**Desenvolvimento**

**Configurações do ambiente**

Para o desenvolvimento desta aplicação foram utilizados:

Plataforma: *Node.Js* – Versão: 10.16.2

*Framework*: *Selenium WebDriver* – Versão: 4.0.0-*alpha*.5

Sistema Operacional: *Windows* 10 *Home*

Bibliotecas internas utilizadas:

*Csv-writer* – Versão: 1.6.0

*Date-pt-br* – Versão: 1.2.2

*Ini* – Versão 1.3.5

Para criação do arquivo executável, foi utilizado a biblioteca pkg.

As bibliotecas acima citadas podem ser encontradas no repositório npm.

É importante salientar também que este software automatiza um processo no navegador *Google Chrome*. Para que o *Selenium* consiga manipular o navegador corretamente, é necessário que as versões de ambos estejam sempre compatíveis e atualizadas.

**Objetivo**

Desenvolver um *software*, com interface gráfica, que capture três *inputs* de dados do usuário: Palavra chave de busca, quantidade de páginas e o tipo de relatório a ser gerado. O *software* então fará uso dessas informações para efetuar a busca no site BuscaPé, recuperando todos os resultados encontrados. No fim, será gerado um relatório em formato texto ou *Excel*, selecionado pelo usuário.

**Iniciando o navegador Google Chrome**

Para inicialização do navegador é instanciado um objeto *WebDriver*, exportado pelo módulo do *Selenium WebDriver*. Este objeto representará o navegador. É utilizando este objeto que o software consegue manipular todas as funcionalidades do *browser*, oferecidos pela biblioteca *Selenium*.

Esta instanciação é feita pela seguinte declaração:

*let driver = new constants.Builder().forBrowser("chrome").build();*

onde *constants* é um arquivo contendo diversas constantes, como a biblioteca importada para o projeto *Selenium WebDriver*, e *Builder*() é a classe responsável por instanciar nosso objeto *WebDriver*. Note que é no método *.forBrowser*() que dizemos ao *Selenium* qual navegador a aplicação utilizará. Pela mesma linha de comando, seria possível instanciar objetos para os navegadores *Firefox* e *Internet Explorer* também, caso necessário.

A partir desta linha, o navegador *Google Chrome* já estará em execução na máquina do usuário.

Figura - Inicialização do driver.

Texto preto sobre fundo branco

Descrição gerada automaticamente

Fonte: Elaborado pelo autor.

Note que o driver apresenta uma mensagem: “O *Chrome* está sendo controlado por um *software* de teste automatizado.”, indicando que está sob efeito de um agente externo.

**Acessando o BuscaPé e realizando a consulta**

Uma vez instanciado em memória, o objeto *WebDriver* do *Google Chrome* pode ser manipulado. O *driver* então é chamado para buscar a página do BuscaPé, pelo método *get*(). A janela também é maximizada para uma maior experiência ao usuário.

*await this.driver.get('https://buscape.com.br');*

É passado como parâmetro o *site* que se deseja navegar. Note aqui, bem como em outras partes do *software*, o uso da palavra ‘*await’*: Esta é uma palavra reservada da linguagem *Javascript* utilizada quando é necessário aguardar o fim de um comando executado de forma assíncrona. Em outras palavras, o comando ‘*await’* pausa a execução do *software* até que o dado comando seja concluído.

Uma vez completamente carregada a página, é inserida a busca do usuário no campo de buscas do BuscaPé.

Para manipulação do campo de buscas e de todos os elementos da página, foi utilizado o DOM (*Document Object Model*) para localização dos elementos.

Localizamos o elemento pela sua propriedade ‘*name’*, definindo um *time-out* para este comando de 1 minuto (60 segundos). Ao localizá-lo, o *software* envia a palavra-chave definida pelo usuário, seguida da tecla *ENTER*, iniciando assim a pesquisa.

*let buscaPeSearchBar = this.driver.wait(constants.until.elementLocated(constants.By.name('q')), 60000);*

*buscaPeSearchBar.sendKeys(searchTags, constants.Key.ENTER);*

**Garimpando dados da página**

Aqui inicia-se o trabalho de extração das informações da página. Serão extraídos, de cada produto, as informações: Nome do produto, loja onde é vendido, preço do produto na respectiva loja e *link* direto para o item.

Todo o processo se dá de forma dinâmica, ou seja, de acordo com a quantidade de resultados encontrados para os parâmetros informados pelo usuário.

A ferramenta primeiro identifica se existem resultados para a busca do usuário. Caso exista pelo menos um resultado, o *software* percorrerá todos os produtos ofertados, armazenando as informações internamente.

A ferramenta também realiza a distinção quanto ao número de lojas que concorrem por um mesmo anúncio: Caso existam duas ou mais lojas concorrendo no mesmo produto, o *software* entrará no anúncio para capturar os dados de todos os concorrentes, para então voltar à página anterior e continuar a pesquisa. Caso contrário, capturará apenas os dados presentes na página atual, prosseguindo então para o próximo anúncio.

Desta forma, a lógica se dá em forma de dois *loops* principais: Um *loop* externo controla o número de páginas que deverão ser analisadas e um *loop* interno controla o número de itens presentes na página em questão.

Para captura dos elementos da página foram utilizados diversos seletores *CSS.* Segundo Mesbah e Mirshokraie (2012), *CSS* é uma linguagem amplamente usada para descrever a semântica de apresentação de elementos *HTML* na *web*.

A linguagem possui várias características, como herança e ordem em formato cascata, o que torna a manutenção do código *CSS* uma tarefa desafiadora para desenvolvedores da *web*.

Desta forma, utilizamos os seletores *CSS* para isolar os elementos de interesse da aplicação.

Figura - Análise do DOM da página.

Tela de celular com publicação numa rede social

Descrição gerada automaticamente

Fonte: BuscaPé.

Para armazenamento interno, foi utilizada o método .*findElements*, proveniente do objeto *WebDriver* do *Selenium*. Este método retorna o tipo de dado *WebElement*. Caso múltiplos elementos se enquadrem na seleção, todos serão retornados em forma de uma lista, passível de iteração.

Foi utilizada também o método .*executeScript* para injeção de códigos *JavaScript*, alternativa ao *framework* *Selenium* onde o mesmo mostrou-se inoperante na versão executável.

**Armazenamento interno**

O armazenamento dos dados extraídos se dá de acordo com a seguinte estrutura:

Temos um vetor principal, representando toda a estrutura. Apelidaremos este vetor principal de Vetor\_A. Cada posição do Vetor\_A é também um vetor de tamanho n, n representando o número de concorrentes em um determinado anúncio. Chamaremos este vetor de Vetor\_B. Por sua vez, cada posição deste Vetor\_B é um objeto chave-valor. Este objeto, que apelidaremos de D\_Prod, armazena 3 variáveis do tipo string.

Figura - Estrutura de Armazenamento.

Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente

Fonte: Elaborada pelo autor.

Cada anúncio no BuscaPé será armazenado em uma posição de Vetor\_A.

No caso em que apenas uma loja esteja concorrendo pelo anúncio, então Vetor\_B será composto por apenas um elemento: um item D\_Prod, contendo os dados do anúncio.

Caso o anúncio contenha N concorrentes, então Vetor\_B será composto por N posições, cada posição com os dados do respectivo concorrente (nome da loja, preço do produto e *link* direto para o anúncio na loja).

**Geração do relatório**

O relatório será gerado de acordo com a escolha do usuário. Na interface, o usuário possui duas opções de exportação dos dados:

1. Exportação em formato texto (extensão .txt)
2. Exportação em formato excel (extensão .csv)

Ao final da execução, toda a informação extraída do website BuscaPé estará armazenada internamente em memória, na estrutura citada em Armazenamento Interno.

Para a criação do relatório em formato texto, o software percorrerá toda a estrutura, identificando cada um dos elementos e os inserindo no arquivo final, um após o outro, utilizando para isso o módulo *File System*, nativo do *Node.Js*. Caso o arquivo não exista inicialmente o software o criará no diretório específico.

No relatório em formato texto também é inserido um cabeçalho com as especificações da busca: nome do item pesquisado, número de resultados encontrados, número de páginas pesquisadas e informações referentes à data e hora da execução.

A geração do arquivo em excel se dá de maneira semelhante, diferindo apenas na maneira em que os dados são manipulados. Para a manipulação de arquivos com extensão .csv foi utilizado a módulo externo *Csv-writer* e, portanto, os dados precisam ser preparados para correta visualização no relatório final.

**Interface**

A interface do software foi desenvolvida com a linguagem de *script* *AutoHotKey*.

Figura - Interface do Software.

Tela de celular com publicação numa rede social

Descrição gerada automaticamente

Fonte: Elaborado pelo autor.

A interface foi composta pelos seguintes elementos:

1. Campo de texto para o item a ser pesquisado. Informado pelo usuário. Preenchimento obrigatório.
2. Campo de texto referente ao número de páginas a serem vasculhadas na *web*. Informado pelo usuário. Preenchimento obrigatório. Formato numérico obrigatório, positivo.
3. Botão para seleção do tipo de relatório. Por padrão selecionado no formato texto.
4. Barra de progresso, atualizada automaticamente pelo *software*.
5. Botão para início de processamento.
6. Botão para interrupção de processamento.

A interface ainda conta com o nome do *software* e versão em execução.

Ao clicar no botão ‘*Start Search*’ o *software* verificará automaticamente o preenchimento dos campos obrigatórios. Caso algum dos critérios sejam descumpridos, uma mensagem de erro será exibida ao usuário.

Figura - Sinalização de campos obrigatórios.

Tela de celular com publicação numa rede social

Descrição gerada automaticamente

Figura Fonte: Elaborado pelo autor.

**Exceções**

Algumas exceções podem ser geradas durante a execução do software. Problemas como perda de conectividade na internet e problemas ao realizar leitura ou escrita em arquivos foram os mais frequentes durante o desenvolvimento desta ferramenta. Todas as exceções são gravadas em um arquivo de log para futura análise.