Renato Zampiere Maciente

WEB SCRAPING E EXTRAÇÃO DE DADOS EM ESTRUTURAS HTML COM VISUALIZAÇÃO EM INTERFACE WEB: UM ESTUDO DE CASO

Formiga - MG

2025

Renato Zampiere Maciente

WEB SCRAPING E EXTRAÇÃO DE DADOS EM ESTRUTURAS HTML COM VISUALIZAÇÃO EM INTERFACE WEB: UM ESTUDO DE CASO

Monografia do trabalho de conclusão de curso apresentado ao Instituto Federal Minas Gerais - Campus Formiga, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais

Campus Formiga

Ciência da Computação

Orientador: Walace de Almeida Rodrigues

Formiga - MG

2025

Maciente, Renato Zampiere

M152w Web Scraping e extração de dados em estruturas HTML com visualização em Interfaces WEB: um estudo de caso / Renato Zampiere Maciente -- Formiga : IFMG, 2025.

64 p. : il.

Orientador: Prof. Me. Walace de Almeida Rodrigues

Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – *Campus* Formiga.

1. Counter-Strike. 2. Páginas WEB. 3. Skins virtuais. 4. WEB Scraping. 5. Selenium. 6. Webdriver. I. Rodrigues, Walace de Almeida. II. Título.

CDD 004

Ficha catalográfica elaborada pela Bibliotecária Lívia Renata Santos- CRB/6-2561

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS
Campus Formiga
Diretoria de Ensino
Docência Área Acadêmica de Computação
Rua São Luiz Gonzaga, s/n - Bairro São Luiz - CEP 35570-000 - Formiga - MG
- www.ifmg.edu.br

DECLARAÇÃO

MEC-SETEC
INSTITUTO FEDERAL MINAS GERAIS - Campus Formiga
Curso de Ciência da Computação

WEB SCRAPING E EXTRAÇÃO DE DADOS EM ESTRUTURAS HTML COM VISUALIZAÇÃO EM INTERFACE WEB: UM ESTUDO DE CASO

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Minas Gerais - Campus Formiga, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

APROVADO em: 23 de julho de 2025.

BANCA EXAMINADORA

Prof.º Walace de Almeida Rodrigues (orientador, IFMG)

Profª. Danielle Costa de Oliveira (IFMG)

Prof.º Everthon Valadão dos Santos (IFMG)

Formiga, 23 de julho de 2025.

Documento assinado eletronicamente por **Walace de Almeida Rodrigues**, **Professor**, em 23/07/2025, às 11:53, conforme Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Everthon Valadao dos Santos**, **Professor**, em 23/07/2025, às 11:54, conforme Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Danielle Costa de Oliveira**, **Professora**, em 23/07/2025, às 14:39, conforme Decreto nº 10.543, de 13 de novembro de 2020.

Agradecimentos

Gostaria de agradecer a Deus e aos meus pais, que me deram todo suporte, apoio e carinho para chegar a esse ponto nessa jornada, à minha família, irmã e cunhado, que sempre deram exemplo de que com luta e esforço conseguimos buscar algo novo e inesperado, à instituição IFMG Campus Formiga pela oportunidade de ter conhecido excelentes professores e profissionais, ao professor Walace de Almeida Rodrigues que se disponibilizou de todo seu conhecimento, experiência e paciência nesse caminho, aos grandes amigos Rafael e Yuri, e por fim, uma das peças mais importantes, a Maga da Computação moderna Rafaela Martins Vieira, sem o seu apoio e sua luz irradiando os meus dias, esse caminho se tornaria muito mais frio e nebuloso.

"Educar verdadeiramente não é ensinar fatos novos ou enumerar fórmulas prontas, mas sim preparar a mente para pensar." (Albert Einstein)

Resumo

O mercado de itens virtuais em jogos eletrônicos tem se expandido significativamente, em títulos como *Counter-Strike*, onde skins podem alcançar alto valor comercial. Diante da oscilação constante de preços entre plataformas, o acesso a dados atualizados é fundamental para subsidiar decisões de compra. Este trabalho apresenta um estudo de caso e a implementação de um sistema automatizado para coleta e centralização, em tempo real, de informações sobre skins de *Counter-Strike*, a partir de cinco plataformas especializadas. A extração foi realizada com a biblioteca Selenium, em conjunto com WebDriver e undetected_chromedriver, visando contornar sistemas anti-bot. As informações coletadas, nome do item, preço, tipo de acabamento, nível de desgaste (float) e imagem, foram extraídas com base em seletores específicos de tags HTML, como `<div>`, `<span>` e `<img>`, e tratadas com a biblioteca pandas para padronização e posterior análise. A visualização dos dados foi realizada por meio de uma interface interativa desenvolvida com Dash Bootstrap, facilitando a navegação e análise por parte do usuário. A aplicação também foi otimizada com o uso de threads, permitindo maior agilidade no processo de coleta, especialmente em ambientes com múltiplas fontes. Adicionalmente, foi implementado um modo de raspagem dinâmica, voltado a usuários com conhecimentos prévios em estruturas HTML. Esse modo permite a personalização do processo de coleta, possibilitando que o usuário defina manualmente os seletores HTML, o tipo de rolagem da página (paginada, infinita ou alternada), e a URL de destino. Tal funcionalidade expande a aplicabilidade da ferramenta, permitindo que novos sites sejam integrados ao sistema conforme os mesmos princípios técnicos adotados no estudo principal. Entre os principais desafios, destacam-se o carregamento assíncrono via JavaScript e mecanismos de proteção contra automação, superados com renderização forçada e uso de navegação não detectável. A proposta visa facilitar o acesso e a análise dos dados pelo usuário, promovendo maior eficiência na interpretação dos valores e características dos itens ofertados.

Palavras-chave: Counter-Strike, Páginas WEB, Skins Virtuais, Web Scraping.

Abstract

The virtual item market in electronic games has grown significantly, particularly in titles such as Counter-Strike, where skins can reach high commercial value. Given the constant price fluctuations across platforms, access to real-time data is essential to support purchasing decisions. This study presents a case study and the implementation of an automated system for the real-time collection and centralization of Counter-Strike skin data from five specialized platforms. Data extraction was conducted using the Selenium library, combined with WebDriver and undetected_chromedriver, to bypass anti-bot mechanisms. Information such as item name, price, finish type, wear level (float), and image was extracted based on specific HTML tag selectors such as , , and <img>, and subsequently processed using the pandas library for standardization and analysis. Data visualization was enabled through an interactive interface developed with Dash Bootstrap, allowing for user-friendly navigation and analysis. The application was further optimized using multithreading to enhance scraping efficiency, particularly in environments with multiple sources. Additionally, a dynamic scraping mode was implemented for users with prior knowledge of HTML structure, allowing manual customization of HTML selectors, page scroll type (paginated, infinite, or alternating), and target URL. This feature broadens the tool's applicability by enabling integration with new websites following the same technical principles used in the main study. Among the key challenges addressed were asynchronous JavaScript content loading and automation detection mechanisms, which were overcome through forced rendering and stealth navigation techniques. The proposed solution aims to facilitate user access and analysis of data, promoting greater efficiency in interpreting item values and characteristics.

Keywords: Counter-Strike, Web Pages, Virtual Skins, Web Scraping.

Lista de ilustrações

Lista de abreviaturas e siglas

APIs	*Application Programming Interfaces (Interfaces de Programação de Aplicações)*
ARM	*Advanced RISC Machine*
BRL	*Brazilian Real (Real Brasileiro)*
CPU	*Central Processing Unit*
CS:GO	*Counter-Strike: Global Offensive*
CSS	*Cascading Style Sheets*
CSV	*Comma-Separated Values*
DOM	*Document Object Model*
GB	*Gigabytes*
GPU	*Graphics Processing Unit*
HTML	*HyperText Markup Language*
JSON	*JavaScript Object Notation*
macOS	*Macintosh Operating System*
RAM	*Random Access Memory*
SSD	*Solid-State Drive*
URL	*Uniform Resource Locator*
USD	*United States Dollar (Dólar dos Estados Unidos)*
WEB	*World Wide Web*

Sumário

1 Introdução

Nas últimas décadas, os jogos eletrônicos deixaram de ser apenas uma forma de entretenimento e passaram a movimentar um mercado bilionário que abrange competições profissionais, produção de conteúdo, e principalmente, a comercialização de itens virtuais (CASTILHO, 2023). Um dos exemplos mais emblemáticos desse fenômeno é o jogo *Counter-Strike: Global Offensive* (CS:GO), que popularizou o uso de "skins" personalizações estéticas de armas que podem ser adquiridas, trocadas ou vendidas em plataformas especializadas.

Esses itens, apesar de não influenciarem diretamente na jogabilidade, possuem forte apelo estético e estão associados à expressão de identidade e status por parte dos jogadores, contribuindo para o sentimento de satisfação pessoal durante a experiência de jogo. Com isso, passaram a ter valor real no mercado digital, com preços que variam significativamente conforme a oferta, demanda e raridade (**??**). Acompanhar os diferentes modelos de skins e suas respectivas variações de preço tornou-se uma atividade cada vez mais complexa e pouco eficiente, exigindo que o usuário acesse manualmente diversas plataformas de venda. Tal processo é frequentemente demorado, repetitivo e suscetível a erros, especialmente em virtude da ausência de padronização na apresentação das informações entre os diversos sites especializados.

Para viabilizar uma solução elegante para a coleta automatizada de dados sobre skins digitais, este trabalho propõe o desenvolvimento de uma aplicação baseada em técnicas de raspagem de dados *(web scraping)*. Inicialmente, foi conduzido um estudo detalhado das estruturas das páginas web de cinco plataformas especializadas, considerando as particularidades na organização de seus elementos e nos mecanismos de carregamento de conteúdo.

Durante a etapa de desenvolvimento do estudo, constatou-se que os sites analisados apresentavam variações significativas em seus formatos de navegação, o que exigiu a adoção de abordagens diferenciadas para garantir a obtenção integral dos dados. Além disso, foram identificados obstáculos que dificultavam o acesso automatizado às informações, tornando necessário o uso de estratégias adaptativas para contornar tais limitações e assegurar a continuidade da coleta.

Os dados obtidos foram devidamente organizados e padronizados, possibilitando sua posterior análise e visualização de forma eficiente. Por fim, as informações são disponibilizadas em uma interface interativa, que visa facilitar a navegação e a compreensão por parte do usuário. A solução contempla ainda um sistema dinâmico de raspagem, que permite a personalização dos critérios de extração, ampliando sua aplicabilidade para

contextos além do universo das skins de jogos eletrônicos.

1.1 Justificativa

A crescente valorização do mercado de jogos eletrônicos impulsionou o surgimento de economias digitais baseadas em bens virtuais, como as skins, personalizações visuais de itens amplamente utilizadas em jogos como *Counter-Strike* (CORPORATION, 2025) (TOMIĆ, 2017). Inicialmente estéticas, essas personalizações passaram a ser comercializadas como ativos digitais, com valores expressivos em marketplaces especializados (VALVERDE, 2023). Nesse contexto, jogadores e colecionadores têm demonstrado interesse em acompanhar continuamente os preços e atributos dessas skins, visando identificar oportunidades de compra mais vantajosas (JÚNIOR *et al.*, 2023). No entanto, essa tarefa ainda é realizada, em grande parte, de forma manual, exigindo o acesso repetido a diferentes plataformas, o que torna o processo moroso e suscetível a erros, sobretudo diante da ausência de padronização dos dados e da existência de barreiras técnicas à automação.

É relevante destacar que as técnicas exploradas neste trabalho, como a renderização forçada de conteúdo, a navegação não detectável e a implementação de diferentes mecanismos de rolagem, representam contribuições significativas, com potencial de aplicação em diferentes contextos de desenvolvimento. Ainda que não representem a totalidade das abordagens possíveis, essas estratégias oferecem subsídios relevantes para investigações futuras e podem colaborar com o aprofundamento das discussões no campo da raspagem automatizada de dados.

1.2 Objetivos

1.2.1 Objetivo Geral

Desenvolver uma aplicação capaz de automatizar a coleta e a apresentação de dados referentes às skins do jogo *Counter-Strike* (CORPORATION, 2025), provenientes de diferentes plataformas de venda, integrando essas informações em um painel visual interativo. A proposta visa facilitar o acesso e a análise dos dados pelo usuário, promovendo maior eficiência na interpretação dos valores e características dos itens ofertados.

1.2.2 Objetivos Específicos

1. Estudar a estrutura HTML de diferentes sites de venda de skins para compreender os padrões de organização dos dados;
2. Estudar e implementar técnicas de *web scraping* utilizando a biblioteca Selenium, com foco na simulação de navegação humana;

3. Utilizar bibliotecas auxiliares para o tratamento, organização e padronização dos dados extraídos;

4. Construir um painel interativo com a biblioteca Dash Bootstrap, permitindo a personalização da busca por parte do usuário;

5. Desenvolver um modo de raspagem dinâmica, no qual o próprio usuário possa definir seletores HTML, tipos de rolagem e preferências de navegador;

6. Avaliar a estabilidade, desempenho e aplicabilidade da ferramenta em contextos reais de uso.

2 Fundamentação Teórica

2.1 Fundamentos da extração de dados da web (Web Scraping)

A extração de dados da web, ou *web scraping*, é uma técnica voltada à coleta automatizada de informações disponíveis publicamente em páginas da internet. Seu principal objetivo é converter conteúdos apresentados de forma visual para usuários humanos em dados estruturados, que podem ser organizados, analisados e aplicados em diferentes contextos, como pesquisas, sistemas digitais e processos de tomada de decisão (FARIAS *et al.*, 2021).

O processo de *web scraping* envolve etapas bem definidas. Inicialmente, realiza-se o envio de requisições HTTP para acessar as páginas-alvo. Em seguida, é feita a interpretação da estrutura do documento HTML retornado, que organiza os elementos visuais da página em uma hierarquia conhecida como DOM. Com base nessa estrutura, utiliza-se seletores CSS, XPath ou outras abordagens para localizar e extrair informações específicas, como textos, imagens, preços, datas, entre outros (KHDER, 2021). Os dados extraídos são então processados e armazenados em formatos apropriados, como arquivos CSV, JSON, tendo em vista a natureza estruturada das informações e a necessidade de interoperabilidade com outras ferramentas analíticas. O formato CSV é amplamente utilizado por sua simplicidade e compatibilidade com planilhas e softwares estatísticos, o JSON é indicado para a representação de dados hierárquicos e intercâmbio entre sistemas web (GOHIL *et al.*, 2022).

A popularização do *web scraping* está associada à crescente necessidade de obtenção e análise de grandes volumes de dados disponíveis em ambientes digitais, especialmente quando não são oferecidas APIs oficiais. Essa técnica é utilizada em setores como comércio eletrônico, análise de mercado, jornalismo de dados, monitoramento de redes sociais, mineração de conteúdo e pesquisa acadêmica (BHATT *et al.*, 2023).

Embora o *web scraping* seja uma ferramenta eficaz e versátil para a aquisição de dados em larga escala, seu uso deve ser pautado por princípios legais e éticos. Muitos sites estabelecem diretrizes específicas quanto ao acesso automatizado por meio do arquivo robots.txt, que indica quais seções do site podem ou não ser exploradas por agentes automatizados, além de orientações presentes nos termos de uso. Barreiras técnicas como CAPTCHAs, limitação de requisições *(rate limiting)* e sistemas de proteção, como o Cloudflare, são frequentemente empregados para restringir esse tipo de acesso. Assim, é fundamental que a prática do *scraping* seja conduzida com responsabilidade, respeitando as políticas estabelecidas pelos sites-alvo e as legislações vigentes relacionadas à proteção de

dados. Quando utilizada de maneira ética, essa técnica desempenha um papel relevante no desenvolvimento de soluções digitais e na geração de conhecimento a partir das informações disponíveis na web (BROWN *et al.*, 2024).

2.2 Navegação Controlada em Páginas Dinâmicas com WebDriver

O WebDriver é empregado para simular interações humanas com páginas da web, possibilitando a navegação em sites que apresentam comportamentos dinâmicos (GARCÍA *et al.*, 2021). Essa abordagem permite acessar conteúdos carregados de forma assíncrona, bem como manipular elementos interativos, como botões, menus e campos de entrada, além de percorrer páginas com rolagem infinita ou múltiplas etapas de navegação (SALUNKE, 2014).

Diferentemente das técnicas de extração que atuam diretamente sobre o código HTML estático, o uso do WebDriver possibilita a reprodução da experiência de navegação de um usuário real. Esse recurso é particularmente relevante em contextos onde a apresentação das informações depende de scripts JavaScript executados dinamicamente (RAGHAVENDRA, 2020). A ferramenta permite controlar o navegador de forma programada, realizando ações como cliques, preenchimento de formulários, espera por carregamentos e coleta de dados exibidos apenas após determinadas interações.

Esse tipo de abordagem torna-se especialmente vantajoso em cenários nos quais as informações não estão disponíveis de imediato no carregamento da página, mas são exibidas conforme o usuário interage com o site (BRITO *et al.*, 2024). Nesses casos, o WebDriver possibilita simular essas interações, como rolagem, cliques e entrada de dados, permitindo o acesso a conteúdos que não seriam extraídos por métodos mais diretos. Com isso, amplia-se significativamente a capacidade de coleta em ambientes web mais complexos, nos quais o conteúdo é carregado sob demanda e requer um nível maior de interação para ser completamente revelado.

2.3 Estrutura de páginas web: HTML, DOM e CSS Selectors

A compreensão da estrutura das páginas web é um requisito fundamental para o desenvolvimento de sistemas de extração automatizada de dados, especialmente os baseados em técnicas de web scraping, que dependem diretamente da organização e codificação das informações no ambiente *online* (KADAM; PAKLE, 2014). Em geral, as páginas da internet são construídas utilizando linguagens de marcação e estilização, sendo a HTML a principal responsável por definir a estrutura lógica e visual de um documento web. Essa linguagem organiza o conteúdo por meio de uma hierarquia de elementos conhecidos como tags, que delimitam títulos, parágrafos, listas, tabelas, imagens, links, entre outros

componentes da interface (MOHOD; MEGHA, 2014).

A representação interna de um documento HTML, após seu carregamento em um navegador, é convertida em uma estrutura denominada DOM. Trata-se de uma interface padronizada que organiza o conteúdo da página como uma árvore hierárquica de nós, na qual cada elemento, como uma tag "div", "p", entre outras, é tratado como um objeto individual. Essa estrutura permite que linguagens de programação, como JavaScript ou Python, interajam diretamente com os elementos da página por meio de bibliotecas específicas, viabilizando a leitura, modificação ou remoção dinâmica de conteúdos (WOOD *et al.*, 1998). Ao oferecer essa representação estruturada, o DOM torna-se essencial para a manipulação e extração programática de informações contidas em páginas web.

A identificação precisa de elementos dentro da árvore DOM é realizada, em grande parte, por meio dos CSS Selectors. Esses seletores consistem em padrões sintáticos utilizados para localizar elementos com base em atributos, classes, identificadores ou relações hierárquicas estabelecidas no documento HTML. Essa abordagem é amplamente adotada em aplicações de automação e extração de dados, pois permite selecionar de forma específica conteúdos relevantes, como preços, nomes, descrições e imagens. A expressividade e flexibilidade dos seletores possibilitam buscas refinadas, mesmo em estruturas altamente aninhadas ou complexas, tornando-os ferramentas essenciais para a manipulação programática de páginas web (ROLE; VERDRET, 1999).

Seletores como "classe", "id", "div", ou "atributo=valor", são utilizados para acessar elementos com base em suas classes, identificadores, estrutura hierárquica ou atributos específicos, respectivamente (ÁLVAREZ *et al.*, 2008). A aplicação direta desses seletores sobre a estrutura DOM permite construir rotinas de extração de dados altamente específicas e eficientes, adaptáveis a diferentes arquiteturas de página. Essa capacidade torna os CSS Selectors fundamentais para o desenvolvimento de sistemas de coleta automatizada robustos e escaláveis, mesmo em contextos com alto grau de complexidade estrutural (NUNES; DORNELES, 2022).

Nesse contexto, o domínio sobre os fundamentos do HTML, da estrutura do DOM e do uso de CSS Selectors constitui a base técnica indispensável para a construção de soluções de extração de dados da web. Esses conhecimentos permitem não apenas identificar e acessar dados de interesse com precisão, mas também adaptar os sistemas de scraping a diferentes arquiteturas de páginas, facilitando a reutilização e a manutenção do código frente a alterações estruturais nos sites analisados.

2.4 Tipos de carregamento de conteúdo em páginas web

A forma como o conteúdo é disponibilizado em páginas web exerce influência direta sobre as técnicas de extração de dados a serem adotadas. Compreender os diferentes

mecanismos de carregamento é essencial para o desenvolvimento de abordagens eficazes de *web scraping*, uma vez que cada estrutura apresenta características específicas e limitações distintas (PARVEZ *et al.*, 2018). Entre os modelos mais recorrentes, destacam-se o carregamento estático, no qual os dados estão incorporados diretamente no HTML fornecido pelo servidor, e o carregamento dinâmico, em que as informações são inseridas posteriormente por meio da execução de scripts JavaScript. Adicionalmente, estratégias como rolagem infinita, paginação tradicional e carregamento sob demanda (lazy load) também são utilizadas e requerem técnicas especializadas para garantir uma coleta completa e precisa das informações.

2.4.1 Carregamento estático vs. carregamento dinâmico

O carregamento estático corresponde a páginas web em que todo o conteúdo é disponibilizado de forma integral no momento em que o código HTML é entregue pelo servidor ao navegador, sem depender da execução de *scripts* adicionais para exibição das informações (ANKITHA *et al.*, 2016; JUNIOR, 2018). Nesse tipo de estrutura, os dados encontram-se incorporados diretamente no código-fonte, o que favorece a aplicação de técnicas convencionais de extração, como o uso de bibliotecas que operam sobre HTML puro, dispensando a necessidade de renderização dinâmica. Essa característica torna o processo de coleta automatizada mais simples e previsível, especialmente em ambientes com estrutura estável.

Por outro lado, o carregamento dinâmico caracteriza-se pela inserção gradual do conteúdo da página por meio da execução de *scripts* JavaScript, geralmente após o carregamento inicial do HTML (FLANAGAN, 1997; MITCHELL, 2018). Essa abordagem é amplamente empregada em aplicações modernas baseadas em **frameworks front-end** que priorizam a experiência do usuário por meio da renderização assíncrona dos dados. Como resultado, as informações nem sempre estão imediatamente disponíveis no código-fonte, sendo carregadas sob demanda por meio de chamadas adicionais à API ou pela manipulação da estrutura do DOM. Para acessar tais dados, torna-se necessário recorrer a ferramentas que simulem o comportamento de um navegador real, permitindo a renderização completa e a interação programada com os elementos da interface.

2.4.2 *Scroll* infinito e paginação tradicional

A paginação tradicional é uma técnica de organização de conteúdo em páginas web que segmenta as informações em diferentes seções, acessíveis por meio de botões de navegação, como Próxima e Anterior. Essa abordagem é amplamente adotada em catálogos de produtos, fóruns e mecanismos de busca, permitindo ao usuário uma navegação sequencial e controlada. No contexto da extração automatizada de dados, a paginação facilita a implementação de estratégias de coleta baseadas na repetição de requisições e na

identificação de padrões nos links de navegação, o que possibilita a iteração programada entre as páginas subsequentes (BERNARD *et al.*, 2002).

Em contrapartida, o *scroll* infinito constitui uma forma dinâmica de apresentação de conteúdo, na qual novos elementos são carregados automaticamente à medida que o usuário realiza a rolagem da página. Essa técnica visa aprimorar a experiência de navegação contínua e é frequentemente utilizada em redes sociais, plataformas de comércio eletrônico e sites de notícias. No entanto, essa estrutura impõe desafios adicionais para o *web scraping*, uma vez que o conteúdo não está presente no carregamento inicial da página e só se torna acessível mediante interações dinâmicas com o DOM. Nesses casos, é necessário simular comportamentos típicos do usuário, como a rolagem da interface, além de monitorar modificações na árvore de elementos da página, a fim de garantir a captura integral das informações pretendidas (MURANO; SHARMA, 2020).

2.4.3 Elementos renderizados sob demanda *(lazy load)*

O carregamento sob demanda, também denominado *lazy load*, é uma técnica de otimização de desempenho adotada em páginas web com o objetivo de adiar o carregamento de determinados elementos, como imagens, listas e blocos de conteúdo, até que estejam prestes a ser exibidos na área visível da tela do usuário *(viewport)*. Essa abordagem visa reduzir o tempo de carregamento inicial, economizar recursos de rede e melhorar a fluidez da navegação, especialmente em dispositivos com conexões mais limitadas ou em plataformas com grande volume de dados a serem exibidos (ZILIOTTI, 2024).

No contexto da extração automatizada de dados, essa técnica impõe desafios específicos, uma vez que os elementos desejados não estão imediatamente disponíveis na estrutura do DOM após o carregamento inicial da página. Para superar essa limitação, torna-se necessário simular interações com a interface da aplicação, como a rolagem progressiva ou o acionamento de eventos que desencadeiem a renderização dos conteúdos ocultos. Ferramentas como Selenium e Puppeteer são amplamente utilizadas nesse cenário, pois permitem replicar com precisão o comportamento de um usuário real, garantindo o acesso a dados que seriam inacessíveis por meio de métodos tradicionais de coleta baseados apenas na leitura do HTML estático.

2.5 Mecanismos de proteção contra *scraping*

Com o avanço e a crescente adoção do web *scraping* como método para coleta automatizada de dados, diversas plataformas digitais passaram a implementar mecanismos específicos de proteção voltados à identificação e bloqueio de acessos automatizados. Essas medidas têm como objetivo dificultar ou impedir a atuação de agentes não humanos por razões que envolvem desde a proteção de dados sensíveis até a preservação do

desempenho dos servidores e o cumprimento de exigências legais (SOUZA; ALVES, 2025). Tais mecanismos apresentam diferentes níveis de complexidade, variando desde técnicas baseadas na observação de padrões de navegação até sistemas mais robustos, como o uso de *firewalls* de aplicação, verificações de integridade do navegador e bloqueios fundamentados na reputação de endereços IP. O conhecimento dessas estratégias é fundamental para compreender os desafios enfrentados durante a coleta de dados em ambientes protegidos, especialmente em cenários nos quais a infraestrutura do site foi projetada para limitar acessos automatizados (TURK *et al.*, 2020).

2.5.1 Detecção de automação por comportamento de navegação

Dentre os mecanismos empregados para impedir o acesso automatizado a páginas web, destaca-se a análise comportamental como uma das estratégias mais recorrentes. Plataformas com sistemas de monitoramento avançado conseguem identificar padrões típicos de agentes automatizados, como a realização de múltiplas requisições em curtos intervalos de tempo, ausência de interações com elementos da interface, como movimentação do cursor, preenchimento instantâneo de formulários e execução de cliques com temporizações exatas. A detecção desses comportamentos não humanos permite que o servidor adote medidas de mitigação, como o bloqueio do endereço IP, encerramento da sessão ou redirecionamento do tráfego para camadas adicionais de verificação, como páginas de desafio ou autenticação. Para superar tais restrições, ferramentas de automação modernas passaram a incorporar simulações avançadas de interação humana. Entre as técnicas utilizadas estão a introdução de atrasos aleatórios entre ações, movimentações não lineares do mouse, preenchimento gradual de campos e variação nos tempos de digitação, a fim de mascarar a presença de robôs e reduzir a probabilidade de bloqueio (POGGI *et al.*, 2007).

2.5.2 Técnicas de ofuscação e atraso no carregamento

Diversas páginas web incorporam estratégias de ofuscação com o intuito de dificultar a identificação e extração de informações relevantes por sistemas automatizados. Entre essas estratégias, destacam-se a utilização de classes e identificadores IDs gerados dinamicamente, a aplicação de camadas visuais que ocultam conteúdos no carregamento inicial e a inserção de dados por meio de *scripts* complexos ou intencionalmente ofuscados. Além disso, técnicas como o *delayed loading*, ou carregamento com atraso, introduzem um tempo de latência ou exigem ações do usuário, como rolagem ou cliques, para que determinados elementos sejam inseridos na estrutura da página. Tais abordagens tornam-se obstáculos significativos para ferramentas de *scraping* que dependem de estruturas estáticas e previsíveis. Nessas circunstâncias, o uso de bibliotecas que suportam a renderização dinâmica, a manipulação direta do DOM e a sincronização com eventos de carregamento se mostra fundamental para viabilizar a coleta de dados (DAMASCENO, 2017).

2.5.3 Sistemas de proteção como Cloudflare e captchas

Soluções avançadas de proteção, como a fornecida pela Cloudflare, são amplamente adotadas por sites que buscam mitigar acessos automatizados, e práticas de web scraping em larga escala. Atuando como uma camada intermediária entre o usuário e o servidor de origem, a Cloudflare realiza a inspeção do tráfego em tempo real, aplicando mecanismos de bloqueio com base em heurísticas, assinaturas de navegação e reputação de endereços IP (OLIVEIRA *et al.*, 2022). Entre os recursos utilizados, destacam-se as verificações de integridade do navegador, desafios baseados em JavaScript e a apresentação de testes CAPTCHA, que visam distinguir usuários humanos de agentes automatizados. Tais medidas impõem obstáculos significativos à automação tradicional, exigindo abordagens mais sofisticadas, como o uso de navegadores *headless* com rotinas antidetecção, bibliotecas para resolução automatizada de CAPTCHAs (VASTEL *et al.*, 2020). Navegadores *headless* são navegadores web que operam sem interface gráfica, permitindo a execução de tarefas de navegação e manipulação de páginas de forma automatizada e em segundo plano. Por funcionarem via linha de comando ou por meio de *scripts*, são amplamente utilizados em testes automatizados, raspagem de dados, monitoramento de sites e simulação de interações de usuários. Esses mecanismos reforçam a necessidade de estratégias adaptativas diante de ambientes web com altos níveis de proteção.

2.5.4 Aspectos éticos e legais da raspagem de dados

A prática do *web scraping*, embora amplamente adotada em contextos técnicos e científicos, envolve implicações legais e éticas que merecem atenção, especialmente no que diz respeito à privacidade de dados e aos direitos autorais sobre o conteúdo extraído. Diversas plataformas definem restrições explícitas a acessos automatizados em seus termos de uso ou por meio do arquivo robots.txt, que deve ser respeitado sempre que disponível.

Além disso, com a vigência de legislações como a Lei Geral de Proteção de Dados (LGPD) no Brasil e o Regulamento Geral de Proteção de Dados (GDPR) na União Europeia, torna-se imprescindível garantir que os dados coletados não envolvam informações pessoais sensíveis, como nomes, e-mails, geolocalização ou identificadores de usuário. Ainda que este trabalho concentre-se exclusivamente em dados públicos e anônimos (como nomes de skins, preços e imagens genéricas de itens), a adoção de boas práticas éticas é essencial para assegurar a conformidade com a legislação vigente.

Entre os princípios éticos recomendados destacam-se:

- Respeitar os limites de acesso definidos pelas plataformas;
- Evitar sobrecarga nos servidores, adotando tempos de espera realistas;
- Garantir a transparência da origem dos dados nas aplicações derivadas;

- Excluir proativamente qualquer dado que possa identificar indivíduos.

Conforme discutido por Brown *et al.* (2024), a raspagem ética requer que desenvolvedores, pesquisadores e empresas considerem não apenas o que é tecnicamente possível, mas também o que é socialmente aceitável e legalmente permitido. Dessa forma, este trabalho adota uma abordagem responsável, limitada a dados visíveis publicamente, sem identificação de usuários, e respeitando os princípios de boa fé e proporcionalidade no uso das informações.

2.6 Normalização e estruturação de dados extraídos

Após a obtenção inicial das informações por meio de técnicas de web scraping, torna-se essencial realizar a normalização e a estruturação dos dados extraídos. Essa etapa tem como objetivo transformar dados heterogêneos, provenientes de diversas fontes, em um formato padronizado e coerente, eliminando inconsistências, ruídos e variações que possam comprometer a qualidade da análise. A normalização compreende atividades como a uniformização de formatos textuais, a conversão de moedas, o tratamento de diferentes idiomas e a padronização de nomenclaturas, garantindo a comparabilidade entre os registros. Já a estruturação envolve a organização lógica das informações em tabelas, listas ou objetos estruturados, contendo atributos bem definidos, como nome do item, valor, estado de conservação e URL de origem (BATISTA *et al.*, 2018).

Esse processo torna-se ainda mais relevante quando os dados são coletados de plataformas com estruturas distintas, pois permite sua consolidação em um modelo comum, facilitando o armazenamento em bancos de dados, a análise estatística e a integração com aplicações analíticas. A qualidade das análises, visualizações e relatórios gerados nas etapas subsequentes está diretamente associada ao nível de consistência, clareza e completude dos dados tratados (GRACIANO; RAMALHO, 2023).

2.7 Integração de dados coletados em aplicações analíticas

A integração dos dados extraídos em aplicações analíticas representa uma etapa fundamental para transformar informações coletadas automaticamente em valor estratégico. Após passarem pelos processos de normalização e estruturação, os dados tornam-se aptos a serem incorporados em sistemas interativos de visualização, painéis dinâmicos, *dashboards*, ou modelos computacionais preditivos, com o objetivo de apoiar a tomada de decisão e gerar *insights* relevantes (PASQUALI *et al.*, 2024).

Para isso, é necessário o uso de ferramentas e bibliotecas que possibilitem tanto a manipulação eficiente dos dados em tempo real quanto sua apresentação de forma compreensível e acessível para o usuário final (LIMA; OLIVEIRA, 2020). Tecnologias

como Pandas e Plotly são amplamente empregadas na análise e visualização de dados, enquanto *frameworks* como Dash e Streamlit permitem o desenvolvimento de interfaces responsivas e dinâmicas, facilitando a interação do usuário com os dados tratados.

Outro aspecto relevante dessa etapa consiste na manutenção do vínculo entre os dados extraídos e suas fontes originais, como por exemplo, os links para os itens analisados, o que assegura transparência no uso da informação e oferece funcionalidades adicionais, como a possibilidade de redirecionamento para as plataformas de origem ou comparação entre diferentes sites. Com isso, a integração analítica completa o ciclo do processo de *web scraping*, conectando a coleta automatizada à aplicação prática e à análise inteligente dos dados (BATISTA *et al.*, 2018).

2.8 Trabalhos Relacionados

No âmbito do web scraping e suas aplicações, a contextualização da pesquisa torna-se imperativa. Nesse sentido, dois estudos relevantes que empregam essa técnica em domínios distintos serão abordados, destacando suas respectivas contribuições para a área.

2.8.1 USO DE METAHEURÍSTICA EM UMA FERRAMENTA DE COTAÇÃO PARA COMPRAS DE CARTAS DE MAGIC: THE GATHERING

O Trabalho de Conclusão de Curso apresenta o ScryX, uma SPA desenvolvida para otimizar a aquisição de cartas de Magic: The Gathering, minimizando custos e o número de fornecedores. A solução integra um Web Crawler para coleta de dados de preços com a metaheurística Simulated Annealing e um algoritmo guloso para otimização combinatória das compras. Embora demonstre significativa economia de 8% a 21% em comparação com ferramentas existentes e empregue uma arquitetura robusta, nota-se como limitação a simplificação da otimização de frete e a potencial aceitação de soluções com pequena porcentagem de cartas faltantes (PEREIRA, 2019).

2.8.2 Protótipo de uma Plataforma de busca de Preços na Web

O presente Trabalho de Conclusão de Curso propõe o desenvolvimento de um protótipo de plataforma de busca e comparação de preços na web, abordando a complexidade da aquisição de informações de produtos em um cenário de vasto comércio eletrônico. A solução, construída com uma arquitetura de microsserviços robusta, que inclui Angular para o frontend, Spring Boot para o backend, e a utilização de PostgreSQL e MongoDB para gerenciamento de dados, distingue-se pela capacidade de indexar dados de qualquer loja aderente ao padrão Schema.org. Métodos de Web Crawling e técnicas de similaridade textual, como TF-IDF e cosseno, são empregados para coletar e classificar informações

eficientemente, culminando em uma ferramenta promissora para otimizar a experiência de compra online, com perspectivas futuras para integração de inteligência artificial e aprimoramento da gestão de dados (SOUZA, 2025).

3 Materiais e Método

3.1 Estratégia de Coleta Baseada em Múltiplos Sites de Skins

A etapa de coleta e seleção de múltiplos sites foi fundamentada em duas métricas principais: relevância e similaridade estrutural. A relevância está diretamente relacionada à confiabilidade do site dentro da comunidade do jogo Counter Strike (CORPORATION, 2025). Sites mais reconhecidos e populares dentro dessa comunidade tendem a possuir uma maior quantidade de itens, o que influencia positivamente tanto a qualidade quanto a consistência da coleta. Sites com maior visibilidade geralmente têm um fluxo constante de atualizações e manutenção dos dados, o que garante a precisão das informações extraídas. Por outro lado, sites menos conhecidos podem apresentar uma oferta reduzida de skins ou dados inconsistentes, comprometendo a confiabilidade da pesquisa, porém, são menos suscetíveis à manutenção e mudança de sua estrutura. Portanto, a seleção dos sites foi um passo crucial para garantir a qualidade dos dados coletados e, consequentemente, a validade das análises subsequentes.

3.2 Identificação da Estrutura dos Elementos HTML por Site

A identificação da estrutura dos elementos HTML foi realizada de forma sistemática, com o objetivo de localizar e mapear as *tags* que contêm as informações desejadas, como o nome da arma, nome da *skin*, preço, desgaste e *float*. A análise das *tags* HTML incluiu a exploração de id, classe e componente, que são atributos fundamentais na construção de páginas web dinâmicas. Cada site possui convenções próprias para a nomeação dessas *tags*, o que tornou necessário um mapeamento detalhado para cada site individualmente. Durante esse processo, foi possível verificar que alguns sites utilizam nomenclaturas padronizadas, enquanto outros optam por nomes mais específicos, o que requer um trabalho extra de adaptação e mapeamento (KADAM; PAKLE, 2014). A principal dificuldade neste estágio foi a diversidade nas nomenclaturas e na organização das *tags* entre os sites, o que exigiu uma adaptação contínua dos *scripts* de *scraping*. Cada site apresentou diferentes formas de referenciar as seções que continham as informações relevantes, como preços e desgastes das *skins*. Para resolver esse problema, foi realizado um processo de normalização da estrutura, no qual foram estabelecidas regras e padrões para a identificação consistente das *tags* entre os diferentes *sites* analisados. Essas diretrizes visam garantir a extração uniforme das informações e serão detalhadas posteriormente na seção de desenvolvimento (DUCKETT; SCHLÜTER, 2011). Essa normalização foi essencial para garantir que os dados extraídos fossem consistentes e comparáveis entre

os *sites*. Inicialmente, a identificação e o mapeamento das *tags* HTML foram realizados manualmente. Posteriormente, essa metodologia foi replicada nos demais *sites* analisados, mantendo-se a consistência no processo de extração das informações.

3.3 Ferramentas Utilizadas

A coleta e análise dos dados foram realizadas utilizando-se de ferramentas e bibliotecas de programação específicas, as quais permitiram a automação das etapas de extração, tratamento e visualização dos dados. A escolha das ferramentas foi pautada pela necessidade de realizar operações de *scraping* em larga escala, além de tratar e organizar os dados de maneira eficiente e apresentar os resultados de forma acessível e visualmente compreensível. A linguagem Python foi a principal linguagem de programação utilizada, devido à sua flexibilidade e vasto ecossistema de bibliotecas. Python é adotada para análise de dados e automação de tarefas, o que a torna ideal para o propósito deste trabalho (CIRILLO; GALANTE, 2025). As bibliotecas Selenium e Pandas foram escolhidas em razão de suas funcionalidades específicas para *scraping* e análise de dados em grandes volumes (OPREA; BÂRA, 2022). Além disso, a biblioteca Threading foi empregada para possibilitar a execução simultânea de múltiplas tarefas, otimizando o tempo de coleta em diferentes fontes e aumentando a eficiência do processo (SILVA; YOKOYAMA, 2011). Por fim, o Dash Bootstrap foi utilizado para o desenvolvimento da interface web, permitindo a apresentação dos dados de forma interativa e visualmente atrativa. A combinação dessas ferramentas garantiu a robustez e a eficiência do sistema de coleta e análise de dados desenvolvido (HOLJEVAC; JAKOPEC, 2020).

3.3.1 Selenium para Automação de Coleta

O Selenium foi utilizado para automação da coleta de dados dos *sites* selecionados. Como muitos dos sites analisados dependem de JavaScript para carregar dinamicamente suas informações, o Selenium se mostrou essencial, pois permite a simulação de interações humanas, como cliques e navegação entre páginas, o que é fundamental para a coleta de dados em páginas complexas. A versão mais recente da biblioteca foi adotada para garantir a compatibilidade com os elementos dinâmicos e para oferecer suporte às interações necessárias para acessar todas as informações disponíveis. O Selenium interage diretamente com o DOM de cada página, o que facilita a extração de dados específicos, como os campos de preço e nome das *skins* (TANAKA *et al.*, 2020). Além disso, o Selenium possibilitou a execução de *loops* automatizados para a coleta de dados em larga escala, o que permitiu a extração de informações de dezenas de páginas de forma eficiente. A execução programada dos *scripts* respeitou os tempos de carregamento das páginas, o que garantiu a integridade dos dados e evitou a coleta de informações incompletas ou incorretas. A capacidade de realizar requisições programadas também assegurou que a coleta fosse feita de forma

consistente e repetível, permitindo que o processo fosse automatizado para futuras coletas de dados ou atualizações no sistema.

3.3.2 Pandas para Tratamento e Organização dos Dados

Após a coleta dos dados brutos, foi necessário realizar um processamento significativo para garantir que os dados fossem organizados de maneira útil e eficaz. Para isso, foi utilizada a biblioteca Pandas, que oferece uma vasta gama de funcionalidades para manipulação e análise de dados (MCKINNEY, 2018). Com o Pandas, os dados extraídos foram armazenados em DataFrames, que são estruturas tabulares adequadas para a organização das informações, permitindo operações rápidas de filtragem, ordenação e agregação. Essa estrutura foi fundamental para permitir uma análise posterior eficiente e para a execução de operações de limpeza e normalização. O tratamento de dados envolveu múltiplas etapas fundamentais para garantir a qualidade e a consistência das informações coletadas. Inicialmente, foi realizada a remoção de duplicatas a fim de evitar redundâncias que poderiam comprometer a análise. Em seguida, foram identificados e tratados os valores ausentes, utilizando abordagens como preenchimento com valores padrão, exclusão de registros incompletos. Por fim, procedeu-se à normalização dos formatos, assegurando uniformidade em elementos como datas, números, nomes e unidades de medida, o que permitiu integrar dados provenientes de fontes distintas em um único conjunto coeso e padronizado. A normalização foi especialmente importante no caso dos preços das *skins*, que apresentavam diferentes formatos em cada site. O Pandas permitiu padronizar esses valores, garantindo a consistência das informações. Além disso, a organização dos dados em categorias, como tipo de *skins* e faixa de preço, facilitou o cruzamento das informações. O uso do Pandas foi crucial para transformar os dados brutos coletados em uma forma que fosse útil para a análise e a construção da interface de visualização.

3.3.3 Dash Bootstrap para Desenvolvimento da Interface Web

O Dash, uma framework em Python, foi utilizado para criar a interface web interativa para a visualização dos dados extraídos. O Dash oferece uma forma simples e eficaz de construir aplicativos web interativos (NOKERI, 2021). O uso do Dash Bootstrap foi especialmente importante para garantir que a interface tivesse um *design* intuitivo e adaptável, utilizando componentes responsivos que são facilmente integrados com o estilo Bootstrap (JIN *et al.*, 2023). O resultado foi uma aplicação web visualmente agradável e funcional, que possibilitou a exploração dos dados de forma dinâmica.

3.4 Construção de um Mecanismo Adaptativo de Extração de Dados

Observou-se que os métodos empregados na resolução do problema poderiam ser estendidos para a criação de um filtro de *scraping* dinâmico, uma vez que envolveram diferentes abordagens de extração de dados. Essa proposta permite ao usuário configurar a coleta conforme a estrutura de cada *site*, desde que possua conhecimentos básicos sobre HTML e seus seletores. Dessa forma, usuários familiarizados com a organização de elementos em páginas web podem explorar com mais autonomia e eficácia as funcionalidades da solução desenvolvida (LAWSON, 2015).

3.5 Infraestrutura Computacional Utilizada

O ambiente computacional utilizado foi um Mac mini (modelo 2024), equipado com chip Apple M4, contendo dez núcleos de CPU e dez núcleos de GPU, baseado na arquitetura ARM, com 16 GB de memória RAM e armazenamento interno de 256 GB SSD. O sistema operacional em uso foi o macOS Sequoia, versão 15.5.

4 Desenvolvimento

Nesta seção, apresenta-se o fluxo de desenvolvimento adotado, destacando as etapas percorridas e as particularidades envolvidas na resolução do problema proposto.

4.1 Coleta de dados: adaptação a diferentes estruturas e restrições

A primeira etapa do processo de desenvolvimento consistiu na definição dos critérios que nortearam a escolha dos sites a serem utilizados como fontes para a coleta de dados. Essa escolha foi essencial para garantir a viabilidade técnica do projeto, a relevância dos dados coletados e a possibilidade de generalização dos métodos aplicados. Foram estabelecidas cinco métricas principais para orientar essa seleção:

- **Similaridade estrutural:** refere-se à presença de elementos comuns na organização das páginas, como o uso de *containers*, *grids*, e *cards* contendo informações relevantes, como imagem, nome do item, tipo de arma e preço;
- **Relevância:** diz respeito à popularidade e influência do site dentro da comunidade de usuários, especialmente jogadores de Counter-Strike e colecionadores de skins;
- **Confiabilidade:** relacionada à reputação e segurança da plataforma, considerando aspectos como histórico de uso, estabilidade e legitimidade das informações;
- **Diversidade nos mecanismos de navegação:** foram considerados sites que apresentassem diferentes formas de carregamento de conteúdo, como paginação tradicional ou rolagem infinita *(infinite scroll)*, a fim de testar a adaptabilidade das soluções de *scraping*;
- **Abrangência geográfica:** buscou-se incluir tanto sites internacionais quanto nacionais, com o objetivo de avaliar possíveis diferenças tecnológicas e estruturais entre eles.

Embora todos os critérios tenham sido levados em consideração, priorizou-se principalmente a similaridade estrutural e a relevância dos sites, por exercerem maior influência direta na aplicação dos métodos de extração de dados. A escolha dos sites não exigiu o cumprimento simultâneo de todos os critérios, mas sim daqueles considerados mais determinantes para o escopo da pesquisa. A ordem de prioridade estabelecida foi: similaridade estrutural, relevância, confiabilidade, diversidade de rolagem/paginação e abrangência internacional *versus* nacional.

Com base nesses critérios, foram selecionados cinco sites para compor o conjunto de fontes de dados do projeto:

- **CS.Money:** foi o primeiro site escolhido, por apresentar alta relevância internacional, ser conhecido pela comunidade global de Counter-Strike e já ter sido utilizado previamente pelo autor, o que contribuiu para validar sua confiabilidade. Além disso, serviu como referência estrutural para a análise dos demais *sites*, pois possui um *layout* padronizado com *cards* contendo imagem da *skin*, nome da arma, desgaste *(wear)*, preço e número do *float*.
- **NeshaStore e NeshaStore Prime:** são plataformas nacionais que, embora tenham menor relevância em comparação com o CS.Money, possuem grande penetração no mercado brasileiro e atendem aos critérios de similaridade estrutural. Ambas exibem os itens em formato de *grid*, com *cards* que contêm imagem, nome da *skin*, nome da arma, desgaste e preço. Destacam-se também por serem exemplos representativos de páginas com paginação em vez de rolagem infinita, o que amplia os testes em relação aos diferentes modelos de navegação.
- **DashSkins:** também um *site* nacional, apresenta uma estrutura semelhante à dos anteriores, com organização clara em *containers*, *grids* e *cards*. O *site* analisado, em virtude de seu extenso catálogo de itens, revela-se particularmente útil para a validação do processo de raspagem de dados. A navegação na plataforma é caracterizada pela utilização de paginação.
- **MarketCSGO:** diferente das outras plataformas nacionais, este é um *site* internacional, assim como o CS.Money. Apesar de sua relevância atual ser relativamente baixa, trata-se de uma plataforma antiga, com presença desde as versões anteriores do Counter-Strike: Global Offensive (CS:GO), lançado em 2012. O *site* utiliza rolagem infinita como mecanismo de navegação, o que o torna valioso para testar o *crawler* em páginas com carregamento dinâmico de conteúdo.

Com a seleção dos *sites* concluída, iniciou-se a etapa de coleta e extração das informações. As principais *tags*-alvo para raspagem foram definidas com base na recorrência dos dados exibidos nas páginas analisadas. As informações extraídas foram:

- **Nome da *skin*:** Em todos os *sites* analisados, foi possível identificar a presença de uma *tag* responsável por exibir o nome da *skin*, informação essencial para a correta localização e comparação dos itens entre as diferentes plataformas. Essa nomenclatura inclui, geralmente, tanto o nome da arma quanto o nome específico da *skin*, compondo um identificador único do item. As informações estavam organizadas em classes específicas no código HTML de cada site. No caso do site CS.Money, por

exemplo, o nome da *skin* foi localizado por meio do seletor ".csm_3f4a05c6". Os dados extraídos apresentavam-se em diferentes idiomas, principalmente em português e inglês, e em determinados casos, acompanhados de ruídos ou elementos indesejados oriundos da estrutura da página. Essas inconsistências foram devidamente tratadas e os dados foram normalizados durante a etapa de pré-processamento, com o objetivo de assegurar a integridade e a uniformidade das informações utilizadas nas análises subsequentes.

- **Estado de desgaste Wear:** As *skins* no jogo Counter-Strike são classificadas de acordo com seu estado de desgaste, o qual influencia diretamente tanto sua aparência visual quanto seu valor de mercado. As categorias utilizadas para essa classificação seguem um padrão definido, sendo elas: Nova de Fábrica *(Factory New)*, Pouco Usada *(Minimal Wear)*, Testada em Campo *(Field-Tested)*, Bem Desgastada *(Well-Worn)* e Veterana de Guerra *(Battle-Scarred)*. Cada uma dessas designações representa um nível distinto de conservação da skin, impactando na atratividade e na precificação do item no mercado. Durante a análise dos *sites* selecionados, constatou-se que todos apresentavam uma tag específica no código HTML destinada à exibição do estado de desgaste das *skins*. Essa informação mostrou-se fundamental tanto para a categorização correta dos dados coletados quanto para a posterior análise de mercado, sendo incorporada de forma estruturada ao processo de extração e organização dos dados.

- **Valor (preço):** Os preços das *skins* estavam disponíveis em todos os *sites* analisados, permitindo sua extração por meio de *tags* específicas no código HTML. No entanto, no caso do site MarketCSGO, os valores estavam expressos em dólar americano (USD), por se tratar de uma plataforma internacional. Para padronizar os dados, foi realizada a conversão cambial para real brasileiro (BRL) utilizando a biblioteca forex_python. Quando essa abordagem não retornava a taxa de câmbio atual, uma requisição alternativa à API pública Frankfurter foi empregada, assegurando a consistência dos valores analisados.

- **Número do *float* (índice de desgaste numérico da *skin*):** No contexto do jogo Counter-Strike, a faixa de *float* das *skin* constitui um atributo visual que expressa o nível de conservação do item. Essa característica é representada por um valor numérico, o qual varia em uma escala contínua de 0.00, indicando um item em estado impecável, até 1.00, representando um nível elevado de desgaste. Esse valor influencia diretamente tanto a aparência da *skin* quanto seu valor de mercado. Durante o processo de coleta de dados, verificou-se que a maioria dos sites analisados disponibilizava o valor do *float* por meio de uma *tag* específica, o que permitiu sua extração e posterior tratamento. A única exceção foi o site MarketCSGO, que, embora exibisse outras informações relevantes no *card* principal

do item, não apresentava o valor do *float* de forma visível ou acessível na estrutura HTML analisada, impossibilitando sua coleta automatizada dentro do escopo deste trabalho.

- **Imagem do item.** Além das informações textuais extraídas, como nome, preço e estado de desgaste das *skins*, a captura da *tag* correspondente à imagem do item também se mostrou relevante. A extração desse elemento visual foi fundamental para possibilitar sua posterior exibição na interface do sistema desenvolvido.

- **Link direcional para o item.** Outro elemento relevante no processo de extração de dados foi o *link* que direciona para a página específica de cada item nas plataformas analisadas. A obtenção dessa informação permitiu associar, a cada *skin*, uma URL única de origem, assegurando rastreabilidade e possibilitando o redirecionamento direto para o *site* original por meio da interface do sistema. Entretanto, foi identificada uma particularidade no *site* CS.Money, que não disponibiliza *links* diretos para visualização individual dos itens. Nessa plataforma, as informações detalhadas são apresentadas exclusivamente por meio de uma janela modal, acionada ao se clicar em um *card* na grade principal de produtos, o que inviabiliza o acesso direto via URL convencional. Para contornar essa limitação, foi adotada uma abordagem baseada no uso do ID exclusivo de cada item, também extraído durante o processo de coleta. A partir desse identificador, foi possível construir uma busca direcionada, utilizando os atributos específicos da *skin*, como nome, preço, estado de desgaste e valor de *float*, de modo a restringir os resultados a um único item exibido na interface. Uma vez localizado, o sistema executa automaticamente uma ação de clique sobre o item correspondente, promovendo a abertura da modal com os dados completos.

A identificação dessas tags foi feita inicialmente de forma manual, por meio da inspeção dos elementos HTML em cada site.

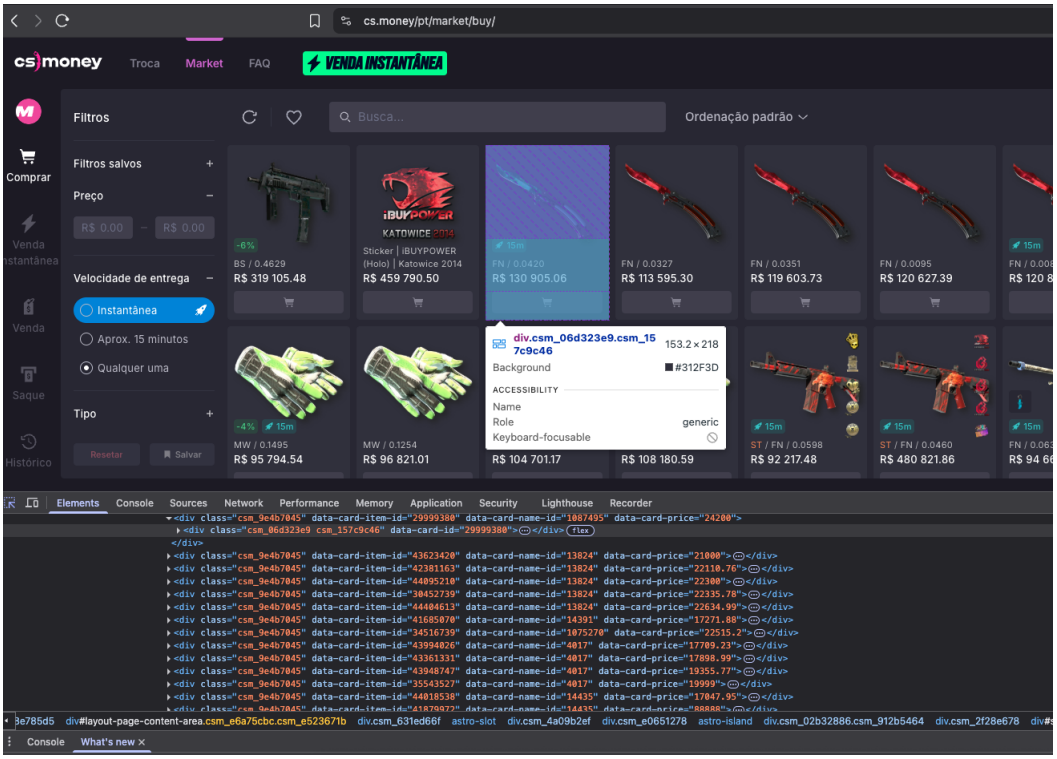

Figura 1 – Exemplo de grid sobre o site

Fonte: (CS.MONEY, 2025)

4.1.1 Site com *Scroll Infinito*: Controle de Carregamento e Última Posição

Durante a etapa de coleta, foi necessário lidar com sites que implementam carregamento dinâmico de conteúdo via *scroll infinito*, uma técnica comum utilizada em interfaces web para carregar novos elementos à medida que o usuário rola a página. Dois dos cinco sites selecionados (CS.Money e MarketCS) adotavam esse modelo, o que demandou soluções específicas para garantir que todos os itens fossem devidamente carregados e extraídos com o Selenium.

No caso do CS.Money, os itens eram exibidos dentro de um *grid* com rolagem interna. Esse *grid* estava embutido em uma div com *overflow* configurado para *scroll*, o que significa que a rolagem não afetava a página inteira, mas apenas o *container* específico onde os itens estavam listados. Um dos principais desafios enfrentados foi que, ao utilizar comandos tradicionais do Selenium para simular a rolagem da página (como driver.execute_script("window.scrollTo(...)), a ação resultava na rolagem do *body* do *site* em vez do *grid*, o que fazia com que a *viewport* fosse jogada ao rodapé da página, sem carregar os novos itens no *grid*. Para contornar esse problema, foi necessário identificar diretamente o elemento HTML do *grid* e aplicar o *scroll* diretamente sobre ele, utilizando *scripts* do tipo: driver.execute_script("arguments[0].scrollTop = arguments[0].scrollTop + 500;". scroll_container), A abordagem empregada consiste em localizar o elemento div do

grid e aplicar diretamente um script JavaScript que força a rolagem vertical de sua barra interna em um valor incremental. Este procedimento assegura o carregamento dinâmico de novos itens, sendo crucial um período de espera subsequente para a completa renderização do conteúdo antes da continuidade da extração de dados.

Esse comando força o *scroll* do *grid* específico, simulando a ação de um usuário que está rolando o *container* manualmente, o que ativa o carregamento de novos itens. Além disso, foi implementada uma estratégia de monitoramento da posição final do *scroll* com base na presença do último item renderizado. A cada iteração, verificava-se se novos elementos haviam sido carregados comparando o ID ou texto do último item anteriormente detectado. O processo era encerrado quando nenhuma nova mudança era observada após uma determinada quantidade de ciclos, indicando o fim do carregamento dinâmico.

O *site* MarketCS apresentava uma peculiaridade adicional, a ordem inversa do carregamento por *scroll*. Nesse caso, os novos itens não apareciam ao rolar para baixo, mas sim ao retornar ao topo da lista, o que indicava um sistema de carregamento reverso, semelhante ao encontrado em alguns *chats* e sistemas de mensagens. Essa abordagem impediu o uso de técnicas tradicionais de rolagem linear. Para resolver esse desafio, foi implementada uma técnica baseada na seleção e monitoramento do último item visível no DOM. A cada ciclo de coleta, o *script* localizava o último elemento carregado e utilizava a função element.location_once_scrolled_into_view do Selenium para forçar o foco visual até esse item, simulando o comportamento de um usuário que rola até aquele ponto. Isso não apenas evitava que o Selenium saísse da área de interesse, mas também induzia o carregamento de novos itens subsequentes.

Adicionalmente, para garantir controle sobre o tempo e volume de coleta, foram estabelecidos critérios de parada. O primeiro consistiu na definição de um número máximo de itens a serem coletados, previamente estipulado como limite superior, evitando execuções infinitas ou coleta excessiva de dados. O segundo critério envolvia a verificação de avanço no carregamento, se após um número determinado de ciclos de *scroll*, por exemplo, 10 tentativas consecutivas, não fossem detectados novos itens adicionados à lista, o algoritmo interrompia a execução automaticamente. Esses mecanismos de controle garantiram a estabilidade do processo, evitando sobrecarga na coleta e reduzindo o risco de bloqueio por parte dos servidores dos *sites*.

Essas estratégias de manipulação do *scroll* e controle do carregamento foram fundamentais para garantir a extração completa e confiável dos dados, respeitando as limitações técnicas impostas pela estrutura dinâmica dos *sites*.

4.1.2 *Site* com Paginação Tradicional: Extração Incremental

Durante o desenvolvimento da rotina de coleta automatizada, identificou-se que alguns *sites* adotavam um sistema clássico de paginação sequencial para a exibição dos itens. Diferentemente do *scroll infinito*, nesse modelo os dados são organizados em páginas discretas, onde um número fixo de resultados é exibido por vez. Os *sites* NeshaPrime, NeshaStore e DashSkin utilizam esse padrão, no qual os produtos são apresentados em um *grid* paginado, e a navegação entre as páginas é realizada por meio de botões de próxima ou avanço. Esse formato, embora mais tradicional, exige uma abordagem sistemática para garantir a extração completa dos dados em todas as páginas disponíveis.

Para lidar com essa estrutura, foi implementada uma estratégia de extração incremental, que consiste em realizar a coleta página por página, respeitando a sequência da navegação e extraindo todas as informações exibidas antes de avançar para a próxima página. O Selenium foi utilizado para automatizar esse processo, simulando o clique nos botões de navegação após a conclusão da coleta de cada página. A detecção dos botões foi realizada por meio de seletores CSS ou expressões XPath, e o clique foi executado com o comando "element.click()". Para assegurar que a página seguinte estivesse completamente carregada antes de iniciar uma nova coleta, utilizou-se a biblioteca WebDriverWait, que aguarda a presença dos novos elementos no DOM antes de prosseguir.

Além disso, foram implementados mecanismos de controle de parada, fundamentais para garantir a robustez e eficiência da coleta. O primeiro critério definido foi um limite máximo de itens a serem extraídos, estabelecido para evitar coletas excessivas e controlar o volume de dados durante o processo. A coleta era automaticamente interrompida quando esse limite era atingido, independentemente de ainda haver páginas disponíveis. O segundo critério baseava-se na detecção do fim da paginação, que ocorria quando o botão de próxima página deixava de estar disponível no DOM, ou quando uma nova tentativa de navegação não resultava em alteração do conteúdo da página, indicando o esgotamento dos itens disponíveis. Essa abordagem incremental proporcionou maior previsibilidade no processo de extração, permitindo também a validação contínua dos dados a cada página e a possibilidade de retomar a coleta a partir de qualquer ponto, caso a execução fosse interrompida. A estratégia demonstrou-se eficaz especialmente em *sites* com estrutura HTML estável, garantindo uma coleta organizada, controlada e alinhada aos objetivos do sistema desenvolvido.

4.1.3 *Site* com Renderização por Visibilidade: Captura Condicionada à Exibição

Durante a implementação do processo de coleta no site Neshaprime, foi identificada uma limitação crítica relacionada ao carregamento das imagens dos itens. Especificamente, observou-se que, quando a aplicação era executada em modo *headless* padrão, isto é, sem exibir graficamente a janela do navegador durante a execução automatizada, as imagens

não eram corretamente renderizadas, resultando em elementos vazios ou imagens quebradas no DOM, mesmo após aguardar os tempos adequados de carregamento. Curiosamente, ao executar o mesmo *script* em modo *headless* falso, com o navegador visivelmente aberto, simulando uma sessão real, as imagens carregavam normalmente, sem apresentar falhas. Esse comportamento está relacionado à forma como muitos *sites* modernos, especialmente *e-commerces* e *marketplaces* visuais, adotam técnicas de *lazy loading* ou renderização condicional por visibilidade. Esse tipo de renderização consiste em só carregar elementos visuais, como imagens, quando eles estão visíveis na *viewport* do navegador, ou seja, quando o usuário, ou o Selenium, rola a página até que aquele elemento esteja efetivamente "à vista". Em modo *headless* tradicional, como o navegador não possui um ambiente gráfico real, alguns mecanismos internos de renderização são desativados ou simulados de forma parcial, o que compromete o carregamento completo de recursos visuais não críticos, como imagens decorativas ou miniaturas de produtos.

Para contornar esse problema, optou-se por implementar uma técnica baseada em captura de tela, *screenshot*, como gatilho de renderização. Essa abordagem consiste em forçar o navegador a renderizar completamente os elementos na tela antes de capturar sua imagem via código, o que, por consequência, obriga a *engine* de renderização a carregar as imagens corretamente. No Selenium, essa técnica foi aplicada com o método "driver.get_screenshot_as_png() ou element.screenshot('nome.png')", que ao solicitar a imagem da interface também força o carregamento real dos recursos visuais, mesmo em modo "*headless*. Essa solução mostrou-se eficaz, restabelecendo o comportamento esperado de renderização das imagens no processo de scraping. Embora o tempo exato da execução do comando de captura de tela *(screenshot)* não tenha sido mensurado com precisão, é possível inferir que seu impacto no desempenho geral da aplicação foi mínimo. Isso se deve ao fato de que a captura ocorre diretamente em nível de código, sem a necessidade de salvar os arquivos gerados em disco. Por operar inteiramente em memória, essa etapa evita operações de escrita em armazenamento persistente, que normalmente representam um dos principais gargalos de desempenho em processos automatizados. Dessa forma, mesmo sem uma medição objetiva, considera-se que a influência dessa ação no tempo total de execução foi pouco significativa.

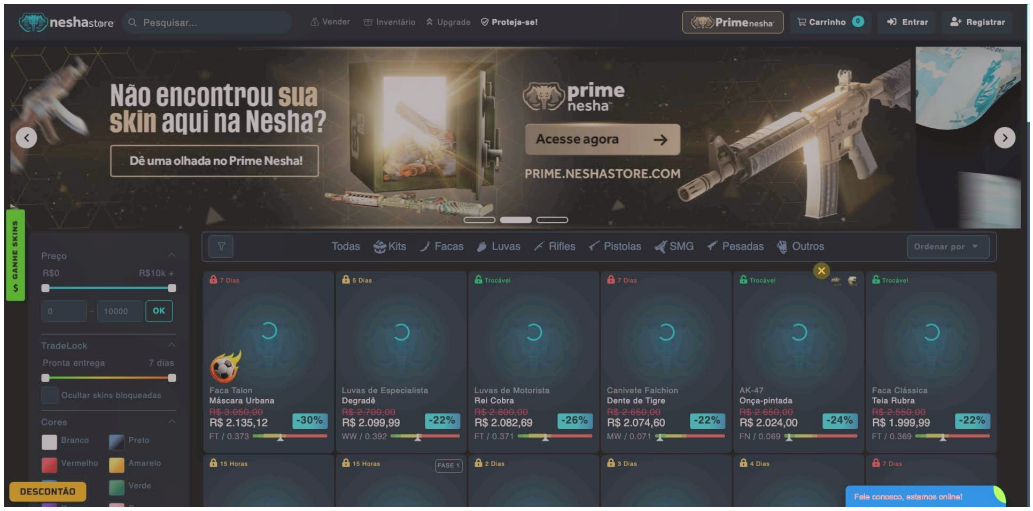

Figura 2 – Identificação do problema na renderização de imagens via screenshot.

Fonte: (NESHASTORE, 2025)

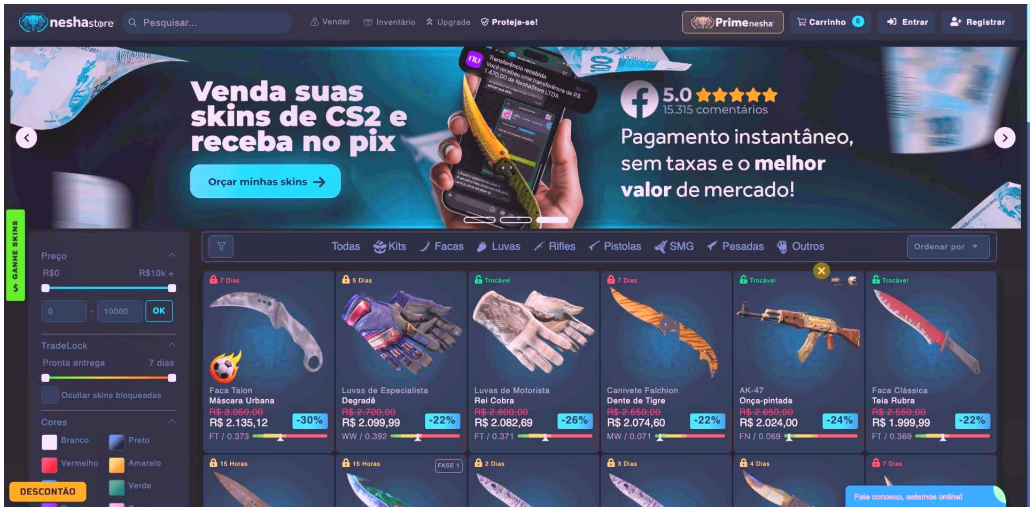

Figura 3 – Renderização correta após captura forçada com screenshot.

Fonte: (NESHASTORE, 2025)

Embora existam outras alternativas para lidar com esse tipo de limitação, como o uso de "element.scrollIntoView()"ou a alteração de atributos, data-src para src, ambas foram testadas e se revelaram ineficazes, não provocavam o carregamento das miniaturas quando em modo *headless*. Outras soluções, como a utilização de Playwright com simulação avançada de visibilidade ou a injeção de *scripts* JavaScript mais complexos, também foram consideradas, mas demandariam reescrita significativa do código existente.

Além disso, o uso de *screenshots* agregou valor ao sistema ao permitir a validação visual da coleta, como parte do processo, as imagens capturadas puderam ser utilizadas

tanto para testes automatizados quanto para validações manuais, conferindo um nível adicional de controle de qualidade sobre os dados extraídos.

Em resumo, a técnica de renderização por visibilidade com uso de captura de tela foi uma solução simples, porém eficaz, para contornar a limitação do carregamento parcial em modo *headless*. Sua adoção garantiu a integridade das imagens coletadas no *site* NeshaPrime e assegurou a completude dos dados, sem comprometer a automação e a escalabilidade do sistema.

4.1.4 Site com Proteção via Cloudflare: Navegação Automatizada com ChromeDriver

Durante o desenvolvimento do projeto, foi identificado um desafio relacionado à proteção implementada por um dos sites analisados. Após um período de manutenção de aproximadamente 24 horas, o site CS.Money retornou ao ar com uma camada adicional de segurança fornecida pela Cloudflare, cujo objetivo é bloquear acessos automatizados e proteger a plataforma contra bots.

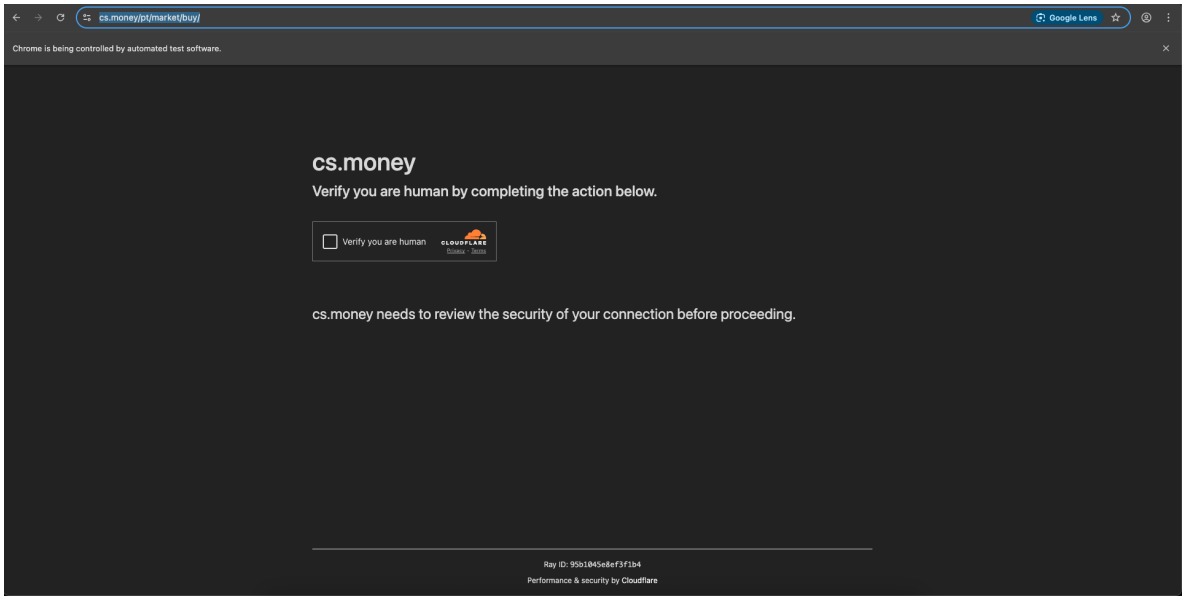

Figura 4 – Cloudflare *Privacy Term*

Fonte: (CS.MONEY, 2025)

Essa mudança impediu temporariamente a execução do processo de coleta automatizada de dados, que até então era realizado utilizando o ChromeDriver em sua versão convencional. A Cloudflare detectava e bloqueava as tentativas de navegação automatizada, redirecionando os acessos para páginas de verificação humana.

Para contornar essa restrição, foi adotada a biblioteca undetected_chromedriver, uma ferramenta projetada para disfarçar a automação do navegador, tornando-a mais semelhante a uma sessão de navegação humana. Essa biblioteca atua ajustando parâmetros

do navegador e mascarando traços comuns de automação, evitando que mecanismos de proteção como o da Cloudflare identifiquem o acesso como sendo feito por um robô.

Com essa solução, foi possível restabelecer a extração automatizada no site CS.Money, garantindo a continuidade da coleta de dados mesmo diante de barreiras de segurança adicionais.

4.2 Padronização e Integração dos Dados Extraídos

Após a conclusão da etapa de extração dos dados brutos provenientes dos diferentes *sites*, foi necessário realizar um processo sistemático de estruturação, limpeza e unificação das informações. Inicialmente, os dados extraídos eram salvos em arquivos temporários no formato .CSV, contendo todas as colunas relevantes, como nome da *skin*, nome da arma, preço, desgaste, *float* e *link* da imagem. No entanto, durante as primeiras execuções, observou-se que parte dessas informações chegava ao destino de forma inconsistente ou contaminada por ruídos, como textos promocionais, espaços desnecessários, símbolos redundantes ou formatos divergentes entre os *sites*.

Essa etapa é comumente conhecida na ciência de dados como *data wrangling* ou *data cleansing*, e representa uma das fases mais críticas para assegurar a qualidade e integridade dos dados antes de sua análise ou visualização. Alguns exemplos de inconsistências detectadas incluem, o campo de preço contendo informações como "R$ 15,00 | 10% OFF", ou nomes de armas com variações como "AK-47 | Redline"versus "AK47 Redline", o que dificultaria o agrupamento e comparação entre *sites*.

Diante dessa necessidade de tratamento, optou-se pelo uso da biblioteca Pandas, uma das mais completas e utilizadas ferramentas para manipulação de dados estruturados em Python. O Pandas oferece funcionalidades robustas para criação e operação sobre DataFrames, uma estrutura de dados tabular altamente eficiente para armazenar, filtrar, transformar e analisar grandes volumes de informações. A flexibilidade do Pandas permitiu o carregamento direto dos dados dos arquivos .CSV ou das coletas em tempo real, facilitando a integração com o restante do *pipeline* de processamento.

O primeiro passo foi a limpeza textual dos dados. Para isso, utilizou-se uma combinação das funções nativas do Pandas com expressões regulares, regex, através do módulo "re", integrado ao Python. Essa estratégia permitiu, por exemplo, extrair somente o valor numérico dos campos de preço, descartando textos promocionais como "10% OFF", além de normalizar espaços em branco, remover caracteres especiais e aplicar a substituição padronizada de termos. O método .str.replace() foi amplamente utilizado para transformar colunas inteiras de texto com regras definidas programaticamente.

Além da limpeza, também foi realizada a padronização semântica de alguns campos.

Essa etapa consistiu em unificar variações de nomenclaturas entre sites diferentes. Por exemplo, armas listadas como "AWP"em um site podiam aparecer como "A.W.P"ou "awp"em outro. A padronização foi aplicada utilizando transformações do tipo .str.lower() e mapeamentos via dicionários Python integrados aos DataFrames, assegurando consistência na base de dados.

Para garantir a uniformidade linguística dos dados exibidos ao usuário final, especialmente no que diz respeito aos nomes das armas e *skins*, foi utilizado um *Seeder* JSON, contendo os nomes de todos os itens relevantes em português e inglês. Esse *Seeder* funcionou como um dicionário de referência para mapeamento e substituição de nomes. Após a concatenação final dos dados provenientes dos diferentes *sites*, os campos de nomes foram mapeados e padronizados conforme a versão em inglês especificada no *Seeder*, o que viabilizou comparações e filtros globais de forma padronizada, independentemente da origem dos dados.

Outro aspecto relevante no processo de unificação dos dados foi a conversão monetária. Como alguns dos sites utilizados apresentavam os preços dos itens em dólares americanos (USD), tornou-se necessário convertê-los para reais brasileiros (BRL), a fim de manter a coerência na apresentação dos resultados. Inicialmente, foi empregada a biblioteca `forex_python.converter`, por meio da classe `CurrencyRates`, que permite obter taxas de câmbio atualizadas de forma prática e integrada ao ambiente de desenvolvimento. No entanto, visando garantir maior robustez ao processo, foi implementado um mecanismo de contingência que recorria à *Frankfurt Exchange Rate API* caso a biblioteca principal apresentasse falhas ou indisponibilidade. O valor em dólar de cada item era então multiplicado pela taxa de conversão obtida e armazenado em uma nova coluna do *DataFrame*, já convertido e formatado em reais, com a devida precisão decimal.

Após a limpeza individual de cada conjunto de dados, foi executada a unificação das tabelas em uma estrutura única, por meio da função "pandas.concat()". Essa função permitiu concatenar os dados horizontalmente, mantendo a integridade das colunas comuns e preenchendo valores ausentes quando necessário. A unificação também permitiu a inclusão de uma coluna adicional identificando a origem do dado através de um *hiperlink* que levavá até o item na página de origem, viabilizando o acesso do usuário ao item desejado para compra.

Antes da consolidação final dos dados em um único conjunto estruturado, foram aplicadas etapas intermediárias de filtragem com o objetivo de otimizar a qualidade e a relevância das informações apresentadas ao usuário. Durante o processo de coleta paralela, os dados extraídos de diferentes fontes foram armazenados temporariamente em estruturas separadas, utilizando execução concorrente para acelerar a obtenção. Após o término da raspagem, os dados foram reunidos em um único *DataFrame*, no qual foram aplicadas filtragens específicas, como restrições de intervalo para o atributo de desgaste (*float value*).

Esse tipo de tratamento antecede a etapa de visualização e visa evitar a sobrecarga do sistema com dados irrelevantes, além de eliminar a necessidade de retrabalho na coleta ou de leitura externa. O código implementado demonstra essa lógica, em que os resultados individuais de cada fonte são concatenados, validados e posteriormente submetidos à filtragem condicional antes do armazenamento definitivo. Essa abordagem garante maior eficiência no fluxo de dados e contribui para a agilidade na resposta da aplicação frente às preferências do usuário.

Ao final desse processo de estruturação, o DataFrame consolidado encontrava-se em seu formato ideal para visualização, contendo apenas dados limpos, padronizados e organizados, prontos para serem apresentados na interface *web*. A adoção do Pandas como núcleo do tratamento de dados se mostrou estratégica, possibilitando não apenas uma limpeza eficaz, mas também uma manipulação extensível e escalável, permitindo futuras integrações com análises estatísticas ou geração de relatórios.

Esse *pipeline* de estruturação e unificação garantiu que os dados coletados pudessem ser utilizados de forma confiável e eficiente, servindo como base sólida para as etapas seguintes de visualização interativa, tomada de decisão por parte do usuário.

4.3 Construção da Interface de Consulta e Visualização Interativa

Com os dados já coletados, tratados e unificados por meio das etapas anteriores, iniciou-se o desenvolvimento da interface gráfica de interação com o usuário, cujo objetivo central era permitir a execução de pesquisas de forma intuitiva, rápida e personalizável. Até então, a interação com os parâmetros de busca era feita de forma textual ou diretamente via código, o que restringia a usabilidade para usuários não técnicos. Dessa forma, tornou-se essencial a criação de uma camada visual acessível e funcional.

Para a construção da interface, optou-se pelo uso da biblioteca Dash Bootstrap Components, um *framework* em Python baseado na combinação entre o Dash, Plotly, e os componentes visuais do Bootstrap. Essa escolha se deu por sua integração nativa com a linguagem principal do projeto, sua simplicidade de uso e pela capacidade de criar *layouts* responsivos e interativos.

A tela principal da aplicação foi inspirada no conceito minimalista de ferramentas como o Google, uma interface limpa e centralizada com foco em um campo de busca. No topo da página, foi posicionada uma logo representativa do sistema, seguida por um campo de entrada, onde o usuário pode digitar o nome da *skin* ou arma que deseja consultar. Ao lado do campo, há um botão de pesquisa, que aciona a coleta e visualização dos dados com base nos critérios fornecidos.

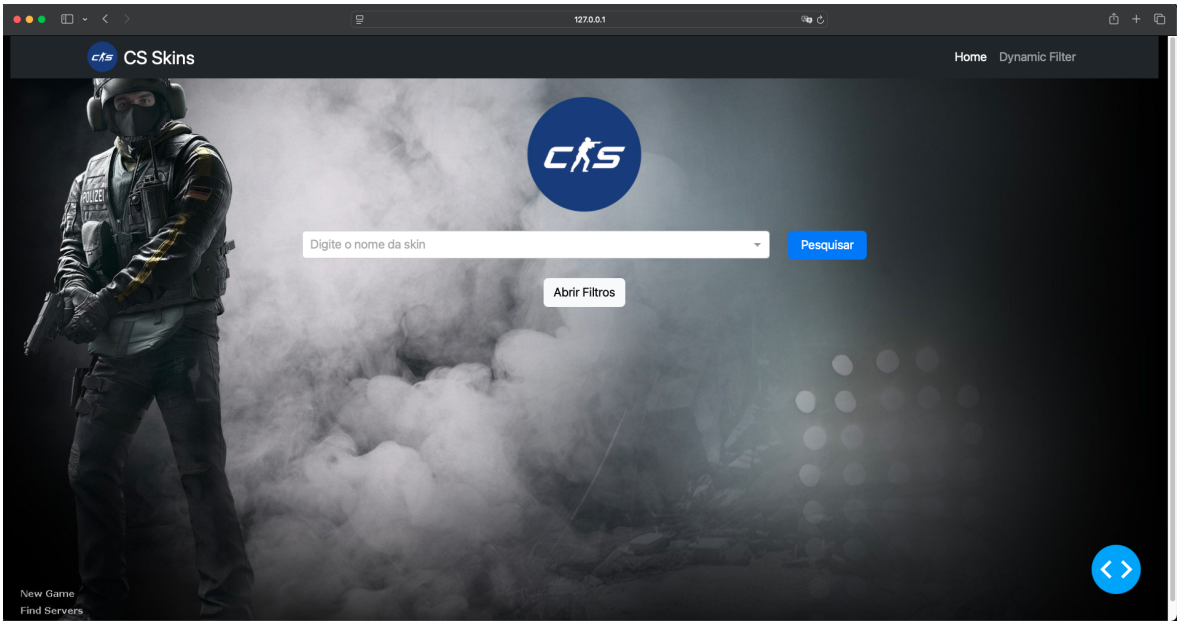

Figura 5 – Visão geral da tela principal da aplicação.

Fonte: Próprio Autor

Para permitir pesquisas refinadas, foi adicionado abaixo do campo principal um botão nomeado "Filtros", o qual, ao ser acionado, expande uma barra lateral, *sidebar*, contendo campos adicionais. Nessa área, o usuário pode definir parâmetros mais específicos como faixa de preço, intervalo de *float* e grau de desgaste. Ao clicar no botão "Pesquisar", todos os parâmetros inseridos, inclusive os filtros opcionais, são aplicados à busca, acionando o sistema de extração e filtragem dos dados.

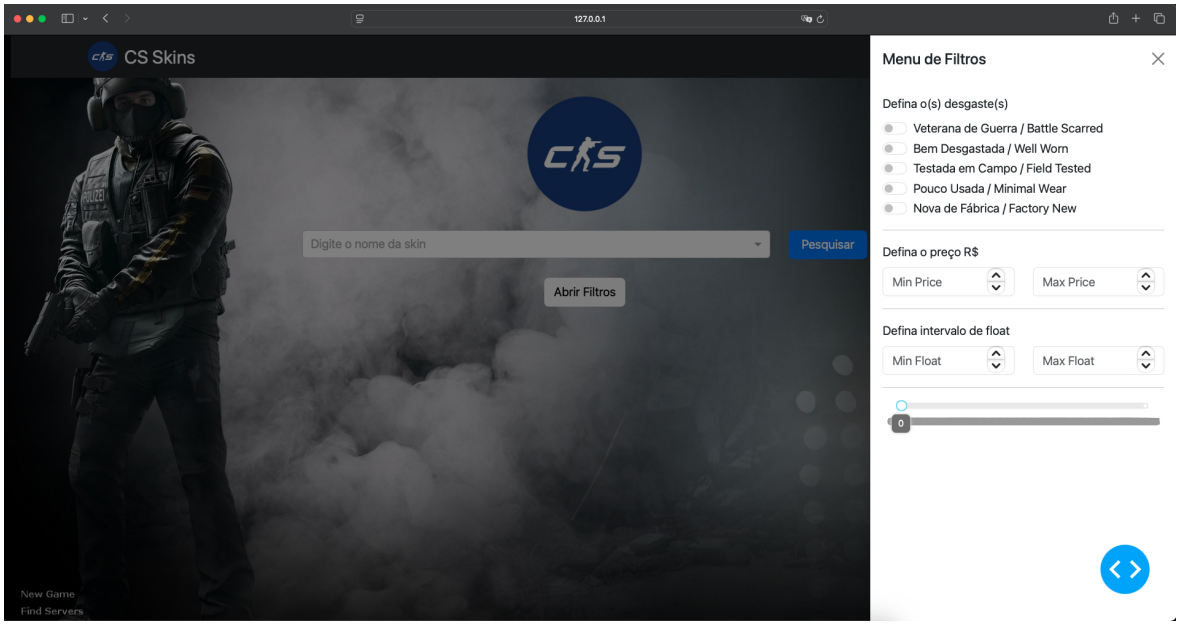

Figura 6 – Menu de filtros com múltiplas opções de escolha.

Fonte: Próprio Autor

Concluída a etapa de coleta de dados, na qual são realizadas as raspagens dos conteúdos dos sites com base nos parâmetros previamente definidos, o sistema organiza os resultados obtidos e os apresenta ao usuário por meio de uma tabela interativa e responsiva. Essa tabela é construída dinamicamente e contém os seguintes campos:

- Nome da arma + nome da *skin* (padronizados em inglês
- Tipo de desgaste (também padronizado)
- Preço (convertido para real)
- Valor de *float*
- *Link* de acesso ao item no *site* de origem
- Foto do item

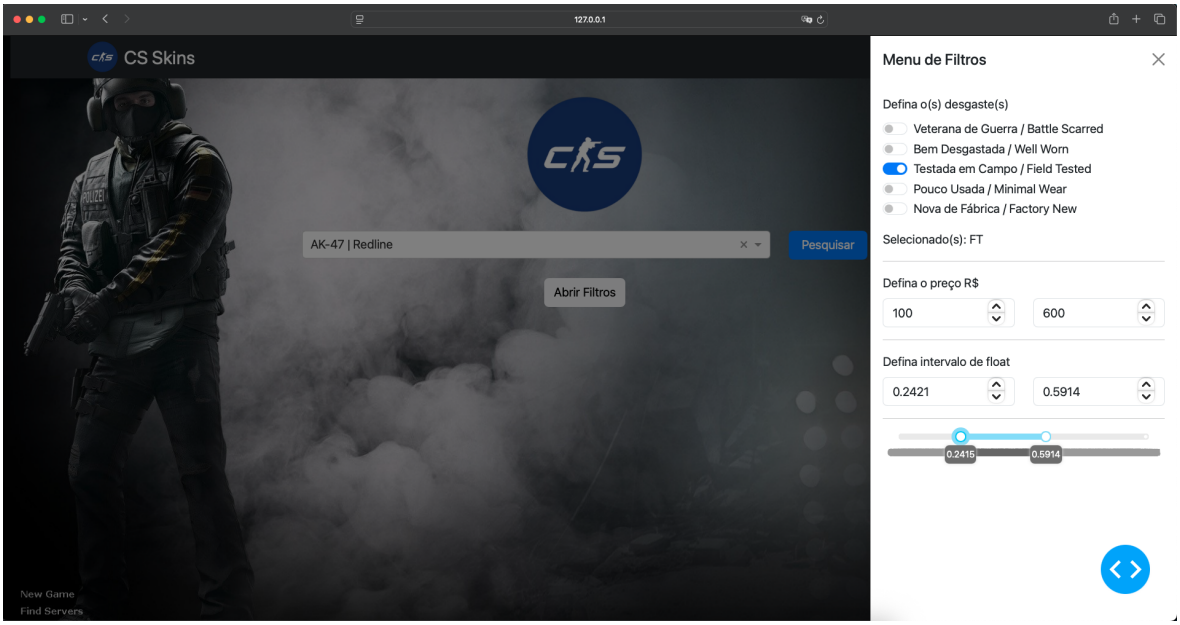

Figura 7 – Interface de filtros com seleção ativa de opções.

Fonte: Próprio Autor

A tabela conta com funcionalidades adicionais de ordenação, crescente, decrescente, em todas as colunas, paginação automática, e uma barra de busca interna, permitindo ao usuário refinar ainda mais os dados visualizados sem a necessidade de refazer a pesquisa.

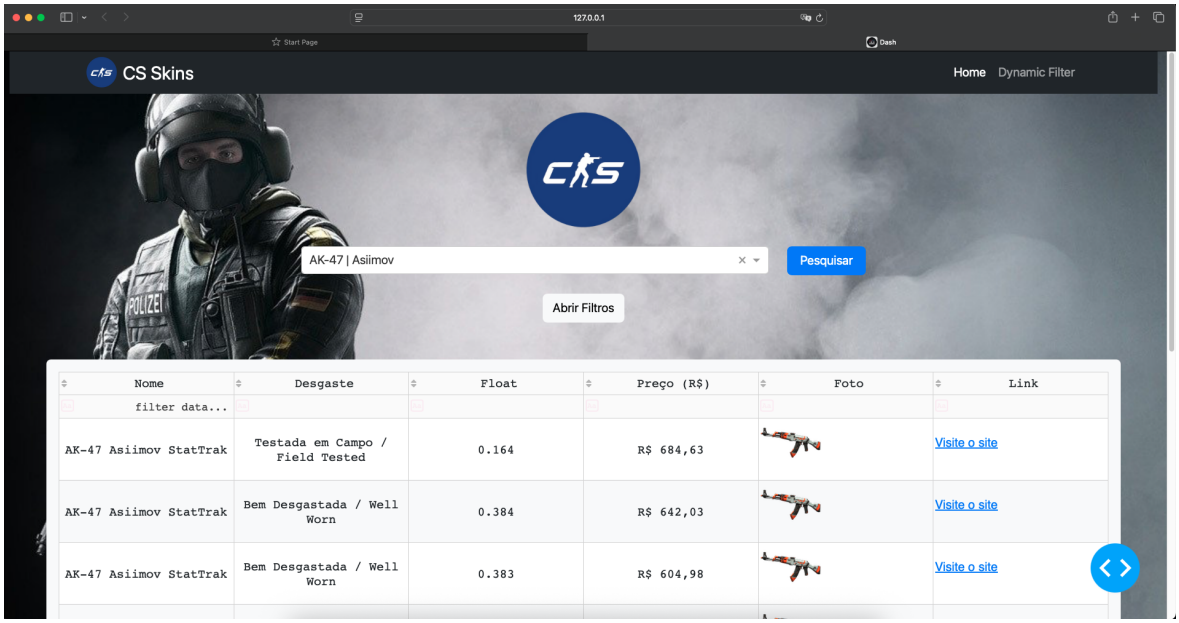

Figura 8 – Visualização dos Itens no Dashboard.

Fonte: Próprio Autor

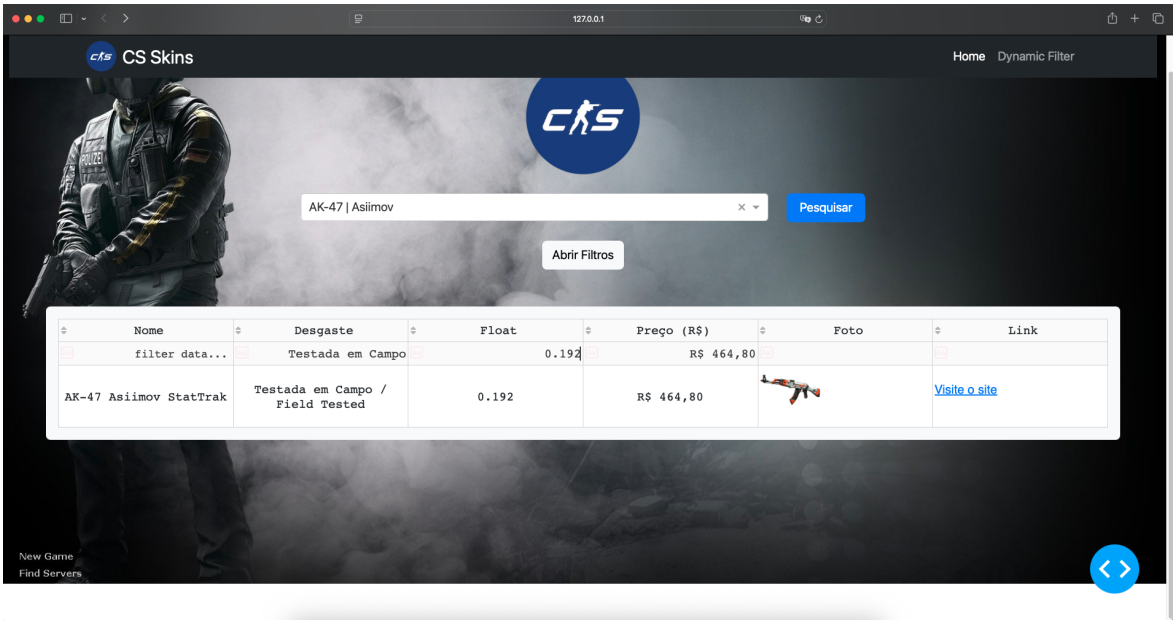

Figura 9 – Visualização de um Item Específico no Dashboard

Fonte: Próprio Autor

Para facilitar a digitação correta dos nomes, especialmente em casos em que o usuário insere os nomes em português, foi implementado um sistema de autocompletar baseado em um "Seeder JSON". Esse arquivo contém o mapeamento completo entre os nomes das armas e skins em português e inglês. Assim, à medida que o usuário digita, o sistema sugere a correspondência correta em inglês, linguagem base utilizada internamente, reduzindo erros e aumentando a precisão da busca.

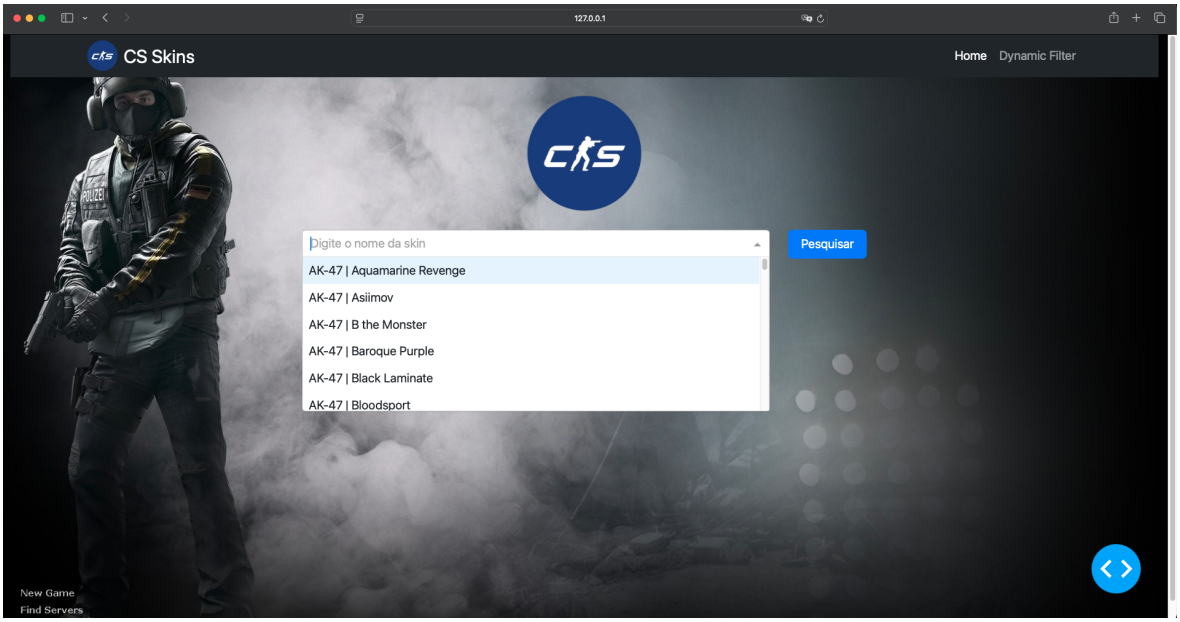

Figura 10 – Campo de busca com sugestão automática baseado em um seeder JSON.

Fonte: Próprio Autor

Outro ponto importante foi o tratamento especial necessário para o *site* CS.Money, que utiliza um modelo de exibição mais complexo. Diferente dos demais, o CS.Money mantém a navegação dos itens sob uma única URL fixa, utilizando carregamento dinâmico por meio de modais, janelas sobrepostas. Por isso, ao invés de redirecionar o usuário diretamente para o *link* de um item específico, o sistema simula a abertura do modal correspondente dentro da própria página via automação. O processo exige que o Selenium posicione o item desejado na tela, visível ao DOM, acione sua visualização e mantenha o modal aberto antes de retornar a interface gráfica simulada ao usuário. Essa exceção foi tratada com atenção no projeto, garantindo uma experiência fluida e equivalente à dos outros sites, apesar da diferença estrutural.

Por fim, uma seção complementar de entrada avançada foi desenvolvida no sistema, permitindo a futura integração de um filtro dinâmico configurável pelo usuário. Essa ferramenta, que será detalhada em seção própria, permite que o próprio usuário insira seletivamente a URL e seletores HTML de um site qualquer, para aplicar a lógica de extração sobre novas plataformas de *skins*. Embora ainda em fase experimental, essa funcionalidade representa uma evolução significativa da proposta, permitindo extensibilidade da ferramenta e adaptabilidade a novos contextos.

4.4 Filtro Dinâmico: Personalização da Extração e Interação pelo Usuário

Ao longo do desenvolvimento deste projeto, diversos desafios técnicos foram enfrentados durante o processo de extração de dados em múltiplos *sites* de *skins*. Problemas como diferentes estruturas de carregamento dos itens, bloqueios de conteúdo dinâmico, padrões de rolagem inconsistentes, validações *anti-bot*, e até mesmo restrições na renderização gráfica em modo *headless* exigiram soluções específicas para cada caso. Embora a estrutura geral das páginas apresentasse semelhanças, especialmente na organização de *grids* e apresentação dos dados, as variações e restrições pontuais se mostraram decisivas no sucesso ou fracasso da coleta automatizada.

Foi a partir dessa constatação que surgiu a proposta de criação de um filtro dinâmico configurável pelo usuário, uma interface que torna possível aplicar as soluções desenvolvidas ao longo deste trabalho de forma modular e reutilizável. Essa funcionalidade foi pensada como produto direto da experiência técnica adquirida, permitindo que o próprio usuário configure o sistema para realizar *web scraping* em *sites* que não foram previamente tratados na aplicação, desde que esses *sites* sigam padrões mínimos de estruturação HTML.

A ideia central do filtro dinâmico é oferecer ao usuário uma ferramenta flexível, sem exigir conhecimento técnico profundo, mas que ainda assim permita o controle de aspectos cruciais do processo de coleta. Tomou-se como inspiração o perfil de usuários da era dos *blogs* nos anos 2000, como os editores do Blogspot, que, mesmo sem formação técnica, conseguiam fazer alterações visuais e funcionais em suas páginas ao interagir com trechos de HTML. Da mesma forma, aqui o usuário é incentivado a inserir seletivamente *tags* ou seletores HTML, como *class*, *id* ou elementos CSS, referentes aos campos essenciais do *scraping*, nome do item, imagem do item e preço.

A interface do filtro dinâmico foi desenvolvida com foco em simplicidade e clareza. O usuário insere:

- A URL do *site* alvo, preferencialmente já filtrada conforme seu interesse
- Nove campos de entrada para os seletores HTML responsáveis por identificar:
  - A *grid* do *container*;
  - O nome do item;
  - A imagem;
  - O preço;
  - O nome do botão de próxima página (em caso de utilização de *scroll* por paginação);

  - A *tag* do elemento de aceitar cookies;
  - A *grid* do banner publicitário;
  - A *div* do banner.

- E seleciona, por meio de caixas de múltipla escolha *(checkboxes)*(, os comportamentos e técnicas específicas que deverão ser aplicados durante a extração.

Entre os comportamentos configuráveis, estão:

- O tipo de carregamento da página (estático, dinâmico ou *"undetect"*, usado para mascarar a ação do bot)
- O formato de rolagem (infinita, híbrida, paginada ou sem rolagem)
- A presença ou ausência de proteções *"anti-bot"* (como validações por *User-Agent* ou *delays* baseados em tempo de interação)
- A necessidade de renderização gráfica (quando a renderização condicional é ativada somente com elementos visíveis)
- E outros parâmetros utilizados ao longo do desenvolvimento, que demonstraram eficácia em situações específicas.

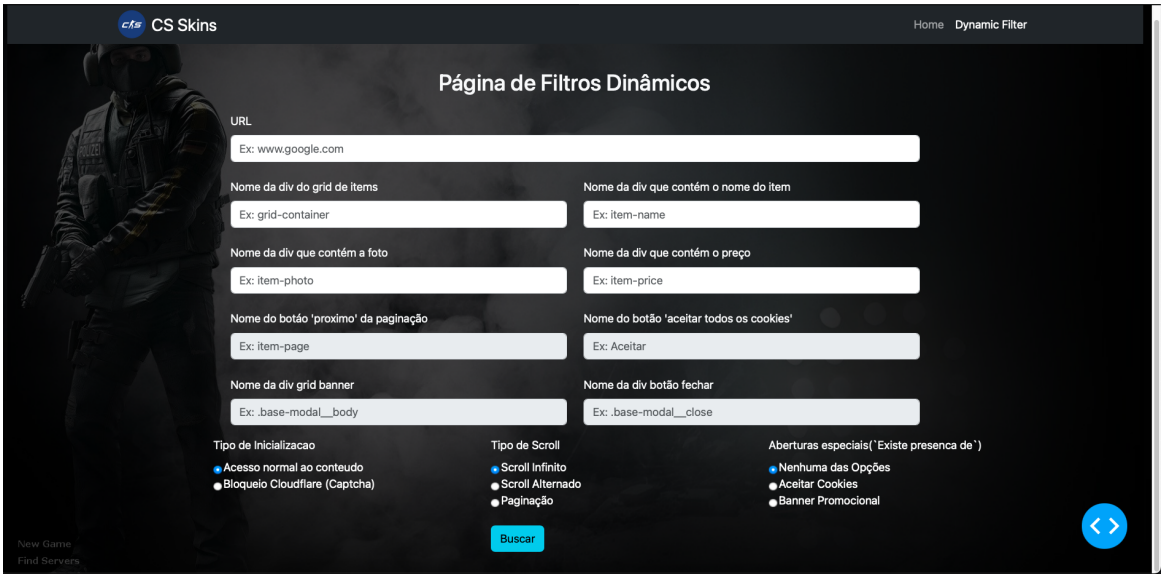

Figura 11 – Filtro Dinâmico

Fonte: Próprio Autor

Essa abordagem confere ao sistema um grau elevado de personalização, transformando-o de uma ferramenta específica para um conjunto fixo de *sites* em uma plataforma semi-genérica

de coleta de dados *web*. Com o filtro dinâmico, o usuário consegue adaptar o funcionamento da aplicação a novos *sites*, desde que identifique corretamente as *tags* relevantes e compreenda minimamente a estrutura do HTML.

Após a coleta, os dados extraídos são organizados internamente em um DataFrame. Finalizada a coleta, os resultados são disponibilizados ao usuário em um arquivo no formato .CSV, pronto para ser analisado, armazenado ou integrado a outras ferramentas.

Essa funcionalidade não apenas consolida os aprendizados adquiridos durante o desenvolvimento, mas também representa uma contribuição prática e robusta para usuários com diferentes níveis de conhecimento técnico. O filtro dinâmico amplia a utilidade da aplicação, tornando-a mais escalável, flexível e autônoma, características essenciais em projetos de extração de dados em ambientes *web* dinâmicos.

4.4.1 Personalização da Estrutura do *Site* Alvo pelo Usuário

A interface do filtro dinâmico, desenvolvida com a biblioteca Dash Bootstrap, oferece ao usuário a possibilidade de configurar manualmente os parâmetros de extração de dados com base na estrutura de um *site* alvo. Essa funcionalidade foi concebida para atender à necessidade de escalabilidade e adaptabilidade do sistema, permitindo a inclusão de novos *sites* de *skins*, ou outros com estrutura semelhante, sem necessidade de modificações diretas no código-fonte.

Por meio da interface, o usuário seleciona ou insere a URL do *site* desejado, desde que ele possua uma estrutura compatível com os modelos analisados durante o desenvolvimento, por exemplo, listagens de produtos organizadas em *grids* ou blocos HTML padronizados. Em seguida, são fornecidos quatro campos principais de entrada, onde o usuário deve informar os seletores HTML, *tags*, classes ou identificadores, correspondentes aos seguintes elementos da página:

- *Grid Container* do item
- Nome do item (exemplo: nome da arma ou da *skin*)
- Preço do item
- Imagem do item
- *Hiperlink* do item

Esses campos permitem ao sistema localizar e extrair as informações corretas de acordo com a estrutura específica da página fornecida. No entanto, considerando que diferentes *sites* utilizam técnicas distintas de carregamento, proteção e renderização, a interface também oferece um conjunto de opções avançadas de personalização, que refletem

diretamente nos mecanismos e estratégias adotadas pelo sistema durante a extração. As opções disponíveis incluem:

- Tipo de *driver* de automação: o usuário pode escolher entre o *WebDriver* tradicional ou o *"UndetectDriver"*, este último sendo mais adequado para *sites* que aplicam mecanismos de detecção de *bots* e automações, como análise de *user agents*, *headless detection* ou requisições suspeitas
- Presença de captcha ou mecanismos *anti-bot*: se o *site* apresentar etapas de verificação por captcha (como reCAPTCHA), o usuário pode sinalizar isso para que a ferramenta aplique temporizações maiores, utilize *drivers* não-*headless* ou solicite intervenção manual, caso necessário
- Tipo de carregamento de conteúdo *(scrolling)*: essa opção é fundamental para que a aplicação compreenda como os itens são exibidos na página. As opções disponíveis incluem:
  - Rolagem infinita padrão: itens carregam à medida que a página é rolada verticalmente
  - Rolagem infinita dinâmica ou híbrida: exige foco em elementos específicos ou depende de modais
  - Paginação tradicional: o conteúdo está dividido em páginas navegáveis por botões próxima ou anterior
- Renderização de imagens via *screenshot*: quando o carregamento visual dos itens depende de estarem visíveis na tela (comum em *frameworks* como React ou Vue), o usuário pode optar por ativar o modo de captura gráfica, utilizando *screenshots* como forma de garantir que os elementos estejam efetivamente carregados.
- Proteções como Cloudflare ou restrições de IP: caso o *site* utilize sistemas como o Cloudflare para bloquear requisições automatizadas, o usuário pode marcar essa opção para que a ferramenta adote uma abordagem mais cautelosa, com simulação de comportamento humano e controle de tempo de navegação.

A partir dessas escolhas, o sistema monta internamente um perfil comportamental da página alvo, permitindo aplicar a combinação ideal de técnicas e ferramentas de extração. Essa abordagem modular e parametrizável garante que o *scraping* ocorra de forma adaptada à realidade de cada *site*, aumentando significativamente as chances de sucesso da coleta sem comprometer a integridade ou a precisão dos dados.

Essa personalização, embora técnica, foi pensada para ser acessível a usuários com conhecimentos básicos de estrutura HTML. O objetivo é empoderar o usuário final,

fornecendo controle sobre a lógica de extração sem exigir alterações no código, tornando a ferramenta versátil, extensível e voltada para aplicações reais em contextos dinâmicos de *e-commerce* e *marketplaces* de jogos.

4.4.2 Avaliação Experimental do Mecanismo de Filtro Dinâmico

A fim de realizar um experimento prático com a ferramenta de extração baseada em filtros dinâmicos, foi conduzido um teste utilizando o *site* ShadowPay, mais especificamente a seção de itens do jogo Dota 2 com filtros aplicados por tipo, nome do item, preço e imagem.

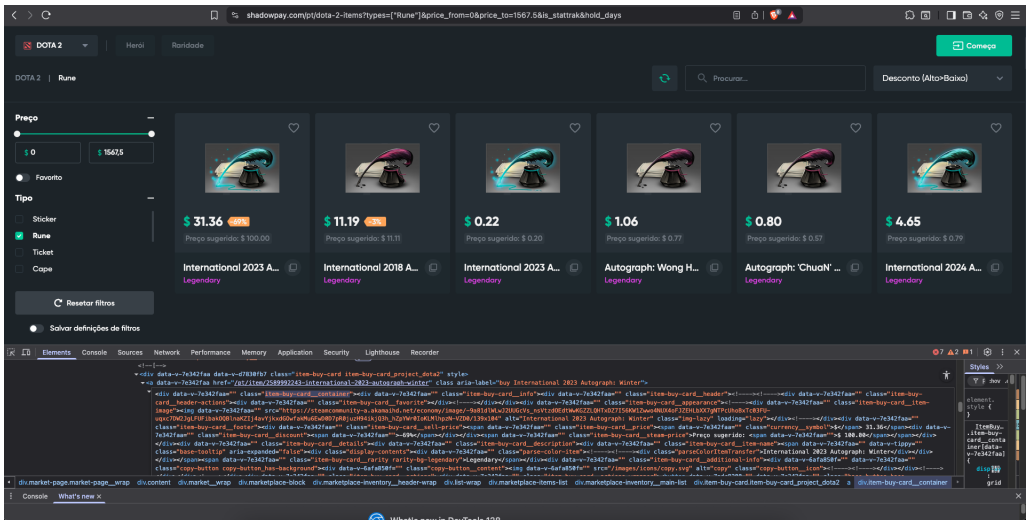

Figura 12 – *Site* ShadowPay

Fonte: (ShadowPay, 2025)

Durante a execução, optou-se pelo modo de inicialização padrão do *WebDriver*. Além disso, foi empregada a técnica de *scroll* alternado.

- A classe da *grid* principal dos itens (.item-buy-card___container)
- A classe da imagem (.img-lazy)
- A classe do nome do item (.parseColorItemTransfer)
- A classe do preço (.item-buy-card___price)

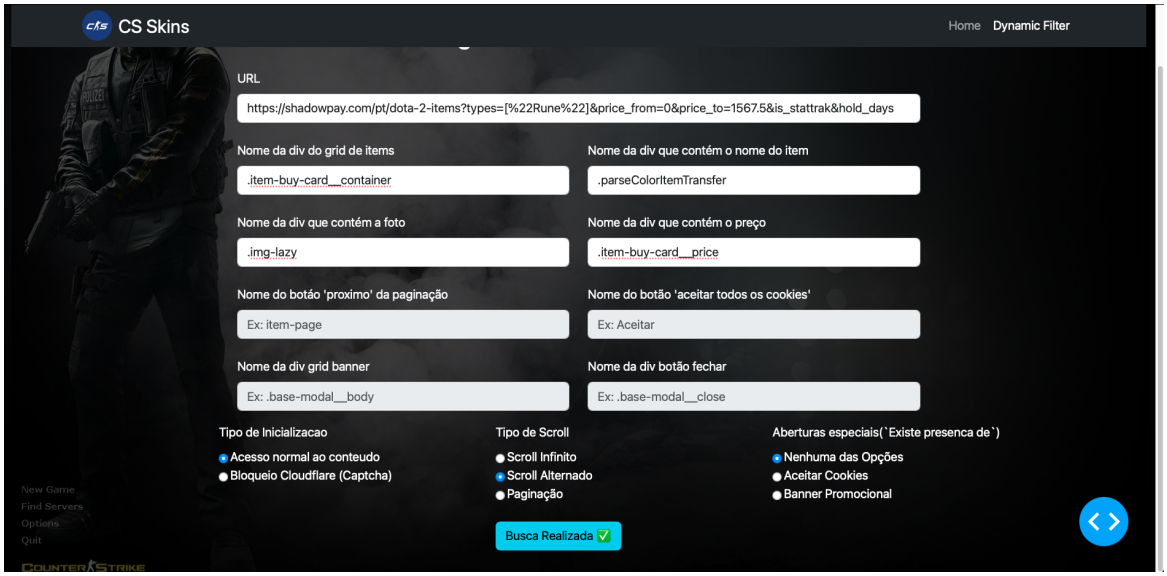

Figura 13 – Filtro Dinâmico - Scraping site Dota 2 (ShadowPay, 2025)

Fonte: Próprio Autor

Como resultado, a ferramenta foi capaz de extrair corretamente os dados dos itens listados, como nome, preço e imagem, e gerar um arquivo .csv contendo as informações estruturadas.

Figura 14 – Arquivo CSV Extraído

Fonte: Próprio Autor

5 Resultados e Análise

5.1 Comparativo entre abordagens aplicadas em cada *site*

Durante a fase de desenvolvimento do sistema de coleta, observou-se que, embora os *sites* compartilhassem estruturas similares em termos de organização dos dados, como *grids* de listagem de itens, imagens, preços e *links*, cada um apresentou particularidades técnicas que exigiram adaptações específicas nas abordagens de *scraping*. Assim, foi necessário aplicar estratégias individualizadas para lidar com diferentes comportamentos de carregamento, renderização e proteção de conteúdo.

O *site* CSMoney foi um dos mais desafiadores no contexto de rolagem e interação automatizada. Ele apresentava um *grid* interno com rolagem própria, separado da rolagem geral da página. Quando se tentava utilizar comandos de *scroll* no Selenium para movimentar o conteúdo do *grid*, o *site* frequentemente redirecionava o foco para o rodapé da página principal, em vez de rolar apenas a seção interna dos itens. Esse comportamento impedia o carregamento correto dos dados desejados. Para contornar o problema, foi implementado um controle refinado com foco no elemento final da lista de itens visíveis, forçando a permanência dentro do *grid* e garantindo que o carregamento de novos itens fosse ativado corretamente. Além disso, o CSMoney mantinha os dados dos itens sob modais dentro de uma única URL, o que dificultava a abertura externa dos produtos, exigindo que o Selenium interagisse diretamente com os filtros internos e abrisse os modais manualmente, garantindo a visibilidade correta das informações antes da coleta.

É pertinente destacar que, durante a fase de testes e ajustes dos *scripts* de *scraping*, o *site* CS.Money passou por um período de manutenção temporária. Esse evento coincidiu com a etapa em que o sistema realizava um número significativo de requisições automatizadas, o que inicialmente levantou a hipótese de que o aumento do tráfego poderia ter contribuído para a interrupção. No entanto, ao retornar à atividade, a plataforma foi atualizada com novas diretrizes, incluindo a imposição explícita de restrições relacionadas à proteção de propriedade intelectual dos dados disponibilizados. Diante dessa mudança, todos os dados previamente extraídos do CS.Money foram prontamente excluídos, e o *site* foi retirado do escopo do estudo de caso, de modo a respeitar suas políticas vigentes. Ainda assim, as técnicas desenvolvidas e aplicadas durante o processo de coleta inicial, como o uso de *drivers* indetectáveis e estratégias adaptativas frente a barreiras automatizadas, foram de grande relevância para o aprendizado técnico e contribuíram significativamente para o amadurecimento metodológico do projeto.

O MarketCS, embora também utilizasse carregamento por rolagem, apresentava

uma lógica distinta. O *site* carregava novos itens com base na visibilidade parcial de elementos anteriores, e não apenas ao atingir o final do *grid*. Tentativas com comandos como scrollIntoView() se mostraram ineficazes. A técnica final adotada foi semelhante à do CSMoney, o sistema mantinha o foco de rolagem no último item visível, evitando que o Selenium perdesse o contexto da área de busca. Isso garantiu que o conteúdo continuasse sendo carregado progressivamente sem interferência da rolagem global da página.

Já os *sites* NeshaPrime, NeshaStore e DashSkin apresentavam uma estrutura baseada em paginação tradicional, em que os itens são divididos em páginas navegáveis por meio de botões de próxima página. Essa abordagem se mostrou mais simples de automatizar, exigindo apenas que o Selenium identificasse e clicasse sequencialmente nesses botões. No entanto, foi necessário implementar condições de parada, o sistema interrompia a coleta caso o número máximo de itens fosse alcançado, ou se não fossem encontrados novos elementos após determinado número de páginas navegadas, o que evitava loops infinitos em caso de falhas silenciosas nos *sites*.

No caso específico do NeshaPrime, surgiu um desafio adicional relacionado à renderização condicional de imagens. Durante a execução do Selenium em modo *headless*, sem abertura visual do navegador, as imagens dos itens não eram carregadas corretamente. Curiosamente, ao executar o mesmo código em modo *headless* falso, com janela visível, os dados eram renderizados perfeitamente. Esse comportamento indicava que o site utilizava técnicas baseadas em visibilidade gráfica para ativar o carregamento dos elementos visuais. Para resolver esse problema, foi adotada a técnica de captura de tela, *screenshot*, como forma de forçar a renderização do conteúdo, validando visualmente os elementos antes de extraí-los. Outras abordagens comuns, como forçar a troca de atributos data-src para src, ou empurrar elementos para a *viewport* via JavaScript, também foram testadas, mas não apresentaram resultados satisfatórios.

Esse conjunto de experiências evidencia que, mesmo diante de estruturas aparentemente padronizadas, os detalhes de implementação variam significativamente entre *sites*. A eficácia do sistema proposto só foi possível devido à adoção de abordagens específicas e combinadas, ajustadas aos obstáculos individuais de cada plataforma. Essa diversidade de soluções serviu de base para o desenvolvimento do filtro dinâmico apresentado nas próximas seções, consolidando os aprendizados adquiridos ao longo do projeto.

5.2 Eficiência do processo de extração com base nos obstáculos encontrados

Com o refinamento progressivo das técnicas implementadas, foi possível observar ganhos substanciais de desempenho ao longo do desenvolvimento. Inicialmente, o processo de *scraping* era feito de forma sequencial e sem paralelismo, o que acarretava tempos de

resposta elevados e experiências pouco fluidas para o usuário. Um dos principais gargalos identificados estava no tempo de espera entre requisições e no carregamento de páginas com renderização dinâmica.

Para mitigar esses efeitos, foi incorporado ao final do desenvolvimento o uso de *Threads* como mecanismo de paralelização das coletas. A ideia central foi dividir a carga de trabalho, principalmente em consultas com múltiplos *sites*, entre diferentes execuções simultâneas, de forma a reduzir o tempo total de resposta. A implementação das *Threads* trouxe resultados concretos e expressivos.

Os testes demonstraram que, sem a utilização de *Threads*, a média de tempo necessário para completar uma busca completa era de 358,4337 segundos. Após a aplicação do paralelismo com múltiplas *Threads* coordenadas, o tempo médio caiu para 215,8118 segundos, representando uma redução de cerca de 39,8% no tempo total. Essa melhoria é significativa não apenas em termos técnicos, mas também sob o ponto de vista da experiência do usuário, que passa a receber os resultados de forma mais ágil e eficiente.

Além disso, com o uso de *Threads*, foi possível manter uma maior responsividade da interface e evitar bloqueios no carregamento de elementos gráficos e na manipulação de grandes volumes de dados via DataFrames. Isso consolidou o uso do paralelismo como uma das otimizações-chave do projeto, tornando o sistema mais escalável e adequado a ambientes com grande volume de informações.

5.3 Análise do filtro dinâmico como generalização dos aprendizados

Uma das maiores contribuições deste trabalho está na consolidação dos aprendizados obtidos durante o desenvolvimento sob a forma de uma ferramenta genérica e adaptável, o filtro dinâmico. A ferramenta foi concebida como uma camada de abstração, capaz de transformar o conhecimento técnico adquirido sobre estruturas HTML, carregamento assíncrono, renderização condicional e proteção *anti-bot*, em uma interface configurável pelo próprio usuário.

A proposta do filtro dinâmico é permitir que usuários com conhecimento básico de estrutura HTML, como reconhecimento de classes ou identificadores de elementos, possam configurar manualmente os parâmetros de coleta em qualquer *site* que possua uma organização visual compatível. Para isso, são fornecidos campos específicos para definição de seletores, *grid* do *container*, nome do item, preço, imagem e link, além de opções de personalização para indicar o tipo de carregamento, presença de captcha, necessidade de screenshot, entre outros.

A aplicação prática da ferramenta demonstrou que a lógica de *scraping* pode, sim, ser generalizada desde que os parâmetros críticos de extração sejam corretamente

identificados e fornecidos pelo usuário. Em testes com *sites* não contemplados originalmente no projeto, o filtro dinâmico foi capaz de realizar coletas funcionais, validando seu potencial como *framework* de coleta de dados *web* semi-estruturados.

Essa abordagem representa não apenas a flexibilidade da solução, mas também sua capacidade de escalabilidade futura. Ao transferir parte da lógica de decisão para o usuário, o sistema amplia seu espectro de uso para além do contexto de *skins* de Counter-Strike, podendo ser aplicado, com as devidas adaptações, em plataformas de *e-commerce*, catálogos visuais ou *marketplaces* especializados.

6 Considerações Finais

O presente trabalho teve como objetivo central desenvolver uma solução automatizada e flexível para a coleta, estruturação e exibição de dados provenientes de *sites* de *skins* do jogo Counter-Strike. Através da utilização de técnicas de *web scraping*, ferramentas de análise e bibliotecas gráficas, foi possível criar uma aplicação funcional e extensível, capaz de lidar com diferentes estruturas de páginas *web*, comportamentos dinâmicos de carregamento e restrições impostas por mecanismos de proteção dos *sites*.

A proposta foi conduzida com base em uma abordagem prática e iterativa, onde cada fase de desenvolvimento foi acompanhada por testes, ajustes e refinamentos. Inicialmente, foi realizada uma análise minuciosa da estrutura de múltiplos *sites* de *skins*, com foco em identificar padrões e divergências em seus sistemas de navegação e exibição de dados. A partir disso, foram aplicadas diferentes estratégias de coleta, de acordo com o comportamento específico de cada plataforma, como rolagem infinita, paginação tradicional ou renderização condicionada por visibilidade.

Entre os desafios enfrentados, destacam-se, o controle de rolagem em *grids* internos, como no caso do CSMoney, a inversão da lógica de carregamento em *sites* como MarketCS, a dificuldade de renderização de imagens no modo *headless*, como observado no NeshaPrime, e a presença de bloqueios automatizados, como captchas e Cloudflare, que surgiram principalmente durante a implementação do filtro dinâmico. Cada um desses obstáculos exigiu o desenvolvimento de soluções sob medida, como manipulação direta do DOM, controle baseado no último item visível, uso de screenshots para forçar renderizações e adoção de navegadores undetectables.

No tocante ao desempenho, a aplicação mostrou-se eficiente e responsiva. A implementação de paralelismo com *Threads* na etapa final do projeto representou um ganho concreto de produtividade e usabilidade, reduzindo significativamente o tempo total de extração de dados. Os testes demonstraram uma redução de aproximadamente 40% no tempo de resposta, evidenciando o impacto positivo dessa decisão arquitetural.

Um dos principais resultados do projeto foi o desenvolvimento de uma interface gráfica interativa, construída com a biblioteca Dash Bootstrap, que permitiu ao usuário realizar buscas de forma intuitiva, utilizando filtros personalizados e recebendo os resultados em tempo real, organizados em tabelas interativas com ordenação, paginação e busca local. Essa interface agregou valor à aplicação ao torná-la acessível mesmo para usuários com pouca ou nenhuma familiaridade com programação.

Além disso, a criação do filtro dinâmico se destacou como um desdobramento inovador do projeto. Ele consolidou os aprendizados adquiridos em uma ferramenta de

alto nível, permitindo que o próprio usuário, munido de conhecimentos básicos de HTML, possa configurar os parâmetros de coleta para qualquer *site* com estrutura similar. Essa funcionalidade amplia o escopo da aplicação, conferindo-lhe um caráter modular, genérico e escalável, capaz de se adaptar a novos domínios sem necessidade de reescrita do código principal. O uso de *tags* personalizadas e opções configuráveis, como tipo de rolagem, necessidade de visibilidade gráfica ou existência de captchas, torna o sistema altamente flexível.

Em termos de padronização dos dados, a utilização da biblioteca Pandas foi fundamental para a limpeza, estruturação e integração dos dados coletados. O uso de expressões regulares, formatação de preços, padronização de nomes, através de um *seeder* com mapeamento de idiomas, e aplicação de filtros sobre dataframes garantiu consistência e qualidade às informações exibidas ao usuário. Outro ponto relevante foi a conversão automática de preços de dólar para real, utilizando uma API de câmbio, o que aprimorou a experiência final para o público-alvo brasileiro.

Do ponto de vista acadêmico e técnico, este trabalho cumpriu seu papel ao explorar de maneira aprofundada os conceitos de *scraping*, automação de navegadores, engenharia de dados e construção de interfaces interativas. Foram aplicados conhecimentos de engenharia de software, análise de sistemas, usabilidade e desempenho, resultando em um produto funcional e aplicável a diferentes cenários reais.

6.1 Trabalhos futuros

Como proposta para trabalhos futuros, destaca-se a possibilidade de expansão do filtro dinâmico para formatos visuais assistidos, permitindo que o usuário selecione elementos diretamente a partir de uma visualização da página, via mouse ou cliques, reduzindo ainda mais a barreira de entrada técnica. Também seria interessante explorar técnicas de *machine learning* para reconhecimento automático de estruturas repetitivas em *sites*, auxiliando na automação da identificação de padrões. A integração com bancos de dados persistentes e o suporte a coletas periódicas programadas também configuram evoluções naturais do projeto.

Em síntese, o sistema desenvolvido não apenas alcançou os objetivos propostos, como também entregou uma ferramenta útil, inovadora e extensível, apta a servir como base para aplicações mais amplas de extração e análise de dados em ambientes *web* dinâmicos.

Embora o presente trabalho tenha atingido plenamente seus objetivos iniciais, ele também revelou uma série de oportunidades de evolução, expansão e aprofundamento técnico que podem ser exploradas em estudos e aplicações futuras. Os tópicos a seguir apresentam sugestões de trabalhos futuros, tanto em termos de aperfeiçoamento da ferramenta desenvolvida quanto de exploração de novas possibilidades acadêmicas e

tecnológicas a partir da base estabelecida neste projeto.

6.1.1 Integração com banco de dados relacional:

Atualmente, os dados coletados são processados em tempo real e apresentados em memória, sendo exportados ao usuário em arquivos CSV. Uma possível melhoria significativa seria a integração com bancos de dados relacionais, como PostgreSQL ou MySQL, permitindo a persistência histórica dos dados, comparações temporais, além de facilitar a criação de relatórios estatísticos e a construção de painéis mais sofisticados com visualização temporal de variações de preços, floats e tendências de mercado.

6.1.2 Coletas programadas e monitoramento automatizado:

Com a infraestrutura de coleta e exibição consolidada, outra proposta futura é a implementação de um sistema de coleta automatizada por agendamento, *cron jobs*. Isso permitiria acompanhar variações de mercado de *skins* ao longo do tempo, criar alertas personalizados, e identificar oportunidades de compra e venda com base em oscilações de preço. Esse recurso transformaria a aplicação em uma ferramenta ativa de monitoramento e tomada de decisão.

6.1.3 Filtro dinâmico com seleção visual de elementos:

Um dos grandes avanços do projeto foi o desenvolvimento do filtro dinâmico, permitindo que usuários configurem seus próprios *scrapers* a partir da entrada de tags HTML. Como evolução dessa funcionalidade, propõe-se a criação de uma interface gráfica interativa, onde o usuário possa selecionar os elementos desejados diretamente com o mouse em uma pré-visualização da página, semelhante ao recurso de inspeção de elementos presente em navegadores. Esse recurso reduziria ainda mais a barreira técnica para usuários não familiarizados com HTML, tornando a ferramenta acessível a um público mais amplo.

6.1.4 Reconhecimento automatizado de padrões de estrutura:

Outra linha de pesquisa interessante seria a aplicação de *machine learning* ou heurísticas de análise de DOM para reconhecimento automático de estruturas repetitivas em páginas *web*, como *grids* de produtos ou listas. Isso poderia permitir que o sistema detectasse automaticamente os elementos que representam preço, imagem ou nome dos itens, mesmo sem intervenção direta do usuário. Técnicas de *clustering* de atributos HTML ou análise de recorrência de blocos poderiam ser exploradas para esse fim.

6.1.5 Otimização da coleta por navegação semântica:

A coleta atual depende fortemente da rolagem ou navegação manual por páginas. Futuramente, seria interessante investigar o uso de navegação semântica baseada em APIs internas dos *sites*, caso disponíveis, ou no consumo direto de *endpoints* JavaScript. Essa abordagem poderia aumentar a performance, reduzir o tempo de carregamento e diminuir a dependência da renderização completa das páginas, sobretudo em *sites* com restrições visuais, como aqueles que bloqueiam acesso *headless*.

6.1.6 Sistema de comparação entre *sites* e recomendação de compras:

Com uma base consolidada de dados e histórico de preços, é viável expandir a aplicação para um comparador de preços entre plataformas, oferecendo ao usuário sugestões sobre onde encontrar a melhor oferta para determinada *skin*. Combinando isso com histórico de variações e filtros personalizados, o sistema poderia atuar também como uma ferramenta de recomendação de compra inteligente, com base em critérios como menor preço, melhor *float*, ou desgaste ideal.

6.1.7 Integração com autenticação e perfis de usuário:

Por fim, um desdobramento relevante seria a criação de um sistema com autenticação de usuários e salvamento de preferências de busca e histórico de interações, permitindo a personalização da experiência e o acompanhamento de itens favoritos. Com isso, cada usuário poderia configurar seus próprios parâmetros, criar alertas personalizados e até mesmo exportar coletas pré-filtradas para seus interesses.

A continuidade deste projeto, por meio dessas possíveis melhorias e expansões, representa uma oportunidade concreta de aplicar e aprofundar conhecimentos em áreas como engenharia de software, inteligência artificial, mineração de dados e experiência do usuário, além de contribuir com soluções práticas para um mercado ativo e em constante transformação como o das *skins* digitais.

Referências

ÁLVAREZ, M.; PAN, A.; RAPOSO, J.; BELLAS, F.; CACHEDA, F. Finding and extracting data records from web pages. In: SPRINGER. *International Conference on Embedded and Ubiquitous Computing*. 2008. Disponível em: <https://link.springer.com/article/10.1007/s11265-008-0270-y>. Acesso em: 28 abr 2025. Citado na página 17.

ANKITHA, B.; RAO, C.; CHANDAVARKAR, N.; BALASUBRAMANI, R. A survey on web data extraction techniques. *IJRET: International Journal of Research in Engineering and Technology*, 2016. Citado na página 18.

BATISTA, N. A.; BRANDÃO, M. A.; PINHEIRO, M. B.; DALIP, D. H.; MORO, M. M. Dados de múltiplas fontes da web: coleta, integração e pré-processamento. In: *Proceedings of the 24rd Brazillian Symposium on Multimedia and the Web*. [S.l.]: Universidade Federal de Minas Gerais (UFMG), 2018. p. 153–192. Citado 2 vezes nas páginas 22 e 23.

BERNARD, M.; BAKER, R.; FERNANDEZ, M. Paging vs. scrolling: Looking for the best way to present search results. *Usability News*, v. 4, n. 1, 2002. Citado na página 19.

BHATT, C.; BISHT, A.; CHAUHAN, R.; VISHVAKARMA, A.; KUMAR, M.; SHARMA, S. *Web Scraping Techniques and Its Applications: A Review*. IEEE, 2023. 1–8 p. Disponível em: <https://ieeexplore.ieee.org/abstract/document/10351298>. Acesso em: 22 abr 2025. Citado na página 15.

BRITO, A. B. d.; SANTOS, R. M. d.; MEDEIROS, S. N. d. Ferramentas de testes: Um estudo comparativo entre cypress, playwright e selenium webdriver. *Instituto Federal de Educação, Ciências e Tecnologia de Pernambuco. Campus Paulista. Curso de Análise e Desenvolvimento de Sistemas.*, Campus Paulista. Curso de Análise e Desenvolvimento de Sistemas., 2024. Disponível em: <https://repositorio.ifpe.edu.br/xmlui/handle/123456789/1675>. Acesso em: 21 abr 2025. Citado na página 16.

BROWN, M. A.; GRUEN, A.; MALDOFF, G.; MESSING, S.; SANDERSON, Z.; ZIMMER, M. *Web scraping for research: Legal. Ethical, Institutional, and Scientific Considerations*. 2024. Disponível em: <https://arxiv.org/abs/2410.23432>. Acesso em: 20 abr 2025. Citado 2 vezes nas páginas 16 e 22.

CASTILHO, F. A. de. *O marketing por trás das vendas de skin em jogos online*. 2023. Disponível em: <https://repositorio.unesp.br/entities/publication/a5f835e5-5f8c-4b44-ae8e-1c5a614cf338>. Acesso em: 29 mar 2025. Citado na página 12.

CIRILLO, G. R.; GALANTE, G. Proposta de comparaçao da eficiência e escalabilidade de bibliotecas python para manipulaçao e análise de dados. In: SBC. *Escola Regional de Alto Desempenho da Região Sul (ERAD-RS)*. 2025. p. 133–134. Disponível em: <https://sol.sbc.org.br/index.php/eradrs/article/view/35378>. Acesso em: 23 jun 2025. Citado na página 26.

CORPORATION, V. *Counter-Strike 2*. 2025. Disponível em: <https://store.steampowered.com/app/730/CounterStrike_2/>. Acesso em: 17 abr 2025. Citado 2 vezes nas páginas 13 e 25.

CS.MONEY. *CS.Money – Mercado de Skins do Counter-Strike*. 2025. Disponível em: <https://cs.money/pt/market/buy/>. Acesso em: 24 jun 2025. Citado 2 vezes nas páginas 33 e 38.

DAMASCENO, A. B. Taintjsec: um método de análise estática de marcação em código javascript para detecção de vazamento de dados sensíveis. *Universidade Federal do Amazonas*, Universidade Federal do Amazonas, 2017. Disponível em: <https://tede.ufam.edu.br/handle/tede/6267>. Acesso em: 14 jun 2025. Citado na página 20.

DUCKETT, J.; SCHLÜTER, J. *Html & Css*. [S.l.]: Wiley, 2011. Citado na página 25.

FARIAS, M. T.; ANGELUCI, A.; PASSARELLI, B. Web scraping e ciência de dados na pesquisa aplicada em comunicação: um estudo sobre avaliações online. *Revista observatório*, v. 7, n. 3, p. 1–22, 2021. Disponível em: <https://repositorio.usp.br/item/003093096>. Acesso em: 23 abr 2025. Citado na página 15.

FLANAGAN, D. *JavaScript: the definitive guide*. "O'Reilly Media, Inc.", 1997. Disponível em: <http://136.175.10.10:8090/ebook/pdf/JavaScript_The_Definitive_Guide.pdf>. Acesso em: 12 abr 2025. Citado na página 18.

GARCÍA, B.; MUNOZ-ORGANERO, M.; ALARIO-HOYOS, C.; KLOOS, C. D. Automated driver management for selenium webdriver. *Empirical Software Engineering*, Springer, v. 26, p. 1–51, 2021. Disponível em: <https://link.springer.com/article/10.1007/s10664-021-09975-3>. Acesso em: 23 abr 2025. Citado na página 16.

GOHIL, A.; SHROFF, A.; GARG, A.; KUMAR, S. A compendious research on big data file formats. In: IEEE. *2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS)*. 2022. p. 905–913. Disponível em: <https://ieeexplore.ieee.org/abstract/document/9788141>. Acesso em: 27 jun 2025. Citado na página 15.

GRACIANO, H. L. d. S.; RAMALHO, R. A. S. Scraperci: Um web scraper para coleta de dados científicos. *Encontros Bibli*, SciELO Brasil, v. 28, p. e92471, 2023. Disponível em: <https://www.scielo.br/j/eb/a/9QYwtw5kgByRpDFFQB778Tj/?lang=pt>. Acesso em: 19 jun 2025. Citado na página 22.

HOLJEVAC, M.; JAKOPEC, T. Web application dashboards as a tool for data visualization and enrichment. In: IEEE. *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*. 2020. p. 1740–1745. Disponível em: <https://ieeexplore.ieee.org/abstract/document/9245289>. Acesso em: 23 jun 2025. Citado na página 26.

JIN, Y.; XU, C.; LIN, T.; LI, W.; ZEGHLACHE, M. L. Python dash for well data validation, visualization, and processing. *Petrophysics-The SPWLA Journal of Formation Evaluation and Reservoir Description*, OnePetro, v. 64, n. 04, p. 568–573, 2023. Citado na página 27.

JÚNIOR, A. G. d. S.; ALMEIDA, L. S. F. d.; MARTINS, M. T. O impacto dos jogos eletrônicos na sociedade e economia. um estudo documental. *PESQUISA & EDUCAÇÃO A DISTÂNCIA*, ESQUISA & EDUCAÇÃO A DISTÂNCIA, v. 1, n. 27, 2023. Disponível em: <http://revista.universo.edu.br/index.php?journal=2013EAD1&page=article&op=viewArticle&path%5B%5D=11061>. Acesso em: 22 abr 2025. Citado na página 13.

JUNIOR, J. L. D. Implementando uma página dinâmica com um gerador de sites estáticos. *UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL INSTITUTO DE INFORMÁTICA CURSO DE CIÊNCIA DA COMPUTAÇÃO*, 2018. Disponível em: <https://lume.ufrgs.br/bitstream/handle/10183/175128/001065210.pdf>. Acesso em: 29 abr 2025. Citado na página 18.

KADAM, V. B.; PAKLE, G. K. A survey on html structure aware and tree based web data scraping technique. *International Journal of Computer Science and Information Technologies*, Citeseer, v. 5, n. 2, p. 1655–1658, 2014. Disponível em: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5d7255655b77afec02e555570bd1813f9943171c>. Acesso em: 21 abr 2025. Citado 2 vezes nas páginas 16 e 25.

KHDER, M. A. Web scraping or web crawling: State of art, techniques, approaches and application. *International Journal of Advances in Soft Computing & Its Applications*, Al-Zaytoonah University of Jordan (ZUJ), v. 13, n. 3, 2021. Disponível em: <https://www.i-csrs.org/Volumes/ijasca/2021.3.11.pdf>. Acesso em: 22 abr 2025. Citado na página 15.

LAWSON, R. *Web scraping with Python*. [S.l.]: Packt Publishing Ltd, 2015. Citado na página 28.

LIMA, N. G.; OLIVEIRA, L. R. de. Uma arquitetura para integração de sistemas com fontes heterogêneas e big data. *Bahia Análise & Dados*, v. 30, n. 2, p. 77–103, 2020. Disponível em: <https://ecd.ufba.br/PosterECD2019/poster-Nelci.pdf>. Acesso em: 19 jun 2025. Citado na página 22.

MCKINNEY, W. *Python para análise de dados: Tratamento de dados com Pandas, NumPy e IPython*. [S.l.]: Novatec Editora, 2018. Citado na página 27.

MITCHELL, R. *Web scraping with Python: Collecting more data from the modern web*. [S.l.]: "O'Reilly Media, Inc.", 2018. Citado na página 18.

MOHOD, V. D.; MEGHA, M. J. A survey on data extraction of web pages using tag tree structure. *Institute of Engineering Technology*, 2014. Citado na página 17.

MURANO, P.; SHARMA, S. A usability evaluation of web user interface scrolling types. *Oslo Metropolitan University*, University of Illinois at Chicago Library, 2020. Disponível em: <https://firstmonday.org/ojs/index.php/fm/article/view/10309>. Acesso em: 05 abr 2025. Citado na página 19.

NESHASTORE. *NeshaStore*. 2025. Disponível em: <https://neshastore.com/>. Acesso em: 18 abr 2025. Citado na página 37.

NOKERI, T. C. Dash bootstrap components. In: *Web App Development and Real-Time Web Analytics with Python: Develop and Integrate Machine Learning Algorithms into Web Apps*. [S.l.]: Springer, 2021. p. 87–97. Citado na página 27.

NUNES, M. C.; DORNELES, C. F. Naewi-non-rendering approach to extract web information. In: SBC. *Simpósio Brasileiro de Banco de Dados (SBBD)*. 2022. p. 161–167. Disponível em: <https://sol.sbc.org.br/index.php/sbbd_estendido/article/view/21859>. Acesso em: 28 abr 2025. Citado na página 17.

OLIVEIRA, L. de S.; SOUSA, J. P. C. de; RIBEIRO, J. V. A. Bypassing cloudflare's reverse proxy: a case study contornando o proxy reverso do cloudflare: um estudo de caso. *Brazilian Journal of Development*, v. 8, n. 4, p. 27250–27259, 2022. Disponível em: <https://core.ac.uk/reader/597568340>. Acesso em: 17 jun 2025. Citado na página 21.

OPREA, S. V.; BÂRA, A. Why is more efficient to combine beautifulsoup and selenium in scraping for data under energy crisis. *Ovidius University Annals, Economic Sciences Series*, Ovidius University of Constantza, Faculty of Economic Sciences, v. 22, n. 2, p. 146–152, 2022. Citado na página 26.

PARVEZ, M. S.; TASNEEM, K. S. A.; RAJENDRA, S. S.; BODKE, K. R. Analysis of different web data extraction techniques. In: IEEE. *2018 international conference on smart city and emerging technology (ICSCET)*. 2018. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8537333>. Acesso em: 28 abr 2025. Citado na página 18.

PASQUALI, T. E.; SILVA, V. R. da; RIBEIRO, F. S.; SANTANA, I. T. S. de; JANKOWITSCH, J.; COSTA, R. A. T.; SILVEIRA, F.; PINHEIRO, W. S. Criação de dashboards analíticos em python para tomada de decisão. *Caderno Pedagógico*, v. 21, n. 8, p. e6539–e6539, 2024. Disponível em: <https://ojs.studiespublicacoes.com.br/ojs/index.php/cadped/article/view/6539>. Acesso em: 19 jun 2025. Citado na página 22.

PEREIRA, L. E. *Uso de Metaheurística em uma Ferramenta de Cotação para Compras de Cartas de Magic: The Gathering*. Dissertação (Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação)) — Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais (Campus Formiga), Formiga - MG, 2019. Citado na página 23.

POGGI, N.; BERRAL, J. L.; MORENO, T.; GAVALDA, R.; TORRES, J. Automatic detection and banning of content stealing bots for e-commerce. In: *NIPS 2007 workshop on machine learning in adversarial environments for computer security*. [s.n.], 2007. v. 2. Disponível em: <https://www.berralgarcia.com/documents/poggi-nips07.pdf>. Acesso em: 14 jun 2025. Citado na página 20.

RAGHAVENDRA, S. *Python Testing with Selenium: Learn to Implement Different Testing Techniques Using the Selenium WebDriver*. Apress, 2020. Disponível em: <https://katalog.ub.uni-heidelberg.de/cgi-bin/titel.cgi?katkey=68662873>. Acesso em: 23 abr 2025. Citado na página 16.

ROLE, F.; VERDRET, P. Le document object model (dom). *Cahiers GUTenberg*, n. 33-34, p. 155–171, 1999. Citado na página 17.

SALUNKE, S. S. *Selenium webdriver in Python: Learn with examples*. CreateSpace Independent Publishing Platform, 2014. Disponível em: <https://dl.acm.org/doi/abs/10.5555/2655462>. Acesso em: 23 abr 2025. Citado na página 16.

ShadowPay. *ShadowPay – Dota 2 Items Filtered by Rune, Price and StatTrak*. 2025. Disponível em: <https://shadowpay.com/pt/dota-2-items?types=[%22Rune%22]&price_from=0&price_to=1567.5&is_stattrak&hold_days>. Acesso em: 15 jun 2025. Citado 3 vezes nas páginas 8, 51 e 52.

SILVA, R. R. da; YOKOYAMA, R. S. Avaliação do desempenho da utilização de threads em user level em linux. *Revista de Informática Teórica e Aplicada*, v. 18, n. 1, p. 112–132, 2011. Disponível em: <https://seer.ufrgs.br/rita/article/view/rita_v18_n1_p112>. Acesso em: 23 jun 2025. Citado na página 26.

SOUZA, E. M. d.; ALVES, W. A. L. Eficiência e precisão na extração de dados científicos: um estudo de caso com robôs automatizados. *Navus-Revista de Gestão e Tecnologia*, v. 16, p. 1–33, 2025. Disponível em: <https://navus.sc.senac.br/navus/article/view/2085>. Acesso em: 14 jun 2025. Citado na página 20.

SOUZA, J. P. d. *Protótipo de uma Plataforma de busca de Preços na Web*. Dissertação (Monografia de Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação)) — Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais (Campus Formiga), Formiga - MG, 2025. Citado na página 24.

TANAKA, T.; NIIBORI, H.; SHIYINGXUE, L.; NOMURA, S.; NAKAO, T.; TSUDA, K. Selenium based testing systems for analytical data generation of website user behavior. In: IEEE. *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 2020. p. 216–221. Disponível em: <https://ieeexplore.ieee.org/abstract/document/9155736>. Acesso em: 23 jun 2025. Citado na página 26.

TOMIĆ, N. Effects of micro transactions on video games industry. *Teaching Assistant, University of Kragujevac, Faculty of Economics, ntomic@kg.ac.rs*, Teaching Assistant, University of Kragujevac, Faculty of Economics, ntomic@kg.ac.rs, v. 14, n. 3, 2017. Disponível em: <https://pdfs.semanticscholar.org/099c/08473fe056170e5b0dd3cfd08bf4e567e29e.pdf>. Acesso em: 22 abr 2025. Citado na página 13.

TURK, K.; PASTRANA, S.; COLLIER, B. A tight scrape: methodological approaches to cybercrime research data collection in adversarial environments. In: IEEE. *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. 2020. p. 428–437. Disponível em: <https://ieeexplore.ieee.org/abstract/document/9229744>. Acesso em: 12 jun 2025. Citado na página 20.

VALVERDE, F. M. *O mercado do jogo Counter-Strike*. 2023. Disponível em: <https://www.maxwell.vrac.puc-rio.br/65448/65448.PDF>. Acesso em: 05 fev 2025. Citado na página 13.

VASTEL, A.; RUDAMETKIN, W.; ROUVOY, R.; BLANC, X. Fp-crawlers: studying the resilience of browser fingerprinting to block crawlers. In: *MADWeb'20-NDSS Workshop on Measurements, Attacks, and Defenses for the Web*. [s.n.], 2020. Disponível em: <https://inria.hal.science/hal-02441653/>. Acesso em: 14 jun 2025. Citado na página 21.

WOOD, L.; HORS, A. L.; APPARAO, V.; BYRNE, S.; CHAMPION, M.; ISAACS, S.; JACOBS, I.; NICOL, G.; ROBIE, J.; SUTOR, R. *et al.* Document object model (dom) level 1 specification. *W3C recommendation*, v. 1, p. 1–212, 1998. Citado na página 17.

ZILIOTTI, F. F. Otimização de aplicações web: Melhorando as métricas core web vitals com nextjs. *Universidade Federal de Uberlândia*, Universidade Federal de Uberlândia, 2024. Disponível em: <https://repositorio.ufu.br/handle/123456789/41807>. Acesso em: 05 abr 2025. Citado na página 19.