segundo projeto, concentra-se no desenvolvimento e uma plataforma para facilitar a inserção e visualização de indicadores a partir de outras fontes de dados [Silva, 2022]. Para o gerenciamento da plataforma, foram implementados controles de usuários, criação e visualização de tarefas de extração de dados, além da identificação e gerenciamento do modo de navegação em uma página *web* utilizado em diferentes etapas de uma tarefa.

Na Figura 1 é ilustrado o processo de criação e edição de uma tarefa de extração de dados. Durante esse processo, o

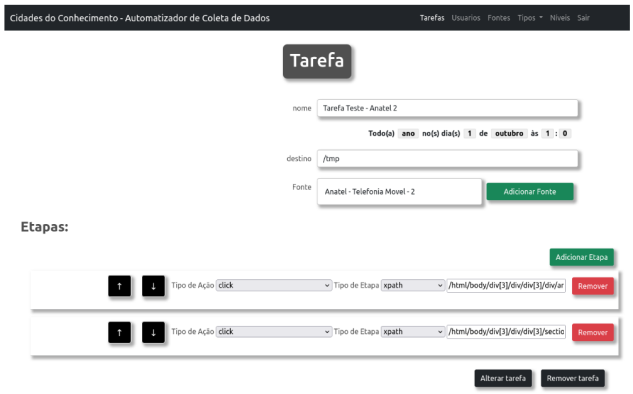

Figura 1. Criação e alteração de uma tarefa de extração.

usuário especifica a fonte de dados e o destino onde esses dados serão armazenados após a extração. Além disso, é necessário configurar as etapas, cada etapa adicionada contém uma ação[9], tipo de etapa como XPATH[10], seletores CSS(*Cascading Style Sheets*) ou *tags* HTML, e também o trecho de navegação para a execução da ação.

Os resultados obtidos com o desenvolvimento desta ferramenta incluem o mapeamento para a coleta de diversos tipos de dados brutos organizados em várias etapas e provenientes de cinco fontes relevantes para a Plataforma de Cidades do Conhecimento.

O terceiro trabalho propôs a virtualização em um único ambiente para integrar as ferramentas desenvolvidas nos trabalhos anteriores (Bregalda e Silva) [Rodrigues, 2023]. No entanto, o autor destaca que a instalação de múltiplos pacotes, a configuração de ambientes distintos e o uso de diferentes SGBDs e *frameworks* para as duas aplicações introduzem uma série de desafios. Entre eles estão a dificuldade de comunicação entre componentes em ambientes separados, a incompatibilidade entre pacotes de versões diferentes e a necessidade de gerenciar vários ambientes de desenvolvimento, o que aumenta os requisitos de recursos computacionais e a complexidade de manutenção [Rodrigues, 2023]. Para simplificar esse processo e facilitar o uso das plataformas, o autor propôs a criação de um único ambiente para ambas as aplicações utilizando contêineres Docker[11]. Foram criados três contêineres: dois para as aplicações, garantindo a independência dos ambientes, e um contêiner geral que hospeda os bancos de dados.

2.5 Web Scraping para extração de dados públicos

Neste trabalho, foi analisado como a utilização de técnicas de *web scraping* pode auxiliar na resolução de divergências na Câmara Municipal de Belo Horizonte, visando aumentar a transparência na publicação de informações sobre os gastos públicos no portal da câmara, em conformidade com as

[5] `https://zoopla.co.uk/`
[6] `https://rightmove.co.uk/`
[7] `https://onthemarket.com/`
[8] `https://judiciary.uk/prevention-of-future-death-reports/`

[9] Nesse cenário, o autor considera ações como cliques, mover o cursor até um elemento específico, entre outras situações.

[10] Linguagem para selecionar nós em documentos XML, utilizando uma sintaxe de caminho similar à de diretórios de arquivos. Ela permite definir expressões para localizar elementos específicos com base em atributos, valores de texto ou sua posição no documento [MDN, 2024b]

[11] Plataforma para criação, gerenciamento e execução de aplicações em contêineres.

diretrizes da Lei de Acesso à Informação (LAI) [Assis and Gomide, 2021]. Como solução para as divergências encontradas, foram empregadas técnicas de web *scraping* para extrair e transformar dados não estruturados em dados abertos.

Para realizar o *scraping*, foram desenvolvidos *scripts* para a extração de dados sobre vereadores e custeio parlamentar. Como cada página web utiliza diferentes estruturas HTML para exibir essas informações, foram adotadas abordagens específicas para cada caso. Por exemplo, para o *scraping* dos vereadores, os nomes e partidos foram obtidos buscando o seletor CSS *"view-content"* nos elementos HTML.

Na extração dos dados de custeio parlamentar, foi necessário identificar o campo de mês/ano de referência, uma vez que a página exige a seleção de um filtro. A busca foi realizada pelo atributo HTML "id"correspondente ao valor *"data"*. A biblioteca Selenium[12] foi utilizada para simular a ação humana de pesquisa, com base em uma lista predefinida de meses/anos. Em seguida, foi extraída a primeira ocorrência da *tag* HTML <*table*>, onde os dados estão armazenados [Assis and Gomide, 2021].

Após a extração das duas fontes de dados, cada uma resultou em um conjunto de dados específico e não relacionado. Posteriormente, essas informações foram agrupadas e relacionadas, permitindo calcular a soma de todos os gastos parlamentares e, também, separar esses gastos por vereador.

Adicionalmente, foram identificadas inconsistências na obtenção dos dados no portal, com resultados divergentes em diferentes extrações. Concluiu-se que certos dados não estavam mais disponíveis, possivelmente devido a manutenções no sistema, alterações no banco de dados ou até mesmo remoção proposital [Assis and Gomide, 2021].

2.6 Ferramenta web para monitoramento de gastos da prefeitura

O trabalho propõe a criação de uma aplicação *web* destinada à visualização, filtragem e comparação de informações extraídas do portal da transparência do município de Santa Cruz do Capibaribe [Dantas, 2021]. Para isso, a aplicação realiza a extração dos dados provenientes do portal da transparência, pois esses dados estão disponíveis por meio de tabelas na página *web*. Além disso, embora o portal ofereça a opção de exportar as tabelas em um arquivo CSV, o arquivo não é gerado em um formato adequado e não possui cabeçalho. Portanto, foi necessário utilizar técnicas de *web scraping* para realizar a extração das informações.

No desenvolvimento do extrator de informações, foi utilizada a linguagem de programação Python, em conjunto com as bibliotecas Selenium e Scrapy. Foram também desenvolvidas ferramentas auxiliares: uma para coletar a listagem dos pagamentos e outra para detalhar cada pagamento, estratégia adotada devido ao limite de tempo de sessão imposto pelo portal da transparência. Após a extração, os dados foram unificados em um único arquivo CSV, mapeando-se os pagamentos aos seus respectivos detalhes. A ferramenta oferece diversos gráficos interativos, como gráficos de dispersão de pagamentos e gráficos de densidade para valores de pagamentos realizados, agrupados por categorias [Dantas, 2021].

[12] `https://www.selenium.dev/`

2.7 Considerações finais

Antes de propor uma solução, analisou-se como os projetos abordados neste capítulo lidavam com a extração de dados e como esse processo era realizado, juntamente com a praticidade de utilizar essas plataformas. Como resultado, foi criada a tabela 2, que reúne as características e/ou processos presentes ou ausentes em cada projeto. Para facilitar a leitura, os trabalhos foram renomeados, conforme demonstrado na tabela 1, que apresenta a relação entre a nova nomenclatura e os respectivos títulos dos projetos.

Tabela 1. Relação de projetos renomeados

Nomenclatura	Título
T1	Coleta dos Indicadores da Plataforma de Cidades do Conhecimento
T2	Automação da Coleta de Dados na Plataforma de Cidades do Conhecimento
T3	Web Scraping para extração de dados públicos
T4	Ferramenta web para monitoramento de gastos da prefeitura

Tabela 2. Comparativo entre trabalhos

	T1	**T2**	**T3**	**T4**
Extração de dados		x	x	x
Extração de múltiplas fontes		x		
Configuração complexa de extração (xpath, seletores...)		x		
Possui Interface gráfica	x	x		x
Uso	x			x

Através da análise apresentada na tabela 2, é possível categorizar aspectos importantes de cada projeto e identificar desafios que podem ser úteis para o desenvolvimento do presente trabalho. Por exemplo, a limitação na extração de múltiplas fontes, observada no projeto T4, simplifica o processo de extração, mas torna o processo dependente da estrutura específica da página, impossibilitando uma abordagem mais genérica para a extração.

Entre os projetos que permitiam a extração de dados, apenas T2, permitia buscar informações de diferentes fontes. No entanto, essas abordagens apresentavam algumas fragilidades. T2 exigia a configuração de uma sequência de instruções para o *web scraper*, tornando o processo complexo e demorado, o que forçava o usuário a extrair manualmente a base desejada. Embora a abordagem resolva o problema da extração de dados, carece de flexibilidade, pois depende da configuração correta pelo usuário.

Além disso, a presença de uma interface gráfica impacta diretamente a usabilidade de uma solução futura. A ausência de uma interface gráfica, como em T3, impede seu uso por qualquer usuário, pois depende de conhecimentos técnicos para a utilização da solução.

Com base nessa análise, buscou-se uma alternativa flexível que não dependesse de elementos HTML específicos e que não exigisse configuração complexa pelo usuário[13], fa-

[13] Nesse contexto, o usuário é aquele que possui familiaridade em lidar com dados, mas não possui formação específica na área da computação.

cilitando assim a usabilidade. No próximo capítulo, essa alternativa será discutida em detalhes.

3 Projeto

Ao longo dos anos, diversos projetos foram desenvolvidos para auxiliar o City Living Lab, incluindo aplicações para mapear bases de dados e realizar o processo de ETL para carregar dados na plataforma de Cidades do Conhecimento [Bregalda, 2021]. Também foram criadas soluções para a inserção e visualização de indicadores a partir de dados brutos coletados por meio de técnicas de *web scraping* [Silva, 2022]. Além disso, houve a unificação dessas soluções em um único ambiente [Rodrigues, 2023] para que facilitasse o acesso às soluções.

No entanto, quando as aplicações foram utilizadas pelo City Living Lab, foram relatadas dificuldades na utilização dessas ferramentas, principalmente devido à ausência de automação na extração de dados e à complexidade do processo, que exigia configurações detalhadas, demandando mais tempo e conhecimento do que a coleta manual.

Para solucionar esses problemas, foi proposto a criação de uma aplicação *web* capaz de automatizar a extração de bases de dados a partir de uma URL fornecida. A aplicação será capaz de analisar o conteúdo das páginas, identificar ações necessárias, como a aplicação de filtros ou navegação adicional, e realizar a extração dos dados de forma automatizada. Além disso, a interface será simplificada e voltada para a usabilidade, permitindo que qualquer integrante do laboratório possa utilizá-la facilmente.

O Diagrama de Casos de Uso (figura 2) apresenta dois atores: o usuário e o módulo *web scraper*. O módulo *web scraper* é considerado um ator devido à sua interação e dependência das ações realizadas pelo usuário para que a extração de dados ocorra.

No primeiro requisito, destaca-se que o usuário deve fornecer uma URL para que o módulo *web scraper* possa iniciar o processo de extração de dados.

No segundo requisito, define-se que após a URL ser fornecida e validada, o módulo *web scraper* analisa o conteúdo da página *web* para identificar se existem elementos que requerem interação, como filtros ou navegação adicional.

No terceiro requisito, se durante a análise do conteúdo da página *web*, for identificado que são necessárias informações adicionais para prosseguir com a extração, o módulo *web scraper* solicita esses dados ao usuário.

O quarto requisito é apresentado que o usuário deve fornecer os parâmetros necessários para que a extração de dados possa continuar. Este requisito é dependente do requisito ”Solicitar informações extras”.

No quinto requisito, mostra-se que após as análises serem concluídas, o módulo *web scraper* inicia o processo de extração dos dados da página *web*.

No sexto requisito, define-se que o usuário pode selecionar quais informações são relevantes e devem ser mantidas. Esta etapa permite ao usuário filtrar os dados extraídos, garantindo que apenas os dados de interesse sejam processados e armazenados.

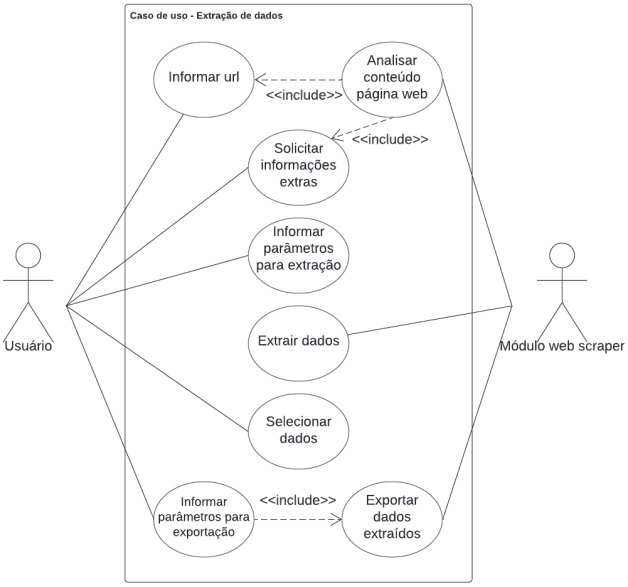

Figura 2. Diagrama de caso de uso - Extração de dados

No sétimo requisito, define-se que o usuário pode escolher a forma de exportação dos dados, nesse caso, devem ser fornecidos os parâmetros que definem como e onde esses dados serão salvos. Isso inclui a escolha do formato de exportação e o local de armazenamento.

No oitavo requisito, o ator módulo *web scraper* exporta os dados conforme a escolha do usuário no requisito ”Informar parâmetros para exportação”. Este requisito é acionado após a seleção dos dados e a definição dos parâmetros de exportação.

3.1 Fluxo de extração de dados

O processo de extração dos dados (Figura 3) é iniciado quando o usuário fornece uma URL válida. Com base nessa URL, o conteúdo da página *web* é obtido e encaminhado para análise.

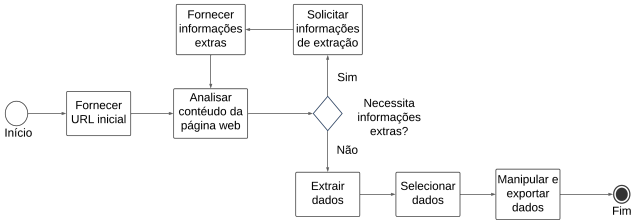

Figura 3. Fluxograma para a extração dos dados

Na etapa de análise ocorre a tentativa de identificar ações que possam depender de interação com o usuário como filtros ou navegações adicionais. Esta etapa é crucial, pois muitas páginas possuem filtros e etapas de navegação que são necessários para se acessar os dados.

A análise do conteúdo página é essencial porque permite a identificação da estrutura específica de cada página *web* fornecida pelo usuário. Sem a realização da análise, o processo de extração seria menos abrangente, sendo necessário a criação de rotinas de extração específicas para cada URL, o que limitaria a quantidade de páginas *web* das quais a aplicação poderia obter dados.

```
{
  "id": "multiSelectTema",
  "class": "multiselect__input",
  "placeholder": "Selecione um tema",
  "type": "text",
  "autocomplete": "off"
}
```

Figura 4. Estrutura construída de um elemento de formulário - Extração de dados

Caso a ferramenta identifique a necessidade de intervenção do usuário para a realização de filtros, será criada uma estrutura com propriedades do campo, contendo informações como tipo do campo, identificador e atributos de validação, por exemplo [MDN, 2024a].

Com a criação dessa estrutura (figura 4), é permitido que o formulário de filtro da página *web* analisada seja recriado, possibilitando ao usuário preencher os campos, caso necessário, diretamente na aplicação.

Se forem detectados possíveis redirecionamentos de URL, o analisador informará ao sistema que será necessário extrair o conteúdo dessa nova página, e o processo de análise deve ser repetido.

Uma vez detectado que não há mais necessidade de etapas de filtro ou redirecionamentos, serão recolhidas informações sobre como deve ser feito para obter os dados, se é preciso realizar a ação de um clique de um elemento <*button*> ou se os dados estão incorporados na página, por exemplo. Após o recolhimento dessas informações, a aplicação extrairá os dados, e uma amostra será enviada ao usuário para que ele possa selecionar quais informações são relevantes e definir como o conjunto de dados final será persistido.

Através dessa abordagem, o processo de extração de dados se torna mais dinâmico e flexível. Embora ainda dependa de intervenções pontuais do usuário para auxiliar a aplicação a obter os dados, elimina-se a necessidade de configurar tarefas complexas de extração de dados.

3.2 Arquitetura de Software

Para a arquitetura de desenvolvimento foi escolhido separação da aplicação em dois projetos, sendo eles *backend* e *frontend*.

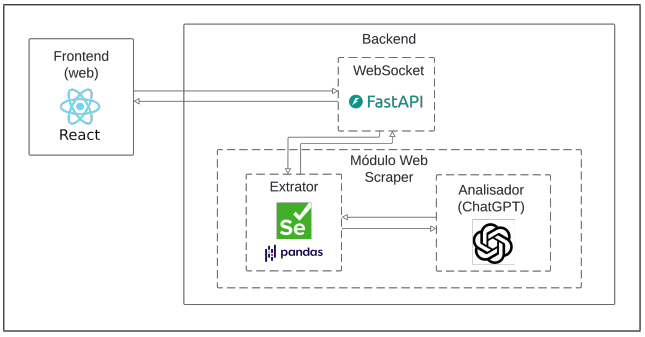

Figura 5. Diagrama de arquitetura

No *backend* ficará concentrado, toda a parte de comunicação com o usuário, análise de conteúdo da página *web* solicitada e extração dos dados. Além disso, o *backend* será desenvolvido utilizando a linguagem de programação Python.

Ademais, com o intuito de organizar e definir melhor o escopo das funcionalidades no *backend*, fora pensado na criação dos módulos API e *web scraper*.

O primeiro, é uma implementação de *web sockets*, utilizando o *framework* FastAPI[14], e é responsável por realizar a comunicação entre o *frontend* e o módulo *web scraper*.Enquanto que o segundo, será responsável pelas etapas extração e análise. Na extração dos dados, será utilizado o *framework* Selenium e também a biblioteca Pandas[15] utilizada para auxiliar na manipulação e exportação dos dados. Em relação a etapa de análise, será utilizada a API do ChatGPT[16] para a identificação de formulários de filtro e ações extras.

A utilização de uma inteligência artificial generativa, como o ChatGPT, justifica-se pela sua flexibilidade na interpretação e análise de conteúdo textual [Hassani and Silva, 2023]. Essa abordagem oferece uma alternativa à criação manual de heurísticas, que demanda o desenvolvimento de um conjunto específico de regras para a identificação de elementos de filtros e navegação. Um dos principais benefícios do uso da IA (Inteligência Artificial) é sua capacidade de aprender e adaptar-se a novas estruturas e padrões de forma autônoma, sem a necessidade de reescrever regras codificadas, proporcionando uma solução mais robusta e resiliente às frequentes mudanças no layout das páginas.

Em relação ao *frontend*, terá como responsabilidade todas as interações com o usuário, para isso, o *frontend* será uma aplicação *web* desenvolvida em Javascript utilizando como principal ferramenta a biblioteca React[17].

3.3 ChatGPT

Conforme discutido na Seção 3.2, o uso de inteligência artificial desempenha um papel fundamental no desenvolvimento deste trabalho. Neste estudo, será utilizada a IA ChatGPT, cuja aplicação será detalhada nesta seção.

Inicialmente, é importante destacar que algumas configurações são necessárias, como a criação de chaves para o uso da API e a inserção de créditos na plataforma da OpenAI[18].

A API da OpenAI oferece diversos tipos de processamento, como geração de texto, conversão de texto para voz, conversão de voz para texto e geração de imagens. No presente trabalho, foi utilizada a funcionalidade de geração de texto.

A classe *OpenAIClient*, mostrada na Figura 6, gerencia a comunicação com a API da OpenAI, permitindo a geração de texto com base nas instruções fornecidas. O método principal, *send_to_openai*, envia a instrução junto ao conteúdo a ser analisado e retorna uma resposta processada pela IA.

É importante ressaltar que o formato da resposta pode ser configurado para retornar em JSON, como ilustrado na Figura 6, por meio do parâmetro response_format. Para isso,

[14] https://fastapi.tiangolo.com/
[15] https://pandas.pydata.org/
[16] https://chatgpt.com/
[17] https://react.dev
[18] https://platform.openai.com/

a instrução fornecida (Figura 7) deve especificar o formato desejado; caso contrário, a API pode retornar um erro.

```python
from typing import Union

from openai import OpenAI
from openai.types import ChatModel


class OpenAIClient:
    def __init__(self):
        self.openAI = OpenAI()

    def send_to_openai(self, text: str, instruction: str, model: Union[str, ChatModel]) -> str:
        try:
            task = self.openAI.chat.completions.create(
                model=model model,
                response_format={"type": "json_object"},
                messages=[
                    {"role": "system", "content": instruction},
                    {"role": "user", "content": text},
                ]
            )

            return task.choices[0].message.content
        except Exception as e:
            return "Error communicating with OpenAI: {}".format(str(e))

```

Figura 6. Código para interação com a API

```
HTML_FORM_FIELD_ANALYSIS = """
Analyze the provided HTML form elements and return their attributes, options, and the field type in a consistent JSON format.

### Structure to follow:

- For each form element, return:
  - `field_type`: A string that indicates whether the element is a "select", "multiselect", or a specific `input` type (e.g., "text", "email", "checkbox").
  - `attributes`: A dictionary containing all attributes of the element (e.g., id, class, name, type, placeholder, etc.).
  - `options`: A list containing all available options for dropdowns or multiselects (with label, value, disabled status, and CSS class).

### Guidelines:

1. Always return `field_type`, `attributes`, and `options` keys, even if one of them is empty.
2. Ensure `field_type` clearly specifies the type of field:
   - `"select"` for a regular `<select>` element.
   - `"multiselect"` for a `<select>` element with the `multiple` attribute.
   - `"input"` followed by the value of the `type` attribute for `<input>` elements (e.g., `"input-text"`, `"input-email"`, `"input-checkbox"`).
3. Ensure `attributes` contains all relevant HTML attributes, like:
   - id
   - class
   - name
   - type
   - placeholder
   - maxlength
   - min, max, step
   - aria-label
4. Ensure `options` includes:
   - label: The visible text for the option.
   - value: The value associated with the option, if available.
   - disabled: Whether the option is disabled.
   - class: The CSS class of the option, if available.
5. Return an empty list `[]` for `options` if no options are present, and an empty dictionary `{}` for `attributes` if no attributes are found.

### Example (select element with options):

```json
{
  "field_type": "select",
  "attributes": {
    "id": "exampleID",
    "class": "exampleClass",
    "name": "dropdown"
  },
  "options": [
    {"label": "Option 1", "value": "1", "disabled": false, "class": "optionClass"},
    {"label": "Option 2", "value": "2", "disabled": true, "class": "optionClass"}
  ]
}
```

"""
```

Figura 7. Exemplo de instrução enviado à API

O tamanho do conteúdo e da instrução enviados influencia diretamente o resultado retornado pela API, pois há um limite de *tokens* que o modelo pode processar. Por exemplo, o modelo *gpt-4o-mini* suporta até duzentos mil *tokens* por requisição[19]. Quando esse limite é excedido, a resposta pode ser truncada. Para contornar essa limitação, o conteúdo pode ser segmentado em partes menores, respeitando o número máximo de *tokens* suportados pelo modelo.

4 Implementação

A implementação da aplicação tem como elemento central o uso de *WebSocket*, que facilita a comunicação em tempo real entre o usuário e o extrator de dados. Na Figura 8, é ilustrada a criação de um *endpoint* utilizando FastAPI. Esse *endpoint* permite uma comunicação contínua entre o usuário e o servidor, adaptando-se aos diferentes estágios do processo de extração de dados.

Esse *endpoint* é responsável por gerenciar o fluxo de extração de dados, disparando ações conforme o recebimento de mensagens. Cada nova conexão cria uma instância única da

[19] `https://platform.openai.com/docs/guides/rate-limits?context=tier-one#usage-tiers`

```python
@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    state_manager = await manager.connect(websocket)

    try:
        while True:
            data = json.loads(await websocket.receive_text())
            state_manager.data = data["data"]

            if data["stage"]:
                response = state_manager.handle_stage(ApplicationStage.WAITING.value)
                await websocket.send_text(f"{response}")

                response = state_manager.handle_stage(data["stage"])
                state_manager.previous_data = data["data"]
                print("Ending of stage")
                await websocket.send_text(f"{response}")

    except WebSocketDisconnect:
        manager.disconnect(websocket)

```

Figura 8. Endpoint WebSocket em uma aplicação FastAPI

classe *WebSocketStageManager*, que coordena o progresso da extração. Os fluxos principais são ativados ao receber uma mensagem do usuário, sendo eles:

- *SEND_INITIAL_URL*: Transmite a URL inicial para iniciar o processo de extração de dados.
- *SEND_ADDITIONAL_INFO*: Fornece os campos de filtro preenchidos ou especifica a ação que deve ser executada pelo extrator.
- *SEND_DATA_DETAILS*: Define quais informações da base de dados devem ser exportadas e o formato de exportação desejado.

À medida que esses fluxos são ativados, eles podem retornar novas instruções ao cliente, utilizando a função *send_text*, caso seja necessário para o próximo passo do processo. As respostas enviadas ao cliente incluem:

- *REQUEST_ADDITIONAL_INFO*: Solicita que o usuário preencha os campos de filtro ou selecione uma das ações identificadas pelo extrator.
- *REQUEST_DATA_DETAILS*: Requisita detalhes específicos dos dados que devem ser extraídos para finalizar o processo.

Conforme ilustrado na Figura 3, o processo de análise do conteúdo *web* começa no estágio *SEND_INITIAL_URL*, quando o usuário envia uma URL. Nesse momento, a página *web* é baixada e passa por uma etapa inicial de limpeza, na qual elementos irrelevantes para a análise são removidos. Tags HTML como *<script>*, *<svg>*, *<meta>* e *<style>*, bem como atributos que não contribuem para o conteúdo analítico como *style* e *tabindex*, são eliminados.

A classe auxiliar *BeautifulSoupManager* (Figura 9) facilita a manipulação de HTML ao fornecer métodos específicos para a remoção de tags e atributos utilizando a biblioteca *BeautifulSoup*. O método *clean_soup* realiza a remoção dinâmica de *tags* e atributos HTML, o que contribui para a diminuição do conteúdo enviado à IA. Além desses métodos, foi implementado um recurso para minificar o conteúdo HTML, eliminando espaços, tabulações e comentários que possam estar presentes na página HTML, reduzindo ainda mais o volume de dados a serem processados.

Após a limpeza inicial, a página *web* é enviada à IA para análise, com o objetivo de identificar possíveis identificadores ou classes CSS associados a campos de filtro. Caso a IA localize esses identificadores, o conteúdo HTML dos campos de filtro é extraído e reenviado à IA para que seja gerado

```python
class BeautifulSoupManager:
    def __init__(self, html: str, features: str = "lxml"):
        self.html = html
        self.soup = BeautifulSoup(html, features=features)

    def remove_tag(self, tags: str | list):
        if isinstance(tags, str):
            tags = [tags]

        for tag in self.soup(tags):
            tag.decompose()

    def remove_attribute(self, attributes: str | list, remove_empty_attributes: bool = False):
        if isinstance(attributes, str):
            attributes = [attributes]

        for tag in self.soup.find_all(True):
            for attr in attributes:
                if tag.has_attr(attr):
                    del tag[attr]

                if remove_empty_attributes:
                    tag.attrs = {attr: value for attr, value in tag.attrs.items() if value}

    def clean_soup(self, tags_to_remove=None,attributes_to_remove=None
    ):

        if tags_to_remove is None:
            tags_to_remove = ["script", "style", "head", "meta", "noscript", "footer", "header", "svg", "iframe", "p",
                              "i"]
        if tags_to_remove is None:
            tags_to_remove = ["script", "style", "head", "meta", "noscript", "footer", "header", "svg", "iframe", "p",
                              "i"]
        if len(tags_to_remove): self.remove_tag(tags_to_remove)
        if len(attributes_to_remove): self.remove_attribute(attributes_to_remove)

```

Figura 9. Classe para manipulação de HTML utilizando BeautifulSoup

um JSON com as características de cada campo, conforme ilustrado na Figura 4.

A identificação dos campos de filtro é realizada em duas etapas devido às limitações dos modelos de IA disponíveis, uma vez que muitos deles não suportam um grande número de *tokens* por requisição. Dependendo do tamanho da página, extrair diretamente os elementos de filtro conforme a estrutura descrita na Figura 4 pode ser inviável. O número de *tokens*(incluindo a página HTML, a instrução e a resposta) frequentemente ultrapassa o limite permitido pelo modelo, o que pode causar erros ou, no pior dos casos, resultar em um JSON incompleto.

No estágio *SEND_ADDITIONAL_INFO*, espera-se que o usuário selecione um filtro, caso algum precise ser aplicado. A interação com o Selenium ocorre nesse ponto, e, inicialmente, é necessário identificar o tipo de campo, pois diferentes abordagens são aplicadas para interagir com campos de texto e de seleção. A Figura 10 ilustra a identificação e interação com esses campos. Para que o Selenium localize o elemento, utiliza-se uma expressão XPath que busca identificadores ou classes CSS. Após localizar o campo, identifica-se seu tipo e, com base nisso, aplicam-se as interações específicas. Por exemplo, para campos de seleção, é necessário primeiro clicar no elemento para expandir a lista de opções e, em seguida, selecionar a opção desejada usando uma expressão XPath.

```python
def handle_field_interaction(self, field):
    attributes = field["attributes"]

    if attributes["id"]:
        element = self.web_interaction_helper.wait_for_element_presence_by_id(attributes["id"])
    else:
        xpath = f"//*[contains(@class, '{attributes["class"]}')]"
        element = self.web_interaction_helper.wait_for_element_presence_by_xpath(xpath)

    if field["field_type"] in ["select", "multiselect"] or attributes["type"] in ["checkbox", "radio"]:
        element.click()

        if "selected_options" in field:
            for option in field["selected_options"]:
                value = option["value"] if "value" in option and option["value"] else option["label"]

                if "class" in option and option["class"]:
                    xpath = f".//*[contains(@class, '{option['class']}') and contains(., '{value}')]"
                else:
                    xpath = f".//*[contains(text(), '{value}')]"

                option_element = self.web_interaction_helper.wait_for_element_presence_by_xpath(xpath)
                self.web_interaction_helper.click(option_element)

        return
    else:
        self.web_interaction_helper.fill_input_field(element, attributes["value"])
        return

```

Figura 10. Código para interação com campos de filtro

A classe auxiliar *SeleniumInteractionHelper* (Figura 11) foi criada para encapsular operações de interação com elementos em páginas *web* usando Selenium. Nessa classe, foram implementados métodos que permitem simular cliques, preencher campos de texto e localizar elementos tanto por identificadores quanto por seletores XPath.

```python
class WebInteractionHelper(SeleniumHelper):

    def __init__(self, state_manager_id: UUID):
        super().__init__(state_manager_id)

    @staticmethod
    def click(element: WebElement) -> bool:
        try:
            element.click()
            return True
        except ElementNotInteractableException:
            return False

    @staticmethod
    def fill_input_field(element: WebElement, value):
        element.clear()
        element.send_keys(value)

    def wait_for_element_presence_by_id(self, element_id: str, timeout: float = 10):
        return WebDriverWait(self.driver, timeout).until(
            expected_conditions.presence_of_element_located((By.ID, element_id))
        )

    def wait_for_element_presence_by_xpath(self, xpath: str, timeout: float = 10):
        return WebDriverWait(self.driver, timeout).until(
            expected_conditions.presence_of_element_located((By.XPATH, xpath))
        )

    def click_using_javascript(self, element):
        self.driver.execute_script("arguments[0].click();", element)

```

Figura 11. Classe para interação com elementos HTML utilizando Selenium

Com base nos filtros preenchidos, o conteúdo atualizado da página é baixado e submetido ao mesmo processo de limpeza anterior. Em seguida, o conteúdo da página *web* é reenviado à IA, instruída a identificar elementos clicáveis relacionados a *links*, ações ou opções de *download* relevantes, descartando itens de menus ou ações irrelevantes. No entanto, esse processo de detecção nem sempre é preciso e, em algumas situações, a IA pode não retornar exatamente o que foi solicitado. Nesses casos, o processo precisa ser repetido.

Quando são detectadas ações de clique ou redirecionamento de página, o estágio da extração é ajustado para *REQUEST_ADDITIONAL_INFO*, e uma lista de ações disponíveis é retornada ao usuário para que ele possa decidir os próximos passos.

Após a escolha de uma ação pelo usuário, a ação de clique é simulada pelo Selenium, em seguida, o conteúdo da página é analisado novamente, repetindo o processo de extração e identificação de novos elementos clicáveis ou de *download*, conforme necessário, para garantir a continuidade da extração dos dados. Os estágios *REQUEST_ADDITIONAL_INFO* e *SEND_ADDITIONAL_INFO* podem ocorrer várias vezes ao longo da extração de uma fonte de dados. Em diversos momentos, uma ação pode abrir uma nova aba, reiniciando o processo desde a obtenção dos filtros até a busca por elementos de ação ou de *download*

Se forem encontradas ações de *download*, a função *handle_download_element* é chamada para cada elemento identificado (Figura 12). Nessa função, similar à interação com campos de filtro, realizam-se buscas específicas usando XPath.

Vale destacar que a função *handle_download_element* incluí adaptações específicas para o Portal Brasileiro de Dados Abertos, onde múltiplos botões de *download* compartilham o mesmo identificador, dificultando a seleção direta dos elementos de *download*. Para superar essa limitação, desenvolveu-se um XPath personalizado que é a primeira tentativa de busca, possibilitando a identificação precisa dos

```python
def handle_download_element(self, download_action, download_dir: str):
    element_xpath = self.web_interaction_helper.create_default_xpath_sentence(download_action)
    xpath_tag = f"{download_action['tag'] if 'tag' in download_action else '*'}"

    if "download_format" in download_action and download_action["download_format"]:
        xpath = (f"//*[normalize-space(text()) = '{download_action['download_format']}']"
                 f"/following::{xpath_tag}[{element_xpath}][1]")
    else:
        xpath = f"//{xpath_tag}[{element_xpath}]"

    try:
        element = self.web_interaction_helper.wait_for_element_presence_by_xpath(xpath)
    except TimeoutException:
        xpath = f"//{xpath_tag}[{element_xpath}]"
        element = self.web_interaction_helper.wait_for_element_presence_by_xpath(xpath)

    if not self.web_interaction_helper.click(element):
        self.web_interaction_helper.click_using_javascript(element)

    self.web_interaction_helper.wait(1)
    self.web_interaction_helper.wait_for_download_completion(download_dir)

```

Figura 12. Código para interação com elementos para baixar dados

botões de *download* e garantindo o acesso automatizado ao conteúdo. Caso essa busca inicial não seja bem-sucedida, o sistema recorre a um XPath padrão, que utiliza o identificar ou classes CSS para localizar os elementos necessários.

Outro aspecto importante é que, para a interação via Selenium, o elemento precisa estar visível na tela. Em alguns casos, isso exige a remoção de estilos e classes CSS ou a garantia de que o elemento esteja na área visível da página *web viewport*[20]. Para garantir a execução do clique, tenta-se primeiramente interagir por meio do Selenium, e, caso essa tentativa falhe, a interação é realizada via JavaScript.

Após a identificação de que a fonte de dados foi baixada, os arquivos são submetidos a uma analise, conforme a Figura 13 com o propósito de identificar o tipo, essa verificação é feita pelo MIME (*Multipurpose Internet Mail Extensions*), com base no MIME pode-se manipular corretamente os arquivos baixados. Além disso, caso algum arquivo esteja compactado, o processo de descompactação é realizado.

```python
def handle_downloaded_data(download_dir):
    file_data_manager = FileDataManager()
    samples = []
    zip_files = [
        os.path.join(download_dir, filename)
        for filename in os.listdir(download_dir)
        if os.path.isfile(os.path.join(download_dir, filename)) and magic.from_file(
            os.path.join(download_dir, filename), mime=True) == "application/zip"
    ]

    for zip_file in zip_files:
        file_data_manager.unpack_file(zip_file, download_dir)
        file_data_manager.delete_file(zip_file)

    for file_name in os.listdir(download_dir):
        file_path = f"{download_dir}/{file_name}"
        mime = magic.from_file(file_path, mime=True)

        if mime in SUPPORTED_MIME_TYPE_FILES:
            df = file_data_manager.open_file(file_path, mime)

            if isinstance(df, dict):
                continue

            df = file_data_manager.create_sample(df)
            samples.append(file_data_manager.convert_dataframe_to_json(df))

    return samples

```

Figura 13. Código para interação com elementos para baixar dados

A classe *FileDataManager* foi desenvolvida para lidar com a leitura de arquivos em múltiplos formatos, incluindo CSV, JSON, XML e XLSX. Ela utiliza métodos específicos para cada tipo de arquivo com base no *mime_type* detectado, retornando sempre um *DataFrame*, estrutura de dados utilizada pela biblioteca Pandas, no final. Isso facilita a integração com o *frontend*, uma vez que *DataFrames* podem ser facilmente convertidos em formatos amigáveis para visualização e manipulação pelo usuário."

[20] Área visível da página *web* exibida na tela do dispositivo, representando a parte do conteúdo que o usuário pode ver sem rolar a página.

Além disso, A função *get_encoding* automatiza a detecção da codificação do conteúdo do arquivo, garantindo que dados em diferentes formatos sejam lidos corretamente, evitando erros de leitura e facilitando o processamento dos arquivos. O método de conversão de CSV possui uma funcionalidade adicional para detectar automaticamente o delimitador dos valores, pois, em alguns casos, o processo de converter o CSV em um *DataFrame* não identifica o delimitador corretamente, o que pode causar inconsistências nos dados.

```python
class FileDataManager:

    def open_file(self, path: str, mime_type: str) -> DataFrame:
        actions = {
            "text/csv": self._read_csv_file,
            "text/plain": self._read_csv_file,
            "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet": self._read_excel_file,
            "application/vnd.ms-excel": self._read_excel_file,
            "application/json": self._read_json_file,
            "application/xml": self._read_xml_file
        }

        return actions.get(mime_type, self._unsupported_file)(path)

    @staticmethod
    def get_encoding(path: str) -> str:
        with open(path, "rb") as file:
            encoding = chardet.detect(file.read())["encoding"]
        return encoding

    def _read_csv_file(self, path: str) -> DataFrame:
        encoding = self.get_encoding(path)
        with open(path, 'r', encoding=encoding) as f:
            amostra = f.readline()  # Lê uma amostra para detecção
            sniffer = csv.Sniffer()
            delimiter = sniffer.sniff(amostra).delimiter

        return pd.read_csv(path, delimiter=delimiter, encoding=encoding, dtype=str)

    @staticmethod
    def _read_excel_file(path: str) -> DataFrame:
        return pd.read_excel(path)

```

Figura 14. Classe para manipulação de arquivos

A partir desse ponto, os arquivos são lidos e transformados em um *DataFrame*. Em seguida, é gerada uma lista contendo amostras dos diversos arquivos baixados, e o estágio da extração é alterado para *REQUEST_DATA_DETAILS*, enviando essas amostras ao usuário.

É importante destacar que, durante o desenvolvimento, várias fontes extraídas não estavam em formatos adequados, especialmente aquelas disponibilizadas no formato XLSX. Em alguns casos, a manipulação do arquivo não apresentou valores válidos, pois a análise do conteúdo revelou que os dados eram representações gráficas, inviabilizando a manipulação direta. Em outra fonte, também em formato XLSX, os nomes das colunas estavam incorretos, o que levou a maioria das colunas a serem renomeadas para *"Unnamed"* durante a exportação, resultando na perda dos nomes originais das colunas.

Por fim, após o retorno do usuário, especificando quais dados são relevantes e como devem ser exportados, o estágio da extração é atualizado para *SEND_DATA_DETAILS*. Nesse momento, com o auxílio da classe *FileDataManager*, os dados são exportados de acordo com os critérios definidos pelo usuário.

5 Estudo de caso

Nesta seção, são apresentados os casos de teste para a extração de dados. Cada estudo de caso descreve o fluxo necessário para acessar a fonte de dados e o resultado esperado da aplicação desenvolvida neste trabalho.

5.1 IBGE - Estimativas da população

A fonte de dados está disponível na página do IBGE (Instituto Brasileiro de Geografia e Estatística)[21]. Ao fornecer a URL para a aplicação, a busca por elementos de filtro é iniciada. Conforme demonstrado pela Figura 15, na página não há possíveis elementos de filtro, nesse momento o extrator começa a realizar buscas de possíveis elementos de ação ou de *download*.

Figura 15. Página de acesso as estimativas da população

A partir da busca, são identificados elementos de ação, sendo assim, o extrator retorna ao usuário que ele deve selecionar uma das ações (Figura 16). Assim que o usuário faz a seleção da ação, o extrator realiza a interação com o elemento escolhido pelo usuário (Figura 17), Em seguida, a página é analisada novamente para verificar a presença de novos elementos de ação ou de *download*.

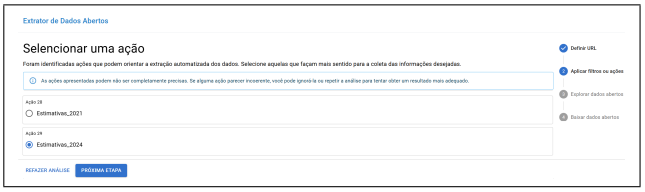

Figura 16. Aplicação desenvolvida exibindo as ações ao usuário

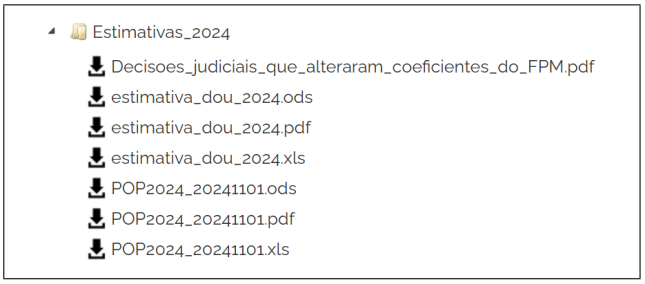

Figura 17. Fragmento da página com o grupo das estimativas de 2024 expandido

Nesse momento, os *links* para *download* estão renderizados, e a análise confirma a presença de possíveis elementos de *download*. O extrator, então, interage com esses elementos identificados pela IA e verifica se os arquivos baixados estão compactados. Caso estejam, é realizada a descompactação, caso contrário, os arquivos são lidos diretamente. Em ambos os casos, os dados são enviados ao usuário, que poderá escolher quais informações são relevantes e definir o formato do arquivo final (Figura 18).

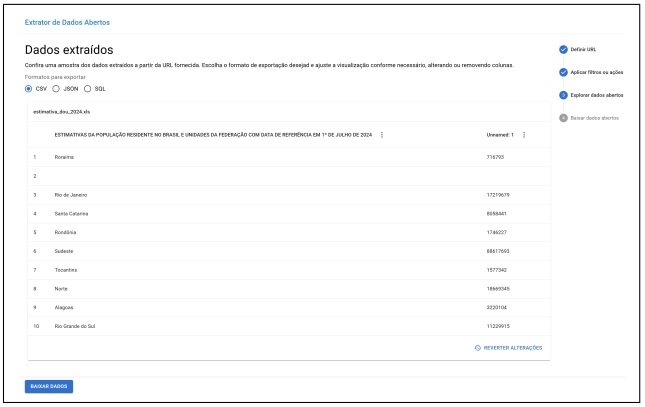

Figura 18. Aplicação desenvolvida exibindo as amostras de dados obtidas

De acordo com a escolha do usuário sobre o formato de exportação e as colunas e nomes relevantes do conjunto de dados, o extrator aplica essas configurações e disponibiliza os links para download dos arquivos processados (Figura 19).

Figura 19. Aplicação desenvolvida exibindo os dados para serem baixados

5.2 ANATEL - Densidade telefônica

A fonte de dados sobre densidade telefônica está disponível no Portal Brasileiro de Dados Abertos[22] (Figura 20). Ao receber a URL fornecida pelo usuário, a aplicação inicia a busca por elementos de filtro. Nesta página específica, existem vários campos de formulário, mas eles estão relacionados a avaliação e comentários sobre a base de dados, não a filtros. A instrução especifica que campos não relacionados a filtros devem ser desconsiderados. Portanto, a IA deve ignorar esses campos e, caso estejam incluídos, o usuário terá a opção de pular essa etapa.

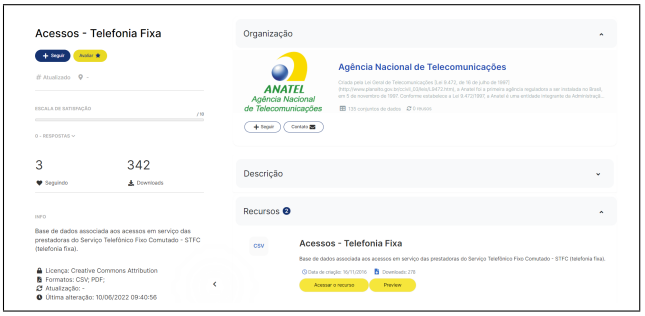

Figura 20. Página de acesso a base de dados de densidade telefônica

[21] https://www.ibge.gov.br/estatisticas/sociais/populacao/9103-estimativas-de-populacao.html?=&t=downloads

[22] https://dados.gov.br/dados/conjuntos-dados/acesso-autorizadas-stfc

Como não foram encontrados campos de filtros, o extrator parte para a próxima etapa, onde ele deve buscar elementos de ação ou elementos de *download*. Conforme na Figura 21, serão retornados dois elementos, a partir desse momento o extrator realizará as ações para tentar extrair os dados.

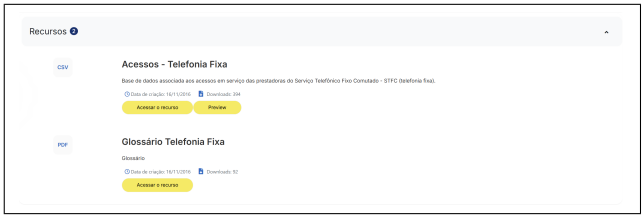

Figura 21. Fragmento da página com elementos para *download*

O resultado da extração é um arquivo compactado que contém diversos arquivos CSV. Dessa forma, o arquivo será descompactado e uma amostra dos dados será enviada ao usuário (Figura 23), para que o mesmo possa selecionar apenas as informações de interesse.

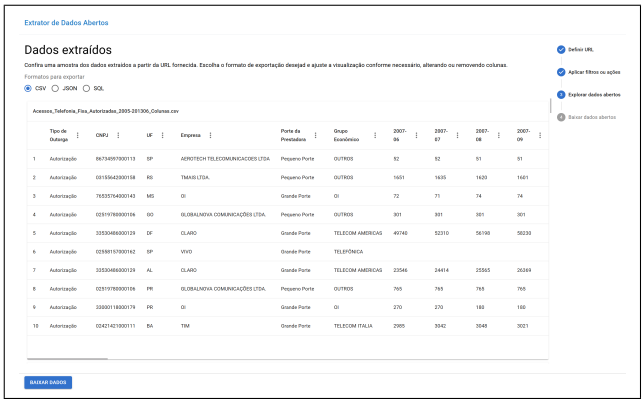

Figura 22. Aplicação desenvolvida exibindo as amostras de dados obtidas

Após a modificação das amostras conforme as preferências do usuário e a definição do formato de exportação, essas configurações são enviadas ao extrator. O extrator, então, ajusta os dados extraídos de acordo com as instruções recebidas e fornece ao usuário os links para download dos dados exportados.

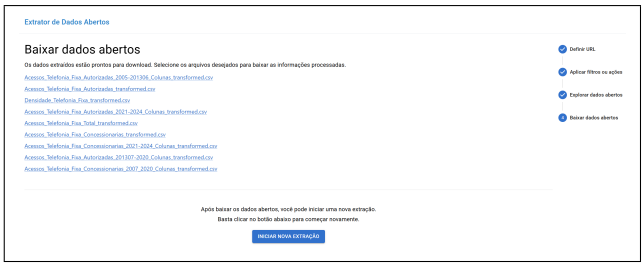

Figura 23. Aplicação desenvolvida exibindo os dados para serem baixados

5.3 SICONFI - Despesas municipais com cultura

A fonte de dados sobre despesas municipais com cultura é disponibilizada pelo sistema SICONFI[23]. Conforme ilustrado na Figura 24, o sistema apresenta diversos campos de filtro que permitem personalizar a consulta.

No entanto, alguns campos, como ”Municípios do Estado” e ”Tabela” (Figura 24), possuem opções dinâmicas, sendo atualizadas automaticamente conforme os valores de outros campos já preenchidos. Esse comportamento torna a identificação dos valores de seleção mais complexa, pois exige fluxos específicos e múltiplas iterações para capturar as alterações em tempo real. Além disso, campos renderizados dinamicamente, como ”Municípios do Estado”, necessitam uma análise contínua a cada interação, o que dificulta a automação desse processo.

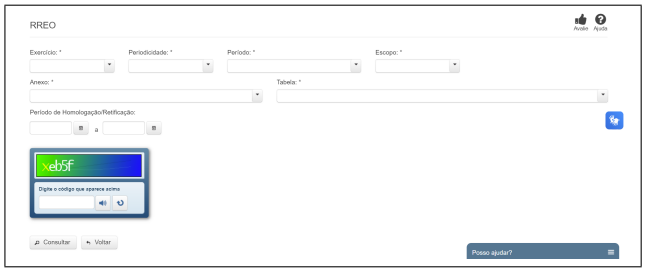

Figura 24. Página de acesso a base de dados de despesas municipais com cultura

Dada a complexidade, optou-se, inicialmente, por não incluir suporte a esses comportamentos no extrator. Assim, sua funcionalidade se limita à interação com campos estáticos, cujo conteúdo não depende de interações prévias, impossibilitando, portanto, a extração de dados provenientes dessa fonte.

6 Conclusão

No presente trabalho, foi proposta a criação de uma aplicação para automatizar parte da extração de dados abertos a partir de múltiplas fontes, com o objetivo de auxiliar o City Living Lab e reduzir a necessidade de intervenção manual no processo de coleta de dados. A aplicação desenvolvida mostrou-se capaz de extrair dados de diferentes fontes de forma autônoma, eliminando a necessidade de o usuário configurar manualmente cada etapa do processo de extração.

Um dos principais fatores para essa flexibilidade foi a utilização de inteligência artificial, especificamente o ChatGPT, que permitiu à aplicação automatizar a identificação de elementos pertinentes a extração dos dados abertos. Isso trouxe benefícios diretos ao processo de extração, uma vez que o usuário deixou de precisar configurar manualmente as expressões XPath, assumindo apenas o papel de orientar o extrator quando necessário.

Durante o desenvolvimento, surgiram desafios relacionados tanto ao uso da IA quanto às características das fontes de dados. No caso do ChatGPT, foi necessário formular instruções para alcançar a melhor resposta possível, pois instruções com muitas regras ou tarefas frequentemente resultavam em respostas menos precisas. Além disso, controlar o tamanho do conteúdo enviado e selecionar um modelo adequado mostraram-se essenciais, já que conteúdos extensos não apenas geravam respostas incompletas, mas também aumentavam os custos.

[23] `https://siconfi.tesouro.gov.br/siconfi/pages/public/consulta_finbra/finbra_list.jsf`

Para melhorar a qualidade das respostas da IA, algumas etapas, como a identificação dos filtros, foram separadas em processos distintos. Com essa abordagem, a IA pôde responder de forma mais precisa a instruções mais diretas e específicas, o que contribuiu para a obtenção de resultados mais alinhados com as necessidades da aplicação.

No entanto, apesar de todo esse empenho em refinar o fluxo de extração e as instruções, a IA ainda apresentou limitações, e em diversos momentos os resultados retornados foram incorretos. Isso evidenciou que, embora o uso da IA traga flexibilidade ao processo, ainda há melhorias há serem feitas no processo.

Em relação às fontes de dados, o trabalho encontrou dificuldades devido ao uso de HTML não semântico em muitas páginas, dificultando a identificação dos elementos. Além disso, algumas bases de dados apresentavam formatos inadequados, como gráficos em vez de dados estruturados, ou rótulos de colunas genéricos, o que comprometeu a qualidade e a integridade dos dados extraídos.

Por fim, embora a Lei de Acesso à Informação e o portal único representem grandes avanços na transparência pública, ainda existem lacunas importantes em relação à formatação e acessibilidade dos dados. Para que a informação pública seja verdadeiramente acessível e reutilizável, é essencial que os dados sejam disponibilizados de forma estruturada e padronizada, permitindo seu uso direto por ferramentas de análise e sistemas automatizados. Essa mudança facilitaria iniciativas que dependem desses dados, já que, conforme discutido anteriormente, algumas fontes ainda disponibilizam conteúdos em formato de gráficos, o que impossibilita certas análises e limita o potencial de uso das informações.

Em trabalhos futuros, seria interessante considerar estratégias para aprimorar a precisão dos resultados obtidos pela IA, incluindo a utilização de modelos mais especializados ou treinados especificamente para a interpretação de HTML. Outra possibilidade seria implementar um sistema de aprendizado contínuo, permitindo que a aplicação aprenda com as correções realizadas pelos usuários. Além disso, seria relevante realizar testes para avaliar o tempo de extração em diferentes cenários, identificando gargalos e otimizando o desempenho. Esses aprimoramentos não apenas aumentariam a precisão ao longo do tempo, mas também reduziriam ainda mais a necessidade de intervenção manual, tornando o processo de extração de dados progressivamente mais autônomo.

Referências

Assis, W. V. d. and Gomide, J. V. B. (2021). Web scraping em dados públicos: método para extração de dados dos gastos públicos dos vereadores da câmara municipal de belo horizonte. *Informação e Informação*.

Bandeira, J. M., Alcantara, W. L., Barbosa Sobrinho, A., Ávila, T. J. T., Bittencourt, I. I., and Isotani, S. (2015). Dados abertos conectados. *III Simpósio Brasileiro de Tecnologia da Informação*.

Battistelo, R. (2018). Aplicação web para indicadores de cidades do conhecimento.

Bregalda, V. (2021). Coleta dos indicadores da plataforma de cidades do conhecimento.

Bricongne, J.-C., Meunier, B., and Pouget, S. (2023). Web-scraping housing prices in real-time: The covid-19 crisis in the uk. *Journal of Housing Economics*.

Dantas, M. C. M. (2021). Monitora santa cruz: uma ferramenta web para auxiliar o cidadão no monitoramento dos gastos da prefeitura de santa cruz do capibaribe.

Hamoud, A., Hashim, A. S., and Awadh, W. A. (2018). Clinical data warehouse: a review. *Iraqi Journal for Computers and Informatics*, 44(2).

Hassani, H. and Silva, E. S. (2023). The role of chatgpt in data science: How ai-assisted conversational interfaces are revolutionizing the field. *Big Data and Cognitive Computing*, 7(2). DOI: 10.3390/bdcc7020062.

Knowledge, O. (2024). The open definition.

MDN (2024a). Atributos - html.

MDN (2024b). Xpath.

Neves, O. M. d. C. (2013). Evolução das políticas de governo aberto no brasil.

Notari, D., Battistelo, R., Molin, L., de Avila e Silva, S., and Fachinelli, A. (2019). AplicaÇÃo web para indicadores de cidades do conhecimento. *Revista Brasileira de Gestão e Inovação*, 7:95–118. DOI: 10.18226/23190639.v7n2.05.

Rodrigues, H. (2023). Virtualização de ferramentas de carga de dados da plataforma cidades do conhecimento.

Silva, A. d. A. P., Monteiro, D. A. A., and Reis, A. d. O. (2018). Qualidade da informação dos dados governamentais abertos: Análise do portal de dados abertos brasileiro. *Revista Gestão em Análise*.

Silva, A. G. d. (2022). Desenvolvimento da automatização da coleta de dados na plataforma de cidades do conhecimento.

Trujillo, J. and Luján-Mora, S. (2003). A uml based approach for modeling etl processes in data warehouses. In *International Conference on Conceptual Modeling*, pages 307–320. Springer.

Zhang, Q. and Richards, G. C. (2023). Lessons from web scraping coroners' prevention of future deaths reports. *Medico-Legal Journal*.

Zhao, B. (2017). *Web Scraping*, pages 1–3. DOI: 10.1007/978-3-319-32001-$4_483 - 1$.