UNIVERSIDADE FEDERAL FLUMINENSE

NOME(S) DO(S) AUTOR(ES)

* ESTA PÁGINA É Capa Dura a ser impressa na gráfica.
* **Com hifenização**.

(remova as caixas amarelas)

TÍTULO DO TRABALHO

Niterói

ANO

NOME(S) DO(S) AUTOR(ES)

TÍTULO DO TRABALHO

Trabalho de Conclusão de Curso submetido ao Curso de Tecnologia em Sistemas de Computação da Universidade Federal Fluminense como requisito parcial para obtenção do título de Tecnólogo em Sistemas de Computação.

Orientador(a):

Nome

NITERÓI

ANO

Folha reservada para a ficha catalográfica

NOME(S) DO(S) AUTOR(ES)

TÍTULO DO TRABALHO

SUBTÍTULO

Trabalho de Conclusão de Curso submetido ao Curso de Tecnologia em Sistemas de Computação da Universidade Federal Fluminense como requisito parcial para obtenção do título de Tecnólogo em Sistemas de Computação.

Niterói, \_\_\_ de \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ de ANO.

Banca Examinadora (provisório):

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. ou Profa. <NOME>, <Título>. – Orientador

<Sigla da Universidade> - <Nome da Universidade>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. ou Profa. <NOME>, <Título>. – Avaliador

<Sigla da Universidade> - <Nome da Universidade>

A dedicatória é opcional, no qual o autor presta homenagem ou dedica seu trabalho a alguém.

Dedico este trabalho a(o) minha(meu) esposa(o) e aos meus estimados filhos.

AGRADECIMENTOS

(EXEMPLO) A Deus, que sempre iluminou a minha caminhada.

A meu Orientador Fulano de Tal pelo estímulo e atenção que me concedeu durante o curso.

Aos Colegas de curso pelo incentivo e troca de experiências.

A todos os meus familiares e amigos pelo apoio e colaboração.

Epígrafe é a inscrição colocada no início de um trabalho, de um capítulo ou partes principais. É opcional para a monografia.

Delete as caixas amarelas

“A Escola é uma arena onde grupos sociais lutam por legitimidade e poder”.

Dinair Leal da Hora

RESUMO

O resumo deve apresentar os pontos relevantes de um texto, no mesmo momento ele terá que dar uma visão rápida e clara dos conteúdos, metodologias e das conclusões do trabalho. As frases têm que ter uma sequência lógica e objetiva. A fonte do estilo é Arial e tamanho 12.

Palavras-chaves: palavra1, palavra2 e palavra3.

*O resumo e abstract* devem ser digitados sem parágrafo inicial e o espaçamento entre linhas é simples. Lembre-se que já existem estilos que formatam o texto adequadamente. Remova as caixas amarelas antes de imprimir.

ABSTRACT (opcional)

The summary must present the excellent points of a text, at the same moment it will have that to give a fast and clear vision of the contents, methodologies and of the conclusions of the work. The phrases have that to have a logical and objective sequence. The source of the style is so great Arial and 12.

Key words: word1, word2 and word3.

LISTA DE ILUSTRAÇÕES

Figura 1: Visão geral do mecanismo de acesso às páginas de internet. 26

Figura 2: Estrutura básica de um documento HTML exibido em uma IDE de programação. 27

Figura 3: Representação visual do código HTML em um navegador de internet. 28

Figura 4: Ilustração da estrutura de composição de um documento HTML contendo estilos CSS e códigos JavaScript. 29

Figura 5: Representação de um documento HTML como texto e como árvore de objetos. 32

Figura 6: Diagrama do caso de uso *Scrap* DrugBank. 34

Figura 7: Diagrama do caso de uso *Scrap* Merckmillipore. 37

Figura 8: Diagrama do caso de uso *Scrap* Scopus. 39

Figura 9: Diagrama de Classes do sistema. 47

Figura 10 – Exemplo de script para execução dos testes. 49

Figura 11 – Modelo de entrada para o *scraper* do site drugbank. 50

Figura 12 - Modelo de arquivo de saída para o *scraper* do site drugbank. 51

Figura 13 - Modelo de entrada para o *scraper* do site merckmillipore. 52

Figura 14 - Modelo de arquivo de saída para o scraper do site merckmillipore. 52

Figura 15 - Modelo de entrada para o *scraper* do site SCOPUS. 53

Figura 16 - Modelo de arquivo de saída para o scraper do site SCOPUS. 53

Figura 17: Exemplo de Figura. 57

LISTA DE TABELAS

Tabela 1: Descrição do caso de uso *Scrap* DrugBank. 34

Tabela 2: Descrição do caso de uso *Scrap* Merckmillipore. 36

Tabela 3: Descrição do caso de uso *Scrap* Scopus. 39

LISTA DE GRÁFICOS

Gráfico 1 – Índice de chaves do json retornado pela api de busca de artigos do SCOPUS. 52

Gráfico 2 – Etapas para processamento do SCOPUS. 53

Gráfico 3: Exemplo de um gráfico 61

LISTA DE ABREVIATURAS E SIGLAS

API – *Application Programming Interface*

ADR – *Adverse drug reaction*

FDA – *Food and Drug Administration (U.S. Federal agency)*

HPSG – *Head-driven phrase structure grammar*

MD5 – *Message-Digest algorithm 5*

ICMS – *Institute of Chinese Medical Sciences*

DNS – *Domain Name System*

IP – *Internet Protocol*

HTTP – *Hypertext Transfer Protocol*

HTML – *Hypertext Markup Language*

CSS – *Cascading Style Sheets*

IDE – *Integrated Development Environment*

JS – JavaScript

DOM – *Document Object Model*

RIA – *Rich Internet Application*

OOP – *Object-oriented Programming*

POO – Programação orientada a objetos

JSON – *JavaScript Object Notation*

CSV – *Comma-separated Values*

SUMÁRIO

Sumário

[AGRADECIMENTOS 6](#_Toc22389544)

[RESUMO 8](#_Toc22389545)

[ABSTRACT (opcional) 9](#_Toc22389546)

[LISTA DE ILUSTRAÇÕES 10](#_Toc22389547)

[Figura 1: Visão geral do mecanismo de acesso às páginas de internet. 21 10](#_Toc22389548)

[Figura 2: Estrutura básica de um documento HTML exibido em uma IDE de programação. 22 10](#_Toc22389549)

[Figura 3: Representação visual do código HTML em um navegador de internet. 23 10](#_Toc22389550)

[Figura 4: Ilustração da estrutura de composição de um documento HTML contendo estilos CSS e códigos JavaScript. 24 10](#_Toc22389551)

[LISTA DE TABELAS 11](#_Toc22389552)

[LISTA DE GRÁFICOS 12](#_Toc22389553)

[LISTA DE ABREVIATURAS E SIGLAS 13](#_Toc22389554)

[1 Introdução 17](#_Toc22389555)

[2 TRABALHOS RELACIONADOs 18](#_Toc22389556)

[2.1 discussão 20](#_Toc22389557)

[3 Fundamentação teórica 21](#_Toc22389558)

[3.1 coleta de dados 22](#_Toc22389559)

[3.2 TIPOS DE DADOS 22](#_Toc22389560)

[3.2.1 Qualitativos 22](#_Toc22389561)

[3.2.2 Quantitativos 23](#_Toc22389562)

[3.2.3 Primários 23](#_Toc22389563)

[3.2.4 Secundários 23](#_Toc22389564)

[3.3 ORGANIZAÇÃO E ARMAZENAMENTO 24](#_Toc22389565)

[3.3.1 Estruturados 24](#_Toc22389566)

[3.3.2 Não estruturados 24](#_Toc22389567)

[3.3.3 Semi estruturados 25](#_Toc22389568)

[3.4 ESTRUTURA DE PÁGINAS E NAVEGADORES DE INTERNET 25](#_Toc22389569)

[3.4.1 PADRÕES DE PÁGINAS DE INTERNET 30](#_Toc22389574)

[3.5 web scraping 31](#_Toc22389575)

[7.2.1 HTML *parsing* 33](#_Toc22389582)

[5 DESENVOLVIMENTO DO TRABALHO 35](#_Toc22389583)

[5.1 TEXTO DO TRABALHO 35](#_Toc22389584)

[5.1.1 REFERÊNCIAS BIBLIOGRÁFICAS 36](#_Toc22389585)

[5.1.2 CITAÇÕES 36](#_Toc22389586)

[5.1.3 IDIOMA ESTRANGEIRO 37](#_Toc22389587)

[5.1.4 FIGURAS, GRÁFICOS E TABELAS. 37](#_Toc22389588)

[5.1.5 NOTAS ENTRE O ORIENTADOR E O ALUNO 38](#_Toc22389589)

[6 CONCLUSÕES E TRABALHOS FUTUROS 39](#_Toc22389590)

[REFERÊNCIAS BIBLIOGRÁFICAS 40](#_Toc22389591)

[ANEXO A – TÍTULO DO ANEXO A 44](#_Toc22389592)

[ANEXO B – TÍTULO DO ANEXO B 45](#_Toc22389593)

# Introdução

Parte inicial do texto, onde devem constar a delimitação do assunto tratado, o problema ou oportunidade, objetivos da pesquisa, motivação para fazer a pesquisa, métodos usados e resultados encontrados, quando for o caso, e outros elementos necessários para situar o tema do trabalho, bem como a estrutura do documento.

A introdução pode ter a seguinte ordem:

* O que é o trabalho? (1 ou 2 parágrafos)
* Motivação para desenvolvê-lo? (1 parágrafo)
* Métodos usados, quando for o caso (linguagens, protocolos, metodologias, etc.) ( 1 ou 2 parágrafos)
* Organização do trabalho:
  + no Capítulo 2....;
  + o terceiro capítulo é dedicado ...;
  + ...; e
  + finalmente, no Capítulo “tal”, temos as conclusões e indicações para futuros trabalhos.

# TRABALHOS RELACIONADOs

Com o objetivo de reunir, como *background*, conteúdo relevante a pesquisa proposta neste trabalho, foram analisadas e listadas abaixo publicações acadêmicas relacionadas (ou, pelo menos, próximas) ao domínio farmacêutico.

Glez-Peña et al. [1] descrevem a dependência de tecnologias, ferramentas e dados *online* da pesquisa biomédica em função do crescimento do volume de informação e da necessidade de otimizar os benefícios resultantes de novas pesquisas através da extração de conteúdos de interesse em um tempo razoavelmente baixo e sem consumir muitos recursos, identificando os elementos necessários para a aplicação das técnicas de *web scraping*, comparando diferentes *softwares* e *frameworks* disponíveis e voltados para este fim, e propondo um escopo para aplicações no domínio biomédico tendo em vista o extenso uso de APIs para disponibilizar informações nesta área. Dentre as diversas tecnologias consideradas, os autores citam a biblioteca jsoup [2] para a linguagem de programação Java [3] e os *frameworks* Scrapy [4] e Beautiful Soup [5], ambos para a linguagem Python [6].

Feldman et al. [7] apresentam uma metodologia para extração de dados relacionados a reações adversas a medicamentos (ADRs) de fóruns médicos da internet e demonstram a habilidade do método em prever reações não listadas em testes clínicos antes que elas sejam reportadas a agência federal Estadunidense *Food and Drug Administration* (FDA), responsável por assuntos relacionados a saúde pública no país. A metodologia se baseia na combinação do conceito *head-driven phrase structure grammar* (HPSG) com padrões relacionais conhecidos do domínio de interesse aplicados a grandes quantidades de textos avulsos pós-processando a informação obtida com algoritmos automatizados para refinar o resultado obtido. A implementação da solução proposta utiliza o *framework* Scrapy [4], a biblioteca (nativa) MD5 da linguagem Python [6], a plataforma NLTK [8], também para a linguagem Python, e o *framework* URE, sobre o qual não foram encontradas informações a respeito, mas que é descrito como uma ferramenta de aproximação entre o HPSG e o processo *Automatic acquisition of lexicon*.

Liu, Liang e Wishart [9] oferecem uma segunda versão para um sistema *online* chamado PolySearch [10] que é apresentado como uma ferramenta de *text-mining* significantemente melhorada para a descoberta de associações entre doenças humanas, genes, medicamentos, toxinas e muitas outras entidades biomédicas através de consultas (*queries*) generalizadas do tipo “dada uma variável X, encontre todas as ocorrências associadas da variável Y” onde X e Y podem assumir qualquer uma das entidades do domínio suportadas pelo sistema, que foi desenvolvido utilizando Python e uma tecnologia de motor de busca chamada ElasticSearch [11]. A partir dos parâmetros informados, uma ampla busca na internet é realizada em múltiplas e variadas fontes como artigos acadêmicos e bancos de dados amplamente utilizados pelas comunidades biomédica e farmacêutica como DrugBank [12], UniProt [13] e Human Metabolome Database [14].

Amalia, Afifa e Herriyance [15] abordam o hábito dos Indonésios de recorrer, como uma medida de primeiros socorros, a motores tradicionais de busca para encontrar informações sobre doenças e possíveis medicamentos ao passo que estes são por vezes incapazes ou ineficientes ao lidar com sinônimos populares para nomes científicos, além de retornar artigos não relacionados com o assunto desejado e propõem, como solução, a construção de um motor de busca baseado em *semantic web* [16] capaz de extrair vocabulário de *websites* populares que sabidamente ou potencialmente tenham o conteúdo desejado, criando um relacionamento entre termos científicos e populares, incluindo sinônimos para as doenças mais pesquisadas (o artigo foca em doenças tropicais, embora o contexto possa ser expandido). A solução utiliza técnicas de *web scraping* para obter dados de fontes como o banco de dados Drugs.com [17], o portal AloDokter [18] e o *website* da agência nacional de controle de alimentos e medicamentos da Indonésia [19] gerando, como resultado, serializações do modelo RDF [20] fazendo uso de *frameworks* do tipo CSV2RDF [21] para a linguagem Python.

Gräßer et al. [22] buscam conhecer a preferência e a experiência de usuários do domínio farmacêutico através da análise automatizada por *web scraping* de fóruns e *websites* que contenham opiniões e *reviews* destes usuários: primeiro uma análise de sentimentos [22, p.121] é feita e uma previsão é gerada quanto a satisfação geral, efeitos colaterais e efetividade de um determinado medicamento. Em seguida, é investigada a compatibilidade dos modelos de classificação obtidos através da sua transferência entre diferentes domínios e fontes de dados, mostrando que esta abordagem pode ser utilizada para apontar, explorar e comparar similaridades e análises de sentimentos entre esses domínios. Os dados são obtidos de duas fontes independentes distintas e amplamente utilizadas no campo farmacêutico por profissionais e consumidores – Drugs.com [17] e DrugLib.com [23] – através de um web *crawler* implementado na linguagem Python utilizando o *framework* Beautiful Soup [5].

## discussão

Como é possível observar no tópico anterior, não foram encontrados muitos artigos no campo farmacêutico, sendo a maior parte do conteúdo relacionado a *data-mining* e *web scraping* disponível na internet voltado para a generalização destas atividades (e suas metodologias) ou para aplicações em domínios próximos como a da biomedicina.

Han et al. [24] trazem um estudo sobre a predição da desintegração da fórmula de medicamentos comprimidos com administração via oral utilizando técnicas de redes neurais, destacando a importância de dois fatores para o sucesso das predições: os dados avaliados e os algoritmos de processamento utilizados, percebendo a extração de dados como uma etapa crítica do processo e citando o uso da tradicional extração manual de dados em pesquisas atuais por cientistas farmacêuticos com experiência na área:

“*In order to ensure a satisfied prediction accuracy, two key factors are to be considered: data and algorithm. The first issue is the reliable data in pharmaceutical research. Deep-learning attempts to learn these characteristics to make better representations and create models from reliable data. Thus, data extraction is a critical step. In current research, reliable formulation datasets were manually extracted and labeled from the research articles of Web of Science by experienced pharmaceutical scientists*.” [24, p. 339]

Entendemos, com isso, que há espaço para soluções mais viáveis de coleta de dados no domínio farmacêutico através uma metodologia simples, fundamentada na determinação de fontes confiáveis para os dados de interesse associada a aplicação das técnicas de *web scraping* para extrair, com velocidade e confiabilidade, as informações desejadas dos dados obtidos, liberando assim os recursos humanos para outras atividades da pesquisa.

# Fundamentação teórica

À medida que surgem novas tecnologias e o aumento do poder computacional, são produzidos mais e mais dados. São aprimorados instrumentos de medição, que geram dados cada vez mais precisos e versam sobre temas cada vez mais amplos. Neste capítulo, discorremos sobre as diversas categorias de dados, as suas formas de armazenamento, e apresentamos técnicas para a coleta e análise destes dados.

## coleta de dados

A coleta de dados é o primeiro processo de obtenção dos dados e informações sobre variáveis de interesse, de forma sistemática, que permite responder perguntas, testar hipóteses e avaliar os resultados [‎‎32]. Esta etapa da pesquisa é comum a todas as áreas de estudo, incluindo ciências físicas e sociais, humanas, negócios, etc. Embora os métodos variem por disciplina, a ênfase em garantir uma coleta precisa e honesta permanece a mesma.

Em muitos casos, a coleta de dados deve ser realizada diretamente da fonte primária, como por exemplo, o censo demográfico realizado pelo IBGE [‎33], que precisa coletar, ainda que por amostragem, os dados diretamente da população por meio de entrevistas que são realizadas diretamente com o objeto da pesquisa. Mas em outros casos, o trabalho de produção dos dados já foi realizado por fontes terceiras confiáveis e precisam apenas ser coletados para compilação e análise.

A técnica a ser empregada na coleta de dados depende substancialmente da forma em que estes dados se apresentam.

## TIPOS DE DADOS

Os dados são divididos em duas grandes áreas: quantitativos e qualitativos [‎32]. Quanto à origem dos dados é de especial relevância, considerarmos a classificação destes em primários e secundários [‎34].

### Qualitativos

São dados não numéricos, geralmente descritivos de características de um objeto de interesse. São exemplos de dados qualitativos, o nome de uma pessoa, sexo, a classificação de uma espécie, a cor de um produto.

### Quantitativos

São valores numéricos. São discretos quando assumem valores enumeráveis, geralmente provenientes de contagem. Contínuos quando assumem valores reais, geralmente provenientes de mensuração.

É de suma importância esta classificação pois ela determina como estes dados podem ser obtidos, processados e finalmente apresentados. Exemplos de dados quantitativos são a idade de um aluno, a altura, o tamanho da população de uma espécie, o peso de um produto.

### Primários

Dados que são coletados em primeira mão. Ainda não foram publicados, e tem a vantagem de, por sua obtenção ter sido planejada especialmente para o estudo de interesse, serem mais confiáveis e adequados ao que se deseja deles. Têm a desvantagem de por vezes o método de extração ser de difícil acesso e custoso. Servem como fontes de obtenção de dados primários, os experimentos que devem ser controlados de forma a eliminar fatores de influência externa, e as entrevistas, normalmente utilizadas em pesquisas sociais.

### Secundários

Dados que são coletados a partir de medições e pesquisas já realizadas anteriormente. Tais dados nem sempre estão disponíveis com a integridade requerida, ou apresentados no formato mais adequado ao tipo de estudo que se deseja fazer deles. Por este motivo, ainda que de mais fácil obtenção, necessitam de igual atenção à sistemática de coleta, desde à definição de critérios para a seleção das fontes, até à metodologia empregada para a sua coleta e manejo.

## ORGANIZAÇÃO E ARMAZENAMENTO

Os dados coletados precisam ser armazenados de forma a garantir acesso de forma pratica. Os dados podem ser estruturados, não estruturados ou semiestruturados [‎35], a escolha para cada tipo depende do problema e da necessidade do domínio aonde se encontram os dados.

### Estruturados

São dados que possuem um formato rígido bem definido. Cada registro obedece a regras fixas onde existe um formato pré-estabelecido em relação a sua inclusão. São, por exemplo, as tabelas de banco de dados relacionais em que cada linha representa um registro, ou seja, uma pessoa, um objeto ou uma instância qualquer, e cada coluna possui registros específicos de dados. A mesma coisa ocorre em um formulário de cadastro que tenha campos fixados para preenchimento. Cada campo que deve ser preenchido foi predeterminado.

* Exemplos: Bancos de dados SQL, formulários de cadastro.
* Como acessar: Através de linguagem de consulta própria chamada SQL no caso dos bancos relacionais, ou busca em ficheiros comumente organizados por ordem alfabética no caso de um formulário físico de cadastro.

### Não estruturados

Não possuem estrutura fixa. Os principais exemplos são os arquivos de texto, que podem conter uma infinidade de informações, variando em língua, tamanho, tema, formatação.

Exemplo: arquivo de texto.

* Forma de acesso: interpretação para a busca da informação.

### Semi estruturados

Não tem uma predefinição no que tange ao conteúdo, mas algumas regras que balizam o seu formato, com a intenção de facilitar a organização e a apresentação das informações alí contidas. Ainda assim possuem alto grau de liberdade para a sua composição.

* Exemplo: xml, html, json.
* Como acessar: através de programas visualizadores que pelo conhecimento prévio das regras que pautam a sua formatação, os apresentam de maneira amigável ao usuário, tal como faz browser de internet com arquivos em formato html.

## ESTRUTURA DE PÁGINAS E NAVEGADORES DE INTERNET



Explicar o surgimento e o funcionamento da internet não faz parte da proposta deste trabalho. Por outro lado, para entender as metodologias e técnicas de extração de dados é essencial elucidar dois de seus principais elementos, ambos tão antigos quanto a própria internet [25]: as páginas e os navegadores.

Todo *website*, independente do seu formato ou conteúdo (portal de notícias, aplicativo de mapas geográficos, rede social, motor de pesquisa, etc.), é composto por uma ou mais páginas de internet organizadas e relacionadas entre si através de referências (*links*) que permitem uma navegação intuitiva por suas partes.

Cada uma dessas páginas de internet são arquivos comuns de computador essencialmente compostos de hipertexto [26] estruturado em blocos de elementos bem definidos principalmente por um par de *tags* que delimita o início e o fim de cada bloco, organizados a partir de duas divisões principais: um cabeçalho contendo metadados sobre o conteúdo da página e um corpo agrupando os diferentes elementos do conteúdo da página em si.

Segundo LONGMAN [26], o conceito de hipertexto existe desde a década de 1940 e começou a se tornar popular na década de 1980 com o surgimento dos computadores pessoais. Ao mesmo tempo, em meados dos anos 1980, é introduzido o sistema de nomes de domínios (DNS), que tem papel fundamental na forma com a qual acessamos páginas de internet até os dias de hoje.

Quando acessamos um endereço na internet através de um navegador (por exemplo: https://www.google.com), uma sequência bem definida de eventos ocorre de maneira transparente ao usuário, conforme a Figura 1 tenta ilustrar: (i) o navegador de internet extrai, do endereço da página, o nome de seu hospedeiro; (ii) com o auxílio de um sistema de DNS, o navegador obtém o endereço IP do computador (servidor) que hospeda a página a ser acessada; (iii) através do endereço do hospedeiro, o navegador acessa (e baixa) a base do documento HTML e todos os arquivos nele referenciados; e (iv), por fim, uma representação visual do conteúdo HTML é exibida para o usuário.



Figura 1: Visão geral do mecanismo de acesso às páginas de internet.

Embora formada por complexas redes de computadores interconectados através de diversos meios e protocolos físicos e lógicos, a internet foi inicialmente concebida simplesmente para compartilhar documentos entre universidades [26] e para que esses documentos pudessem oferecer uma estrutura clara, alguma formatação visual e, principalmente, vínculos diretos (*links*) entre partes de um mesmo documento ou entre diferentes documentos, o conceito de página de internet foi introduzido através da criação de uma linguagem de marcação de hipertexto chamada de HTML (*Hypertext Markup Language*) [27]. A Figura 2 mostra a estrutura básica de um documento HTML visualizado em uma IDE de programação.



Figura 2: Estrutura básica de um documento HTML exibido em uma IDE de programação.

Uma das principais vantagens da utilização de hipertexto, além do mecanismo de referências (introduzido no HTML), é a possibilidade de mesclar textos com arquivos multimídia [28] (como imagens, por exemplo) produzindo um conteúdo não apenas mais rico, mas também potencialmente mais visual.

A partir da aceitação do HTML e com o início da popularização da internet, já no início da década de 1990 [29], começaram a surgir os navegadores de internet: programas de computador capazes de traduzir HTML em representações visuais (através, por exemplo, da renderização de imagens) tornando a visualização de documentos mais amigável para os usuários, conforme ilustrado na Figura 3.



Figura 3: Representação visual do código HTML em um navegador de internet.

Após alguns anos de amadurecimento [30], com o aprimoramento das tecnologias existentes (podendo-se destacar a introdução do DOM – *Document Object Model* – em 1998, que permite a interpretação e a manipulação de elementos de documentos HTML como uma coleção de objetos independentes) e o surgimento de novas tecnologias como o CSS – que permite, dentre outras coisas, classificar grupos de elementos dentro de um mesmo documento HTML e estilizar, amplamente e com muita flexibilidade, os elementos de uma página – e a linguagem de programação JavaScript, capaz de executar códigos de alto nível diretamente nos navegadores de internet permitindo, dentre outras coisas, criar efeitos visuais como animações e modificar o conteúdo de uma página em tempo real (durante a navegação do usuário, reagindo a eventos como o clique de um botão), bastante dinamismo foi adicionado às páginas de internet ampliando assim o seu papel até se tornarem verdadeiras aplicações (RIA) [31] suportadas pelos avanços igualmente feitos nos navegadores de internet. A Figura 4 ilustra a composição de um documento HTML que faz uso dessas tecnologias citadas.



Figura 4: Ilustração da estrutura de composição de um documento HTML contendo estilos CSS e códigos JavaScript.

Embora a forma de acesso às páginas de internet mais conhecida e utilizada por usuários seja através dos navegadores, qualquer programa de computador pode agir como um cliente para servidores de páginas de internet através dos mesmos princípios pelos quais os navegadores de internet são criados e, como vimos no Capítulo 2, as linguagens de programação de alto nível atualmente oferecem bibliotecas e *frameworks* que possibilitam implementações deste tipo de forma bastante direta e simplificada permitindo que o código HTML de praticamente qualquer página na internet (partindo do princípio de que se tenha acesso a ela) possa ser extraído com poucas linhas de código.

A partir destes conceitos, têm-se então não apenas um padrão sólido e consolidado para a estrutura de páginas na internet (e, consequentemente, uma definição clara do papel dos navegadores), mas também um conjunto muito bem definido de elementos que representam a base de absolutamente todo conteúdo na internet fazendo com que a avaliação do conteúdo de uma “página de internet” (ou partes específicas de uma determinada página), uma vez que seu código hipertexto tenha sido obtido, seja possível e muito viável (como, por exemplo, dentre todo conteúdo de uma página, considerar apenas elementos do tipo “tabela” ou selecionar apenas elementos que contenham uma determinada classe CSS associada) sendo necessário observar apenas as possíveis alterações dinâmicas de conteúdo resultantes da execução de códigos JavaScript, quando houver.

### PADRÕES DE PÁGINAS DE INTERNET

Dentre as muitas *tags* que hoje compõem o HTML, é possível destacar às específicas para divisões de áreas, parágrafos, listas, tabelas e formulários, além das estruturais, responsáveis por delimitar, no corpo do documento, seções para cabeçalho, rodapé, *menu* de navegação e conteúdo principal.

A partir da interpretação de um agrupamento de páginas de internet como um aplicativo *web*, temos o encontro do HTML (e tecnologias relacionadas) com os padrões de *software* e as boas práticas de desenvolvimento que, buscando motivar a criação de produtos estáveis e de fácil manutenção, permitem a esperança de que *websites* criados por equipes de profissionais sérios buscarão respeitar as definições da linguagem permitindo diversos filtros e predições assertivas, desde que:

1. O código HTML da página seja minimamente válido;
2. Informações de um mesmo tipo, ainda que exibidas em páginas diferentes, sejam identificadas por uma mesma classe CSS ou elemento HTML (padronização);
3. A escolha de *tags* para determinada parte de um documento HTML seja lógica: embora, por exemplo, seja possível criar uma tabela sem o uso da *tag* apropriada “table”, a *tag* mais apropriada (neste caso, “table”) deverá ser eleita;
4. Por mais dinâmico que o conteúdo de uma página seja, com ou sem o uso de códigos JavaScript, a organização do código HTML (escolha de *tags*) tende a mudar com pouca frequência (a estrutura e o *layout* são pouco dinâmicos).

Partindo da ideia de que as afirmativas acima são verdadeiras para uma determinada página (ou *website*) eleita como fonte para extração de um determinado tipo ou conjunto de dados, podemos intuitivamente perceber que boa parte do trabalho de extração de dados fica simplificado pela possibilidade de se obter facilmente apenas as informações desejadas (relevantes) de qualquer documento HTML.

## web scraping

Dados são comumente publicados na web como documentos HTML. Tais documentos tem regras e definições próprias. Este formato tem o objetivo de permitir a apresentação do documento dentro de um *web browser*, como o google chrome [36], mozilla firefox [37] ou o microsoft internet explorer [38].

Alguém interessado em coletar dados publicados originalmente para serem apresentados em um *browser*, terá a tarefa inicial de converter estes dados em estruturas próprias adequadas ao seu estudo. Este processo pode ser feito manualmente ou automatizado por um programa de computador. Manualmente, utilizando um *web browser*, navega-se até a página pretendida; busca-se a informação visualmente; coleta-se o dado para futuro processamento. Automaticamente, o programa faz uma requisição HTTP e recebe um documento HTML; analisa o documento em busca da informação pretendida; retorna o trecho do documento que contém o dado.

### Buscando informações em um documento HTML

O documento HTML, é essencialmente um documento de texto e, portanto, técnicas de busca de uso geral em documentos de texto podem ser aplicadas. Dentre elas estão a busca por palavras chave simples, e a busca por padrões de texto mais complexos, como os padrões definidos com a linguagem de busca por padrões regex (*regular expression*) [39]. Há ainda as técnicas que utilizam a estrutura do documento para alcançar o dado pretendido. Exemplos de técnicas deste tipo são a linguagem XPath [40], que define um caminho dentro do documento até a informação, e a utilização de ferramentas que realizam o *parsing* do documento-o em uma representação de objetos encadeados, como a biblioteca Beautiful Soup [41].

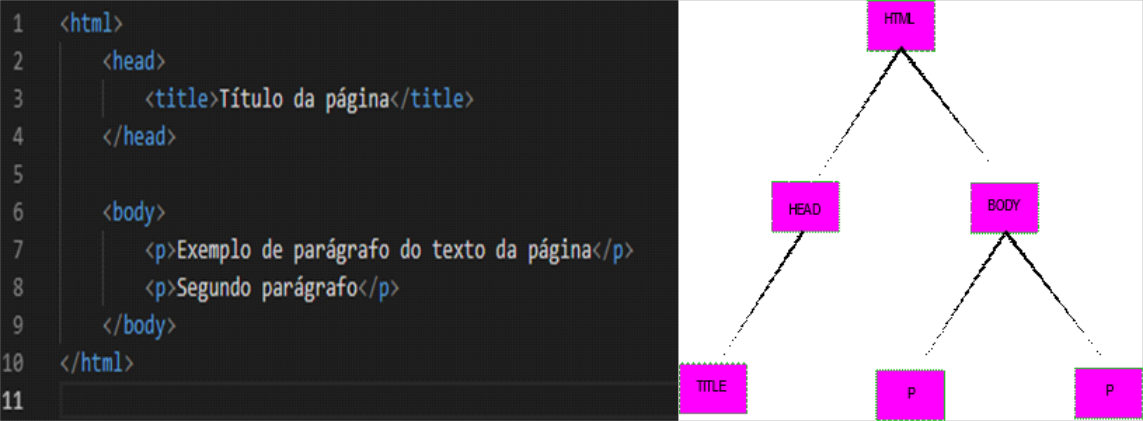


Figura 5: Representação de um documento HTML como texto e como árvore de objetos.

O *parsing* de um documento HTML é a análise sintática do texto. Nesta análise é identificada a estrutura da página, e verificado se está de acordo com as regras de composição de um arquivo HTML. É uma etapa importante para a tradução do arquivo texto para uma estrutura que pode ser compreendida e manipulada pelo computador. No exemplo da figura 5, o título da página pode ser extraído seguindo-se o caminho: html; head; title.

No exemplo da figura 5, Neste capítulo colocamos algumas informações relevantes para a elaboração do TCC.

# IMPLEMENTAÇÃO

Com base nas definições feitas em reunião junto a um pesquisador farmacêutico especialista no domínio de interesse deste trabalho (conforme detalhado no Anexo A), pudemos identificar todas as demandas necessárias para a implementação de uma ferramenta de *web scraping* que tem como objetivo automatizar a busca e a coleta de dados específicos sobre as propriedades e os excipientes de determinados fármacos e suas respectivas composições.

Neste capítulo abordaremos os detalhes técnicos desta implementação: apresentaremos os casos de uso disponíveis, falaremos sobre as tecnologias eleitas, forneceremos uma visão geral do sistema através de um diagrama de classes e definiremos as fontes a serem utilizadas pela ferramenta desenvolvida, conforme a orientação do especialista farmacêutico consultado.

## CASOS DE USO

### Diagrama do caso de uso *Scrap* DrugBank

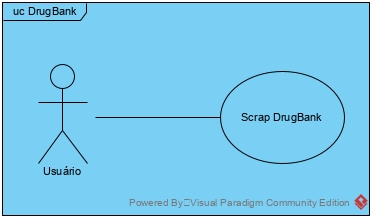


Figura 6: Diagrama do caso de uso *Scrap* DrugBank.

### Descrição do caso de uso *Scrap* DrugBank

Tabela 1: Descrição do caso de uso *Scrap* DrugBank.

|  |  |
| --- | --- |
| **ID:** | **UC01** |
| **Objetivo:** | Extrair dados de interesse da fonte “DrugBank”. |
| **Requisitos:** | Estar conectado à internet. |
| **Atores:** | Usuário. |
| **Pré-condições:** | Os pacotes, linguagens, *frameworks* e bibliotecas usados pela ferramenta precisam estar instalados no ambiente.  Os dados de interesse precisam estar definidos no arquivo (JSON) de configurações/entradas da fonte. |
| **Pós-condições:** | As informações obtidas deverão estar salvas em arquivo(s) no disco. |
| **Fluxo Principal:** | **Scrap DrugBank**  1. O ator executa a ferramenta, através de linha de comando, com os parâmetros que definem a fonte “DrugBank”;  2. A ferramenta busca os dados de interesse (definidos em um arquivo configurável do tipo JSON);  3. A ferramenta gera arquivo(s) de saída (conforme os parâmetros informados) no disco com as informações encontradas. |
| **Fluxos Alternativos:** | Não há. |
| **Erros/Exceções:** | O tratamento de erros (como requisições inválidas e *timeouts*) é feito pelo *framework* utilizado (mensagens de erro podem ser exibidas para o usuário).  Em caso de falha(s) na execução as pós-condições não serão verdadeiras (os dados de interesse não serão obtidos). |
| **Mensagens:** | *Logs* de saída do *framework* utilizado. |
| **Regras de negócio:** | **[RN01]** O endereço de internet “<https://www.drugbank.ca/unearth/q>" deve aceitar requisições do tipo GET com os parâmetros “utf8”, “searcher” e “query” e os respectivos valores “✓”, “drugs” e “{variável\_de\_interesse}” (este último contendo a variável desejada) e deve ser utilizado para buscar os dados de interesse.  **[RN02]** O resultado da requisição anterior (**RN01**) deverá ser uma página HTML válida contendo um elemento do tipo “h1” e outro do tipo “table” com ID “drug-moldb-properties”, de onde os dados de interesse devem ser extraídos.  **[RN03]** O resultado da requisição feita pela **RN01** poderá ser uma página HTML válida contendo um elemento “a” dentro de um elemento “h2” que, por sua vez, deverá estar dentro de um elemento “div” com a classe CSS “search-result” atribuída. Neste caso, uma nova requisição do tipo GET deverá ser feita ao endereço contido no atributo “href” da referida *tag* “a” e o resultado obtido desta nova requisição deverá ser o descrito na **RN02**. |

### Diagrama do caso de uso *Scrap* Merckmillipore

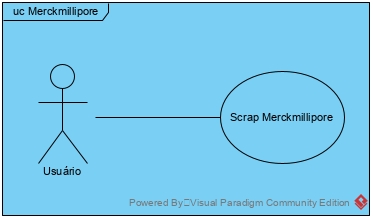


Figura 7: Diagrama do caso de uso *Scrap* Merckmillipore.

### Descrição do caso de uso *Scrap* Merckmillipore

Tabela 2: Descrição do caso de uso *Scrap* Merckmillipore.

|  |  |
| --- | --- |
| **ID:** | **UC02** |
| **Objetivo:** | Extrair dados de interesse da fonte “Merckmillipore”. |
| **Requisitos:** | Estar conectado à internet. |
| **Atores:** | Usuário. |
| **Pré-condições:** | Os pacotes, linguagens, *frameworks* e bibliotecas usados pela ferramenta precisam estar instalados no ambiente.  Os dados de interesse precisam estar definidos no arquivo (JSON) de configurações/entradas da fonte. |
| **Pós-condições:** | As informações obtidas deverão estar salvas em arquivo(s) no disco. |
| **Fluxo Principal:** | **Scrap Merckmillipore**  1. O ator executa a ferramenta, através de linha de comando, com os parâmetros que definem a fonte “Merckmillipore”;  2. A ferramenta busca os dados de interesse (definidos em um arquivo configurável do tipo JSON);  3. A ferramenta gera arquivo(s) de saída (conforme os parâmetros informados) no disco com as informações encontradas. |
| **Fluxos Alternativos:** | Não há. |
| **Erros/Exceções:** | O tratamento de erros (como requisições inválidas e *timeouts*) é feito pelo *framework* utilizado (mensagens de erro podem ser exibidas para o usuário).  Em caso de falha(s) na execução as pós-condições não serão verdadeiras (os dados de interesse não serão obtidos). |
| **Mensagens:** | *Logs* de saída do *framework* utilizado. |
| **Regras de negócio:** | **[RN04]** O endereço dinâmico de internet “<http://www.merckmillipore.com/BR/pt/search/>{variável\_de\_interesse}” (onde “{variável\_de\_interesse}” deve conter a variável desejada) deve aceitar requisições do tipo GET e deve ser utilizado para buscar as informações desejadas.  **[RN05]** O resultado da requisição anterior (**RN04**) deverá ser uma página HTML válida contendo um elemento do tipo “h1” que, por sua vez, deve conter um elemento do tipo “span” e, também, um elemento do tipo “table” com a classe CSS “attribute-group-table” atribuída, de onde os dados de interesse devem ser extraídos.  **[RN06]** O resultado da requisição feita pela **RN04** poderá ser uma página HTML válida contendo um elemento “a” dentro de um elemento “h2” que, por sua vez, deverá estar dentro de um elemento “div” com a classe CSS “container-serp” atribuída. Neste caso, uma nova requisição do tipo GET deverá ser feita ao endereço contido no atributo “href” da referida *tag* “a” e o resultado obtido desta nova requisição deverá ser o descrito na **RN05**. |

### Diagrama do caso de uso *Scrap* Scopus

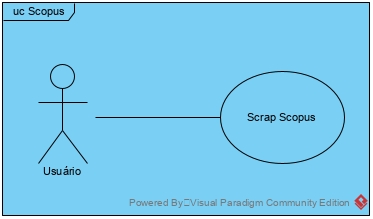


Figura 8: Diagrama do caso de uso *Scrap* Scopus.

### Descrição do caso de uso *Scrap* Scopus

Tabela 3: Descrição do caso de uso *Scrap* Scopus.

|  |  |
| --- | --- |
| **ID:** | **UC03** |
| **Objetivo:** | Extrair dados de interesse da fonte “Scopus”. |
| **Requisitos:** | Estar conectado à internet na rede da Universidade Federal Fluminense (ou em outra rede que forneça acesso ao site scopus.com e à sua API).  Ter uma chave de acesso válida à API do “Scopus”. |
| **Atores:** | Usuário. |
| **Pré-condições:** | Os pacotes, linguagens, *frameworks* e bibliotecas usados pela ferramenta precisam estar instalados no ambiente.  A busca a ser realizada precisa estar definida no arquivo (JSON) de configurações/entradas da fonte.  A chave de acesso à API do “Scopus” precisa estar definida no arquivo (JSON) de configurações/entradas da fonte. |
| **Pós-condições:** | As informações obtidas deverão estar salvas em arquivo(s) no disco. |
| **Fluxo Principal:** | **Scrap Scopus**  1. O ator executa a ferramenta, através de linha de comando, com os parâmetros que definem a fonte “Scopus”;  2. A ferramenta busca os dados de interesse (definidos em um arquivo configurável do tipo JSON);  3. A ferramenta gera arquivo(s) de saída (conforme os parâmetros informados) no disco com as informações encontradas. |
| **Fluxos Alternativos:** | Não há. |
| **Erros/Exceções:** | O tratamento de erros (como requisições inválidas e *timeouts*) é feito pelo *framework* utilizado (mensagens de erro podem ser exibidas para o usuário).  Em caso de falha(s) na execução as pós-condições não serão verdadeiras (os dados de interesse não serão obtidos). |
| **Mensagens:** | *Logs* de saída do *framework* utilizado. |
| **Regras de negócio:** | **[RN07]** Apenas artigos publicados no últimos 10 (dez) anos devem ser considerados.  **[RN08]** As palavras-chave buscadas devem aparecer no título e/ou no *abstract* do artigo para que o mesmo seja considerado.  **[RN09]** Cada artigo deve ter uma fonte associada e cada fonte deve ter uma pontuação (*score*) que determina seu fator de impacto – apenas fontes que tenham a média dos últimos 3 (três) anos da sua respectiva pontuação maior que 1 (um) devem ser consideradas.  **[RN10]** Cada artigo deve ter um código DOI único associado – este código deve ser utilizado para evitar a coleta de artigos duplicados.  **[RN11]** O endereço de internet “https://api.elsevier.com/content/search/scopus" deve aceitar requisições do tipo GET com os parâmetros “query” e “count” com os valores das variáveis “termo buscado” (ou consulta – *query* – definida conforme os padrões da fonte “Scopus”) e “número desejado de resultados”, respectivamente, e deve ser utilizado para buscar os dados de interesse.  **[RN12]** A chave de acesso à API do “Scopus” deve estar contida no cabeçalho das requisições de busca (**RN11**), no campo de nome “X-ELS-APIKey”.  **[RN13]** O número de resultados obtidos pode ser limitado pelo usuário através das definições no arquivo de configurações/entradas da fonte.  **[RN14]** Os resultados das requisições de busca (**RN11**) devem estar no formato JSON e os registros encontrados devem estar na chave de nome “search-results” e conter, para cada artigo retornado pela busca, os campos “source-id”, “doi” e “url” com o ID da fonte, o código DOI e o endereço de internet associados, respectivamente.  **[RN15]** Os endereços de internet referentes aos artigos encontrados (obtidos pelas requisições definidas na **RN11** e conforme descrito na **RN14**) devem aceitar requisições do tipo GET com os parâmetros “httpAccept” e “fields” com os respectivos valores “application/json” e “description” e devem ser utilizados para coleta dos dados de interesse.  **[RN16]** Os resultados das requisições de acesso aos artigos (**RN15**) devem estar no formato JSON contendo, dentro da chave aninhada “abstracts-retrieval-response → item → bibrecord → head”, o *abstract* e o endereço da fonte do artigo nas chaves “abstracts” e “source → website”, respectivamente.  **[RN17]** O endereço dinâmico de internet “https://doi.org/{doi}” (onde “{doi}” deve conter a código DOI único de cada artigo obtido pelas requisições definidas na **RN11** e conforme descrito na **RN14**) deve aceitar requisições do tipo GET e deve ser utilizado para verificar a validade de cada artigo e, também, evitar coletas duplicadas.  **[RN18]** O endereço dinâmico de internet “https://www.scopus.com/source/citescore/{id\_da\_fonte}.uri” (onde “{id\_da\_fonte}” deve conter o valor do campo “source-id” obtido pelas requisições definidas na **RN11** e conforme descrito na **RN14**) deve aceitar requisições do tipo GET e deve ser utilizado para obter os dados de pontuação (*score*) das fontes dos artigos encontrados.  **[RN19]** Os resultados das requisições definidas na **RN18** devem estar no formato JSON e conter o campo “rp”, definido na chave aninhada “yearInfo → {ano} → metricType”, onde {ano} assume o valor do ano desejado.  **[RN20]** As requisições definidas na **RN19** só devem ser consideradas caso a resposta contenha o campo ““documentType” com o valor “all”. |

## TECNOLOGIAS

A biblioteca python scrapy [4] foi construída com o objetivo de ser um framework completo para a tarefa de extrair dados de websites a partir do documento HTML. Para isto, ela gerencia as requisições HTTP ao site, expõe uma ferramenta para navegação pelo conteúdo do documento HTML via xpath [40] ou seletores css [48], e ainda, exporta os dados coletados para um arquivo com diferentes opções de formato: csv [21], json [44], xml [49].

A figura 9 representa um diagrama de atividades para uma implementação mínima, mas ainda extremamente flexível de um programa baseado no scrapy.

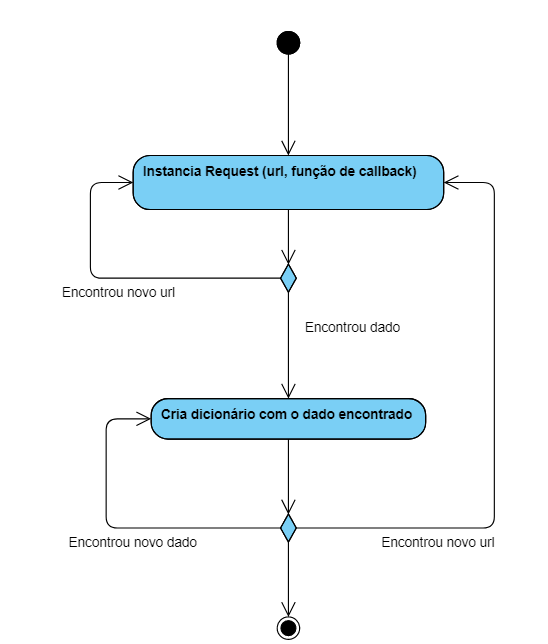


Figura 9 - Diagrama atividades de um programa scrapy mínimo.

Esta implementação consiste em estender a classe Spider e sobrescrever o método start\_request. O método start\_request deve realizar criações sucessivas de instâncias da classe Request. A requisição para o endereço indicado na criação do objeto é gerenciada pelo framework. O framework decide o momento adequado de realizar a requisição e se será necessário tentar novamente em caso de falha. Após uma requisição bem sucedida, a função que é passada como parâmetro na criação do objeto de Request é chamada e é responsável por realizar a procura desejada dentro do documento. A função deve criar, então, mais instâncias da classe Request ou objetos do tipo dicionário, que serão tratados pelo framework como um item do resultado final da coleta de dados.

## DIAGRAMA DE CLASSES

O diagrama de classes do sistema (Figura 9) representa a estrutura geral, a organização e os relacionamentos entre as classes da ferramenta implementada com base no *framework* Scrapy[4] (conforme visto no tópico anterior).

Recorrendo aos conceitos de herança e sobreposição de métodos do paradigma de programação orientada a objetos (POO), as classes dno sistema estendem determinadas classes generalizadas do *framework* utilizado obtendo assim um mecanismo de execução das tarefas relacionadas a atividade de *web scraping*, e especializam, também, uma classe especial responsável pela lógica individual de cada fonte de dados (se beneficiando dos recursos oferecidos pelo *framework* e permitindo fácil expansão através da inclusão de novas fontes de dados, se necessário).

### Descrição das classes

* **JSONExporter –** Esta classe é responsável pela escrita no disco do(s) arquivo(s) de saída no formato JSON, utilizando a codificação UTF-8, com os dados obtidos de cada fonte.
* **CSVExporter –** Com o auxílio da biblioteca Pandas[41], esta classe realiza a escrita no disco do(s) arquivo(s) de saída no formato CSV, utilizando a codificação ANSI, com os dados obtidos de cada fonte.
* **JSONExportPipeline –** Respeitando a estrutura do *framework* utilizado, esta classe implementa um conjunto pré-definido de métodos de auxílio ao processo de exportação de dados[42]. No sistema proposto, esta classe apoia a geração de arquivos no formato JSON.
* **CSVExportPipeline –** Esta classe sobrescreve um pequeno conjunto de propriedades e métodos da classe JSONExportPipeline para apoiar a exportação de dados no formato CSV.
* **BaseSpider** – Esta é uma classe intermediária que estende a classe especial “Spider” do *framework* utilizado (responsável pelas atividades específicas de extração de dados), agregando as propriedades comuns às outras classes do sistema deste mesmo tipo (descritas a seguir) e garantindo que a classe genérica (provida pelo *framework*) seja inicializada corretamente com os parâmetros necessários. Idealmente, esta classe deveria ser abstrata; em função das particularidades das tecnologias eleitas, trata-se de uma classe concreta que não deve ser instanciada ou utilizada diretamente, mas especializada para sobrepor seus principais métodos[43] com a lógica relacionada a cada diferente fonte de dados.
* **DrugBankSpider –** Esta classe é responsável pela extração de dados da fonte “DrugBank”, contendo toda lógica relacionada as regras de negócio desta fonte. A classe realiza a busca pelos dados de interesse através de requisições do tipo GET ao *website* da fonte em questão e extrai as informações relevantes encontradas com base na estrutura HTML já conhecida da página acessada pelo sistema.
* **MerckSpider –** Análoga a classe DrugBankSpider, a classe MerckSpider desempenha o mesmo papel descrito acima, porém para a fonte “Merckmillipore”: requisições do tipo GET são feitas à página de busca do *website* desta fonte e as variáveis de interesse encontradas são extraídas do conteúdo da página HTML retornada com estrutura também já conhecida.
* **ScopusSpider –** A classe ScopusSpider desempenha o mesmo papel das duas últimas classes descritas acima, porém para a fonte “Scopus”. Seu funcionamento difere das duas classes anteriores na forma com que os dados são obtidos: as requisições são feitas a uma API que disponibiliza os dados no formato JSON, com estrutura conhecida, de onde as informações são extraídas; a classe também faz requisições ao *website* da fonte para atender as regras de negócio referentes a pontuação (*score*) da origem de cada artigo encontrado e consulta o *website* https://doi.org para validar os artigos considerados relevantes pelo sistema e evitar a coleta de artigos duplicados.

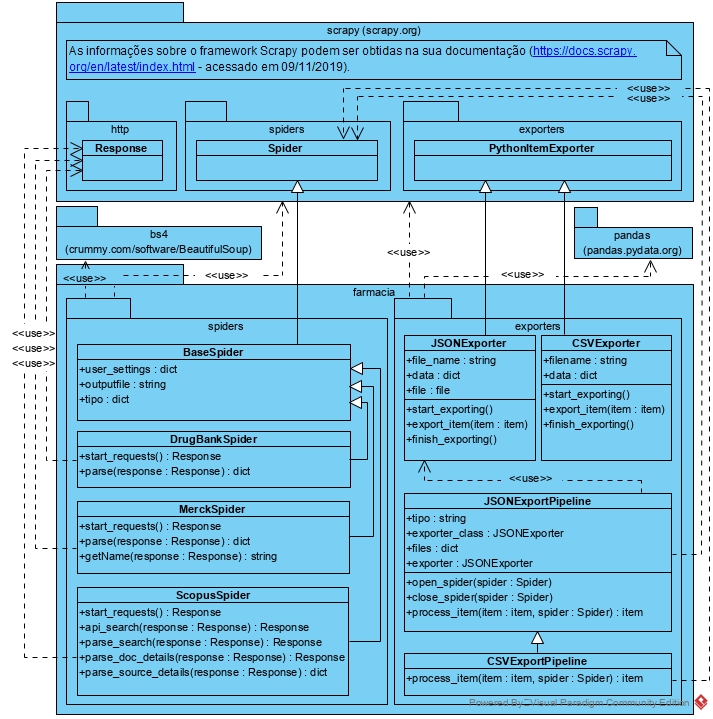


Figura 10: Diagrama de Classes do sistema.

## FONTES

Esta seção detalha para cada site utilizado, a sequência de atividades desenvolvidas para alcançar os objetivos de extração de dados.

### DrugBank

O processamento é iniciado com a busca pelo nome do fármaco. O primeiro objeto do tipo request é criado para o endereço https://www.drugbank.ca/unearth/q?utf8=%E2%9C%93&query=nome\_do\_farmaco&searcher=drugs. Onde o parâmetro query indica qual é o fármaco a ser pesquisado. Esta requisição pode retornar uma lista de resultados, um resultado único ou nenhum resultado.

Caso o retorno seja uma lista de resultados, o primeiro item da lista é seguido, abrindo uma página de detalhes para o item. Caso seja um resultado único, não é apresentada lista e o site redireciona automaticamente para a página de detalhes do fármaco. A figura 12 mostra, como exemplo, a página de detalhes resultado da busca para o fármaco tylenol.

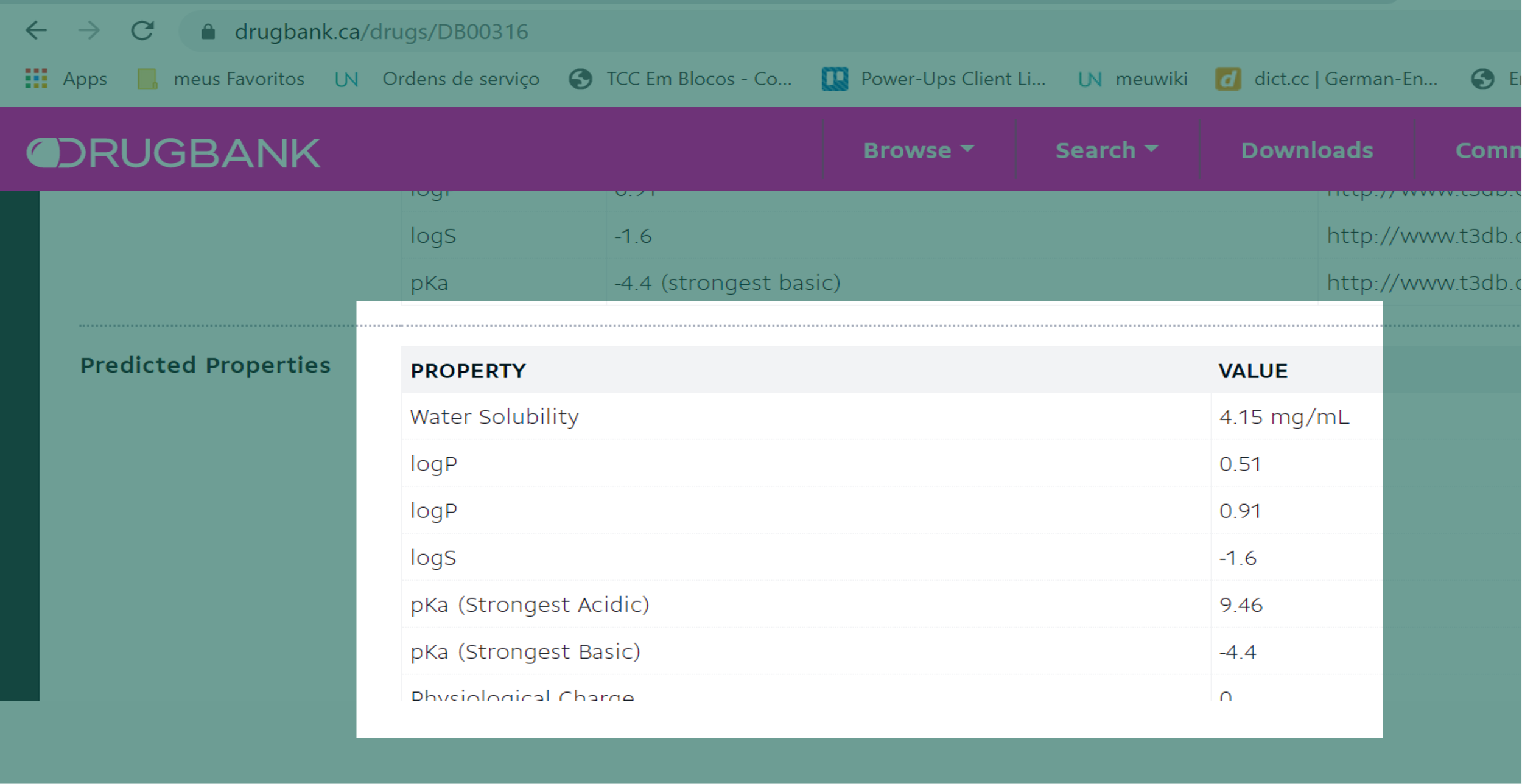


Figura 11 - página de detalhe do fármaco tylenol no drugbank.

A tabela em evidência é identificada pelo id da tag HTML igual a "drug-moldb-properties". A primeira célula de cada linha da tabela indica o nome da propriedade e a segunda, o valor.

### MerckMillipore

O processamento é iniciado com a busca pelo nome do fármaco. O primeiro objeto do tipo request é criado para o endereço http://www.merckmillipore.com/BR/pt/search/**nome\_do\_farmaco**. Onde o termo **nome\_do\_farmaco** indica qual é o fármaco a ser pesquisado. Esta requisição pode retornar uma lista de resultados, um resultado único ou nenhum resultado.

Caso o retorno seja uma lista de resultados, o primeiro item da lista é seguido, abrindo uma página de detalhes para o item. Caso seja um resultado único, não é apresentada lista e o site redireciona automaticamente para a página de detalhes do fármaco. A figura 12 mostra, como exemplo, a página de detalhes do resultado da busca para o fármaco Aerosil.

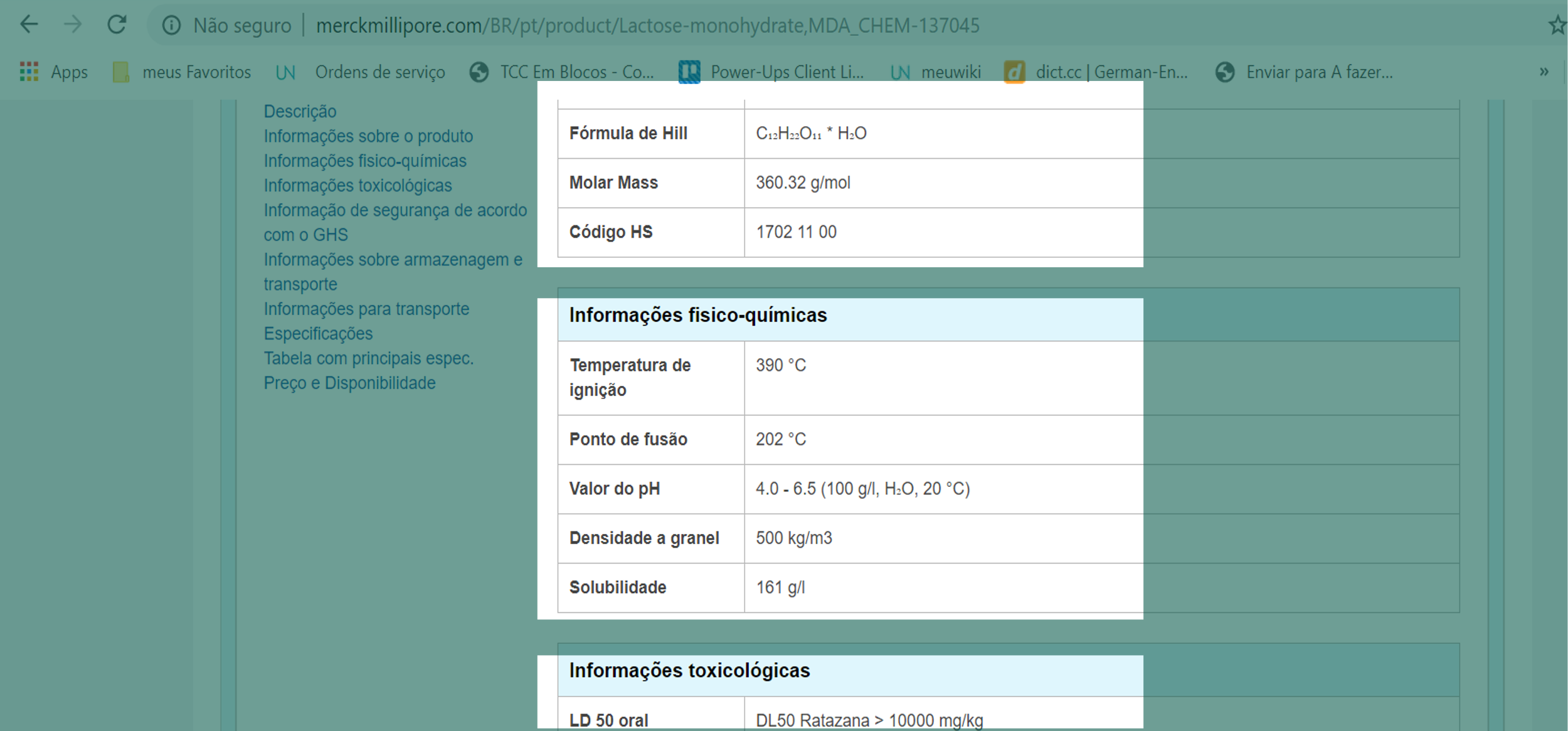


Figura 10 - Página de detalhes do fármaco Lactose monohidratada no merckmilliporo.

O grupo de tabelas em evidência é selecionado como todos os elementos de tag table que contém a classe attribute-group-table. A primeira coluna de cada linha indica o nome da propriedade e a segunda, o valor.

### Scopus

O Scopus disponibiliza uma API [50] para facilitar o trabalho de busca de dados em seus serviços, que retorna um documento em formato JSON. Apesar de o framework scrapy fornecer ferramentas específicas para trabalhar apenas com documentos HTML, esta deficiência foi superada utilizando a biblioteca padrão da linguagem python json [51] para navegação pelo documento retornado. Esta biblioteca transforma o texto em formato JSON em um objeto ou lista de objetos e, desta forma, permite acesso às informações diretamente através de suas propriedades. Seguindo as instruções da documentação da API do Scopus, utilizamos a url https://api.elsevier.com/content/search/scopus?query=**QUERY\_STRING**, onde o termo **QUERY\_STRING** define os parâmetros de busca. Para que a requisição seja processada pelo Scopus, o header da requisição deve conter a chave X-ELS-APIKey com um valor de chave de api habilitada para utilizar o serviço. Obtemos uma chave de api do Scopus após cadastro no próprio site. O Scopus restringe, ainda, o acesso ao serviço somente a IPs previamente habilitados, como é o caso da rede interna da Universidade Federal Fluminense, utilizada neste projeto para acessar o Scopus. O termo de busca fica definido pelo usuário no momento da execução. O gráfico 2 mostra um índice de chaves do arquivo json retornado pela API.

search-results

search-results **->** opensearch:totalResults

search-results **->** opensearch:startIndex

search-results **->** opensearch:itemsPerPage

search-results **->** opensearch:Query

search-results **->** opensearch:Query -> @role

search-results **->** opensearch:Query -> @searchTerms

search-results **->** opensearch:Query -> @startPage

search-results **->** link

search-results **->** link **-> []** **->** @\_fa

search-results **->** link **-> []** **->** @ref

search-results **->** link **-> []** **->** @href

search-results **->** link **-> []** **->** @type

search-results **->** entry

search-results **->** entry **-> []** **->** @\_fa

search-results **->** entry **-> []** **->** link

search-results **->** entry **-> []** **->** link **-> [] ->** @\_fa

search-results **->** entry **-> []** **->** link **-> [] ->** @ref

search-results **->** entry **-> []** **->** link **-> [] ->** @href

search-results **->** entry **-> []** **->** prism:url

search-results **->** entry **-> []** **->** dc:identifier

search-results **->** entry **-> []** **->** eid

search-results **->** entry **-> []** **->** dc:title

search-results **->** entry **-> []** **->** dc:creator

search-results **->** entry **-> []** **->** prism:publicationName

search-results **->** entry **-> []** **->** prism:issn

search-results **->** entry **-> []** **->** prism:volume

search-results **->** entry **-> []** **->** prism:pageRange

search-results **->** entry **-> []** **->** prism:coverDate

search-results **->** entry **-> []** **->** prism:coverDisplayDate

search-results **->** entry **-> []** **->** prism:doi

search-results **->** entry **-> []** **->** pii

search-results **->** entry **-> []** **->** citedby-count

search-results **->** entry **-> []** **->** affiliation

search-results **->** entry **-> []** **->** affiliation **-> [] ->** @\_fa

search-results **->** entry **-> []** **->** affiliation **-> []** **->** affilname

search-results **->** entry **-> []** **->** affiliation **-> []** **->** affiliation-city

search-results **->** entry **-> []** **->** affiliation **-> []** **->** affiliation-country

search-results **->** entry **-> []** **->** prism:aggregationType

search-results **->** entry **-> []** **->** subtype

search-results **->** entry **-> []** **->** subtypeDescription

search-results **->** entry **-> []** **->** article-number

search-results **->** entry **-> []** **->** source-id

search-results **->** entry **-> []** **->** openaccess

search-results **->** entry **-> []** **->** openaccessFlag

Gráfico 1 – Índice de chaves do json retornado pela api de busca de artigos do SCOPUS.

A lista de artigos desejada é encontrada seguindo as chaves search-results e entry. Além das propriedades já retornadas pela api para cada artigo, adicionamos o texto completo do abstract, um link para o artigo gerado a partir do DOI [52] do mesmo e a pontuação dada pelo Scopus para a publicação nos últimos três anos. O gráfico 3 resume as etapas do processamento realizado na fonte Scopus.

Gráfico 2 – Etapas para processamento do SCOPUS.

# TESTES

Neste capítulo descrevemos em detalhes a execução de um caso concreto de cada um dos *scrapers* desenvolvidos. Medimos o tempo de execução e verificamos a ocorrência de eventuais falhas.

Os testes foram executados em um computador com as seguintes configurações:

Tabela 4: Configurações do sistema.

|  |  |
| --- | --- |
| Nome do sistema operacional | Microsoft Windows 8.1 Enterprise |
| Tipo de sistema | x64-based PC |
| Processador(es) | [01]: Intel64 Family 6 Model 60 Stepping 3 GenuineIntel ~1900 Mhz |
| Memória física total | 16.213 MB |
| Versão do Python instalada | Python 3.6.8 :: Anaconda 4.4.0 (64-bit) |

Para cada site, foi criado um programa com objetivo específico de navegar pela estrutura do site até a obtenção dos dados de interesse. Para isto, o usuário especialista deve fornecer uma série de parâmetros encapsulados em um arquivo do tipo JSON [44] que é fornecido como entrada para o programa. O usuário, na chamada do programa define ainda o tipo de arquivo de saída (se CSV [21] ou JSON), e o nome e caminho onde será salvo o arquivo com o resultado do processamento.

A medição do tempo de execução de cada programa foi realizada através um arquivo de script do Windows, demonstrado na figura 10.

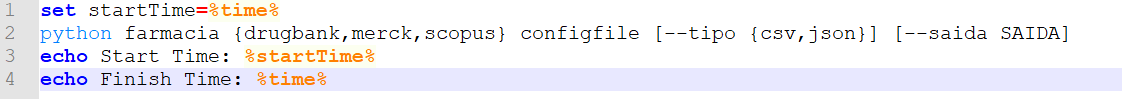


Figura 11 – Exemplo de script para execução dos testes.

O script registra os tempos antes e após a execução do comando. O comando genérico exibido na linha 2 da figura 10, é composto pela chamada do programa, um parâmetro para especificar qual fonte será executada – drugbank [12], merck [47] ou scopus [44]; o nome do arquivo de configuração que define os termos de busca e saída de acordo com a especificação de cada fonte; o tipo de saída; e o nome do arquivo de saída.

## DRUGBANK

O objetivo do programa referente ao site drugbank [12] é encontrar uma lista de fármacos pelo nome. Encontrada a página específica de cada fármaco, buscar por uma lista de propriedades. Ambas as entradas: lista de fármacos e de propriedades são de responsabilidade do usuário especialista. O usuário edita um arquivo em formato JSON [44] seguindo o modelo demonstrado na figura 11 e o referencia como entrada para o programa.



Figura 12 – Modelo de entrada para o *scraper* do site drugbank.

A lista de fármacos a serem buscados se encontra na chave "*inputs*" e a lista de propriedades, na chave "*outputs*". O programa foi executado com o comando ilustrado na figura 12 e totalmente processado em 2,92 segundos e produziu como saída um arquivo em formato CSV [21] conforme o modelo apresentado na figura 12.

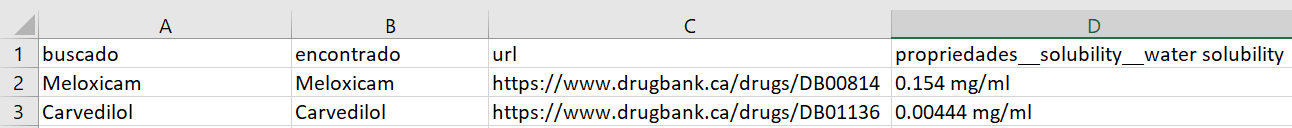


Figura 13 - Modelo de arquivo de saída para o *scraper* do site drugbank.

## merckmillipore

O objetivo do programa referente ao site merckmillipore [45] é encontrar uma lista de fármacos pelo nome. Encontrada a página específica de cada fármaco, buscar por uma lista de propriedades. Ambas as entradas: lista de fármacos e de propriedades são de responsabilidade do usuário especialista. O usuário edita um arquivo em formato JSON [44] seguindo o modelo demonstrado na figura 13, e o referencia como entrada para o programa.

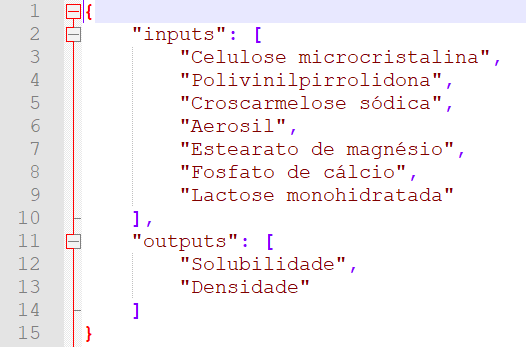


Figura 14 - Modelo de entrada para o *scraper* do site merckmillipore.

A lista de fármacos a serem buscados se encontra na chave "*inputs*" e a lista de propriedades, na chave "*outputs*". O programa foi executado em 9,72 segundos e produziu como saída um arquivo em formato CSV [21] conforme o modelo apresentado na figura 14.

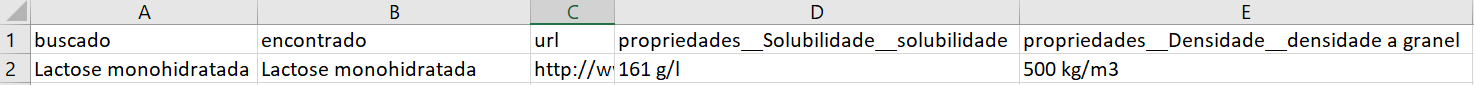


Figura 15 - Modelo de arquivo de saída para o scraper do site merckmillipore.

## SCOPUS

O objetivo do programa referente ao site scopus [46] é encontrar uma lista de artigos por um critério de busca definido pelo usuário especialista. O usuário edita um arquivo em formato JSON [44] seguindo o modelo demonstrado na figura 15, e o referencia como entrada para o programa. O site scopus realiza autenticação por IP. Para que esta execução fosse possível, a máquina responsável foi conectada à rede da Universidade Federal Fluminense.

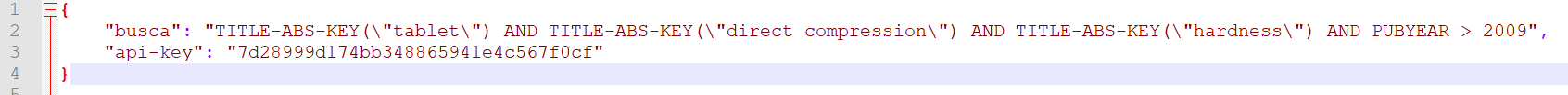


Figura 16 - Modelo de entrada para o *scraper* do site SCOPUS.

O critério de busca a ser utilizado encontra-se na chave "busca". Em nossas primeiras tentativas de processar uma busca completa pelo scopus com a chave descrita na figura 19, o resultado esperado era um arquivo com 1257 artigos visto que este era o número de resultados que o sistema do SCOPUS indicava tanto através de sua API como de sua interface web. No entanto, após diversas tentativas, identificamos um comportamento inesperado no *scraper* que não conseguia processar 100% dos resultados. Nosso arquivo de saída continha entre 1210 e 1250 resultados. Por este motivo, decidimos acionar a extensão do *framework* scrapy [4] chamada *autothrotle* [47] que tem o objetivo de evitar a super utilização dos recursos do site, aumentando o tempo de espera entre as requisições. Após o ajuste, o programa foi executado em 21 minutos e 18 segundos e produziu como saída um arquivo em formato CSV [21] contendo todos os 1257 resultados esperados conforme o modelo apresentado na figura 15 com os principais campos.

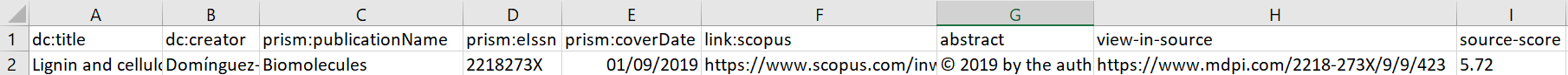


Figura 17 - Modelo de arquivo de saída para o scraper do site SCOPUS.

# CONCLUSÕES E TRABALHOS FUTUROS

Neste capítulo devem ser colocadas as conclusões que o aluno obteve durante a elaboração do trabalho, bem como o que pretende após sua conclusão (especialização, mestrado, aplicar os conhecimentos em alguma área...).

REFERÊNCIAS BIBLIOGRÁFICAS

1. Daniel Glez-Peña, Anália Lourenço, Hugo López-Fernández, Miguel Reboiro-Jato, Florentino Fdez-Riverola. **Web scraping technologies in an API world**, 2014. Briefings in Bioinformatics, Volume 15, Issue 5, September 2014, Pages 788–797.
2. HEDLEY, Jonathan. **Biblioteca Java para trabalhar com HTML.** <[https://jsoup.org](https://jsoup.org/)> Acesso em 06 set. 2019.
3. ORACLE. **Linguagem de programação**. <[https://java.com](https://www.java.com/)> Acesso em 06 set. 2019.
4. SCRAPINGHUB; et al. ***Framework* Python para extração de dados de *websites***. <[https://scrapy.org](https://scrapy.org/)> Acesso em 06 set. 2019.
5. RICHARDSON, Leonard. **Biblioteca Python para “*scraping* de tela”**. <[https://crummy.com/software/BeautifulSoup](https://www.crummy.com/software/BeautifulSoup/)> Acesso em 06 set. 2019.
6. PYTHON SOFTWARE FOUNDATION. **Linguagem de programação**. <[https://python.org](https://www.python.org/)> Acesso em 06 set. 2019.
7. Ronen Feldman , Oded Netzer , Aviv Peretz , Binyamin Rosenfeld, Utilizing Text Mining on Online Medical Forums to Predict Label Change due to Adverse Drug Reactions, Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 10-13, 2015, Sydney, NSW, Australia.
8. NLTK PROJECT. **Plataforma para criar programas Python para trabalhar com dados de linguagem humana**. <https://www.nltk.org> Acesso em 09 set. 2019.
9. Yifeng Liu, Yongjie Liang, David Wishart, PolySearch2: a significantly improved text-mining system for discovering associations between human diseases, genes, drugs, metabolites, toxins and more, Nucleic Acids Research, Volume 43, Issue W1, 1 July 2015, Pages W535–W542.
10. CANADIAN INSTITUTES OF HEALTH RESEARCH; et al. **Ferramenta *online* para *text-mining***. <http://polysearch.ca> Acesso em 09 set. 2019.
11. ELASTICSEARCH B. V.. **Motor de busca**. <[https://elastic.co](https://www.elastic.co/)> Acesso em 09 set. 2019.
12. CANADIAN INSTITUTES OF HEALTH RESEARCH; et al. **Banco de dados de medicamentos e substâncias bioquímicas**. <[https://drugbank.ca](https://www.drugbank.ca/)> Acesso em 09 set. 2019.
13. NATIONAL INSTITUTES OF HEALTH, U.S. Department of Health and Human Services; et al. **Banco de dados de informações funcionais e sequências de proteínas**. <[https://uniprot.org](https://uniprot.org/)> Acesso em 09 set. 2019.
14. CANADIAN INSTITUTES OF HEALTH RESEARCH; et al. **Banco de dados de informações sobre o metabolismo de pequenas moléculas encontradas no corpo humano**. <[http://hmdb.ca](http://www.hmdb.ca/)> Acesso em 09 set. 2019.
15. A. Amalia, R. M. Afifa and H. Herriyance, "Resource Description Framework Generation for Tropical Disease Using Web Scraping," 2018 IEEE International Conference on Communication, Networks and Satellite (Comnetsat), Medan, Indonesia, 2018, pp. 44-48.
16. WORLD WIDE WEB CONSORTIUM, W3C. **Padrões da “*web* semântica”**. <[https://w3.org/standards/semanticweb](https://www.w3.org/standards/semanticweb/)> Acesso em 09 set. 2019.
17. DRUGS.COM. **Banco de dados independente com informações sobre medicamentos**. <[https://drugs.com](https://www.drugs.com/)> Acesso em 09 set. 2019.
18. ALODOKTER. **Portal *web* de informações médicas e sobre saúde**. <https://alodokter.com> Acesso em 09 set. 2019.
19. AGÊNCIA NACIONAL DE CONTROLE DE ALIMENTOS E MEDICAMENTOS DA INDONÉSIA. ***Website* oficial da instituição**. <http://pom.go.id> Acesso em 09 set. 2019.
20. WORLD WIDE WEB CONSORTIUM, W3C. **Padrões do modelo *Resource Description Framework***. <[https://w3.org/RDF/](https://www.w3.org/RDF/)> Acesso em 09 set. 2019.
21. WORLD WIDE WEB CONSORTIUM, W3C. **Procedimentos e regras para conversão de dados tabulares em RDF**. <[https://w3.org/TR/csv2rdf/](https://www.w3.org/TR/csv2rdf/)> Acesso em 09 set. 2019.
22. Felix Gräßer, Surya Kallumadi, Hagen Malberg, and Sebastian Zaunseder. 2018. Aspect-Based Sentiment Analysis of Drug Reviews Applying Cross-Domain and Cross-Data Learning. In DH’18:2018 International Digital Health Conference, April 23–26, 2018, Lyon, France. ACM, New York, NY, USA, 5 pages.
23. DRUGLIB.COM. **Banco de dados sobre medicamentos e pesquisas farmacêuticas**. <http://druglib.com> Acesso em 09 set. 2019.
24. HAN, Run. ***Predicting oral disintegrating tablet formulations by neural network techniques***, 2018. Artigo Científico (publicado no Asian Journal of Pharmaceutical Sciences 13) – Institute of Chinese Medical Sciences (ICMS), University of Macau, Macau, China.
25. BERNERS-LEE, Tim. ***WWW: past, present, and future***, 1996. Publicação acadêmica – Computer, vol. 29, no. 10, pp. 69-77.
26. LONGMAN, Addison Wesley. ***A History of HTML***, 1998**.** <https://www.w3.org/People/Raggett/book4/ch02.html> Acesso em 27 set. 2019.
27. BERNERS-LEE, Tim. ***Information Management: A Proposal***, 1989. <https://www.w3.org/History/1989/proposal.html> Acesso em 27 set. 2019.
28. WORLD WIDE WEB CONSORTIUM, W3C. **Definições e recomendações do HTML 5.2 para conteúdos embutidos**. <https://www.w3.org/TR/html52/semantics-embedded-content.html> Acesso em 27 set. 2019.
29. CAILLIAU, Robert; ASHMAN, Helen. ***Hypertext in the Web – a History***, 1999. Artigo científico – Laboratório Europeu de Partículas Físicas, Suíça e Universidade de Nottingham, Reino Unido.
30. TO-DO: \*MISSING REFERENCE\*
31. FRATERNALI, Piero; ROSSI, Gustavo; SÁNCHEZ-FIGUEROA, Fernando. ***Rich Internet Applications***, 2010. Publicação acadêmica – IEEE Internet Computing, vol. 14, no. 3, pp. 9-12.
32. Muhammad Sajjad Kabir, Syed. **Basic Guidelines for Research: An Introductory Approach for All Disciplines**, Edition: First, Chapter: 9, Publisher: Book Zone Publication, Chittagong-4203, Bangladesh, pp.201-275.
33. **Instituto Brasileiro de Geografia** <<https://censo2010.ibge.gov.br/materiais/guia-do-censo/questionarios.html>> Acesso em 17 de outubro de 2019.
34. Zozus, Meredith Nahm. **The data book collection and management of research data-Chapman and Hall CRC Press**, 2017, p. 36.
35. TO DO: FALTA REFERÊNCIA, DADOS ESTRUTURADOS, SEMI ESTRUTURADOS, NÃO ESTRUTURADOS.
36. **Google Chrome** <[https://www.google.com/chrome](https://www.google.com/chrome/)> Acesso em 17 de outubro de 2019.
37. **Mozilla Firefox** <[https://www.mozilla.org/en-US/firefox/new](https://www.mozilla.org/en-US/firefox/new/)> Acesso em 17 de outubro de 2019.
38. **Microsoft Intenet Explorer** <<https://www.microsoft.com/pt-br/download/internet-explorer.aspx>> Acesso em 17 de outubro de 2019.
39. Aho, Alfred V.; Ullman, Jeffrey D. **Foundations of Computer Science**, 1992, capítulo 10 - Patterns, and Regular Expressions.
40. **XML Path Language (XPath) 3.0** <<https://www.w3.org/TR/xpath-30/>>
41. Pandas. **Biblioteca Python para análise de dados**. <<https://pandas.pydata.org/>> Acesso em 27 nov. 2019.
42. *Item Exporters*. **Documentação do *framework* Scrapy**. <<https://docs.scrapy.org/en/latest/topics/exporters.html>> Acesso em 27 nov. 2019.
43. *Spiders*. **Documentação do *framework* Scrapy**. <<https://docs.scrapy.org/en/latest/topics/spiders.html>> Acesso em 28 nov. 2019.
44. **JSON - JavaScript Object Notation.** < <http://www.json.org/>> Acesso em 21 de novembro de 2019.
45. **Merckmillipore.** <<http://merckmillipore.com/>> Acesso em 21 de novembro de 2019.
46. **SCOPUS - Abstract and citation database of peer-reviewed literature: scientific journals, books and conference proceedings.** <<https://www.scopus.com/>> Acesso em 21 de novembro de 2019.
47. **Scrapy Autothrotle extension.** <<https://docs.scrapy.org/en/latest/topics/autothrottle.html>> Acesso em 21 de novembro de 2019.
48. **Seletores CSS.** <<https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors>> Acesso em 21 de novembro de 2019.
49. **XML.** <<https://www.w3.org/XML/>> Acesso em 21 de novembro de 2019.
50. **Scopus Search API.** <<https://dev.elsevier.com/documentation/ScopusSearchAPI.wadl>> Acesso em 21 de novembro de 2019.
51. **Biblioteca Python json.** <<https://docs.python.org/3/library/json.html>> Acesso em 21 de novembro de 2019.
52. **DOI - Digital Object Identifier.** <<https://www.doi.org/>> Acesso em 21 de novembro de 2019.

ANEXOS

A função das duas definições, Anexo e Apêndice, é semelhante, mas com uma grande diferença entre elas: a autoria. O **ANEXO** de um trabalho acadêmico deve ser aquele texto ou documento que **não foi elaborado por você**, tendo como objetivo servir de legitimação. Já o **APÊNDICE** se configura como texto ou documento **elaborado por você**, tendo como objetivo complementar a sua argumentação.

ANEXO A – TÍTULO DO ANEXO A

ANEXO B – TÍTULO DO ANEXO B