UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E ELETRÔNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELETRÔNICA

Marina Rocha Guimarães

BUSCADOR DE COMPONENTES ELETRÔNICOS EM WEBSITES

Florianópolis
2022

Marina Rocha Guimarães

BUSCADOR DE COMPONENTES ELETRÔNICOS EM WEBSITES

Trabalho de Conclusão de Curso de Graduação em Engenharia Eletrônica do Centro Tecnológico da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Engenharia Eletrônica.
Orientador: Prof. Richard Demo Souza, Dr.

Florianópolis
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Guimarães, Marina Rocha
Buscador de componentes eletrônicos em websites / Marina Rocha Guimarães ; orientador, Richard Demo Souza, 2022.
46 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Eletrônica, Florianópolis, 2022.

Inclui referências.

1. Engenharia Eletrônica. 2. busca por componentes eletrônicos. 3. crawler. 4. scraper. 5. sistema de recomendação. I. Souza, Richard Demo. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Eletrônica. III. Título.

Marina Rocha Guimarães

BUSCADOR DE COMPONENTES ELETRÔNICOS EM WEBSITES

Este Trabalho Conclusão de Curso foi julgado adequado para obtenção do Título de Bacharel em Engenharia Eletrônica e aprovado em sua forma final pelo Curso de Engenharia Eletrônica.

Florianópolis, 09 de março de 2022.

Documento assinado digitalmente
Fernando Rangel de Sousa
Data: 16/03/2022 22:38:47-0300
CPF: 884.649.114-91
Verifique as assinaturas em https://v.ufsc.br

Prof. Fernando Rangel de Sousa, Dr.
Coordenador do Curso

Banca Examinadora:

Documento assinado digitalmente
Richard Demo Souza
Data: 16/03/2022 18:58:25-0300
CPF: 004.267.379-89
Verifique as assinaturas em https://v.ufsc.br

Prof. Richard Demo Souza, Dr.
Orientador
Universidade Federal de Santa Catarina

Documento assinado digitalmente
MARIO DE NORONHA NETO
Data: 16/03/2022 14:01:33-0300
CPF: 003.859.519-22
Verifique as assinaturas em https://v.ifsc.edu.br

Prof. Mario de Noronha Neto, Dr.
Avaliador
Instituto Federal de Santa Catarina

Documento assinado digitalmente
EDUARDO LUIZ ORTIZ BATISTA
Data: 16/03/2022 19:10:04-0300
CPF: 036.521.889-85
Verifique as assinaturas em https://v.ufsc.br

Prof. Eduardo Luiz Ortiz Batista, Dr.
Avaliador
Universidade Federal de Santa Catarina

AGRADECIMENTOS

À minha mãe, Vera Guimarães, que me incentivou e me apoiou para que eu pudesse dar o melhor de mim durante não apenas a elaboração deste projeto, mas também durante toda a graduação.

Ao meu pai, Zeloir Guimarães, minha maior fonte de inspiração.

Aos meus irmãos e cunhados, Thaís, Renata, Renan, Viviane e Paulo que me aconselharam todas as vezes que precisei.

Ao meu marido, Luiz Gustavo Abou Hatem, que esteve pacientemente presente em todos os momentos mais difíceis da elaboração deste projeto, sempre me motivando a seguir em frente.

À Katusha, minha cadela, que me acompanhou em todas as manhãs dos finais de semana que passei desenvolvendo este projeto e eventualmente me fazendo dar pausas necessárias.

Aos meus amigos e colegas de trabalho que sempre acreditaram em mim e me incentivaram a finalizar o projeto.

Ao Professor Richard Demo Souza que me inspirou durante a graduação e me auxiliou na elaboração deste trabalho.

"O trabalho duro é inútil para quem não acredita em si mesmo."
(Naruto Uzumaki)

RESUMO

Este trabalho apresenta o processo de criação de um extrator de dados de lojas que vendem componentes eletrônicos na modalidade *on-line* através da criação de *crawlers* e *scrapers* utilizando, para isso, a ferramenta Scrapy do Python. Além disso, o trabalho também mostra a criação do front-end e do back-end de um *website* (http://ebusca.link) que, além de disponibilizar uma ferramenta de busca que mostra os resultados obtidos com os *crawlers*, também apresenta um sistema de recomendação o qual, por sua vez, pode ser dividido em um recomendador principal (por popularidade) e em um recomendador secundário (por um modelo de *clustering*).

Palavras-chave: crawler, scraper, busca por componentes eletrônicos, sistema de recomendação.

ABSTRACT

This work presents the process of creating a data extractor for stores that sell electronic components in the online modality through the creation of crawlers and scrapers using the Scrapy tool from Python. In addition, the work also shows the creation of the front-end and back-end of a website (http://ebusca.link) that provides a search tool that shows the results obtained with the crawlers, as well as presents a recommendation system which, in turn, can be divided into a main recommender (by popularity) and a secondary recommender (by a clustering model).

Keywords: crawler, scraper, search for electronic components, recommendation system.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Durante os cursos de Engenharia Eletrônica e Elétrica na Universidade Federal de Santa Catarina os alunos precisam comprar componentes eletrônicos para a realização de diversos projetos acadêmicos ou pessoais e, uma forma de realizar essa compra, é através de lojas que vendem produtos na modalidade *on-line*. Um passo importante na compra desses componentes é a busca em diversos sites para entender qual possui os produtos desejados e, além disso, qual possui o melhor custo benefício.

Tendo isso em vista, nota-se a importância de uma ferramenta de busca acessível na qual os alunos - principalmente aqueles sem experiência na compra de componentes eletrônicos - possam pesquisar em um só lugar todos os componentes que desejam comprar e, assim, entender de uma forma menos trabalhosa qual o melhor *website* para realizar seu pedido online.

1.1 OBJETIVOS

O objetivo principal deste trabalho é a criação do processo de extração de dados de lojas *on-line* que vendem componentes eletrônicos para posterior disponibilização em um *website*, o qual terá, além de uma ferramenta de busca capaz de mostrar esses resultados, um sistema de recomendação de produtos de acordo com o termo buscado.

Os objetivos específicos identificados que guiam a elaboração deste trabalho são:

a) Criação de *crawlers* e *scrapers* para extração de dados de *websites* de lojas que vendem componentes eletrônicos na modalidade *on-line*;

b) Automatização do processo de extração de dados;

c) Criação do *website* que irá apresentar a ferramenta de busca para o usuário;

d) Implementação de toda a estrutura para acesso remoto por parte dos usuários;

e) Testes iniciais com a ferramenta a fim de ingerir dados de buscas dos usuários;

f) Criação do sistema de recomendação com base nos dados ingeridos nos testes iniciais do *website*.

1.2 ESTRUTURA DO TRABALHO

Este trabalho é dividido em cinco capítulos: o primeiro contém a introdução, enquanto o segundo apresenta toda a fundamentação teórica necessária para compreender os conceitos citados no trabalho. Já o terceiro capítulo explica todas as etapas do desenvolvimento do projeto, ao passo que o quarto mostra os resultados obtidos com o trabalho. Por fim, o quinto capítulo apresenta a conclusão final do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação teórica do presente trabalho, envolvendo uma visão geral acerca das ferramentas e técnicas utilizadas no desenvolvimento do mesmo.

2.1 WEB CRAWLING E WEB SCRAPING

Nos últimos anos o número de usuários da *internet* cresceu consideravelmente - de acordo com um estudo produzido em janeiro de 2021 pela empresa We Are Social, 4.66 bilhões de pessoas no mundo utilizam a internet (KEMP, 2021). Com o aumento de usuários cresce também a quantidade de informação disponível no meio *on-line* e, desse modo, é compreensível que ocorra a criação de ferramentas e técnicas para facilitar a extração dessas informações.

Uma técnica bastante utilizada é o *web crawling* que pode ser definida como "*a program/software or programmed script that browses the World Wide Web in a systematic, automated manner. (...) By following the linked structure of the Web, web crawler may traverse several new web pages starting from a webpage.*" (MD. ABU KAUSAR V. S. DHAKA, 2013). Para encontrar as informações relevantes, o algoritmo começará com uma certa página da *web* e irá então rastrear os próximos *links* que irão levar a outras páginas. Utilizando esses *links* e chegando em novas páginas *web*, o processo vai se repetindo e os *crawlers* encontram várias informações (provenientes de diversas páginas). Essas informações têm, por sua vez, o potencial de responder às perguntas que deram origem à criação do algoritmo.

Um ponto importante é que o *crawler* captura todas as informações encontradas na página em questão, ou seja, ele não é capaz de selecionar informações específicas do *website*. Para esse fim, é utilizado uma técnica conhecida como *web scraping*, a qual é capaz de extrair informações específicas do HTML da página encontrada pelo *web crawler* de forma automatizada. Para isso, o *scraper* utiliza ferramentas denominadas seletores de dados - alguns exemplos são o XPath (*XML Path Language*)[1], seletores CSS[2] e RegEx (*Regular Expressions*)[3]. No atual trabalho foi aplicado uma combinação de XPath com RegEx, afim de facilitar o processo de extração.

Vale ressaltar que o *crawler* é um mecanismo de busca que tenta se aproximar da pesquisa realizada por um usuário em um navegador de internet. Assim, algumas definições precisam ser passadas para esse *script*, como o *user agent* que será utilizado (é a identificação que o navegador passa para os sites para que eles saibam qual suporte ou *layout* adequado devem retornar), o método da requisição (*GET*, *POST*, etc), informações

[1] O XPath é uma linguagem de consulta para selecionar nós de um documento XML.
[2] Os Seletores definem quais elementos um conjunto de regras CSS se aplica.
[3] As expressões regulares proveem uma forma concisa e flexível de identificar cadeias de caracteres de interesse em um texto.

de dispositivo e, quando necessário, configurações de *proxy* - sendo este um serviço que atua como intermediário entre o cliente que solicita um recurso e o servidor que o fornece, podendo mascarar o IP (*Internet Protocol*) do cliente quando necessário.

Além disso, existe um importante conceito a ser citado quando se fala de *crawler* que é o *sitemap*: "*A sitemap is a file where you provide information about the pages, videos, and other files on your site, and the relationships between them. (...) A sitemap tells Google which pages and files you think are important in your site, and also provides valuable information about these files*" (GOOGLE, 2021). Isto é, existem *websites* que possuem esse arquivo denominado *sitemap* no qual são definidos os *links* que direcionam para as páginas específicas no site. Quando um *website* disponibiliza esse arquivo, o *crawler* é mais simplificado, já que os *links* para as páginas são mais acessíveis.

Atualmente existem diversas maneiras de se criar um *script* com a finalidade de busca e extração de dados da web, uma delas é através da biblioteca Scrapy escrita em Python.

2.1.1 Scrapy

Para a realização do presente trabalho, escolheu-se a ferramenta Scrapy para o desenvolvimento dos *crawlers* e *scrapers* utilizados. Essa escolha foi baseada em alguns fatores como a robustez da ferramenta - ela foi citada no livro Learning Scrapy (KOUZIS-LOUKAS, 2016) como "*a robust web framework for scraping data from various sources*"; o fato dela ser escrita em Python - linguagem principal utilizada no desenvolvimento do trabalho em questão; e a vasta documentação existente - tanto oficial quanto não oficial.

Vale ressaltar que o principal motivo para existir uma boa documentação sobre essa biblioteca é o fato de ela ser *open source*, isto é, qualquer pessoa pode acessar o código fonte, vê-lo, modificá-lo e distribuí-lo conforme ache necessário.

2.2 BANCOS DE DADOS

De acordo com o livro Introdução a Sistemas de Bancos de Dados (DATE, 2004) "um banco de dados é (...) um sistema computadorizado cuja finalidade geral é armazenar informações e permitir que os usuários busquem e atualizem essas informações quando as solicitar". Isto é, além de ser um tipo de armazenamento centralizado de informações, ele também disponibiliza para os usuários a possibilidade de atualizar, inserir ou deletar os dados presentes ali. Atualmente, é muito comum em empresas e projetos pessoais existirem diversas informações que precisam ser salvas de uma forma organizada e acessível para uma consulta posterior - daí a importância do banco de dados.

Existem alguns tipos de banco de dados e um deles é o banco de dados relacional o qual é utilizado no presente trabalho. Esse tipo de banco possui tabelas que se relacionam através de uma ou mais colunas, ou seja, existem diferentes informações em tabelas que

se complementam de certa forma. Um bom exemplo disso é a utilização (no comércio) de uma tabela para armazenar informações de vendas (data da venda, preço e quantidade do produto vendido) e de outra para as informações de cada produto existente (nome, fornecedor, preço de custo, etc). Além disso, em um banco de dados relacional as tabelas possuem sua estrutura bem definida, ou seja, todas as linhas de uma tabela possuem as mesmas colunas com os mesmos tipos de dados para cada uma delas.

Vale ressaltar que a comunicação entre o usuário e o banco de dados relacional é realizada com consultas em SQL (*Structured Query Language*) através de um SGBDR (Sistema de Gerenciamento de Banco de Dados Relacional) ou RDBMS (*Relational Data Base Management System*) em inglês. Esse sistema é formado por um conjunto de algoritmos e é responsável pela manipulação das informações existentes no banco de dados. Ou seja, o SGBDR é o que possibilita a requisição por parte do usuário da criação, modificação ou eliminação de tabelas e dados dentro do banco. Tendo isso em vista, vale dizer que existem diversos tipos de sistemas de gerenciamento como Oracle, MySQL, SQL Server, PostgreSQL dentre outros. Para o atual trabalho escolheu-se utilizar o MySQL, sendo o principal motivador dessa escolha a facilidade de lidar com esse sistema.

Um ponto interessante desse sistema de gerenciamento de banco de dados é que ele permite que arquivos CSVs sejam inseridos em tabelas, caso esses arquivos estejam de acordo com o esquema definido para a tabela em questão. Além disso, uma das formas para que o MySQL aceite essa operação é o arquivo estar localmente armazenado em */var/lib/mysql-files/* - que é uma pasta padrão criada por esse gerenciador no sistema operacional Linux.

2.3 INSTÂNCIA AWS

Atualmente existem diversos serviços de computação em nuvem, como a AWS (Amazon Web Services), Microsoft Azure e GCP (Google Cloud Platform). Cada um deles possui seus pontos fortes e fracos, mas no geral eles são similares - já que todos possuem o objetivo de disponibilizar uma grande variedade de serviços em nuvem. No entanto, a AWS é o serviço de computação em nuvem mais utilizado atualmente (CHAND, 2021).

Dentre os serviços disponibilizado pela AWS está o EC2, o qual possibilita que o usuário alugue uma máquina (conhecida na AWS como instância) a fim de rodar suas aplicações. A maior vantagem do EC2 é que existem diversos tipos de instâncias, fazendo com que o usuário consiga escolher a melhor máquina com relação ao seu custo benefício para seu caso específico. Além disso, as configurações das instâncias são bem flexíveis, indo desde escolhas do processador, até escolhas de armazenamento e sistema operacional. Um outro ponto interessante é que o pagamento do aluguel de uma EC2 é sob demanda, isto é, o usuário deve pagar a máquina pelo tempo que usar, podendo desligá-la sempre que quiser.

Vale ressaltar que normalmente os serviços que rodam em uma EC2 provém de *scripts* existentes em algum repositório do GitHub. A fim de realizar o *deploy* (colocar em produção alguma aplicação que teve seu desenvolvimento concluído), existem ferramentas de CI/CD (*Continuous Integration / Continuous Deployment*) que realizam esse processo automaticamente toda vez que ocorre uma mudança na *master* (ou qualquer *branch* configurada) do repositório em questão. Um exemplo de ferramenta que é bastante utilizada para automatizar esse processo é o CircleCI.

Um outro ponto interessante é que, para acessar uma instância EC2 do computador local, pode-se utilizar o protocolo SSH (*Secure Shell*) - basta ter os acessos necessários. O SSH é um protocolo de rede criptográfico muito utilizado em processos de operação de serviços de rede.

2.4 APACHE AIRFLOW

Durante muito tempo utilizou-se a ferramenta do Linux denominada Cron para programar a execução de comandos e processos de forma rotineira. Essa ferramenta possui alguns pontos positivos, como sua simplicidade, facilidade para utilizar e o fato de não precisar de uma instalação - basta criar um arquivo com o *scheduler* das tarefas desejadas.

No entanto, recentemente o Cron foi sendo substituído por uma ferramenta denominada Apache Airflow, a qual foi criada pela empresa Airbnb e logo se tornou um projeto *open source*. Um dos grandes motivos dessa substituição foi a simples interface do Airflow que facilita a localização de *logs*, a reexecução de tarefas que falharam e a organização do projeto no geral. Ou seja, mesmo que inicialmente esta ferramenta seja mais complexa que o Cron (já que precisa ser instalada e configurada), a visibilidade e facilidade disponibilizada por sua interface gráfica supera os pontos negativos.

Outro ponto interessante é a definição de DAGs dentro do Airflow, sendo "*A DAG (Directed Acyclic Graph) is the core concept of Airflow, collecting Tasks together, organized with dependencies and relationships to say how they should run*" (FOUNDATION, 2022). Ou seja, uma DAG é um *script* que disponibiliza as informações acerca das tarefas que serão executadas, informando também a ordem em que isso ocorrerá e qual a relação entre elas. Por exemplo, se existe uma tarefa de baixar dados de um local para depois inseri-los em um outro lugar, a tarefa de realizar o *upload* de arquivos depende que a tarefa extração de dados tenha sido concluída com sucesso.

Esse conceito de DAG é muito interessante, já que assim pode-se criar uma DAG diferente para cada *crawler* específico, definindo as tarefas a serem executadas para cada um deles e suas dependências. Vale ressaltar que o Apache Airflow ainda apresenta um outro ponto posivito: ele permite a configuração do envio de alertas para quando alguma tarefa dentro da DAG falhe. Esse alerta pode ser enviado por *e-mail*, SMS, ligação telefônica dentre outros - basta ser configurado.

2.5 FRONT-END E BACK-END

Um *website* (conjunto de hipertextos acessíveis geralmente pelo protocolo HTTP ou HTTPS na internet) pode ser dividido em duas principais partes: o *front-end* e o *back-end*. É comum em empresas existirem profissionais diferentes para cada uma dessas duas áreas, sendo essa divisão pontuada por Tomasevic e Pavicevic como: "*the front-end development deals with the presentation of the web content to the users via their web browser. (...) Front-end developers task is to turn designer ideas into a functional website. (...) The back-end development of a website deals with communicating with the front-end side by sending and receiving data, organising and storing data, security and all the other aspects that user can not see*" (D. TOMASEVIC, 2021).

Vale ressaltar que o *back-end* geralmente é desenvolvido utilizando uma ou mais APIs (*Application Programming Interface*), que são um conjunto de rotinas e padrões que facilitam a comunicação entre sistemas. Em um *website*, uma API permite que a interface do *front-end* (parte visível para o usuário) possa se comunicar com o banco de dados, buscando e inserindo informações quando necessário. Essa comunicação acontece através de requisições HTTP que podem ser de diversos tipos como GET (usada para receber dados, sem modificar nenhuma informação no banco de dados)e POST (usada para modificar informações no banco).

No presente trabalho, o *front-end* foi desenvolvido com a linguagem de programação JavaScript e com a *framework* denominada Vue (ferramenta *open source* do JavaScript utilizada para auxiliar no desenvolvimento de interfaces de usuário), ao passo que o *back-end* foi desenvolvido com Python (mesma linguagem do restante do projeto) utilizando a *framework* denominada *FastAPI*. A escolha dessas duas linguagens teve como maior incentivador seus usos no mercado atual e a facilidade de utilização de ambas.

2.6 SISTEMA DE RECOMENDAÇÃO

Sistemas de recomendação são subclasses do sistema de filtragem de informações que busca prever a classificação ou preferência que o usuário daria a um item (SEN, 2020). Pode-se criar um sistema de recomendação de diversas formas, dentre elas existem os sistemas baseados em popularidade e os sistemas baseados em um modelo de *clustering*.

2.6.1 Sistema de Recomendação por Popularidade

Um sistema de recomendação por popularidade, como o nome sugere, funciona com a tendência do momento (SEN, 2020). Esse tipo de sistema de recomendação pode ser considerado o mais simples dentre os sistemas existentes, já que se se baseia em cálculos realizados através de alguma métrica de retorno obtida com a utilização do usuário. Um exemplo disso - que acontece em *ecommerces* - é o sistema recomendar itens a partir da

nota de avaliação deixada pelos usuários ponderado pelo número de vezes que o item foi pesquisado e avaliado.

2.6.2 Clustering

Esse tipo de modelo não supervisionado de *machine learning*, denominado *clustering*, é muito utilizado em sistemas de recomendação, já que, "*unlike supervised learning (like predictive modeling), clustering algorithms only interpret the input data and find natural groups or clusters in feature space*" (BROWNLEE, 2020). Essa capacidade é exatamente o que um recomendador busca: conseguir identificar o grupo ao qual o item pesquisado pelo usuário pertence, para então conseguir oferecer boas sugestões.

Um exemplo de modelo de *clustering* utilizado atualmente é o *K-means*, no qual define-se o número de *clusters* e, de início, o algoritmo define aleatoriamente um centroide para cada deles. Após isso, calcula-se para cada ponto existente o centroide de menor distância e refaz-se os grupos em torno desses centroides, sendo que cada ponto pertencerá ao *cluster* do centroide mais próximo. Depois, os centroides são reposicionados a fim de ficarem na posição que possui a distância média de todos os pontos do seu *cluster* e todo o processo é repetido até nenhum ponto precisar mudar novamente de grupo. O processo de *clustering K-means* descrito pode ser observado na Figura 1 abaixo.

Um ponto importante é o tipo de *feature* utilizada como entrada para o modelo *K-means* e uma boa escolha é o *TF-IDF* (term frequency-inverse document frequency) já que é "*a numerical statistic method which allows the determination of weight for each term (or word) in each document. (...) The method determines the weight, measure which evaluates importance of terms (or words) in document collection*" (B. TRSTENJAKA S. MIKACB, 2014). Ou seja, ao utilizar o *TF-IDF* em uma descrição de produto, teremos as palavras consideradas mais importantes em relação não apenas a esse texto, mas também em relação a todos os demais.

No presente trabalho foi utilizada o *TF-IDF* da biblioteca *scikit-learning* do Python e sua fórmula matemática é:

$$F_{\text{tfidf}}(t, d) = F_{\text{tf}}(t, d) * F_{\text{idf}}(t), \tag{1}$$

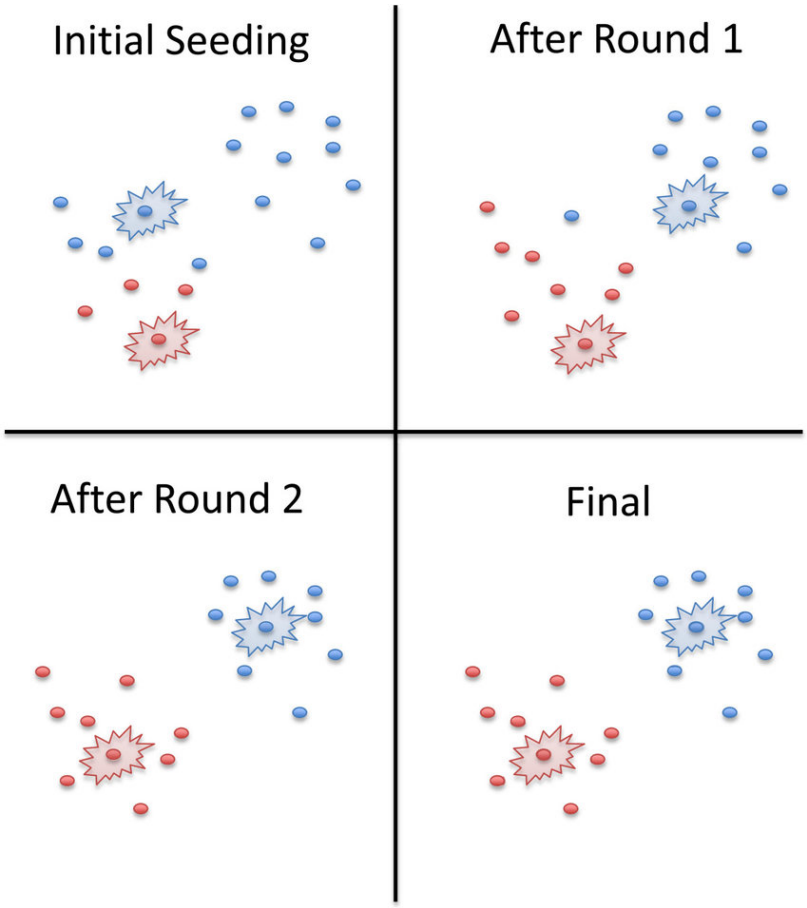
sendo

$$F_{\text{tf}}(t, d) = \text{freq}(t, d), \tag{2}$$

$$F_{\text{idf}}(t, D) = \log\left(\frac{N}{df(t) + 1}\right) \tag{3}$$

Em que t é a palavra, d é o texto, D é o conjunto dos textos, N é o número de textos do conjunto analisado e $df(t)$ é a frequência do documento de t, isto é, o número de documentos no conjunto de documentos que contêm a palavra t.

Figura 1 – Processo de *clustering K-means*.



Fonte: **PAGE**; **JUSTIN**; **LIECHTY**; **ZACH**; **HUYNH**; **MARK e UDALL (2014**; **p. 829)**

3 DESENVOLVIMENTO

Neste capítulo é apresentada a elaboração do projeto como um todo, dividido por tópicos. Vale ressaltar que todos os códigos citados no texto estão armazenados em repositórios do GitHub no perfil *Marinarg*. São eles:

- my_crawlers;
- airflow;
- search_eletronics_website;
- search_eletronics_api.

3.1 EXTRAÇÃO DE DADOS

A etapa inicial do presente trabalho foi a escolha da extração de dados através de *crawlers* e *scrapers*. Um ponto importante sobre *crawlers* é que eles possuem bastante relação com o *website* em que buscam dados, ou seja, é bem provável que o *crawler* que funcione em um site não funcione em outro.

A ideia inicial era ter um número de *crawlers* próximo de cinco e, para isso, iniciou-se a criação desse *script* para seis *websites* diferentes, sendo eles: Baú da Eletrônica, FilipeFlop, SoldaFria, TiggerCOMP, Digi-Key e Mouser. No entanto, os *websites* Digi-Key e Mouser apresentaram dificuldades significativas com relação ao barramento de *bots*. Ou seja, com a criação dos *crawlers*, percebeu-se que esses sites começavam a bloquear as *requests* realizadas pelo *script* após a aquisição de um certo número de dados retornando alguns *status* de erro como o 429 (*Too Many Requests*). Esse problema tentou ser resolvido rotacionando o *user agent*, ou seja, utilizando diferentes *user agents* para diferentes *requests*, mas os *websites* continuaram a retornar erro.

O próximo passo para testar seria utilizar um serviço de *proxy*, já que com ele o IP utilizado poderia ser rotacionado tentando, assim, diminuir o número de retornos de erros. No entanto, como apenas esses dois *websites* apresentaram esse problema, propôs-se que para a primeira versão do presente trabalho iria ser utilizado apenas aqueles outros quatro *crawlers* citados e estes dois seriam deixados como uma melhoria para a próxima versão do projeto.

Cada um dos quatro *crawlers* criados e utilizados foi desenvolvido separadamente utilizando a ferramenta Scrapy. No entanto, eles podem ser divididos em dois principais grupos com relação ao seu processo de criação: os *crawlers* que usam *sitemap* e os que não usam, sendo a principal diferença entre eles o fato que o *sitemap* já disponibiliza todos os *links* que levam diretamente às páginas de produtos, ao passo que, quando esse arquivo não existe, deve-se criar um algoritmo que percorra todo o caminho que um usuário faria para acessar um produto (para todos os produtos do site).

3.1.1 Crawlers com Sitemap

O único *website* que possuía um *sitemap* atualizado no momento de criação dos *scripts* era o SoldaFria. Vale ressaltar que o *sitemap* pode ser encontrado na parte inferior da página resultante da ação de adicionar o texto *robots.txt* ao final da URL do *website* em questão (por exemplo https://www.soldafria.com.br/robots.txt).

A criação desse *crawler* iniciou-se com a busca pelos *links* relevantes no arquivo *sitemap* através de regras estabelecidas após analisar o arquivo. Por exemplo, observou-se que os *links* que possuíam a palavra *blog* na URL não levavam a páginas de produtos - então esses *links* foram desconsiderados.

Após encontrar as páginas de produto, iniciou-se a criação do *scrapper* para extrair as informações necessárias do produto como nome, preço, descrição, imagem, moeda de compra, se estava disponível em estoque, dentre outras. Percebeu-se que a maioria dessas informações poderiam ser extraídas com um mesmo padrão de XPath adicionado a um RegEx - então criou-se uma função principal que realiza essa busca para ser utilizada - com apenas alguns ajustes - para cada tipo de informação necessária.

3.1.2 Crawlers sem Sitemap

Os outros três *crawlers* (Baú da Eletrônica, FilipeFlop e TiggerCOMP) não possuíam o arquivo *sitemap* disponível, então eles seguiram outro fluxo de criação. O primeiro passo foi buscar através da página inicial os *links* que levavam a todas as categorias existentes dentro de cada site utilizando, para isso, lógicas de XPath e RegEx no HTML de retorno. Esse passo é similar ao usuário entrar no site e clicar no nome de uma categoria para então encontrar a lista dos produtos que pertencem a ela - e fazer isso para todas as categorias existentes.

Após isso, criou-se uma outra lógica para encontrar as páginas de produtos dentro de cada categoria e, um detalhe importante nesse passo, é que em algumas categorias existem mais de uma página de resultados de produto, ou seja, existe paginação. Para isso, teve que ser adicionado no *script* uma lógica para checar se existia paginação a cada nova página de categoria encontrada e, em caso afirmativo, extrair os produtos das próximas páginas também. Por fim, extraiu-se os dados de cada página de produto utilizando uma estrutura semelhante àquela criada no *scrapper* do *website* SoldaFria.

Uma outra informação interessante de ser buscada são os valores de frete para cada um desses *websites*, mas tanto o site SoldaFria quanto o TiggerCOMP pediam um *login* do usuário antes de mostrar esses valores, impossibilitando assim a aquisição desses dados pelo *crawler*. Já para os *websites* FilipeFlop e Baú da Eletrônica, encontrou-se uma URL específica que retornava essas informações como se o usuário estivesse finalizando uma compra, ou seja, não era necessário realizar um *login* bastava apenas passar algumas informações como CEP, código de produto que seria comprado e quantidade desse produto.

Criou-se então outros dois *crawlers* para a extração dos dados de frete dos sites FilipeFlop e Bau da Eletrônica, utilizando para isso o CEP da Universidade Federal de Santa Catarina e o código de um produto qualquer.

3.2 EXTRAÇÃO AUTOMATIZADA DOS DADOS

Um ponto importante a ser comentado é que os *crawlers* são apenas *scripts* que possuem a lógica de extração de dados, mas é necessário ativar essa lógica (de forma manual ou automatizada) para que ela realmente busque essas informações. Normalmente, quando o *script* precisa ser rodado apenas esporadicamente, esse processo ocorre através de comandos via terminal. No entanto, quando torna-se necessário adquirir esses dados com uma certa periodicidade - seja diária, semanal ou mensal - esse processo costuma acontecer através de uma estrutura que consiga programar a ativação do *crawler* para as datas desejadas. Visto isso, foi escolhido o orquestrador de fluxos Apache Airflow.

A fim de utilizar a ferramenta, algumas configurações precisaram ser realizadas como a definição do local em que as DAGs ficam armazenadas e a criação de um banco de dados que guarda tanto o histórico da execução das DAGs quanto os dados resultantes da execução dos *crawlers* - escolheu-se utilizar o sistema de gerenciamento de banco de dados MySQL para ambos os casos. Um ponto importante é que para o *back-end* do Airflow apenas a criação do banco de dados foi suficiente, mas para o armazenamento dos dados dos *crawlers* teve que ser definido e criado os esquemas das tabelas que serviriam para esse propósito. Isso porque o Airflow já possui em sua instalação comandos que criam as tabelas necessárias automaticamente, sem que o usuário tenha que se preocupar com isso. Visto que os *crawlers* de extração de dados gerais produzem uma saída diferente dos *crawlers* de pesquisa de informações de frete, criou-se duas tabelas diferentes no banco de dados para receber essas informações. A tabela que armazena os dados gerais possui os seguintes campos:

- product name;
- product description;
- product labels;
- product id;
- product price;
- product image;
- product url;
- currency iso;

- currency symbol;
- in stock;
- execution date;
- website domain;
- website url.

Já a tabela que armazena as informações de frete possui os seguintes campos:

- response;
- execution date;
- website domain;
- website url.

Vale notar que o campo *response* na tabela de dados de frete possui todo o resultado extraído da página de retorno após a requisição ser realizada, ou seja, não existe um *scrapper* para esses *crawlers*. Além disso, as duas tabelas citadas acima se relacionam através do campo *website domain*, o qual se refere ao nome do *website* em questão como, por exemplo, "filipeflop".

Após a instalação e configurações iniciais do Airflow e do banco de dados, criou-se uma DAG para cada *crawler* - incluindo os *crawlers* de extração de dados de frete. Cada DAG possui a tarefa de rodar o *script* do *crawler*, salvar o resultado em um arquivo CSV, mover esse arquivo para a pasta padrão de entrada de dados do MySQL, inserir esses dados em uma tabela específica do banco de dados e deletar o arquivo CSV após ele ser utilizado. Com as DAGs criadas, definiu-se que elas rodariam todos os dias pela manhã e dispariariam um alerta caso alguma delas falhasse, enviando assim uma mensagem para o e-mail do criador do *script*.

3.3 WEBSITE

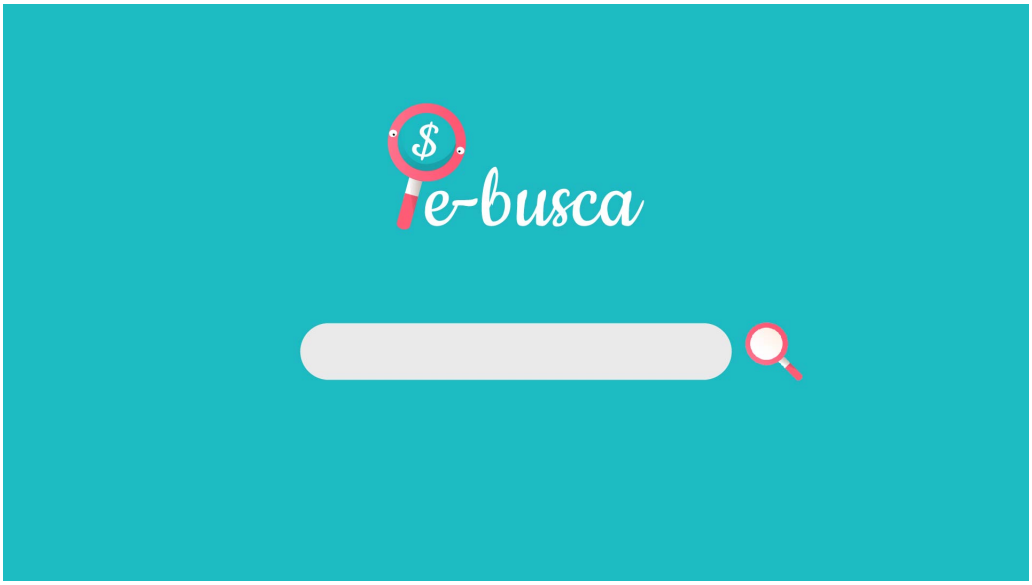
A fim de criar uma boa visualização dos dados extraídos, construiu-se um *website* próprio com as funcionalidades desejadas. Para isso, o desenvolvimento pode ser dividido em duas partes: o *front-end* e o *back-end*.

3.3.1 Front-end

O primeiro passo para a criação do *front-end* foi o desenho do *website*, ou seja, de como ele apresentaria os resultados, quais cores, ícones e fontes utilizaria, dentre outras

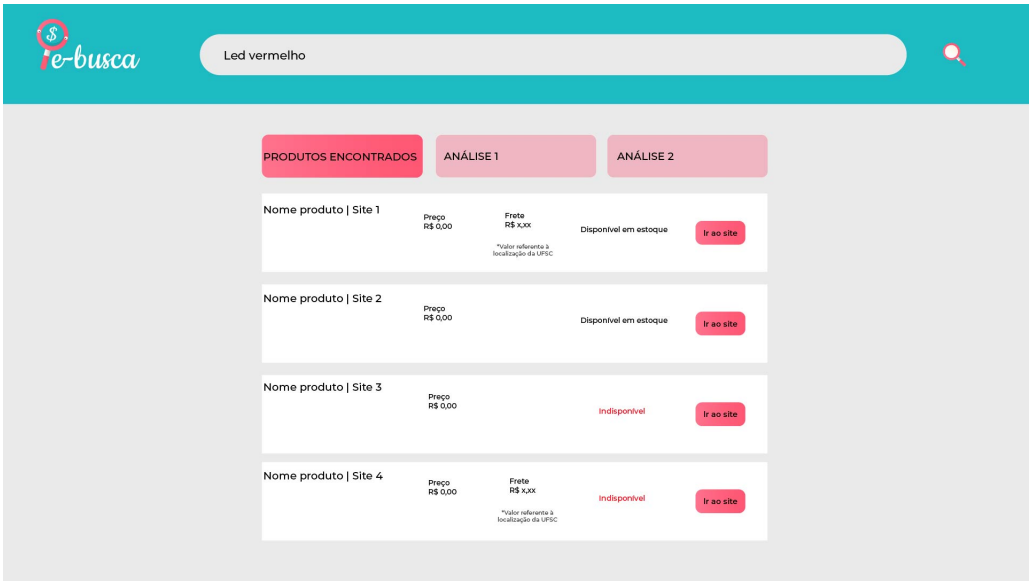
escolhas relacionadas com o *design* do projeto em si. Uma primeira versão foi criada no software de desenhos vetoriais denominado Adobe Illustrator e ela pode ser vista nas Figuras 2 e 3 abaixo.

Figura 2 – Primeira versão do design da página inicial.



Fonte: do Autor.

Figura 3 – Primeira versão do design da página de resultados.



Fonte: do Autor.

Nessa primeira definição, a ideia era possuir um *website* que tivesse uma tela inicial (como mostrado na Figura 2) para o usuário poder realizar sua primeira busca. Após isso,

ele seria redirecionado para a página de resultados (Figura 3) na qual, além de observar os resultados de sua pesquisa, ele poderia inserir novas buscas e ter acesso a algumas análises sobre os resultados. Essas análises foram pensadas em um primeiro momento para adicionar valor na página de resultados, mas não foi decidido exatamente o que teria ali. Após essa primeira definição, começou-se a criar o algoritmo em JavaScript que entregaria o resultado desejado utilizando, para isso, a *framework* Vue.

Vale ressaltar que, após algumas ideias do que seria colocado nas seções de análises, decidiu-se que seria melhor removê-las e adicionar uma seção de recomendações, a qual será explicada posteriormente. Um ponto importante para ser comentado é que, para as recomendações, precisou ser criado um identificador único de usuário no *front-end*, o qual foi criado gerando um número aleatório variando de 0 a 1000 - a utilização desse identificador também será comentada posteriormente.

3.3.2 Back-end

Já a criação do *back-end* foi realizada utilizando a linguagem Python e a *framework FastAPI*, a fim de criar uma API com as rotas necessárias. Em um primeiro momento, criou-se duas rotas: uma para trazer do banco de dados as informações resultantes dos *crawlers* para o termo pesquisado pelo usuário e outra para inserir no banco de dados informações de pesquisas do usuário.

A primeira rota funciona através de uma requisição do tipo GET e utiliza uma lógica de SQL que busca o termo pesquisado pelo usuário na tabela de resultados dos *crawlers*, unindo a isso a informação de frete presente em outra tabela - para os *websites* que possuem essa informação. Nessa lógica, considera-se nomes de produtos que contenham o termo pesquisado, por exemplo, se a busca for “led” e existir um resultado com o nome “led vermelho” ele irá retornar. Além disso, a lógica busca o resultado mais recente que existe para cada um dos *websites* utilizados como base na extração de dados, descartando produtos que não contenham informações obrigatórias, como nome do produto, preço, URL e se ele está disponível em estoque ou não. Esse descarte é importante para não trazer possíveis erros de extração de dados dos *crawlers* para o resultado mostrado no *website*.

A segunda rota, funciona através de uma requisição do tipo POST e também utiliza uma lógica SQL. No entanto, essa lógica tem como objetivo inserir os termos buscados pelo usuário em uma outra tabela, a qual também foi criada utilizando o MySQL e possui os seguintes campos:

- unique id;
- search;
- execution date.

A coluna denominada *unique id* é um id único criado para o usuário no *front-end* (comentado anteriormente) e ele é único para cada dispositivo. Ou seja, se o usuário acessar o site sempre de um mesmo dispositivo ele sempre terá o mesmo identificador - a não ser que ele o apague manualmente do seu armazenamento local. Já a coluna *search* é o termo buscado pelo usuário e o *execution date* é a data da busca.

Vale ressaltar que a tabela citada acima foi utilizada no sistema de recomendação principal, o qual será explicado posteriormente.

3.4 IMPLEMENTAÇÃO DO PROJETO DE FORMA REMOTA

Todo o processo citado anteriormente (rotina e *scripts* de *crawlers* e *website*) foi realizado em ambiente local, sem a possibilidade de acesso externo ao resultado produzido. Então, o próximo passo foi inserir a estrutura criada em um ambiente remoto, o qual pudesse ser acessado externamente pelos usuários através de qualquer dispositivo.

Essa etapa iniciou-se criando uma instância EC2 gratuita na AWS do tipo t3.micro contendo 1 GiB de memória. No entanto, a medida que a estrutura foi sendo montada nessa instância, notou-se que ela não seria o suficiente para suportar o Airflow, banco de dados e o *website* rodando ao mesmo tempo. Decidiu-se então optar para uma instância paga denominada t3a.large, já que ela possui uma memória de 27 GiB. Um outro ponto que vale ser mencionado é que inicialmente testou-se utilizar o sistema operacional Ubuntu pela facilidade trazida por ele, no entanto o sistema operacional Debian é mais indicado para casos em que o usuário deseja rodar uma aplicação em uma máquina alugada, já que o Ubuntu vem com uma variedade de pacotes pré-definidos pelos mantenedores, o que ocupa um certo espaço desnecessário na memória da máquina.

Após essa mudança, todos os repositórios do GitHub foram clonados para a EC2 e foram instalados e configurados o Airflow e o banco de dados - esse processo foi similar ao que foi realizado localmente. Depois, o *back-end* e o *front-end* do *website* foram modificados, já que a conexão entre eles ocorre através do endereço de IP da máquina em que ambos se localizam. Um detalhe importante na configuração do *website* remoto é que foi adquirido um domínio pela AWS para o nome do site fazer sentido com o seu objetivo: configurou-se o domínio ebusca.link, em que a letra “e” é referente a componentes eletrônicos e a palavra “busca” refere-se ao objetivo do site.

Um detalhe importante é que atualmente o *deploy* é manual, ou seja, não existe nenhuma ferramenta de CI/CD presente no processo que inclua na EC2 as mudanças colocadas em produção no repositório do GitHub. Esse processo ocorre através de comandos pelo terminal do Linux dentro da instância EC2 (acessada localmente com conexão SSH) que utilizam os arquivos com extensão *.service* (arquivos que disponibilizam informações sobre como iniciar cada serviço) criados dentro de cada repositório a fim de facilitar o *deploy* na instância da AWS. Vale ressaltar que cada serviço roda em uma porta diferente

da EC2.

3.5 TESTES COM A PRIMEIRA VERSÃO

Com toda a estrutura rodando de forma remota, iniciaram-se os testes do *website* para receber os dados de busca dos usuários. Nesse processo, alguns colegas utilizaram o site e pesquisaram componentes pensando em projetos que já realizaram ou pretendiam realizar. Esse primeiro teste foi interessante também para receber devolutivas sobre possíveis melhorias, tanto de funcionalidades quanto de *design*.

Algumas melhorias que foram implementadas após esse teste:

- Limitação da busca para ter pelo menos dois caracteres e não ser apenas números - antes disso quando um usuário digitava apenas uma letra o buscador retornava todos os nomes de produtos que possuíam essa letra;
- Mudança no *design* do texto "PRODUTOS ENCONTRADOS" - ele parecia ser um botão e na verdade não era;
- Disponibilização do número de produtos encontrados para o termo buscado.

Além dessas melhorias citadas acima, criou-se também o sistema de recomendação que será explicado abaixo.

3.5.1 Sistema de Recomendação

Criou-se dois sistemas de recomendação: um por popularidade e outro por *clustering*. O modelo por popularidade foi criado a partir de uma nova rota na API utilizada no *back-end* do *website*. Essa nova rota funciona através de uma requisição do tipo GET e utiliza uma lógica SQL que, por sua vez, usa a tabela do banco de dados referente aos dados de buscas de usuários - adquiridos no primeiro teste realizado com a disponibilização do *website* para os colegas. Um ponto importante a ser citado é que independente do sistema de recomendação utilizado, ele está relacionado apenas com a busca atual realizada pelo usuário, não utilizando como parâmetros de entrada buscas anteriores realizadas pelo mesmo usuário.

3.5.1.1 Sistema de Recomendação por Popularidade

O sistema de recomendação por popularidade inicia quando o usuário realiza uma busca e aperta o botão "produtos relacionados". Após isso, o *front-end* do *website* envia uma requisição para a API do *back-end* indicando qual foi o termo buscado pelo usuário e a API utiliza um *script* em SQL que funciona através de quatro etapas:

1. Traz da tabela de registro de buscas dos usuários todos os registros que contenham esse mesmo termo;

2. Seleciona os identificadores únicos de usuários juntamente com a data e hora dos registros buscados no passo anterior - considerando apenas registros únicos;

3. Realiza uma nova pesquisa na tabela de registros de buscas a fim de encontrar outros termos pesquisados com os mesmos identificadores únicos e data e hora encontrados no passo anterior;

4. Agrupa e ordena os resultados de forma a encontrar no máximo cinco produtos que mais foram pesquisados juntamente com o termo passado pelo *front-end*.

Vale ressaltar que a lógica citada acima ainda possui um filtro para retornar apenas termos pesquisados juntos por pelo menos três usuários diferentes (ou o mesmo usuário em dias e horários diferentes). Além disso, existem alguns descartes no resultado, como o filtro para analisar se o termo retornado possui mais de dois caracteres (para evitar possíveis erros de digitação) e se não são apenas números.

Após isso, a API analisa se houve algum retorno do sistema de recomendação por popularidade e, se existir, é ele que será mostrado ao usuário no *website*. Caso contrário, será usado o segundo modelo de recomendação - aquele que utiliza um modelo de *clustering*.

3.5.1.2 Sistema de Recomendação por Clustering

Esse segundo sistema de recomendação funciona a partir de um modelo de *K-means* que é treinado e atualizado todos os dias através de uma DAG criada no Airflow. Esse modelo utiliza as descrições de produto existentes na tabela de resultado dos *crawlers* no momento em que a DAG está rodando, já que os *crawlers* rodam todos os dias e, com isso, as descrições dos produtos estão em constante mudança.

No treinamento do modelo, as descrições de produtos são vetorizadas (utilizando a extração de *feature TF-IDF*) levando em conta as *stop words* previamente definidas. Essas *stop words* são uma junção de palavras provenientes da biblioteca de Python denominada nltk com palavras criadas através da observação das descrições de produtos como, por exemplo, as palavras “características”, “linear” e “utilizar”. Após isso, utiliza-se a biblioteca sklearn do Python para a criação do modelo, o qual fica salvo em um arquivo na instância EC2 que também é utilizada pelo *back-end* do *website*.

A fim de mostrar o resultado desse sistema de recomendação, a mesma rota da API que produz a saída do sistema de popularidade e verifica se existem resultados para serem retornados, também utiliza o modelo de *clustering* quando necessário. Essa API importa o modelo e as *stop words* (que também ficam salvas em um arquivo local), vetoriza o termo buscado pelo usuário (utilizando o *TF-IDF*) e realiza a predição para retornar qual é o *cluster* ao qual essa busca pertence.

Vale notar que a lógica foi criada para retornar ao usuário apenas a palavra mais popular do *cluster* que não é (e não contém) o termo buscado, sendo normalmente o centroide. A decisão de retornar apenas uma recomendação deve-se ao fato que esse é um sistema de recomendação secundário com o intuito de complementar o resultado não existente do primeiro sistema.

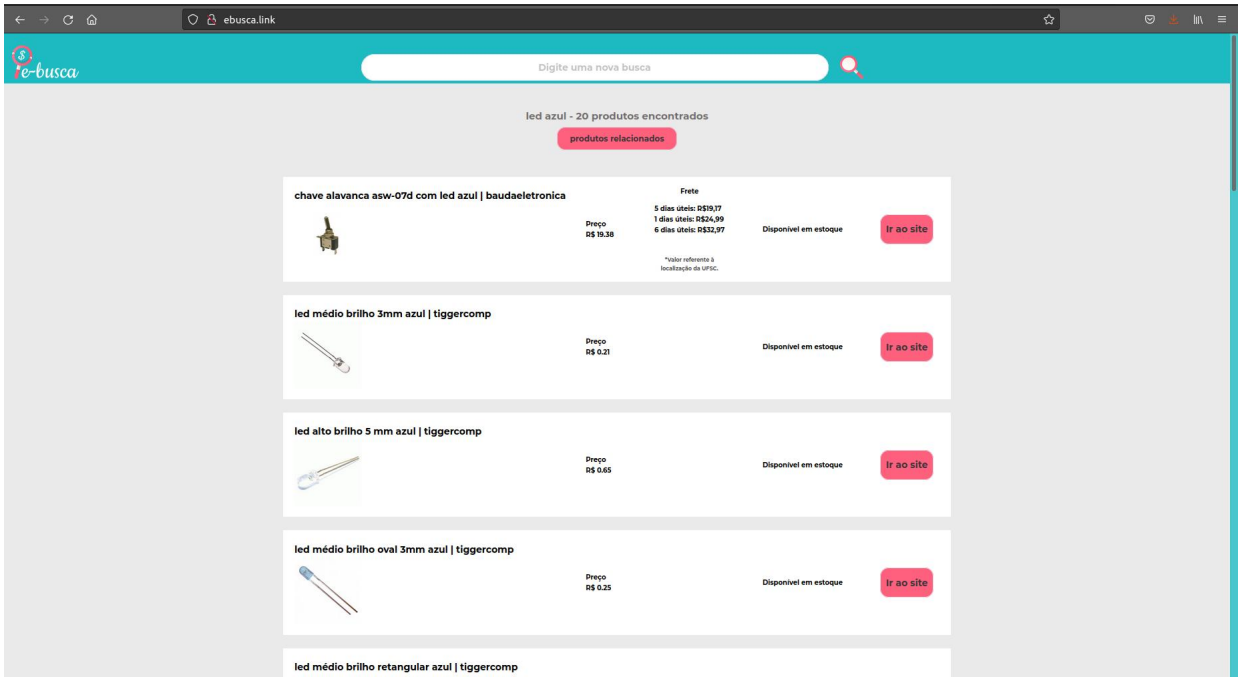
4 RESULTADOS

Neste capítulo serão apresentados os resultados finais do trabalho. Vale ressaltar que o *website* criado pode ser acessado no endereço eletrônico http://ebusca.link.

4.1 WEBSITE

A versão final da página de início do *website* ficou similar a imagem mostrada anteriormente na Figura 2. Já a versão final da página de resultados pode ser observada na Figura 4 abaixo.

Figura 4 – Versão final do front-end da página de resultados.



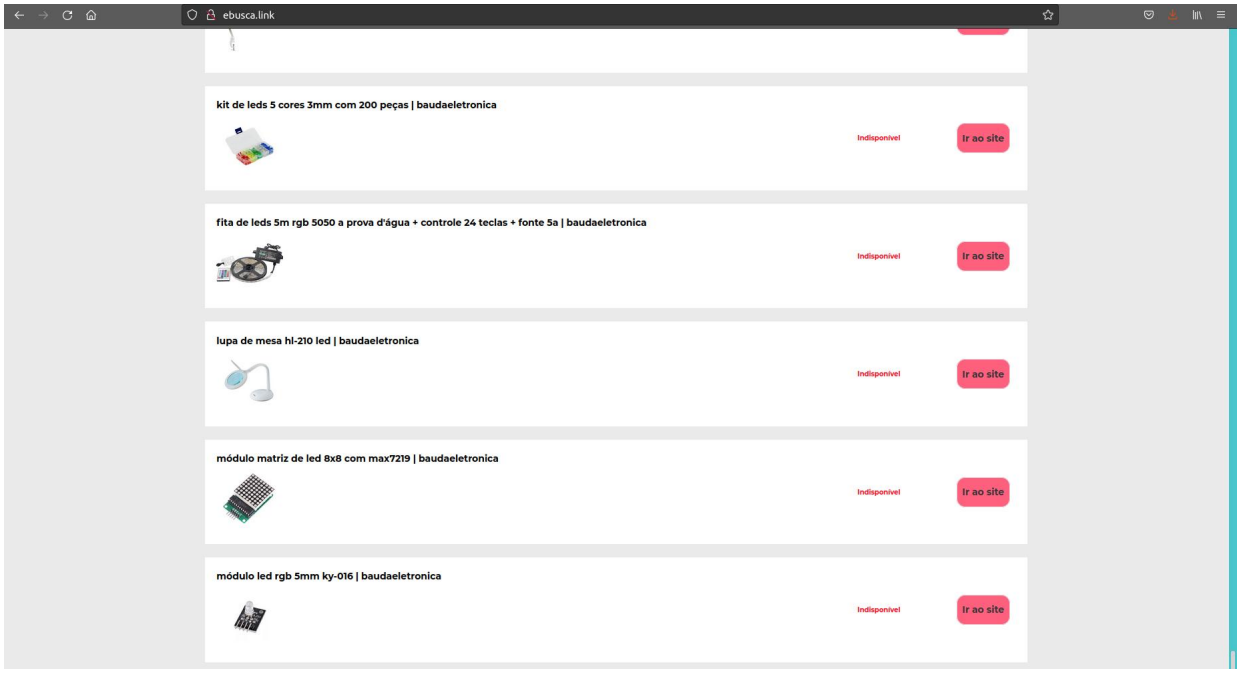
Fonte: do Autor.

Alguns pontos interessantes a serem mencionados é que, como nem todos os *websites* utilizados como base para a extração de dados disponibilizam os dados de frete de maneira acessível pelos *crawlers*, nem todos os itens mostram essas informações - apenas aqueles provenientes dos sites Baú da Eletrônica e Filipeflop. Além disso, a Figura 5 abaixo mostra como fica o *front-end* do site quando um produto está indisponível na loja em questão - esses produtos ficam ordenados para aparecem na parte inferior da lista de resultados.

4.1.1 Recomendações

Abaixo serão mostrados nas Figuras 6 e 7 resultados de recomendação para as buscas "led vermelho" e "resistor" as quais utilizam o sistema de recomendação por popularidade.

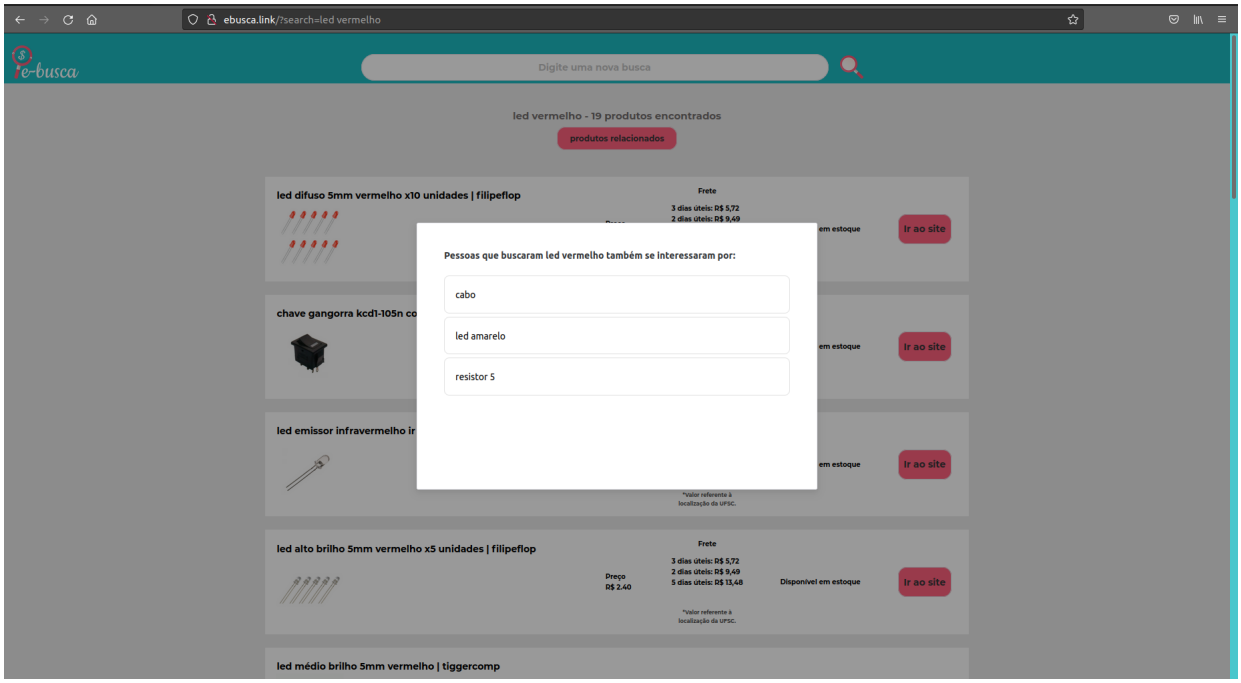
Figura 5 – Versão final do front-end para produtos indisponíveis.



Fonte: do Autor.

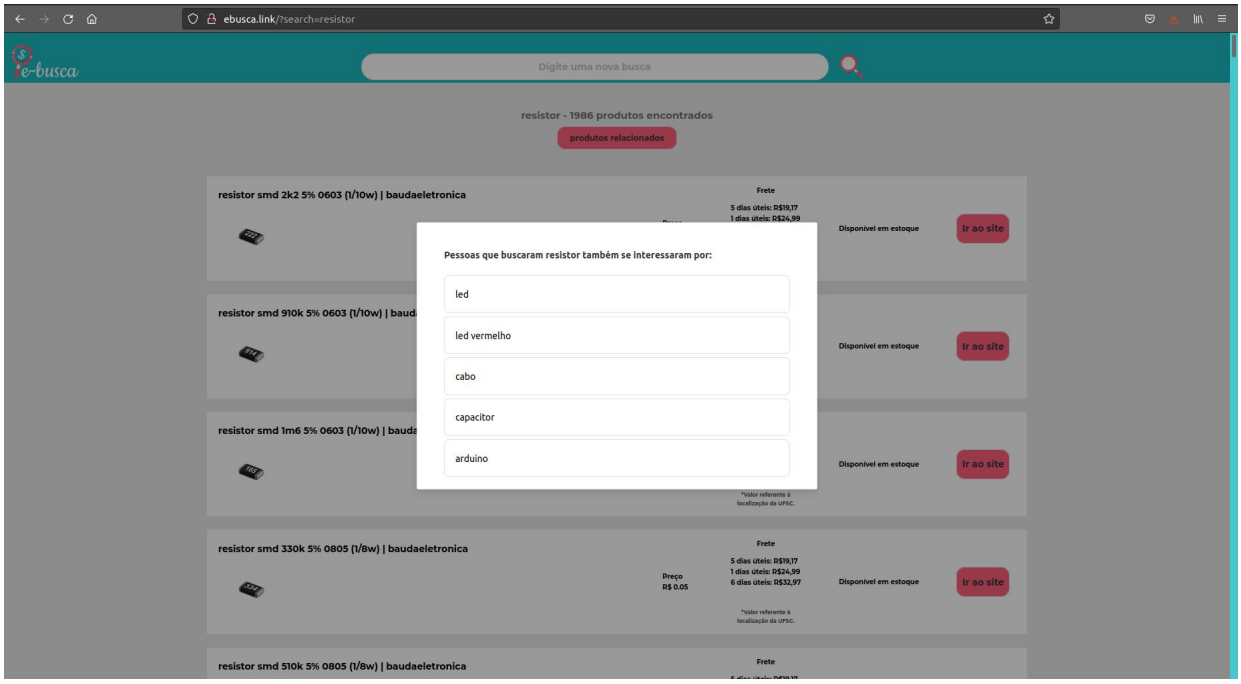
Já as Figuras 8 e 9 mostram os resultados para as buscas "sensor de movimento" e "circuito integrado" que utilizam o sistema de recomendação por *clustering*.

Figura 6 – Resultado da recomendação por popularidade para a busca "led vermelho".



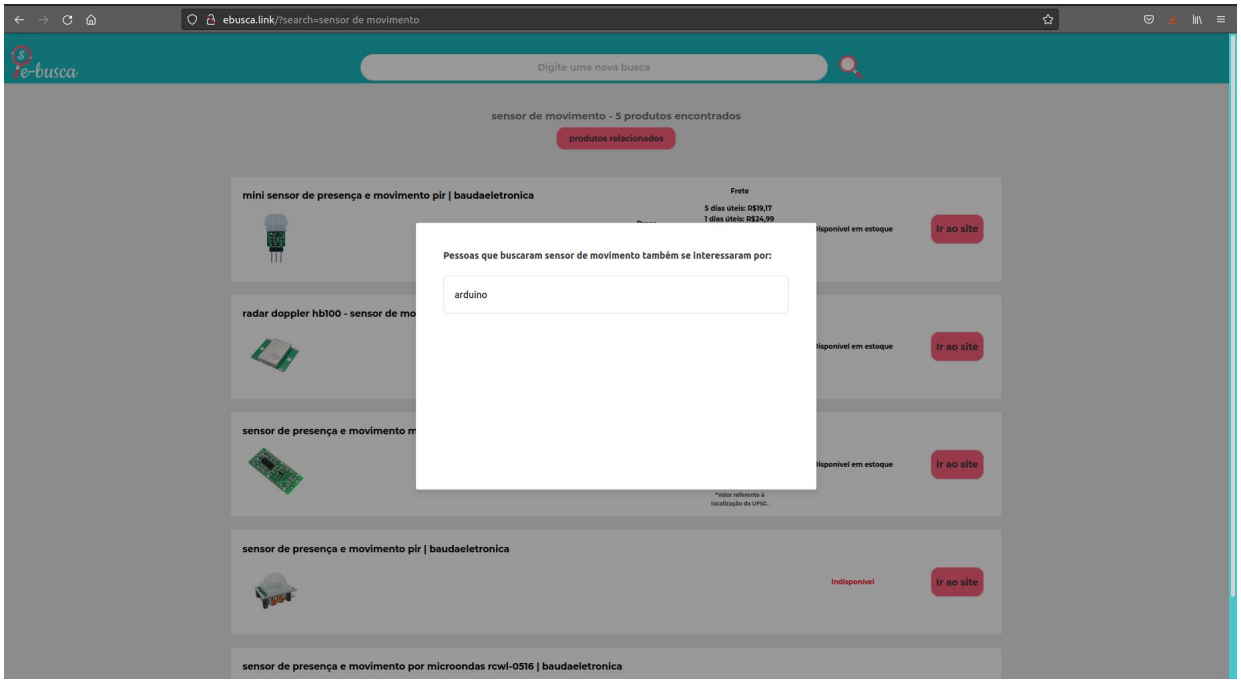
Fonte: do Autor.

Figura 7 – Resultado da recomendação por popularidade para a busca "resistor".



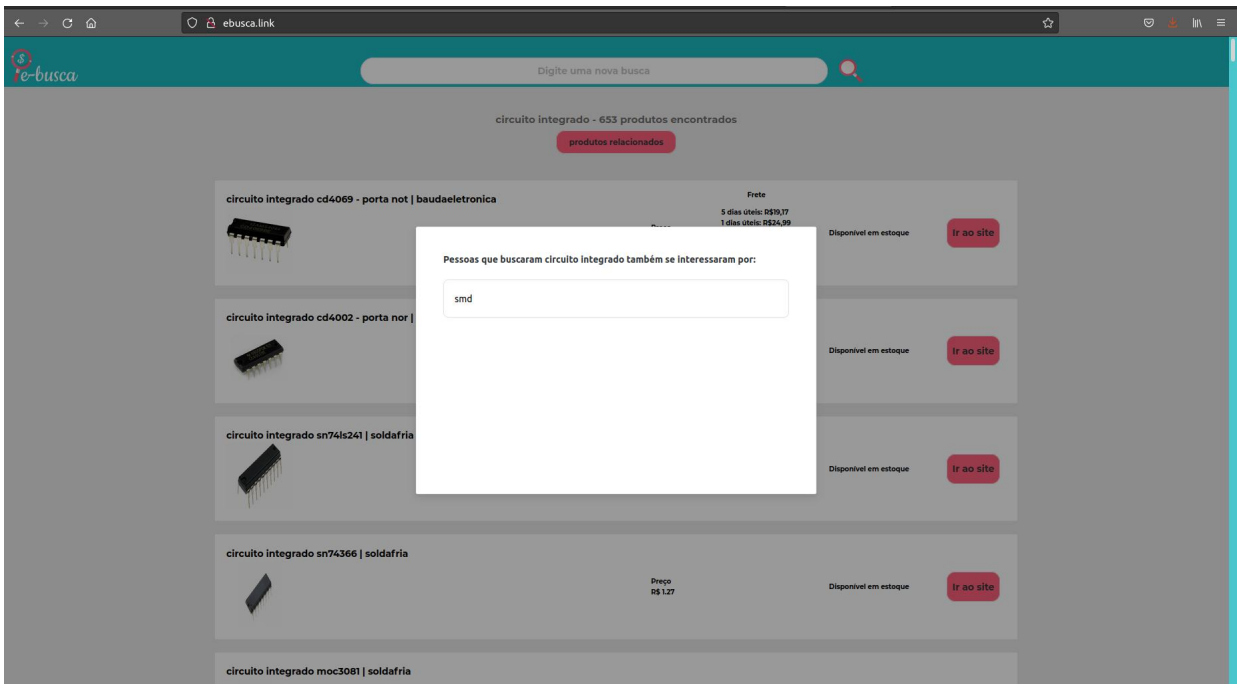
Fonte: do Autor.

Figura 8 – Resultado da recomendação por clustering para a busca "sensor de movimento".



Fonte: do Autor.

Figura 9 – Resultado da recomendação por clustering para a busca "circuito integrado".



Fonte: do Autor.

5 CONCLUSÃO

Neste trabalho, foi criado um sistema automatizado para extração de dados de quatro *websites* de venda de componentes eletrônicos na modalidade *on-line* através de *crawlers* e *scrappers*. Além disso, a fim de disponibilizar uma boa visualização dos resultados dos *crawlers*, criou-se um *website* que possui uma ferramenta de busca e um sistema de recomendação que, por sua vez, utiliza dois diferentes sistemas: o de popularidade e o de *clustering*. Um ponto importante a ser mencionado sobre o sistema de recomendação é que, a medida que o *website* for utilizado, o sistema de recomendação secundário (recomendação por modelo de *clustering*) será cada vez menos necessário. Isso porque, com o aumento da quantidade de buscas no site, aumentará também a quantidade de retorno para o sistema de recomendação por popularidade.

Para a construção do projeto como um todo foram necessários conhecimentos sobre componentes eletrônicos adquiridos no curso de Engenharia Eletrônica a fim de validar os resultados mostrados no *website* construído. Além disso, algumas matérias optativas do curso foram essenciais para auxiliar no aprendizado de Python.

5.1 TRABALHOS FUTUROS

Com o intuito de aumentar a quantidade de dados apresentados no *website* final, seria interessante criar mais *crawlers* e utilizar, para isso, serviços de *proxy* a fim de ter uma diversidade maior de *websites* que podem ser utilizados como base para a extração. Além disso, alguns outros pontos que melhorariam o resultado final seriam:

- Extração de informações sobre formas de pagamentos para cada uma das lojas;
- Utilização de paginação para a tela de resultados;
- Criação de filtros no *website* para que o usuário consiga escolher resultados de lojas específicas ou ordenar produtos pelo preço;
- Utilização de responsividade no *front-end* do *website* final para que ele se adapte em qualquer dispositivo;
- Cálculo do valor do frete a partir do *input* do usuário com seu endereço.

REFERÊNCIAS

B. TRSTENJAKA S. MIKACB, D Donkoc. KNN with TF-IDF Based Framework for Text Categorization. **Procedia Engineering 69**, p. 1356–1364, 2014. DOI: <10.1016/j.proeng.2014.03.129>.

BROWNLEE, J. **10 Clustering Algorithms With Python**. 2020. Disponível em: <https://machinelearningmastery.com/clustering-algorithms-with-python/>. Acesso em: 22 jan. 2022.

CHAND, M. **Top 10 Cloud Service Providers In 2021**. 2021. Disponível em: <https://www.c-sharpcorner.com/article/top-10-cloud-service-providers/>. Acesso em: 5 jan. 2022.

D. TOMASEVIC, T. Pavicevic. **RESEARCHING THE CONFLICTS BETWEEN USER EXPERIENCE, FRONT-END AND BACK-END IN SOFTWARE DEVELOPMENT PROCESS**. [*S.l.*: *s.n.*], 2021.

DATE, C. J. **Introdução a sistemas de bancos de dados**. [*S.l.*]: ELSEVIER EDITORA, 2004. ISBN 9788535212730. Disponível em: <https://books.google.com.br/books?id=xBeO9LSlK7UC>.

FOUNDATION, The Apache Software. **DAGs**. 2022. Disponível em: <https://airflow.apache.org/docs/apache-airflow/stable/concepts/dags.html#dags>. Acesso em: 29 jan. 2022.

GOOGLE. **Learn about sitemaps**. 2021. Disponível em: <https://developers.google.com/search/docs/advanced/sitemaps/overview>. Acesso em: 29 jan. 2022.

KEMP, S. **Digital 2021: the latest insights into the 'state of digital'**. 2021. Disponível em: <https://wearesocial.com/uk/blog/2021/01/digital-2021-the-latest-insights-into-the-state-of-digital/>. Acesso em: 2 jan. 2022.

KOUZIS-LOUKAS, D. **Learning Scrapy**. [*S.l.*]: Packt Publishing, 2016. ISBN 9781784390914. Disponível em: <https://books.google.com.br/books?id=EF8dDAAAQBAJ>.

MD. ABU KAUSAR V. S. DHAKA, Sanjeev Kumar Singh. Web Crawler: A Review. **International Journal of Computer Applications**, v. 63, p. 31, fev. 2013. ISSN 0975-8887.

SEN, S. **Recommender Systems**. 2020. Disponível em: <https://medium.com/the-owl/recommender-systems-f62ad843f70c>. Acesso em: 22 jan. 2022.