Journal of Artificial Intelligence and Engineering Applications

Website: https://ioinformatic.org/

15th October 2025. Vol. 5. No. 1; e-ISSN: 2808-4519

Implementation of a Data-Testid Attribute-Based Web Scraping Method for Accommodation Data Extraction from a Dynamic E-Commerce Website (Case Study: Traveloka)

Abdul Latif[1], Siti Khotimatul Wildah[2*], Sarifah Agustiani[3], Eka Herdit Juningsih[4]

[1,4]*Information Systems, Universitas Bina Sarana Informatika, Indonesia*
[2]*Computer Technology, Universitas Bina Sarana Informatika, Indonesia*
[3]*Informatics, Universitas Bina Sarana Informatika, Indonesia*
Abdul.bdl@bsi.ac.id[1], siti.ska@bsi.ac.id[2,*], sarifah.sgu@bsi.ac.id[3], eka.ekj@bsi.ac.id[4]

Abstract

With the growing dominance of Online Travel Agent (OTA) platforms in the digital tourism industry, the data presented therein has become a crucial asset for market analysis, competitive intelligence, and academic research. However, automatic data extraction (web scraping) from these modern platforms faces significant challenges due to the use of JavaScript frameworks that generate dynamic HTML structures with non-semantic and frequently changing CSS class names. This study proposes and validates a robust web scraping methodology to address these issues. Using Traveloka's website as a case study, this research leverages the data-testid attribute—a stable marker deliberately implemented by developers for automated testing purposes—as the primary selector for data extraction. The implementation was carried out using the Python programming language, with Selenium for browser automation and Beautiful Soup for HTML parsing. The results demonstrate that this method successfully and consistently extracts accommodation data, including hotel names, locations, rating scores, number of reviews, and pricing structures. The study concludes that utilizing the data-testid attribute offers a superior and more stable alternative to traditional scraping techniques that rely on CSS selectors or DOM structures, thereby providing an effective blueprint for data acquisition from contemporary dynamic web applications.

Keywords: *Accommodation Data Extraction; data-testid Attribute; Dynamic Website; Selenium; Web Scraping*

1. Introduction

Online Travel Agent (OTA) platforms such as Traveloka have transformed the landscape of the tourism and hospitality industry. Through digital marketing strategies, OTAs are able to enhance hotel visibility, simplify the booking process, and positively impact room occupancy rates [1]. Beyond serving merely as transactional intermediaries, these platforms have evolved into dynamic data repositories of immense richness. Information such as pricing, room availability, user sentiment through reviews, and facility attributes has become a vital source of insight for various disciplines, ranging from economic analysis and hotel marketing strategies to computer science. The ability to systematically and automatically acquire such data is key to unlocking deeper insights.

Information from OTAs is also considered highly valuable as it directly represents user purchasing behavior and is consistently available in a standardized format [2]. However, such data is dynamic and cannot be retrieved retroactively if not collected in real-time, thus requiring an appropriate data acquisition method to ensure that the information can be optimally utilized in analysis.

To obtain data from OTA platforms in a structured and consistent manner, one commonly used technical approach is web scraping. This method enables the automated extraction of information from web pages, thereby replacing manual procedures that are repetitive and time-consuming. Web scraping offers flexibility in collecting factual and numerical data, converting it into formats suited for analytical needs, and storing it locally. This technique has been widely applied across various domains, such as brand monitoring, sentiment analysis, and data augmentation [3].

Nevertheless, the implementation of this technique on modern websites faces increasingly complex technical challenges. This is due to the dynamic nature of web pages controlled by JavaScript, in which content is loaded and updated in real time, rendering it inaccessible through conventional scraping approaches based on static HTML parsing [4].

One of the main challenges in implementing web scraping on modern applications is the brittle nature of scripts that rely on static DOM structures, such as CSS class naming. Changes in the user interface (UI) or updates to front-end libraries can drastically alter the structure of page elements, resulting in failures in the data extraction process. This challenge has been identified as a common cause of breakage in web-based automation systems [5].

To address these challenges, this study proposes a methodological approach that leverages the use of the data-testid attribute. This attribute is explicitly embedded by developers into key HTML elements to support automated end-to-end (E2E) testing. Since functional testing requires stable and consistent markers, this attribute (e.g., data-testid="tvat-hotelName") serves as an ideal anchor point. The stability required by developers to ensure software quality through automation also provides a similar benefit for researchers seeking to perform reliable data extraction.

This study aims to design, implement, and evaluate a scraping method that utilizes data-testid as the primary selector. The contribution offered is twofold: from a technical perspective, it provides a robust, practical, and replicable framework for extracting data from dynamic web applications; and from an academic standpoint, it documents and systematically examines a practice that is well established within the developer community but remains underrepresented in scholarly literature, thereby helping to bridge the gap between industry practice and academic inquiry.

2. Literature Review

Web scraping is a widely used method for automatically extracting data from web pages [6]. This technique serves as an alternative solution for obtaining information that is not available through Application Programming Interfaces (APIs) [7]. Various studies have demonstrated the effectiveness of web scraping for different purposes. For instance, [6] applied it to extract data from e-commerce websites, [8] from online news portals, [9] from financial reports of the Indonesian stock exchange, [10] for tax potential mining, [11] from Indonesian marketplaces, [12] to analyze digital wallet users, [13] for social media data collection, [14] for collecting criminal news, and [15] for retrieving data from the SINTA Journal portal related to health analyst journals.

On the other hand, modern website HTML structures often use CSS class names that are obfuscated or automatically changed by frameworks such as React or Next.js, making element selection unstable. In related studies, the data-testid attribute is introduced as a potential solution. This attribute is generally inserted by developers to support automated testing and is considered more stable compared to typical CSS class names [16].

Another common issue encountered is the reliance on obfuscated and non-semantic CSS class attributes. This reliance complicates stable element selection because minor changes to the user interface or updates to frontend frameworks like React can lead to extraction failures. [17] identifies this issue as a key factor contributing to the brittleness of web-based testing systems and recommends using more stable attributes such as data-testid.

The data-testid attribute is a semantic marker commonly used in automated end-to-end testing. Due to its function requiring stability, this attribute tends to remain unchanged even when the user interface is modified. Although not many scientific studies have explicitly explored the use of data-testid in the context of scraping, this approach offers significant potential for improving the robustness and replicability of data extraction processes from modern web applications.

In another study targeting OTA (Online Travel Agent) platforms [18], most approaches have not yet adopted a selector system based on semantic attributes such as data-testid, making them still vulnerable to structural layout changes. Therefore, this study aims to address that gap by proposing the use of data-testid as the primary selector in the scraping process of accommodation data from dynamic OTA websites, specifically Traveloka. By adopting an approach aligned with modern testing practices, this research is expected to deliver a more reliable scraping method that is resilient to structural changes in websites.

3. Research Methods

The methodology in this study is systematically structured and replicable to ensure result consistency and enable application in similar follow-up studies. The methodological design involves three main components. First, the system architecture design outlines the technical flow, starting from initiating data requests to web pages, the acquisition and extraction of information, to the storage of data in a structured format. Second, an analysis of the target web page is conducted, which includes identifying relevant HTML elements based on the data-testid attribute and mapping the Document Object Model (DOM) structure to ensure the sustainability and accuracy of element selection. Third, data extraction and processing algorithms are applied, encompassing parsing, cleaning (normalization), and data validation processes to ensure the accuracy, completeness, and consistency of the extracted results.

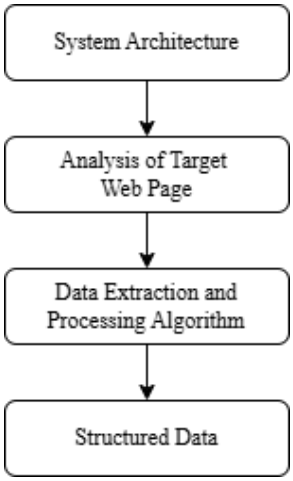

Fig. 1: Research Methods

3.1. Data Retrieval System Architecture

The system developed in this study employs an integrated technology stack selected based on its stability, community support, and compatibility with the requirements of data extraction and processing from dynamic web applications. The core components of this technology stack are described as follows:

1. Programming Language
   This study utilizes Python (version 3.9 and above) as the primary programming language. The selection is based on its mature library ecosystem in data analysis, ease of integration with various web automation tools, and concise and expressive syntax that enhances code development productivity [19].
2. Web Browser Automation
   To perform interactions and navigation on dynamic web pages, Selenium WebDriver is employed in combination with Google ChromeDriver. Selenium is an industry-standard for browser automation, supported by extensive documentation and a large, active user community [20][21][22].
3. HTML Parsing
   The extraction of HTML elements is carried out using the Beautiful Soup version 4 library. This library is chosen for its capability to handle malformed HTML documents and its Pythonic, intuitive interface, which facilitates the navigation and querying of elements within the DOM structure [23].
4. Data Structuring and Analysis
   Extracted data is processed and organized using the Pandas library, which is widely recognized as the de facto standard for tabular data manipulation and analysis in the Python environment. Pandas' DataFrame object enables efficient and structured data processing, supporting various data transformation and aggregation operations [24][25].

3.2. Object Analysis and Selector Identification

This stage represents a core component of the proposed methodology, in which a structural analysis of the HTML document is conducted on the hotel listing elements within the Traveloka search results page. The analysis focuses on understanding the structure of elements that represent each relevant data attribute, such as hotel name, price, and rating. As an illustration, the following HTML element is used to display the hotel name:

<h3 aria-level="3" dir="auto" role="heading" class="css-4rbku5 css-901oao r-1w9mtv9 r-fdjqy7" data-testid="tvat-hotelName">ARTOTEL Gelora Senayan Jakarta</h3>

In this structure, a distinction is evident between the characteristics of the two types of attributes used. The class attribute consists of a set of auto-obfuscated values (e.g., css-4rbku5, css-901oao, and so on), which are generally non-semantic and highly susceptible to changes—either due to user interface updates or automated build processes in modern frontend frameworks. In contrast, the data-testid attribute contains descriptive, semantic, and stable values, such as "tvat-hotelName", which are explicitly designed by developers to support automated end-to-end testing processes.

Based on this analysis, a systematic mapping process is carried out between each target data element and the most stable selector attribute. The highest priority is given to the data-testid attribute due to its stability. However, in cases where data-testid is not available, other stable selectors such as data-id, or as a last resort, specific CSS class selectors, are utilized. This mapping is then used as the foundation for the extraction logic design and is presented in detail in Table I as a blueprint for the data acquisition process.

Table 1: Mapping of Data Fields to HTML Structure and Selector Attributes

Data Field	HTML Tag	Selector Attribute	Selector Value	Raw Text Example
Hotel Name	h3	data-testid	tvat-hotelName	ARTOTEL Gelora Senayan Jakarta
Property Type	div	class	r-1vvnge1 div.r-1w9mtv9	Hotel
Star Rating	div	data-id	tvat-starRating	Inferred from the number of <svg> elements
Rating Score	div	data-testid	tvat-ratingScore	9.1 (778 reviews)
Location	div	data-testid	tvat-hotelLocation	“Senayan, Jakarta”
Original Price	div	data-testid	tvat-originalPrice	Rp 2,017,033
Discounted Price	div	data-testid	tvat-hotelPrice	Rp 1,512,775
Facilities	div	data-testid	(starts with) hotel-feature-badge	WiFi; Parking; Restaurant
Image URL	img	data-testid	list-view-card-main-image	(URL from the src attribute)

3.3. Data Extraction and Processing Algorithm

The data extraction process in this study is implemented through a series of structured and logical algorithmic steps. These stages are described as follows:

1. WebDriver Initialization
   The process begins with the configuration and launch of Selenium WebDriver, which is used to automatically open a Google Chrome browser window (Project 2024b).
2. Target Page Navigation
   The WebDriver is directed to the specific URL of the hotel search results page on the Traveloka website, which serves as the data extraction target. A static delay (time.sleep) is applied to allow sufficient time for the initial page load.
3. Handling Dynamic Content via Scrolling
   Traveloka loads hotel listings dynamically as users scroll down the page. To accommodate this behavior, a scrolling loop is implemented. Within the loop, the script repeatedly sends the Keys.END command to the browser to scroll to the bottom of the page. After each scroll, the script checks whether the page height (document.body.scrollHeight) has increased. If the page height does not increase after several attempts, the loop terminates, indicating that all content has been loaded.
4. Interactive CAPTCHA Handling
   During extended scraping sessions, the site may display CAPTCHA challenges to verify human presence. The script is designed to detect the presence of a CAPTCHA iframe. If detected, execution is paused and a message is displayed in the console, instructing the operator to manually solve the CAPTCHA. Once completed, the operator may press Enter to resume the scraping process. This mechanism significantly improves the script’s resilience for large-scale data collection.
5. Page Source Retrieval and Parsing
   After each scrolling iteration, the page’s HTML source (page_source) is retrieved from the WebDriver and converted into a parsable object using the Beautiful Soup library, which transforms the HTML document into a navigable object (Abodayeh et al., 2023).
6. Iteration and Data Deduplication
   The system extracts all elements with the attribute data-testid="tvat-searchListItem". To prevent duplicate entries due to re-rendered elements during scrolling, a set data structure is used to store previously processed hotel names. Before extracting data from a hotel element, the script checks whether the hotel name already exists in the set. If not, the data is processed; otherwise, it is skipped.
7. Item-Level Data Extraction
   For each detected hotel card, the script extracts specific data elements using the data-testid attribute mappings defined in Table I. For example, the hotel name is extracted using the command container.find('h3', {'data-testid': 'tvat-hotelName'}).text.
8. Implicit Exception Handling
   To account for missing elements (e.g., not all hotels display a discounted price), conditional checks (if element) are used during data extraction. If an element is not found, default values such as 'N/A' or 0 are recorded. This approach ensures the scraping process remains stable.
9. Data Cleaning and Transformation
   The raw data extracted is subsequently cleaned and reformatted to suit further analysis needs. This process involves the use of regular expressions and string manipulation operations. Example transformations include:
   a. Converting price text such as 'Rp 1.512.775' into the integer 1512775 by removing the currency symbol, periods, and spaces.
   b. Splitting rating text like '9.1 (778 reviews)' into two separate variables: the rating score (float) and the number of reviews (integer).The section should be formatted as left, bold, Times New Roman, and 12pt font size. For subsection (left, bold, Times New Roman, and 10pt), the initial letter of first word should be capitalized and also similarly for other sub-subsections (left, bold, Times New Roman, and 9pt).

3.4. Data Structuring and Storage

The cleaned extracted data is organized into a Python dictionary structure, where each dictionary represents a single hotel entity along with its associated attributes. All dictionary entries are aggregated into a list, forming a list of dictionaries data structure that represents a collection of observations obtained from the scraping process.

To facilitate further data analysis, this structure is converted into a DataFrame using the Pandas library, which is the standard tool for tabular data manipulation and analysis within the Python environment (Team 2025). The DataFrame provides a structured, consistent, and readily processable representation of the data for subsequent analytical stages.

As a final step, the DataFrame is saved in Comma-Separated Values (CSV) format using the df.to_csv() method. This format is chosen for its high compatibility with various statistical and data analysis tools, as well as for its convenience in documenting and distributing the extraction results.

4. Results and Discussion

4.1. Data Structuring and Storage

The data acquisition process was carried out using a web scraping approach applied to the hotel search results page on the Traveloka website, with the search region explicitly filtered to the DKI Jakarta area. This restriction was intended to produce a localized and uniform dataset, allowing for more focused spatial and pricing analyses of the capital city's hospitality landscape.

Scraping was performed using Selenium WebDriver with Google Chrome running in automated mode. The WebDriver was used to open web pages with dynamically rendered content (JavaScript-rendered) and wait until all elements were fully loaded. To avoid reliance on obfuscated and unstable class attributes, element selection was performed using the data-testid attribute as the primary selector.

The fully loaded HTML content was extracted via the page_source property and processed using the BeautifulSoup4 library for parsing and DOM element navigation. Extraction was performed on hotel entities with the following attributes:

1. Hotel name
2. Property type
3. Location (district/subdistrict in Jakarta)
4. Rating score
5. Number of reviews
6. Star category
7. Original price (before discount)
8. Discounted price (if available)

From the scraping process, a total of 1,809 valid and complete hotel records were collected. The data includes hotels located across various areas of Jakarta such as Mangga Besar, Kuningan, Cikini, Cipinang, Kemayoran, Thamrin, Tebet, Rawasari, and others, reflecting a geographically representative distribution across Central, South, East, West, and North Jakarta.

The data cleaning and preprocessing steps included:

1. Normalization of price values by removing "Rp" and thousand separators (“.”), then converting them into integer type
2. Conversion of rating scores to float type
3. Extraction of review counts from formats such as "5.000+" into integer values
4. Location name consolidation for consistency
5. Handling of missing values in discounted price and star category attributes

The cleaned data was stored in a list of dictionaries structure, converted into a Pandas DataFrame object, and finally exported in Comma-Separated Values (CSV) format with UTF-8 encoding. This dataset is ready for use in further analysis such as visualization, modeling, and recommendation systems. An example of the extracted results is presented in Subsection 4.2 as a representation of the final scraping output.

4.2. Presentation of Extracted Data

The implementation of the described methodology successfully executed the scraping process on the target page. Data from each hotel listing was extracted, cleaned, and structured effectively. Table II below presents a sample of the structured data that was produced and stored in tabular format.

Table 2: Sample of Structured Extracted Data

Hotel Name	Property Type	Stars	Location	Rating Score	Number of Reviews	Original Price (Rp)	Discounted Price (Rp)
ARTOTEL Gelora Senayan Jakarta	Hotel	5	Senayan	9.1	778	2,017,033	1,512,775
Hotel Indonesia Kempinski Jakarta	Hotel	5	Thamrin	8.9	5,000+	4,558,558	3,418,919
Merlynn Park Hotel	Hotel	5	Petojo Utara	8.6	17,000+	965,741	724,306
Millennium Hotel Sirih Jakarta	Hotel	4	Tanah Abang	8.7	6,000+	1,041,861	781,396
Urbanest Inn House Slipi	Homestay	3	Kemanggisan	8.4	1,700+	309,473	232,105
Oakwood Apartments PIK Jakarta	Apartment	5	Kamal Muara	8.2	987	2,042,042	1,531,532

This table demonstrates that the end-to-end process—from dynamic page navigation, element identification via data-testid, raw text extraction, cleaning, to structuring—performed as designed and successfully produced clean output suitable for further analysis.

4.3. Comparative Analysis and Method Validation

The primary validation of the proposed approach was conducted through comparison with conventional scraping methods. Traditional web scraping practices typically rely on CSS class-based selectors, such as soup.select_one('.css-4rbku5.css-901oao.r-1w9mtv9'). However, these class names are non-semantic and obfuscated, as they are automatically generated by CSS-in-JS frameworks commonly used in modern applications built with React or similar technologies.

Reliance on such classes renders scrapers highly brittle; minor changes to the UI structure, front-end refactoring, or style library updates can cause the entire scraper to fail, as the targeted elements may no longer be recognizable.

In contrast, this study employs the data-testid attribute as an element identifier (e.g., data-testid="tvat-hotelName"). This attribute is explicitly added by developers to support automated end-to-end (E2E) testing. Since functional testing requires a stable interface, data-testid attributes are typically stable, unaffected by visual modifications, and thus more reliable as the foundation for element selection in the scraping process.

Nonetheless, this study also acknowledges that not all desired data elements are marked with a data-testid attribute. For instance, to extract "Property Type," the implementation had to fall back on CSS class selectors due to the absence of a corresponding data-testid. While less ideal and potentially more fragile, this hybrid approach illustrates a practical reality in web scraping: prioritizing the most stable selectors while maintaining flexibility to use alternatives when necessary.

By aligning the scraping strategy with the testing-oriented design principles employed by site developers, this method leverages structural stability explicitly built into the system. This reinforces the claim that data-testid-based approaches offer functional advantages and greater technical resilience compared to conventional scraping methods.

5. Conclusion

This study successfully identified and addressed the main challenges in web scraping practices for modern web applications, particularly the instability of selectors caused by the use of dynamic and non-semantic CSS classes. By proposing and implementing a methodology based on the data-testid attribute, this research demonstrates that it is possible to build scrapers that are robust, reliable, and resilient to changes in the user interface. The case study on the Traveloka platform confirms the successful extraction of rich and structured accommodation data, thereby validating the effectiveness of this approach over conventional methods.

The primary contribution of this study lies in the systematic formulation and evaluation of a scraping technique that aligns with contemporary web application development practices. By providing a reproducible framework, this research offers practical value for the advancement of applied data science, particularly in the context of data acquisition from dynamic web applications. This methodology can be utilized by researchers, data analysts, and developers to efficiently and sustainably collect data from complex platforms.

6. Suggestion

Based on the results obtained, several directions for future research can be explored to broaden the scope and depth of this methodology's application:

1. Longitudinal analysis**:** Running the scraper periodically to observe price fluctuations, scarcity message dynamics, and seasonal demand patterns over an extended period.
2. Scalability and automation enhancement**:** Designing a scraping system capable of handling pagination or infinite scrolling automatically, and deploying it within a cloud computing environment to support large-scale data acquisition.
3. Advanced text-based analysis**:** Integrating Natural Language Processing (NLP) techniques to analyze user review texts—such as sentiment and dominant topics—to gain deeper insights beyond numerical rating scores.
4. Cross-platform comparative study**:** Applying the data-testid-based methodology to other OTA or e-commerce platforms to evaluate the generalizability and effectiveness of this approach across various technical and structural environments.

References

[1] J. M. Polgan, N. Khairunnisa, A. Hermawan, R. G. Guntara, U. P. Indonesia, and I. H. Bandung, "Strategi Pemasaran Untuk Meningkatkan Occupancy Kamar Hotel Melalui Online Travel Agent Di Indies Hotel Bandung," vol. 13, no. 2023, pp. 2417–2423, 2025.

[2] S. Suzuki, "Use of online travel agencies as a data source for tourism marketing," *J. Glob. Tour. Res.*, vol. 5, no. 2, pp. 167–171, 2020, doi: 10.37020/jgtr.5.2_167.

[3] P. Salvi and S. Pawar, "Issues and challenges of Web Scraping : Health Care Industry Case Study Approach ISSUES AND CHALLENGES OF WEB SCRAPING : HEALTHCARE INDUSTRY," vol. 11, no. 1, p. 7, 2023.

[4] K. Sharma and G. M. Borkar, "Comparative Analysis of Dynamic Web Scraping Strategies: Evaluating Techniques for Enhanced Data Acquisition," *Adv. Commun. Syst.*, pp. 241–252, 2024, doi: 10.56155/978-81-955020-7-3-22.

[5] S. Brisset, R. Rouvoy, L. Seinturier, R. Pawlak, and S. E. Jun, "Erratum : Leveraging Flexible Tree Matching to Repair Broken Locators in Web Automation Scripts," 2021.

[6] A. Z. Rizquina and C. I. Ratnasari, "Implementasi Web Scraping untuk Pengambilan Data Pada Website E-Commerce," *J. Teknol. Dan Sist. Inf. Bisnis*, vol. 5, no. 4, pp. 377–383, 2023, doi: 10.47233/jteksis.v5i4.913.

[7] F. F. Rahanto and I. Kharisudin, "Analisis Sentimen Data Ulasan Menggunakan Metode Naive Bayes Studi Kasus The Wujil Resort & Conventions Pada Situs Tripadvisor," *UNNES J. Math.*, vol. 10, no. 1, pp. 55–62, 2021.

[8] Y. A. Hafiz and E. Sudarmilah, "Implementasi Web Scraping Pada Portal Berita Online," *Inisiasi*, pp. 55–60, 2023, doi: 10.59344/inisiasi.v12i1.120.

[9] N. Javier, B. D. Satoto, Y. Dwi, and P. Negara, "IMPLEMENTASI TEKNIK WEB SCRAPING UNTUK PENGUMPULAN DATA LAPORAN KEUANGAN PERUSAHAAN DI BURSA EFEK INDONESIA (IDX)," vol. 9, no. 2, pp. 2789–2795, 2025.

[10] B. Pendidikan, D. Pelatihan Keuangan, K. Keuangan, M. Djufri, and P. Pajak, "JURNAL BPPK PENERAPAN TEKNIK WEB SCRAPING UNTUK PENGGALIAN POTENSI PAJAK (Studi Kasus pada Online Market Place Tokopedia, Shopee dan Bukalapak)," vol. 13, pp. 65–75, 2022.

[11] F. Sembiring and D. P. Sari, "Penerapan teknik scraping python pada website marketplace indonesia," *Integr. (Journal Inf. Technol. Vocat. Educ.*, vol. 2, no. 1, pp. 15–22, 2020, doi: 10.17509/integrated.v2i1.28243.

[12] E. Yuniar, D. S. Utsalinah, and D. Wahyuningsih, "Implementasi Scrapping Data Untuk Sentiment Analysis Pengguna Dompet Digital dengan Menggunakan Algoritma Machine Learning," *J. Janitra Inform. dan Sist. Inf.*, vol. 2, no. 1, pp. 35–42, 2022, doi: 10.25008/janitra.v2i1.145.

[13] L. Hidayati, L. P. Kusuma, D. Agustini, and V. Y. P. Ardhana, "Implementasi Web Scraping Untuk Pengumpulan Data Media Sosial Lingkup Pemerintah Provinsi Ntb," *J. Sist. Inf. dan Inform.*, vol. 7, no. 1, pp. 63–72, 2024, doi: 10.47080/simika.v7i1.3200.

[14] S. Satriajati, S. B. Panuntun, and S. Pramana, "Implementasi Web Scraping Dalam Pengumpulan Berita Kriminal Pada Masa Pandemi Covid-19," *Semin. Nas. Off. Stat.*, vol. 2020, no. 1, pp. 300–308, 2021, doi: 10.34123/semnasoffstat.v2020i1.578.

[15] A. Ulfah and I. Najiah, "Implementasi Web Scraping Pada Situs Jurnal Sinta Menggunakan Framework Selenium Webdriver Python," *JIKA (Jurnal Inform.*, vol. 7, no. 1, p. 29, 2023, doi: 10.31000/jika.v7i1.7037.

[16] A. S. Yondra, D. Triyanto, and S. Bahri, "Implementasi Web Scraping Untuk Mengumpulkan Informasi Produk Dari Situs E-Commerce," *J. Komput. Dan Apl.*, vol. 10, no. 01, pp. 93–102, 2022.
[17] X. Wang, L. Xiao, T. Yu, A. Woepse, and S. Wong, "An automatic refactoring framework for replacing test-production inheritance by mocking mechanism," *ESEC/FSE 2021 - Proc. 29th ACM Jt. Meet. Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, pp. 540–552, 2021, doi: 10.1145/3468264.3468590.
[18] A. Mulyana, I. ayu M. Er Meytha Gayatri, and W. Wagini, "Pemanfaatan Online Travel Agency (OTA) di Indonesia," *EKOMBIS Rev. J. Ilm. Ekon. dan Bisnis*, vol. 11, no. 2, pp. 1179–1194, 2023, doi: 10.37676/ekombis.v11i2.3776.
[19] P. S. Foundation, "Python 3.13.5 documentation." Accessed: Jun. 19, 2025. [Online]. Available: https://www.python.org/doc/
[20] BrowserStack, "Selenium Python Tutorial (with Example)." Accessed: Jun. 19, 2025. [Online]. Available: https://www.browserstack.com/guide/python-selenium-to-run-web-automation-test
[21] T. S. Project, "Selenium 4.22.0." Accessed: Jun. 19, 2025. [Online]. Available: https://pypi.org/project/selenium/
[22] T. S. Project, "Selenium with Python." Accessed: Jun. 19, 2025. [Online]. Available: https://selenium-python.readthedocs.io/
[23] P. S. Foundation, "Beautiful Soup." Accessed: Jun. 19, 2025. [Online]. Available: https://wiki.python.org/moin/BeautifulSoup
[24] T. pandas development Team, "pandas: powerful Python data analysis toolkit." Accessed: Jun. 19, 2025. [Online]. Available: https://pypi.org/project/pandas/
[25] T. pandas development Team, "pandas documentation." Accessed: Jun. 19, 2025. [Online]. Available: https://pandas.pydata.org/docs/