RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 1 2024

SISTEMA PARA VALIDAÇÃO DE HIPERLINKS EM ARTIGOS PUBLICADOS NA WEB

Lucas Henrique Barbosa Silva
Graduando em Engenharia de Software – Uni-FACEF
lucas.barbosa3680@gmail.com

Débora Pelicano Diniz
Mestre em Ciência da Computação – UFSCar
deboradiniz@facef.br

Resumo
O trabalho apresentado neste artigo teve o objetivo a criação de um algoritmo para validar o uso de hiperlinks em documentos digitais, visto que nos dias atuais é muito comum o uso de hiperlinks para conectar diferentes recursos em páginas da web. Assim, na escrita do artigo foi explicado como criar links, comentando os benefícios que uma navegação eficiente e fluida proporciona, bem como foram apresentadas suas desvantagens, como distração do leitor e links quebrados. Também foram detalhadas maneiras de verificar se um hiperlink está funcionando corretamente, utilizando ferramentas online e expressões regulares (Regex). Foram apresentados conceitos, formas de implementações, como também as tecnologias utilizadas, como Node.js, FileSystem (FS) e a biblioteca Chalk para estilização de texto no terminal. Além disso, Foram detalhadas as funções que foram criadas durante o desenvolvimento do algoritmo de *web scraping* que percorre um determinado arquivo markdown e retorna no terminal uma lista dos links que foram encontrados e o seus respectivos *status code*, bem como os testes realizados para comprovar a eficiência de sua funcionalidade.

Palavras-chave: *Web Scraping*. Regex. Hiperlink.

Abstract
The work presented in this article aimed to create an algorithm to validate the use of hyperlinks in digital documents, given that nowadays it is very common to use hyperlinks to connect different resources on web pages. Thus, when writing the article, it was explained how to create links, commenting on the benefits that efficient and fluid navigation provides, as well as illustrations such as reader distraction and broken links. Ways to check whether a hyperlink is working correctly have also been implemented, using online tools and regular expressions (Regex). Concepts, forms of implementation were presented, as well as the technologies used, such as Node.js, FileSystem (FS) and the Chalk library for text styling in the terminal. Furthermore, the functions that were created during the development of the web scraping algorithm that scans a given markdown file and return to the terminal a list of links that were found and the respective status code were developed, as well as the tests carried out to prove the efficiency of its functionality.

Keywords: *Web Scraping, Regex, Hiperlink*.

1 Introdução

Um hiperlink estar funcionando corretamente é crucial para a integridade de um artigo ou site na web, pois é a partir dele que o usuário irá conseguir obter

informações relevantes para a sua busca ou interagir com a plataforma, melhorando a experiência do usuário.

Quando um link está quebrado ou leva o usuário a uma página inexistente, isso pode causar frustração e diminuir a confiança do usuário no conteúdo do site ou serviço que o mesmo está acessando. Isso se dá por conta de uma falta de gerenciamento dos hiperlinks, em que não é possível rastrear dentro do site, quais deles estão ativos ou não.

Considerando o que foi exposto, o objetivo do trabalho é a criação de um algoritmo para validar o uso de hiperlinks em documentos digitais, utilizando boas práticas das expressões regulares (Regex), que são essenciais para fazer a captura de links dentro de um arquivo markdown, de forma a aumentar a confiança do usuário no momento em que está lendo um texto que possui hiperlinks. Assim, foi desenvolvido um algoritmo que faz a captura desses hiperlinks e retorna em uma lista, todos os links capturados, juntamente com o seu *status code* (Maia. 2020).

De forma a atingir o objetivo proposto, o artigo está estruturado asseguinte forma: na Seção 2 são apresentados conceitos e funcionalidades de um hiperlink, como sua criação, funcionalidade, importância, vantagens e desvantagens; na Seção 3 é feita uma descrição das expressões regulares, também trazendo seus benefícios e como utilizá-la; na Seção 4 são apresentadas as demais ferramentas que foram utilizada para o desenvolvimento do algoritmo, como Node.js, FileSystem (FS), arquivo markdown e a biblioteca Chalk; na Seção 5 são apresentadas as considerações finais e perspectivas futuras deste projeto.

2 Hiperlinks em textos

Um hiperlink é um elemento em um documento digital que faz referência a outro recurso, seja um outro ponto no mesmo documento, um documento diferente ou uma localização na web. Quando ativado (normalmente por meio de um clique), o hiperlink permite ao usuário navegar diretamente para o recurso referenciado (MDN Contributors, 2023).

2.1 Como criar

Para que um hiperlink possa ser criado é necessário analisar a anatomia de um link. Um link básico é criado envolvendo o texto (ou outro conteúdo) que se deseja transformar em um link, dentro de um elemento **<a>** (MDN contributors, 2023).

Dê ao link um atributo **href** (também conhecido como *Hypertext Reference* ou destino), que conterá o endereço da web para o qual o link apontará. No quadro 1 está apresentado um exemplo de hiperlink.

Quadro 1: Exemplo de hyperlink

Link para o meu perfil no GitHub.. Isso resultará em: “Link para o meu perfil no GitHub.”

Fonte: Autoria própria

Existem outras maneiras de criar um hiperlink, a mais comum delas é a criação de hiperlinks dentro de e-mail, em que é necessário apenas escrever um texto, copiar o link de uma página de destino e inserir no texto escrito anteriormente (MDN Contributors, 2023).

2.2 Vantagens

Os hiperlinks permitem que os usuários naveguem entre diferentes páginas da web de forma rápida e eficiente. Com apenas um clique, pode-se redirecionar o usuário para outra página, seja dentro do mesmo site ou para um site terceiro.

Eles podem conectar uma página a outra, permitindo que o usuário acesse conteúdo relacionado. Imagine um índice em um livro: cada entrada é um hiperlink para uma seção específica. Isso facilita a localização de informações relevantes.

Antes dos hiperlinks, a navegação era linear. Passava-se por todas as páginas para chegar à última. Agora, com hiperlinks, pode-se ir diretamente para a página desejada, tornando a experiência de navegação mais fluida.

Hiperlinks geralmente são mostrados em cor azul e/ou sublinhados, o que facilita a identificação. Ao passar o mouse sobre um hiperlink, ele se transforma em uma mão, indicando que é clicável (Cloudmarket. 2022).

2.3 Desvantagens

Os hiperlinks são elementos essenciais na web, mas como qualquer tecnologia, também têm suas desvantagens, como (Cloudmarket. 2022):

- **Quebra de fluxo de leitura:** quando um hiperlink é inserido em um texto, ele pode interromper o fluxo natural da leitura. Os leitores podem ser distraídos e desviados para outras páginas antes de terminarem de ler o conteúdo original.
- **Links quebrados: h**iperlinks podem se tornar obsoletos ou apontar para páginas que não existem mais. Isso pode causar frustração para os usuários que esperam encontrar informações relevantes.
- **Acessibilidade**: nem todos os usuários conseguem clicar em hiperlinks. Pessoas com deficiências visuais, por exemplo, podem depender de leitores de tela que não interpretam links da mesma forma que os usuários videntes.
- **Poluição visual:** muitos hiperlinks em uma página podem torná-la visualmente confusa e desorganizada. Isso pode prejudicar a experiência do usuário.
- **Risco de *phishing*:** hiperlinks podem ser usados para direcionar os usuários para sites maliciosos. Os usuários podem ser enganados por links que parecem legítimos, mas na verdade levam a páginas falsas.
- **Dependência externa:** quando um site contém muitos hiperlinks para outros domínios, ele se torna dependente desses sites externos. Se um desses sites sair do ar ou mudar seu conteúdo, os links podem ficar quebrados.

Apesar dessas desvantagens, os hiperlinks continuam sendo uma parte fundamental da experiência de navegação na web. Eles permitem a conexão entre diferentes páginas e recursos, facilitando a descoberta de informações relevantes. Portanto, é importante usá-los com sabedoria e considerar as necessidades dos usuários ao projetar um site.

2.4 Como verificar se o hiperlink está funcionando corretamente

Existem algumas maneiras para verificar se um hiperlink está funcionando da maneira correta, como (MDN Contributors, 2023):

- **Verificação manual:** clique no link e verifique se ele abre a página da web correta. Se o link estiver quebrado ou não funcionar, uma mensagem de erro (404 ou página inexistente) será exibida. Verifique também se o URL exibido na barra de endereços corresponde ao destino esperado.
- **Ferramentas online:** existem várias ferramentas online que podem verificar a validade de um hiperlink. Alguns exemplos incluem:
    - **Verificador de Links da NordVPN: e**ssa ferramenta verifica se um URL é seguro, detectando malwares, sites falsos e ataques de *phishing*.
    - **Verificador de URL segura do linkbioBR:** essa ferramenta verifica se uma URL está bloqueada ou marcada como segura/insegura pelo Google.
    - **Verificador de link de site do Site24x7: b**usca por links quebrados em uma página da web ou site.
    - **Algoritmos de busca:** também é possível verificar se um hiperlink está ativo ou não com algoritmos de busca que, por meiode expressões regulares, são capazes de identificar um link e retornar se o mesmo está ativo ou não.

3 Regex: Expressão Regular: Conceitos e implementações

Expressões regulares, também conhecidas como **Regex**, são uma sequência de caracteres utilizadas para lidar e identificar utilizando padrões, outros caracteres dentro de um determinado texto. Elas permitem buscar, extrair e manipular texto com base em um padrão pré-definido (MDN, 2024).

As expressões regulares são usadas com os métodos **test** e **exec** do objeto **RegExpe** e com os métodos **match**, **replace**, **search** e **split** do objeto **String**.

Quando se deseja saber se um padrão é encontrado em uma **string**, usa-se o método **test** ou **search**; para mais informações (mas execução mais lenta) usa-se o método **exec** ou **match**. Caso tenha sido usado os métodos **exec** ou **match** e se houver correspondência, estes métodos retornam um *array* e atualizam as propriedades do objeto da expressão regular associada e também do objeto da expressão regular predefinida **RegExp**. Se não houver correspondência, o método **exec** retorna **null** (convertido para ***false***).

3.1 Criando uma expressão Regular

Existem duas maneiras de criar uma expressão regular. A mais comum e mais utilizada por ser mais performática é a expressão literal, que consiste em um padrão criado entre barras **const regex = /ab+c/;**. Essa forma de construção possui melhor performance quando a expressão regular utilizada é uma constante, ou seja, não muda durante a execução (MDN, 2024).

A outra maneira de criar uma expressão regular é chamando o construtor **RegExp**, **let regex = new RegExp("ab+c")**. Esse construtor é utilizado quando sabe-se que o padrão da expressão regular irá mudar ou quando o padrão for desconhecido, por exemplo, na entrada de dados de um usuário (MDN, 2024).

3.2 Benefícios

Utilizar Regex para verificar se um hyperlink está funcionando corretamente traz alguns benefícios, tais como (MDN, 2024):

- **Flexibilidade:** permitem criar padrões complexos para corresponder a uma ampla variedade de cadeias de caracteres.
- **Eficiência:** ferramentas poderosas para buscas e substituições rápidas em grandes volumes de texto.
- **Consistência:** padronização nas operações de correspondência de padrões e substituição em diversas linguagens e ferramentas.
- **Aplicações Diversas:** Utilizadas em validação de entrada de dados, extração de dados, manipulação de texto, *parsing*, entre outras utilizações.

3.3 Utilização

Na Figura 2 está apresentada a função que contém a expressão regular usada para identificar links no formato Markdown, capturando o texto entre colchetes, seguido por uma URL entre parênteses, onde a URL começa com "http" ou "https".

Figura 2: Descrição estrutural do regex

```
4  function extraiLinks(texto) {
5      const regex = /\[([^\]]*?)\]\((https?:\/\/[^\s?#.].[^\s]*)\)/gm;
6      const capturas = [...texto.matchAll(regex)];
7      const resultados = capturas.map(captura => ({[captura[1]]: [captura[2]]}))
8      return resultados.length !== 0 ? resultados : chalk.red('não há links no arquivo');
9  }
```

Fonte: Autoria própria

As barras / no início e no final fazem a identificação de uma expressão regular. As flags “g” e “m” são usadas para modificar o comportamento da busca, sendo elas:

- g (global): permite encontrar todas as correspondências no texto, não apenas a primeira.
- m (multiline): altera o comportamento dos caracteres ^ e $, que passam a corresponder ao início e fim de uma linha, respectivamente, em vez do início e fim de todo o texto.

O caractere de escape “\” é usado para indicar que estamos procurando o caractere literal “[“ (colchete esquerdo). No regex, os colchetes são especiais, então precisam ser escapados para serem tratados como literais.

“**([...])**”: O grupo de captura é utilizado para capturar o que for correspondido dentro dele. O conteúdo capturado poderá ser recuperado posteriormente (é útil para manipulações).

“**[^[\]]**”: Dentro de um conjunto de colchetes “[...]”, o “^” na primeira posição significa "não". Portanto, “[^[\]]” significa qualquer caractere que não seja um colchete esquerdo ou direito. O “\” é para escapar os colchetes, tornando-os literais. Basicamente, está capturando qualquer coisa que não seja “[“ ou “]”. O “*” significa "nenhum ou mais" (zero ou mais ocorrências), já o “?” após o “*” tenta capturar o menor número possível de caracteres, em vez do maior número (que é o comportamento padrão).

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 1 2024

O segundo grupo de captura que procura o link (URL). Sendo eles:

“**https?**”: Corresponde a "http" ou "https". O ? significa "zero ou um" do caractere que o precede, neste caso, o "s". Portanto, pode corresponder a "http" ou "https".

“**:\/\/**”: Escapa os caracteres :// na URL (é a forma literal que eles aparecem após "http" ou "https").

“**[^\s?#.]**”: Dentro dos colchetes [...], o ^ no início significa "não". Estamos dizendo que queremos qualquer caractere que não seja um espaço (\s), ponto de interrogação (?), cerquilha/hash (#), ou ponto (.).

“**.**”: Corresponde a qualquer caractere (exceto nova linha), mas como foi colocado fora do conjunto anterior, será qualquer caractere após os primeiros caracteres do domínio da URL.

“**[^\s]***”: Corresponde a qualquer sequência de caracteres que não contenha espaço, ou seja, a parte restante da URL. O * significa "zero ou mais" ocorrências.

Seguindo esse sequência de caracteres, o algoritmo irá fazer a busca desse tipo de padrão: “**[texto de ancoragem](https://link.com)**”.

4 Ferramentas

Nessa seção estão apresentadas as ferramentas e metodologias utilizadas durante o desenvolvimento do projeto.

4.1 Lógica de programação

A lógica de programação é a base para o desenvolvimento de qualquer software, e aprender a aplicá-la usando Node.js pode ser extremamente vantajoso. Node.js é um ambiente de execução JavaScript que permite executar código JavaScript no lado do servidor, o que é ideal para construir aplicações web escaláveis e de alto desempenho (Miranda, 2024).

4.2 Node.js

Node.js é um ambiente de execução JavaScript que permite rodar código JavaScript no lado do servidor, fora do navegador (Kinta. 2023). Ele usa o navegador para interpretar o código e oferece uma arquitetura assíncrona e baseada em eventos, o que o torna ideal para aplicações escaláveis e em tempo real, como APIs, servidores web e chatbots. (Bessa, 2023).

Ele foi criado em 2009 Ryan Dahl e foi a primeira forma de criação de npm (Node packge). Após um ano de sua criação, foi desenvolvido o Express, um dos frameworks mais utilizados na área de desenvolvimento backend utilizando Node.js. Já em 2017,na sua versão 8, o node começou a ter mais foco na área de segurança, porém ainda não superando o Java. Por fim, a versão mais atual do node é a versão V21.7.3, porém a mais utilizada pelos desenvolvedores é a versão V18.20.4, pois existe diversas melhorias ligadas ao Chrome V8 (Bessa, 2023).

4.3 Regex

Regex (ou expressão regular), como já visto na Seção 3, é uma sequência de caracteres que define um padrão de busca. Ela é usada principalmente para encontrar e manipular texto com base em padrões específicos, como identificar números, e-mails, datas ou validar entradas em um formulário. Por exemplo, usa-se uma regex para verificar se um e-mail está no formato correto ou para substituir partes de um texto. Regex é amplamente usada em programação para realizar buscas e substituições de maneira eficiente (Regex Tutorial. 2023).

4.4 File System (FS)

O FS (File System) é um módulo integrado do Node.js que fornece uma API para interagir com o sistema de arquivos do computador em que o Node.js está sendo executado. Ele permite a leitura, gravação, exclusão e manipulação de arquivos e diretórios (Alves, 2023).

Esse módulo é muito utilizado em aplicações fullstack, pois os usuário podem querer enviar imagens, textos entre outros tipos de arquivos para o servidor e através das APIs que são fornecidas pelo File System. Os módulos Node.js fornecem funcionalidades como APIs para serem usadas por programas, como o módulo fs para interagir com o sistema de arquivos e o módulo http para criar servidores. Node.js abstrai muitas operações de baixo nível, facilitando o desenvolvimento. Além disso, é possível criar módulos personalizados. A partir da versão 14, o Node.js permite criar módulos de duas maneiras: CommonJS (CJS) e ESM (MJS), sendo que o artigo foca em exemplos no estilo CommonJS (Kinsta, 2023).

4.5 Chalk

A biblioteca Chalk, no Node.js é usada para estilizar o texto exibido no terminal, adicionando cores, negrito, sublinhado e outros efeitos de forma simples. Ela permite destacar saídas de log ou mensagens de erro de maneira visualmente atraente, sem modificar o comportamento do código, como pode ser visto na Figura 3.

Ela é leve, sem dependências externas, e amplamente utilizada para melhorar a legibilidade de saídas no terminal, especialmente em projetos de CLI (*Command Line Interface*)

Figura 3: Utilização do chalk

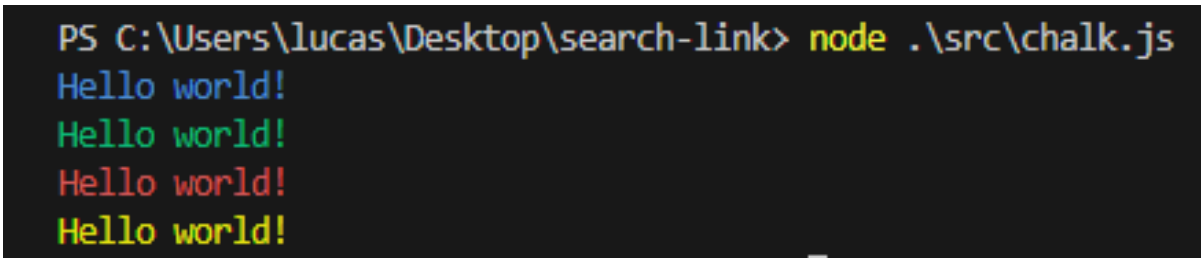
src > chalk.js

```
1  import chalk from 'chalk';
2
3  console.log(chalk.blue('Hello world!'));
4  console.log(chalk.green('Hello world!'));
5  console.log(chalk.red('Hello world!'));
6  console.log(chalk.yellow('Hello world!'));
```

Fonte: Autoria própria

A execução do código apresentado na Figura 3 gera o que está apresentado na Figura 4.

Figura 4: Impressão do chalk



Fonte: Autoria própria

4.6 Markdown

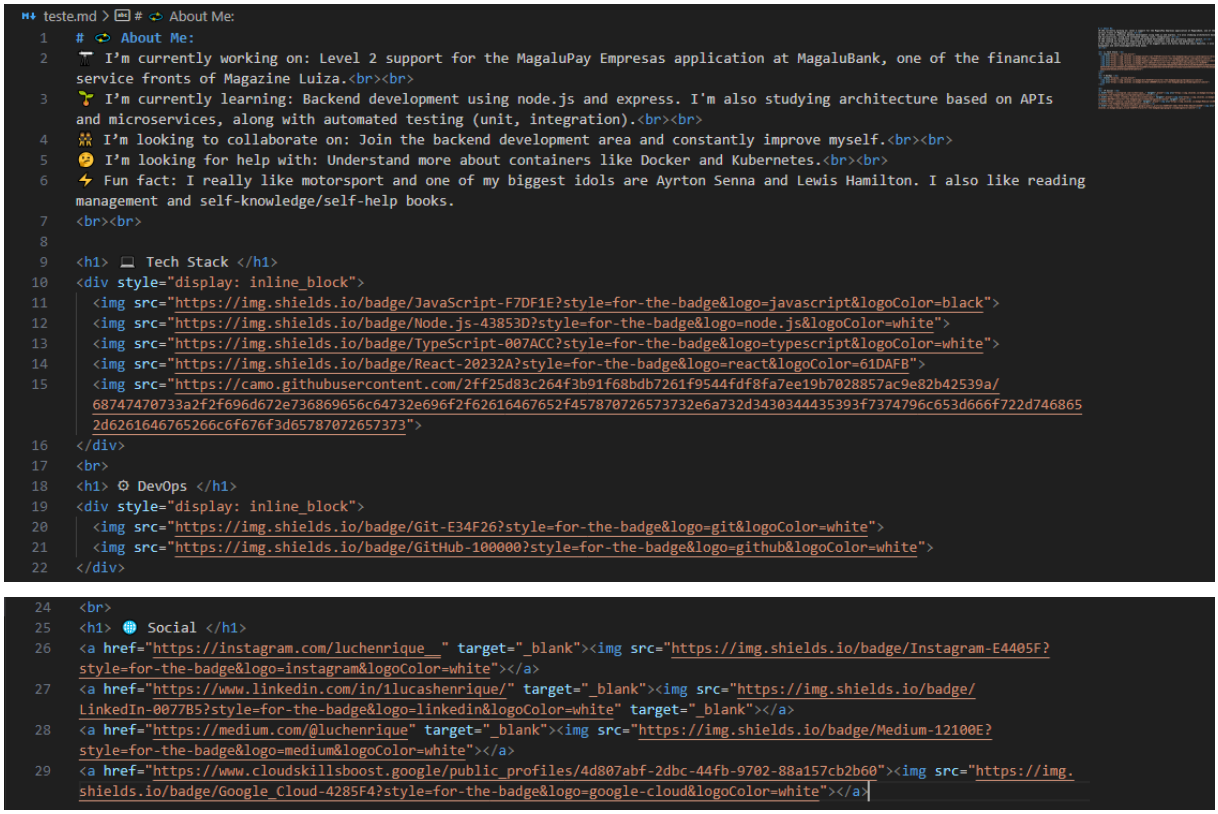
Markdown é uma linguagem de marcação simples e fácil de usar, projetada para formatar texto de maneira rápida e eficiente. Criada por John Gruber em 2004, ela permite que você escreva usando um formato de texto simples que é convertido em HTML válido. Aqui estão alguns dos elementos básicos do Markdown (Silveira. 2022):

- **Cabeçalhos:** Use o símbolo # para criar cabeçalhos. Por exemplo, # Cabeçalho 1 cria um cabeçalho de nível 1.
- **Ênfase:** Use asteriscos ou sublinhados para ênfase. Por exemplo, *itálico* ou _itálico_ para itálico e **negrito** ou __negrito__ para negrito.
- **Listas:** Crie listas não ordenadas com -, + ou *, e listas ordenadas com números seguidos de um ponto.
- **Links:** Crie links usando texto do link.
- **Imagens:** Insira imagens usando !texto alternativo.
- **Blocos de código:** Use três crases (```) para criar blocos de código.

Um dos exemplos mais simples de utilização de markdown é a descrição dos perfis do GitHub, utilizando o arquivo README.md. Esses arquivos no modelo markdown são utilizados para documentar algum código no GitHub, fazendo a identidade do perfil, como pode ser visto na Figura 5.

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 1 2024

Figura 5: exemplo de arquivo no formato markdown (README.md)



```
teste.md > # About Me:
# About Me:
I'm currently working on: Level 2 support for the MagaluPay Empresas application at MagaluBank, one of the financial service fronts of Magazine Luiza.<br><br>
I'm currently learning: Backend development using node.js and express. I'm also studying architecture based on APIs and microservices, along with automated testing (unit, integration).<br><br>
I'm looking to collaborate on: Join the backend development area and constantly improve myself.<br><br>
I'm looking for help with: Understand more about containers like Docker and Kubernetes.<br><br>
Fun fact: I really like motorsport and one of my biggest idols are Ayrton Senna and Lewis Hamilton. I also like reading management and self-knowledge/self-help books.
<br><br>

<h1> Tech Stack </h1>
<div style="display: inline_block">
  <img src="https://img.shields.io/badge/JavaScript-F7DF1E?style=for-the-badge&logo=javascript&logoColor=black">
  <img src="https://img.shields.io/badge/Node.js-43853D?style=for-the-badge&logo=node.js&logoColor=white">
  <img src="https://img.shields.io/badge/TypeScript-007ACC?style=for-the-badge&logo=typescript&logoColor=white">
  <img src="https://img.shields.io/badge/React-20232A?style=for-the-badge&logo=react&logoColor=61DAFB">
  <img src="https://camo.githubusercontent.com/2ff25d83c264f3b91f68bdb7261f9544fdf8fa7ee19b7028857ac9e82b42539a/68747470733a2f2f696d672e736869656c64732e696f2f62616467652f457870726573732e6a732d343034343533f7374796c653d666f722d7468652d62616467652266c6f676f3d65787072657373">
</div>
<br>
<h1> DevOps </h1>
<div style="display: inline_block">
  <img src="https://img.shields.io/badge/Git-E34F26?style=for-the-badge&logo=git&logoColor=white">
  <img src="https://img.shields.io/badge/GitHub-100000?style=for-the-badge&logo=github&logoColor=white">
</div>

<br>
<h1> Social </h1>
<a href="https://instagram.com/luchenrique__" target="_blank"><img src="https://img.shields.io/badge/Instagram-E4405F?style=for-the-badge&logo=instagram&logoColor=white"></a>
<a href="https://www.linkedin.com/in/1lucashenrique/" target="_blank"><img src="https://img.shields.io/badge/LinkedIn-0077B5?style=for-the-badge&logo=linkedin&logoColor=white" target="_blank"></a>
<a href="https://medium.com/@luchenrique" target="_blank"><img src="https://img.shields.io/badge/Medium-12100E?style=for-the-badge&logo=medium&logoColor=white"></a>
<a href="https://www.cloudskillsboost.google/public_profiles/4d807abf-2dbc-44fb-9702-88a157cb2b60"><img src="https://img.shields.io/badge/Google_Cloud-4285F4?style=for-the-badge&logo=google-cloud&logoColor=white"></a>
```

Fonte: Autoria própria

5 Resultados da solução proposta

Nessa seção será apresentada a solução proposta para validação de hyperlinks de artigos publicados na internet.

5.1 Arquivo de leitura

Para que a função regex possa percorrer e capturar os links de um arquivo lido e validá-los, é importante que o arquivo lido esteja no modelo MarkDown, que, de acordo com Habbema (2023) é “uma linguagem de marcação que permite que você formate texto de maneira simples usando uma sintaxe fácil de ler e escrever”. Na Figura 6 está apresentado o exemplo de um arquivo no modelo MarkDown, utilizado para os testes do algoritmo.

5.2 Arquivo index.js

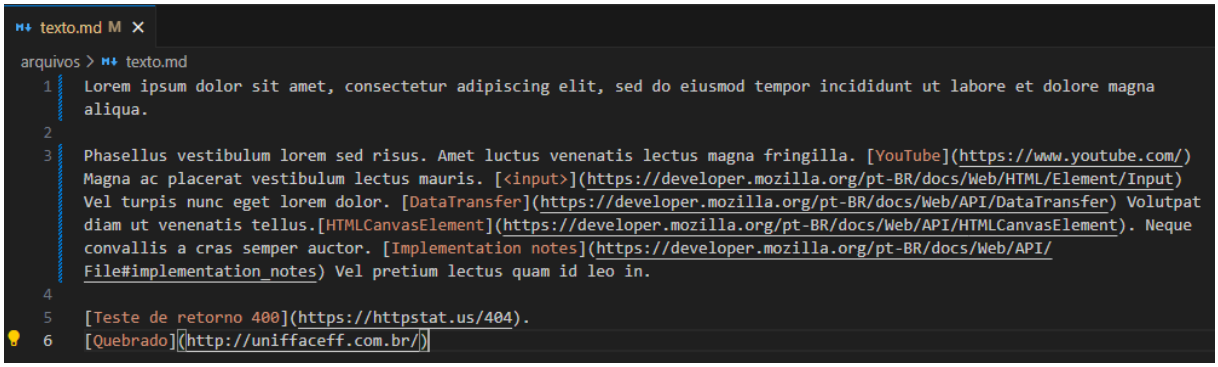
Esse arquivo é composto por três funções e uma das principais é a **extraiLinks**, pois nela é definida a expressão regex, a partir da qual o algoritmo irá fazer a captura dos links.

A segunda função que compõe o arquivo **index.js** é a função **trataErro,** que é utilizada para capturar possíveis erros durante a leitura do arquivo. E por fim, também compondo o arquivo **index.js**, existe a função **pegaArquivo**, que é utilizada

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 1 2024

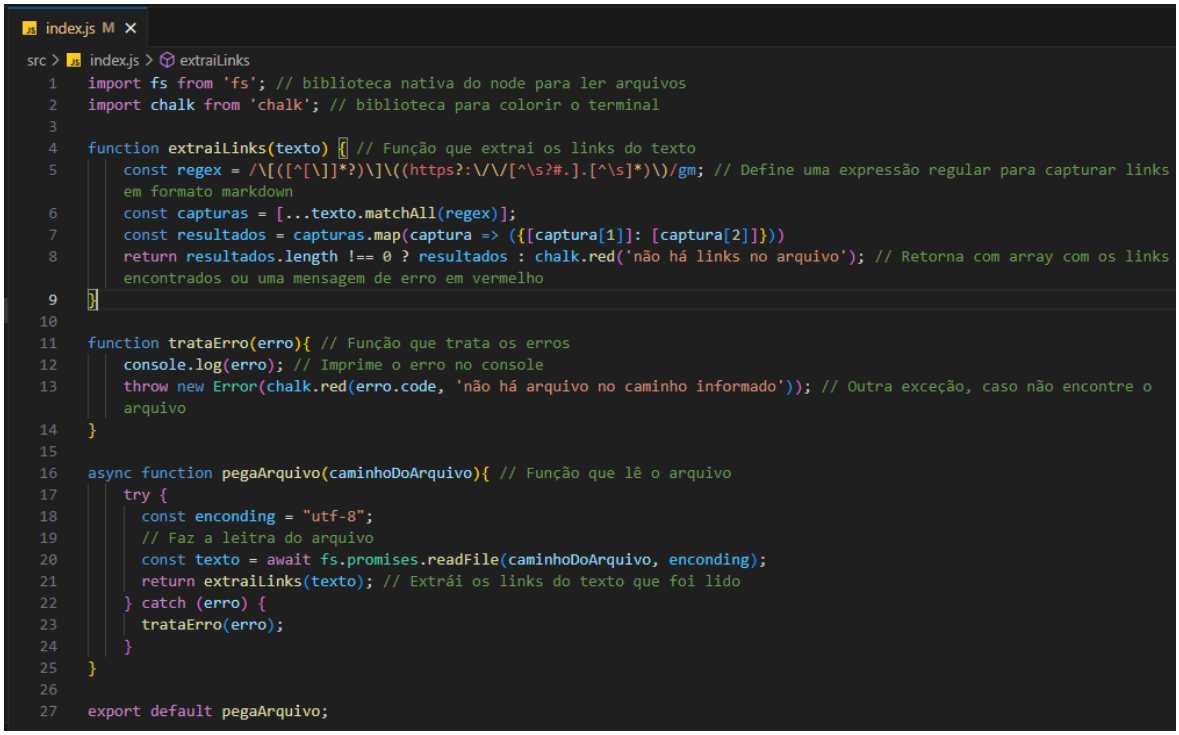
para encontrar o arquivo que deverá ser lido e dentro dessa função, possui uma função *callback* que chama a função **extraiLinks**. Na Figura 7 estão apresentados os códigos do arquivo **index.js**.

Figura 6: Arquivo no modelo MarkDown que será percorrido e lido pelo algoritmo.



Fonte: Autoria própria

Figura 7: Código do arquivo index.js



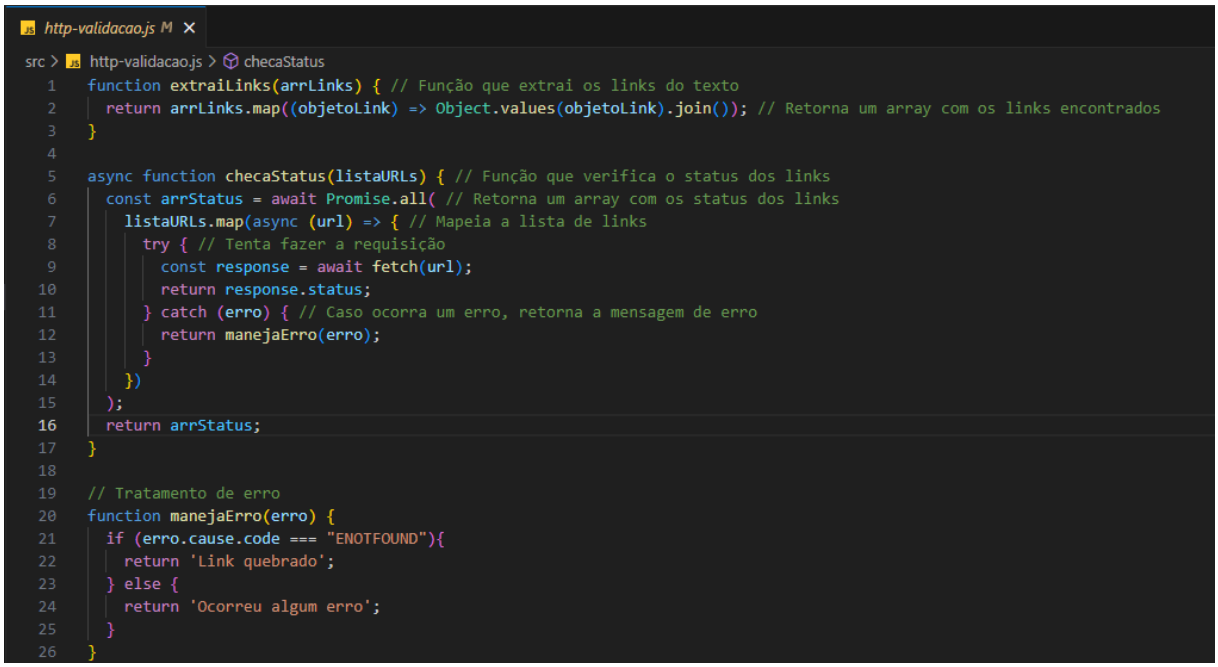
Fonte: Autoria própria

5.3 Arquivo de extração de link

Após o link ser encontrado, a função **checaStatus** (apresentada na Figura 8), irá testar se o link está ativo (retornando o código 200), se o link não está ativo (retornando o código 404) ou se o link está quebrado, chamando a função **manejaErro,** que retornará a string '**Link quebrado'** (no caso de tentar fazer a leitura

e não encontrar o status code) ou,caso haja algum outro erro, retornará a string **‘Ocorreu algum erro’**.

Figura 8: Função checaStatus e manejaErro do arquivo de validação

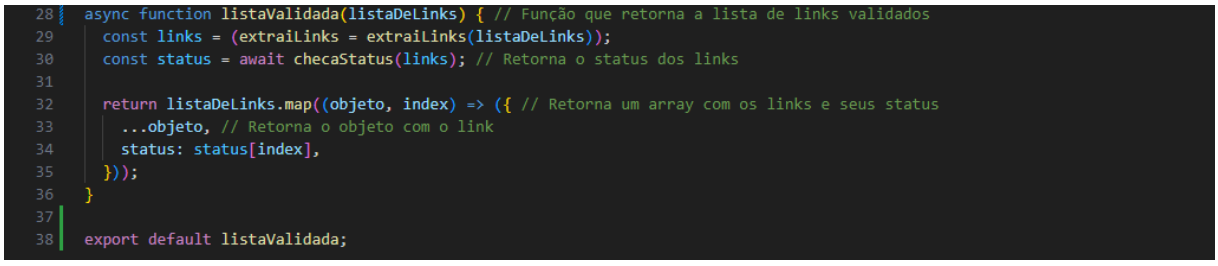


Fonte: Autoria própria

.

A função **listaValidada** (apresentada na Figura 9) irá retornar a lista dos links que foram encontrados dentro do arquivo markdown que foi lido.

Figura 9: Retorno da lista após os links serem encontrados



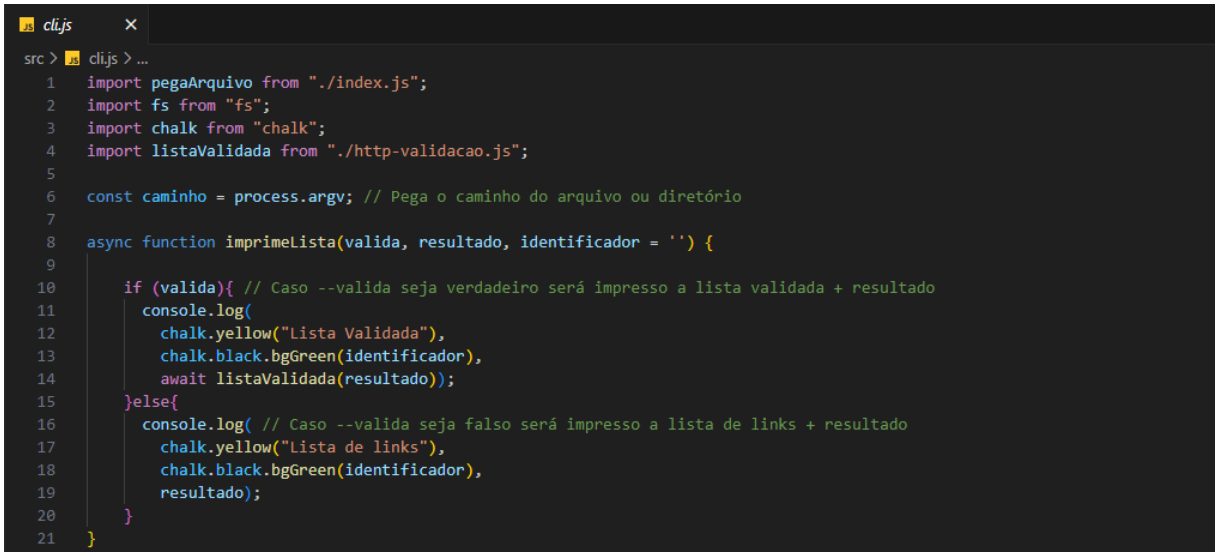
Fonte: Autoria própria

5.4 Arquivo de validação

Arquivo da figura 10 irá efetuar a validação da lista de array. Caso “valida” for verdadeiro, será retornado o resultado da Figura 12, porém caso seja falso, será retornada a lista apresentada na Figura 13.

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 1 2024

Figura 10: Validação de entrega no terminal

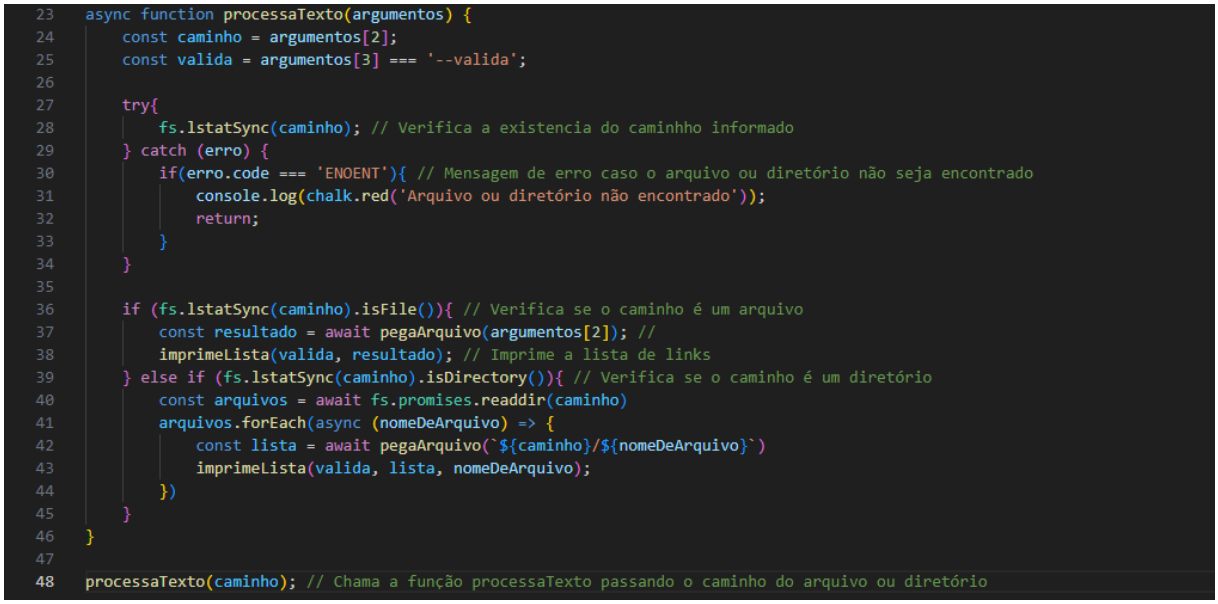


```
cli.js
src > cli.js > ...
1  import pegaArquivo from "./index.js";
2  import fs from "fs";
3  import chalk from "chalk";
4  import listaValidada from "./http-validacao.js";
5
6  const caminho = process.argv; // Pega o caminho do arquivo ou diretório
7
8  async function imprimeLista(valida, resultado, identificador = '') {
9
10     if (valida){ // Caso --valida seja verdadeiro será impresso a lista validada + resultado
11       console.log(
12         chalk.yellow("Lista Validada"),
13         chalk.black.bgGreen(identificador),
14         await listaValidada(resultado));
15     }else{
16       console.log( // Caso --valida seja falso será impresso a lista de links + resultado
17         chalk.yellow("Lista de links"),
18         chalk.black.bgGreen(identificador),
19         resultado);
20     }
21 }
```

Fonte: Autoria própria

A função processaTexto (Figura 11) verifica se o caminho é um arquivo ou diretório, processa seus links e os imprime em formato de lista. Caso for passado com flag --valida, a lista também será impressa no terminal, porém acompanhada com o *status code* do respectivo link.

Figura 11: Verifica o caminho passado



```
23 async function processaTexto(argumentos) {
24     const caminho = argumentos[2];
25     const valida = argumentos[3] === '--valida';
26
27     try{
28         fs.lstatSync(caminho); // Verifica a existencia do caminhho informado
29     } catch (erro) {
30         if(erro.code === 'ENOENT'){ // Mensagem de erro caso o arquivo ou diretório não seja encontrado
31             console.log(chalk.red('Arquivo ou diretório não encontrado'));
32             return;
33         }
34     }
35
36     if (fs.lstatSync(caminho).isFile()){ // Verifica se o caminho é um arquivo
37         const resultado = await pegaArquivo(argumentos[2]); //
38         imprimeLista(valida, resultado); // Imprime a lista de links
39     } else if (fs.lstatSync(caminho).isDirectory()){ // Verifica se o caminho é um diretório
40         const arquivos = await fs.promises.readdir(caminho)
41         arquivos.forEach(async (nomeDeArquivo) => {
42             const lista = await pegaArquivo(`${caminho}/${nomeDeArquivo}`)
43             imprimeLista(valida, lista, nomeDeArquivo);
44         })
45     }
46 }
47
48 processaTexto(caminho); // Chama a função processaTexto passando o caminho do arquivo ou diretório
```

Fonte: Autoria própria

Ao executar o comando “npm run cli” no terminal, será exibido um array com o nome de “Lista de links” e abaixo será listado o nome, juntamente com a URL do link capturado.

RECA
REVISTA ELETRÔNICA DE COMPUTAÇÃO APLICADA
Vol. 5 Nro. 1 2024

Figura 12: Execução do comando “npm run cli”

```
Lista de links  [
  { YouTube: [ 'https://www.youtube.com/' ] },
  {
    '<input>': [
      'https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/Input'
    ]
  },
  {
    DataTransfer: [ 'https://developer.mozilla.org/pt-BR/docs/Web/API/DataTransfer' ]
  },
  {
    HTMLCanvasElement: [
      'https://developer.mozilla.org/pt-BR/docs/Web/API/HTMLCanvasElement'
    ]
  },
  {
    'Implementation notes': [
      'https://developer.mozilla.org/pt-BR/docs/Web/API/File#implementation_notes'
    ]
  },
  { 'Teste de retorno 400': [ 'https://httpstat.us/404' ] },
  { Quebrado: [ 'http://uniffaceff.com.br/' ] }
]
```

Fonte: Autoria própria

Ao executar o comando “npm run cli --valida” será exibido uma lista dos links que foram capturado, juntamente com o seu “Status code”, podendo exibir o código de 200, para os links que estão ativos que o acesso foi permitido. O código 404, para um link em que existe o domínio da página, porém o caminho passado na URL não existe e o erro de “Link quebrado” que será exibido quando a URL não for existente.

Figura 13: Execução do comando “npm run cli --valida”

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                    powershell
Lista Validada  [
  { YouTube: [ 'https://www.youtube.com/' ], status: 200 },
  {
    '<input>': [
      'https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/Input'
    ],
    status: 200
  },
  {
    DataTransfer: [ 'https://developer.mozilla.org/pt-BR/docs/Web/API/DataTransfer' ],
    status: 200
  },
  {
    HTMLCanvasElement: [
      'https://developer.mozilla.org/pt-BR/docs/Web/API/HTMLCanvasElement'
    ],
    status: 404
  },
  {
    'Implementation notes': [
      'https://developer.mozilla.org/pt-BR/docs/Web/API/File#implementation_notes'
    ],
    status: 200
  },
  {
    'Teste de retorno 400': [ 'https://httpstat.us/404' ],
    status: 404
  },
  {
    Quebrado: [ 'http://uniffaceff.com.br/' ],
    status: 'Link quebrado'
  }
]
```

Fonte: Autoria própria

6 Conclusão

Esse projeto se propôs a abordar a importância de validar os hiperlinks dentro de artigos ou sites na web. Os resultados obtidos mostram que um simples algoritmo é capaz de validar todos os hiperlinks encontrados, aumentando a eficácia dos documentos digitais.

Por meio de um desenvolvimento de algoritmo com ferramentas de capturas de hiperlinks, foi possível criar um protótipo que auxilia os usuários a ter um maior controle de seus artigos publicados.

Vale ressaltar que, como se trata de um algoritmo com saída apenas no terminal, foram utilizadas algumas bibliotecas, que auxiliam a visualização das informações, para que o usuário não se perca nas informações apresentadas no final da execução.

No entanto, é reconhecido que existem desafios e limitações a serem enfrentados, como a necessidade de adaptar um determinado texto, para a estrutura markdown e limitação nos tipos de dados a serem retornados no terminal. Em trabalho futuros, serão incluídos uma escalabilidade no projeto, permitindo que o algoritmo capture um texto em qualquer formato e o transforme automaticamente na estrutura markdown para que o algoritmo possa efetuar a leitura.

Referências

ALVES, Valdir. Guia para o File System do Node.js. 2023. Disponível em <https://www.dio.me/articles/guia-para-o-file-system-do-nodejs>. Acesso em 31 mai. de 2024

BARBOSA, Ana. CAVALCANTI, Alexsandro. Web Scraping e Análise de dados. CONAPESC
Disponível em <https://www.editorarealize.com.br/editora/anais/conapesc/2020/TRABALHO_EV138_MD4_SA24_ID1284_24112020001516.pdf> Acesso em 30 mai. de 2024.

BESSA, André. Node.JS: o que é, como funciona esse ambiente de execução JavaScript e um Guia para iniciar. 2023. Disponível em <https://www.alura.com.br/artigos/node-js?srsltid=AfmBOor7nLwKYm_H4HIWjamudApE0q1uSEMl128VIEOYXgqQJzx4ZCPm> Acesso em 31 mai. de 2024

CLOUDMARKET. 2022. O que é: Hiperlink. Disponível em: <https://www.cloudmarket.com.br/marketing-digital/blog/glossario/o-que-e-hiperlink/>. Acesso em 15 mai. 2024.

KINSTA. Entendendo o Módulo do Sistema de Arquivos Node.js (FS). 2023. Disponível em: <https://kinsta.com/pt/base-de-conhecimento/nodejs-fs/>. Acesso em 20 jul. de 2024

MAIA, Victor. Top 20 Ferramentas de Web Crawling para raspar dados de maneira rápida. Medium, 2020. Disponível em <https://medium.com/@victormaiam/top-20-ferramentas-de-web-crawling-para-raspar-dados-de-maneira-r%C3%A1pida-7bdc44784385> Acesso em 31 mai. de 2024

MARKDOWN Guide. Markdown Guide. Disponível em: <https://www.markdownguide.org/>. Acesso em 20 jul. de 2024.

MDN contributors. Criando Hyperlinks. 2023. Disponível em: <https://developer.mozilla.org/pt-BR/docs/learn/html/introduction_to_html/creating_hyperlinks>. Acesso em 15 mai. 2024.

MDN web docs. Expressões Regulares. 2024. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Regular_expressions>. Acesso em 31 de mai. de 2024

MIRANDA, Luiz. JavaScript e TypeScript do básico ao avançado JS/TS. 2024. Disponível em: <https://www.udemy.com/course/curso-de-javascript-moderno-do-basico-ao-avancado/?couponCode=ACCAGE0923>. Acesso em: 27 jul. de 2024.

REGEX TUTORIAL. Tutorial Regex - Uma folha de dicas com exemplos! 2023. Disponível em: <https://regextutorial.org/>. Acesso em 21 ago. 2024

SILVEIRA, Paulo. O que é Markdown? E para que serve?. 2022. Disponível em: <https://www.alura.com.br/artigos/como-trabalhar-com-markdown?srsltid=AfmBOor98G5_29U3m14ouBDdT_r9CpKuMK_VKWToQ-WLSRBY2RtiPmFN> . Acesso em 20 jul. de 2024.