Pengembangan Aplikasi Web Scraping untuk Crawling Web Data dari Situs E-Commerce Properti

Jerry Livano Arsianto[#1], Tjatur Kandaga Gautama[*2]

[#]*Program Studi S1 Teknik Informatika, Universitas Kristen Maranatha*
Jl. Prof. Drg. Surya Sumantri No. 65, Bandung 40164, Indonesia

[1]2172002@maranatha.ac.id

[2]tjatur.kandaga@it.maranatha.edu

Abstract — **Technological developments drive the need for automated web data collection solutions for market analysis on real-estate e-Commerce sites, so this study will discuss the development of a web scraper application as a solution to overcome the challenges of manual data analysis from the original site. The web scraping feature in this application uses Selenium Webdriver and BeautifulSoup technology, and MongoDB as a flexible storage database. This system is also equipped with features for filtering, sorting, and exporting data to Excel for more efficient comparative analysis of real-estate data. This project was developed based on a website with REST API implementation to integrate the frontend with the backend. The frontend was developed using React and Tailwind, while the backend was developed using Flask. Based on the project that has been developed, the test results show that the application is able to dynamically scrape data from various domains at a fairly fast speed. In addition, functional testing also proves that the features in the application function according to specifications and are able to provide effective solutions for real-estate data analysis needs. With this research and development, it is hoped that application users can analyze real-estate data efficiently and be the first step for further development.**

Keywords— **BeautifulSoup, MongoDB, Real-Estate Data Analysis, Selenium Webdriver, Web Scraping**

I. PENDAHULUAN

Kemajuan teknologi dan internet telah mendorong pertumbuhan *e-Commerce*, termasuk *marketplace* properti seperti ‘rumah123’, ‘raywhite’, ‘belirumah.co’, dan situs sejenisnya. *Platform* ini menjadi preferensi utama bagi konsumen dalam mencari produk properti secara *online*, sehingga data yang tersedia di dalamnya sangat bermanfaat bagi pengembang properti dalam menganalisis tren pasar berdasarkan preferensi konsumen. Kebutuhan data properti berdasarkan preferensi dari situs aslinya merupakan salah satu tantangan bagi pengembang perumahan dalam melakukan analisis dan perbandingan produk [1]. Hal ini disebabkan karena metode pengumpulan data properti secara manual akan membutuhkan waktu cukup lama akibat kompleksitas data, perbedaan struktur informasi antar situs, serta sistem navigasi berbasis *pagination*.

Teknik *web scraping* dapat menjadi solusi dengan mengotomatisasi ekstraksi data dalam jumlah besar dan mengonversinya ke dalam format terstruktur, sehingga kebutuhan dalam pengumpulan data produk sebagai media untuk melakukan analisis pasar bagi pengembang perumahan dapat terpenuhi untuk meningkatkan efisiensi dalam pengambilan keputusan, terutama dari segi waktu dan ketepatan [2].

Penelitian ini bertujuan untuk merancang sistem berbasis *web scraping* yang mampu mengumpulkan data properti dari berbagai situs *marketplace e-Commerce* secara otomatis dan terstruktur. Perancangan ini juga didukung oleh permintaan dari perusahaan pengembang perumahan yang membutuhkan sistem untuk mengumpulkan data properti dari berbagai situs *e-Commerce*. Hasil yang diperoleh diharapkan dapat membantu pengembang properti dalam menganalisis tren pasar, menentukan spesifikasi produk sesuai preferensi konsumen, serta merancang strategi bisnis yang lebih efektif.

Manfaat dari penelitian ini mencakup peningkatan efisiensi dalam pengumpulan dan analisis data properti, optimalisasi pengambilan keputusan bisnis, serta pengembangan teknologi berbasis data bagi industri property berdasarkan preferensi dari *e-Commerce*. Dengan demikian, sistem yang diusulkan dapat menjadi langkah awal dalam transformasi digital bagi pengembang properti dalam menganalisis preferensi pasar secara lebih akurat dan cepat.

II. KAJIAN LITERATUR

A. Penelitian dan Pengembangan Sejenis

Beberapa penelitian dan pengembangan sejenis sudah pernah dilakukan untuk menangani permasalahan sejenis. Pada tahun 2023, Apriza Z. N. dan Chanifah I. R. juga telah melakukan analisis penggunaan *web scraper* menggunakan *library BeautifulSoup* dan *framework selenium* untuk melakukan penarikan data dari dua situs *e-Commerce*, yaitu 'shopee' dan 'tokopedia' [3]. Pada tahun 2021, Tomasz Jach melakukan penelitian untuk mengumpulkan data yang tersebar dari berbagai portal menggunakan teknik *web scraping* pada pasar properti di Polandia [4]. Pada tahun 2018, João Manuel Azevedo Santos melakukan pengembangan untuk melakukan *scraping* data perumahan di Australia dari beberapa *platform*, seperti 'Imovirtual', 'Idealista', 'Casas Sapo', 'OLX', dan lain-lain [5]. Namun, kekurangan dari beberapa pengembangan yang pernah dilakukan tersebut adalah pada fleksibilitas situs yang dapat diakses masih bersifat statis.

B. Teknologi Web Scraping

Web scraping atau *web extraction* adalah suatu teknik untuk melakukan ekstraksi data dari *World Wide Web* (WWW), kemudian data tersebut disimpan ke sistem lokal untuk dianalisis dan diolah nantinya. Umumnya, *scraping web* data dilakukan melalui *Hypertext Transfer Protocol* (HTTP) atau *web browser* [6]. Menurut Turland, *web scraping* adalah teknik untuk memperoleh informasi dari *website* secara otomatis tanpa harus menyalinnya ulang secara manual [7].

Tujuannya adalah untuk mengonversi *web* data menjadi terstruktur dan menyimpannya dalam penyimpanan yang terorganisir. Sedangkan manfaatnya adalah untuk mengumpulkan *web* data sesuai keinginan pengguna secara akurat, cepat, dan mudah digunakan untuk analisis dengan meminimalkan waktu yang digunakan untuk pencarian. Terdapat tiga tahapan utama dalam metode *web scraping*: *fetching*, ekstraksi data, dan transformasi data [8].

C. HTML Parsing dengan Library BeautifulSoup

HTML *parsing* merupakan metode untuk mengekstrak data dari *tag* atau atribut HTML tertentu. Metode ini umumnya digunakan untuk mengambil dan menganalisis konten yang dimiliki pada halaman *website*, khususnya pada proses pengambilan *web data*. Proses melakukan *parsing* dimulai dari generasi *query* tanam berupa Uniform Resource Locator (URL) dari *web* dan memperoleh label HTML. Selanjutnya, data dan *metadata* yang tersedia akan ditarik menggunakan teknik *web scraping* dan diurai menggunakan HTML *Document Object Model* (DOM) [2].

BeautifulSoup merupakan salah satu *library* dari *Python* yang berfokus untuk melakukan parsing konten HTML dengan kapasitas *memory* tingkat menengah dan penggunaan CPU yang rendah, sehingga menjadi pilihan yang efisien untuk melakukan *scraping* data dengan cepat dan ringan. Objek pada *BeautifulSoup* memiliki dua argumen, yaitu sumber dari *web page* dan *parser*-nya. Beberapa *parser* yang termasuk dalam *BeautifulSoup* adalah: html.parser, lxml, dan html5lib [9].

D. HTML Parsing dengan Library BeautifulSoup

Selenium adalah *testing framework* yang mendukung otomatisasi pengujian aplikasi *web* dalam berbagai bahasa pemrograman, seperti *Java*, *Python*, dan *C#*. Umumnya, *Selenium* digunakan untuk pengujian regresi fungsional dengan empat komponen utama: *Selenium IDE*, *Selenium Remote Control*, *Selenium WebDriver*, dan *Selenium Grid* [10]. *Framework* ini memungkinkan interaksi otomatis dengan aplikasi *web* layaknya pengguna, termasuk membuka URL dan menjalankan pengujian. *Selenium* bekerja dengan menginterpretasi operasi *Selenese commands* yang dienkode dalam tabel HTML, terdiri dari *command*, *target*, dan *value* [11].

1) Selenium Webdriver: Selenium Webdriver atau *Selenium* 2 menggabungkan *Selenium* dan *Webdriver* dalam satu proyek. Arsitekturnya menggunakan API untuk mengirimkan skrip lintas *browser* ke *driver* yang sesuai. Perintah dikirim dalam format JSON *Wire Protocol* atau W3C *Webdriver*, kemudian diterjemahkan dan dieksekusi oleh *browser* dengan hasil interaksi dikembalikan ke skrip *Selenium Webdriver*. Namun, *Selenium Webdriver* memiliki keterbatasan dalam pengujian karena tidak memiliki fungsionalitas bawaan untuk menggenerasi hasil pengujian, sehingga memerlukan aplikasi pihak ketiga [12].

E. Teknologi Pengembang Website

Dalam pengembangan *website*, dibutuhkan teknologi yang saling melengkapi untuk menciptakan satu kesatuan *website*, yaitu dari sisi *frontend*, *backend*, dan penyimpanan data.

1) Python: Python merupakan bahasa pemrograman dinamis dan *interpreter multi-programming* yang melibatkan *functional programming* dan Object-Oriented Programming (OOP). Python merupakan bahasa pemrograman yang *user-friendly* dan tidak terlalu sensitif terhadap *error*, sehingga *user* masih dapat melakukan *compiling* program hingga *error* yang bersangkutan ditemukan [13].

2) Flask: Flask adalah *micro-framework Python* yang ringan karena tidak memerlukan banyak *tools* atau *library* eksternal. *Framework* ini terdiri dari *static files* untuk kode status dan *template files* berbasis *Jinja* untuk halaman HTML, serta memerlukan *virtual environment* untuk mengelola dependensi. Dengan kode minimal, *Flask* tetap fleksibel untuk kustomisasi dan mendukung pengembangan *web* API secara efisien [14].

3) JavaScript: JavaScript adalah bahasa pemrograman *client-side* berorientasi objek yang menggunakan *prototype* untuk berbagi perilaku antar *instance*. *JavaScript* tidak memerlukan *compiler* dan dieksekusi melalui *interpreter* seperti *browser*, sehingga teknologi ini memungkinkan pengembangan aplikasi dengan antarmuka berbasis *website* [15].

4) React JS: React adalah *open-source JavaScript library* yang populer untuk pengembangan *frontend* karena kemampuannya untuk menggunakan kembali komponen dengan data berbeda untuk meningkatkan skalabilitas dan fleksibilitas. *React* berfungsi sebagai *View* dalam arsitektur MVC dengan mengabstraksi Document Object Model (DOM). Teknologi ini menggunakan *virtual* DOM untuk merender perubahan secara *efisien* dengan hanya memperbarui bagian yang berubah untuk meningkatkan kinerja [16].

5) Tailwind: Tailwind adalah CSS *framework* berbasis *utility-first* yang memungkinkan *styling* langsung melalui *class* pada elemen HTML. *Framework* ini fleksibel dalam konfigurasi dan meningkatkan kualitas elemen HTML. *Tailwind* mempercepat pengembangan *frontend* modern serta mendukung desain responsif dan *mobile-friendly* dengan sintaks yang efisien pada elemen yang digunakan [17].

6) MongoDB: MongoDB merupakan *open-source document database* dengan performa tinggi dan penskalaan otomatis. Penyimpanan data dilakukan dengan format *Binary* JSON di dalam sebuah *documents* atau *collections* hingga 16 MB per dokumen. Sebagai *database NoSQL*, *MongoDB* tidak bergantung pada tabel atau skema tetap, sehingga cukup fleksibel dalam menangani data yang tidak terstruktur [18].

F. Representational State Transfer Application Programming Interface

Application Programming Interface (API) adalah antarmuka yang memungkinkan aplikasi berkomunikasi dengan sistem lain tanpa mengubah struktur internalnya melalui sistem operasi yang dikendalikan *user* [19]. Dalam *web service*, API berperan memberi akses *user* melakukan operasi melalui arsitektur *Representational State Transfer* (REST) menggunakan protokol *Hypertext Transfer Protocol* (HTTP) untuk memuat sebuah *file* bertipe *JavaScript Object Notation* (JSON) [19].

REST merupakan pendekatan desain yang mendorong pengguna protocol dan fitur aplikasi dalam mengakses dan mengelola data secara representatif melalui permintaan sumber daya melalui *Uniform Resource Identifier* (URI). REST memiliki lima sifat dasar, yaitu *uniform interface*, *stateless*, *cacheable*, *client-server*, dan *layered system*. Dalam penggunaan REST, terdapat beberapa metode HTTP yang umum digunakan: GET, PUT, POST, dan DELETE [20].

III. ANALISIS DAN RANCANGAN SISTEM

A. Basis Data MongoDB

Basis data yang digunakan pada proyek menggunakan jenis *non-relational database*, yaitu *MongoDB* dengan penyimpanannya yang bertipe *collection*. Rancangan basis data yang digunakan pada proyek direpresentasikan pada Tabel I.

TABEL I
BASIS DATA COLLECTION MONGODB PADA PROYEK WEB SCRAPER

Nama Collection	Nama Field	Tipe Data
Site	guid	string
	account_guid	string
	site_name	string
	site_url	string
	space_rule	string
	limit_data	int
	is_active	boolean
	url_pattern	dict []
	created_date	datetime

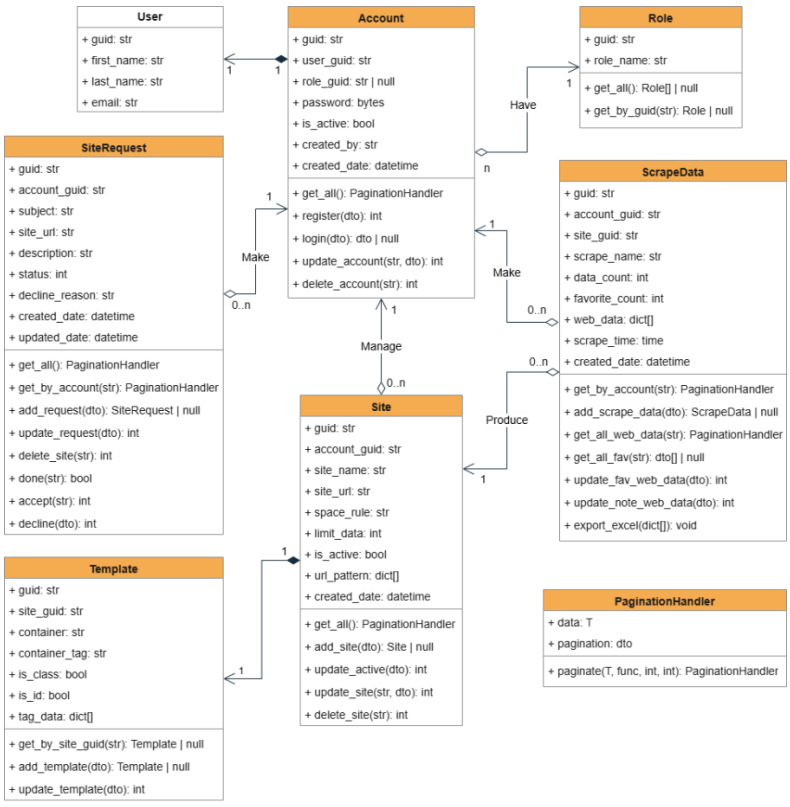
Nama Collection	Nama Field	Tipe Data
SiteRequest	guid	string
	account_guid	string
	subject	string
	site_url	string
	description	string
	status	int
	decline_reason	string
	created_date	datetime
	updated_date	datetime

Nama Collection	Nama Field	Tipe Data
ScrapeData	guid	string
	account_guid	string
	site_guid	string
	scrape_name	string
	data_count	int
	favorite_count	int
	web_data	dict []
	scrape_time	time
	created_date	datetime
User	guid	string
	first_name	string
	last_name	string
	email	string
Role	guid	string
	role_name	string

Nama Collection	Nama Field	Tipe Data
Account	guid	string
	user_guid	string
	role_guid	string
	password	string
	is_active	boolean
	created_by	string
	created_date	datetime
Template	guid	string
	site_guid	string
	container	string
	container_tag	string
	is_class	boolean
	is_id	boolean
	tag_data	dict []

B. Class Diagram

Gambar 1 merupakan susunan *class diagram* dari aplikasi, dimana terdapat 7 *class* yang berelasi dan 1 *class* tunggal. Tiap *class* memiliki atribut terkait dan 6 *class* memiliki *method* untuk melakukan aksi terhadap tiap *class*-nya.



Gambar 1. Class Diagram Proyek Web Scraper

C. Use-Case Diagram

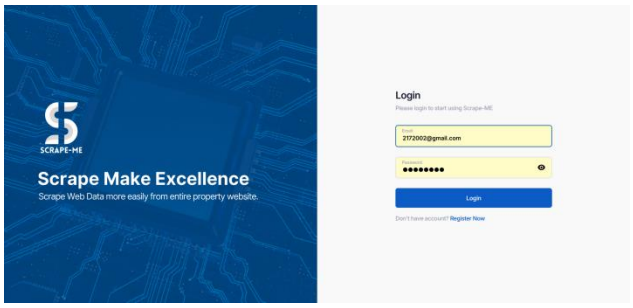
Gambar 2 merupakan representasi dari *use-case diagram* pada aplikasi. Pada diagram tersebut, terdapat dua jenis *role* atau *actor* yang dapat terlibat, yaitu *admin* dan *user* yang masing-masing memiliki otorisasi terhadap *use-case* yang terkait.

Gambar 2. Use-Case Diagram Proyek Web Scraper

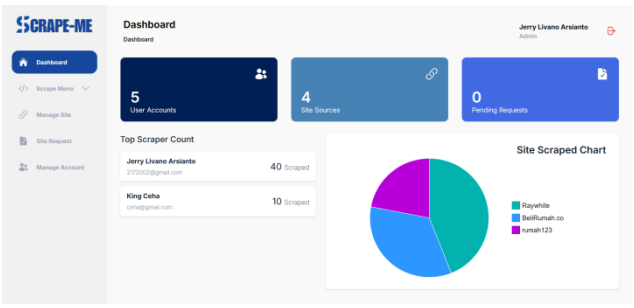
IV. IMPLEMENTASI

A. Tampilan Aplikasi

Tampilan aplikasi ini merupakan hasil keseluruhan aplikasi yang sudah dibuat dan diimplementasikan fitur-fitur yang dibutuhkan dari *backend* ke bagian *frontend*.

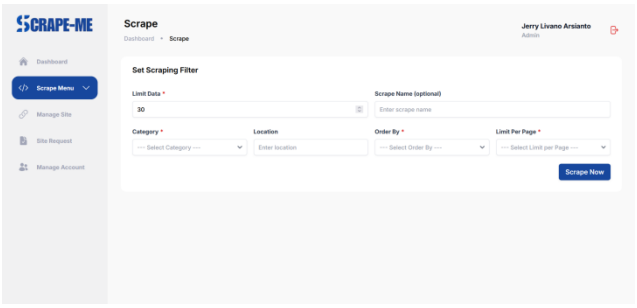


Gambar 3. Tampilan Halaman Login

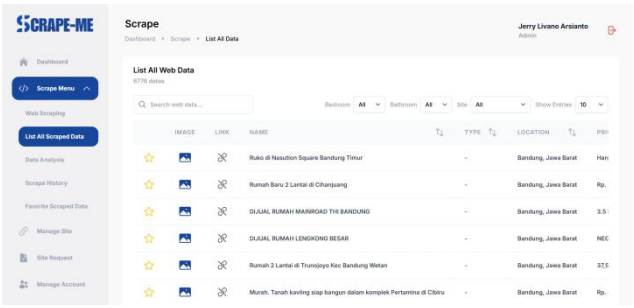


Gambar 4. Tampilan Halaman Dashboard

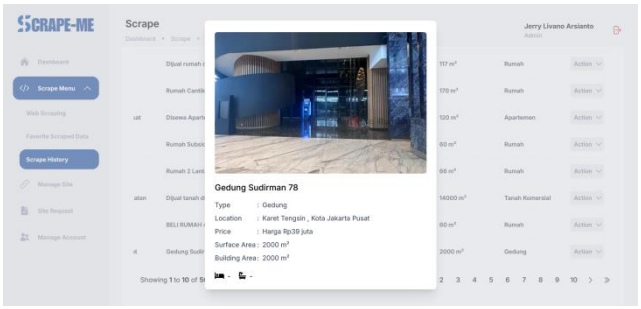
e-ISSN : 2443-2229



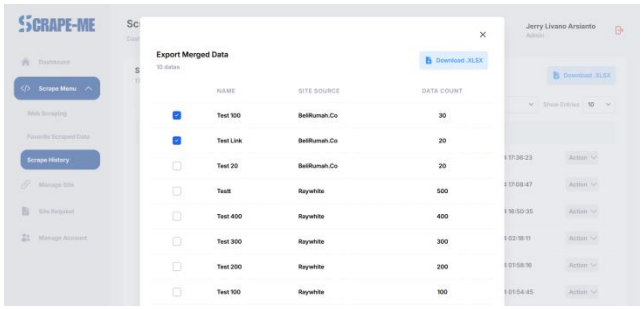
Gambar 5. Tampilan Halaman Input Filter Scraping Data



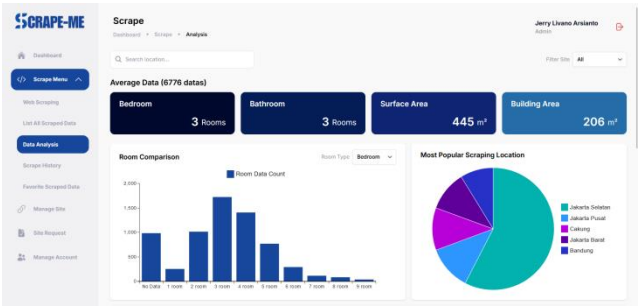
Gambar 6. Tampilan Halaman Web Data Hasil Scraping



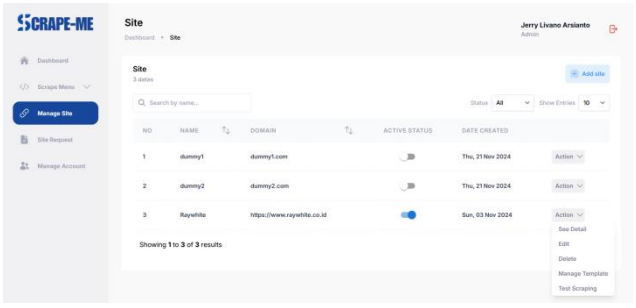
Gambar 7. Tampilan Halaman Detail Web Data



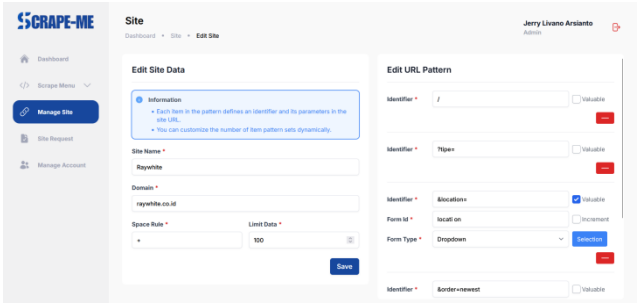
Gambar 8. Tampilan Filter Merge Export Excel



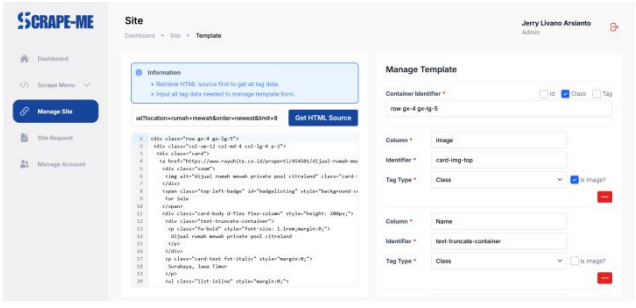
Gambar 9. Tampilan Halaman Data Analysis



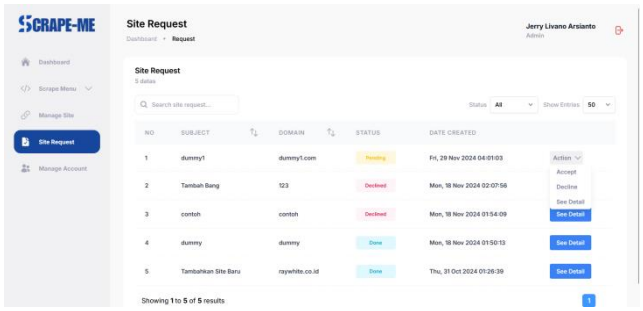
Gambar 10. Tampilan Halaman Manajemen Site



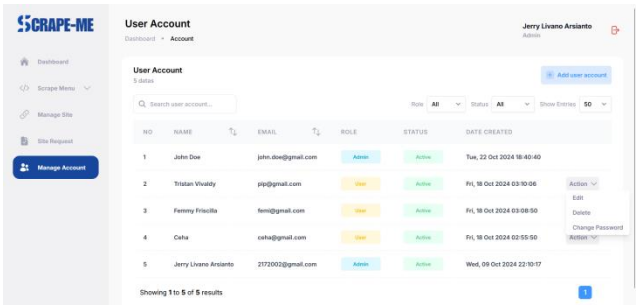
Gambar 11. Tampilan Halaman Manajemen URL Pattern



Gambar 12. Tampilan Halaman Manajemen Site Template



Gambar 13. Tampilan Halaman Manajemen Site Request



Gambar 14. Tampilan Halaman Manajemen Account User

Gambar 3 sampai Gambar 14 merupakan hasil tampilan dari setiap halaman aplikasi *web scraping* yang sudah dibuat.

B. Penjelasan dan Alur Kerja Sistem Web Scraping yang Digunakan

Terdapat beberapa tahapan dan alur kerja untuk melakukan *scraping web* data yang digunakan oleh sistem. Tahapan tersebut terbagi menjadi 5 segmen sebagai berikut:

1) Membuat Custom URL: Tahap ini akan mengonversikan data 'url_pattern' dari *collection* 'Site' untuk membentuk URL dinamis dari *website* yang dituju beserta *pagination*-nya untuk beberapa halaman *page* yang berbeda.

```
def create_site_url(request: CreateSiteUrlDto) -> str | None:
    try:
        result = (request.site_url + ("".join([
            f"{str(item['identifier'])}{f"{str(item.get('form_id')).replace(" ", request.space_rule)}" if item.get('form_id') else ""}"
            for item in request.url_pattern]))) if request.url_pattern else None
        return result
    except ValueError:
        return None
```

Gambar 15. Fungsi Membuat Custom URL Dinamis

Gambar 15 merupakan fungsi yang digunakan untuk mengkustomisasi URL dengan menggabungkan seluruh potongan URL yang ada pada *list* data. Di awal URL juga ditambahkan *domain* dari *website* yang dituju.

2) Akses Website dengan Selenium Webdriver: Setelah URL dinamis dibuat, *Selenium Webdriver* akan menjalankan fungsinya untuk mengakses URL tadi melalui jenis *browser* yang digunakan oleh *Webdriver* dan dijalankan otomatis. Sebelum dijalankan, *Selenium Webdriver* juga harus diinisialisasi terlebih dahulu dari *file* yang sudah diinstal pada sistem.

```
def _get_driver_path(self):
    project_root = os.path.dirname(os.path.abspath(__file__))
    driver_path = os.path.join(project_root, '../driver/chromedriver-win64/chromedriver.exe')
    return driver_path
```

Gambar 16. Fungsi Mengembalikan Path Selenium Webdriver

Pada Gambar 16 terdapat fungsi yang digunakan untuk menginisialisasi *Selenium Webdriver*, dimana fungsi tersebut akan mengembalikan *path directory* dari *file Selenium Webdriver* disimpan pada sistem.

```
def _initialize_driver(self):
    options = Options()
    options.add_argument('--headless')
    options.add_argument('--disable-gpu')
    options.add_argument('--no-sandbox')
    options.add_argument('--disable-popup-blocking')
    service = Service(self.driver_path, port=0)
    return webdriver.Chrome(service=service, options=options)
```

Gambar 17. Fungsi Menginisialisasi dan Mengonfigurasi Selenium Webdriver

Pada Gambar 17, terdapat *object* 'Options' yang digunakan sebagai konfigurasi yang digunakan menggunakan *method* 'add_argument()'. *Object* 'Service' digunakan untuk menjalankan *chrome webdriver* melalui *path file* tadi pada *port* 0 supaya sistem dapat memilih *port* secara acak. Setelah konfigurasi selesai, fungsi mengembalikan *webdriver* melalui *browser Chrome* dengan konfigurasi 'Service' dan 'Options' tadi.

3) HTML Parsing dengan BeautifulSoup: Setelah URL yang sudah dibuat diakses, selanjutnya *page source* dari halaman yang diakses tersebut diambil dan di-*parsing* untuk *web* data yang dibutuhkan menggunakan Teknik HTML *parsing* menggunakan *library BeautifulSoup*.

```
def get_html_source(self, url) → BeautifulSoup | None:
    driver = None
    try:
        self.kill_chromedriver()
        driver = self._initialize_driver()
        driver.get(url)
        content = driver.page_source
        soup = BeautifulSoup(content, 'html.parser')
        comments = soup.find_all(string=lambda text: isinstance(text, Comment))
        for comment in comments:
            comment.extract()
        return soup.body
    except WebDriverException as e:
        print(f"WebDriver error: {e}")
        return None
    finally:
        if driver:
            driver.quit()
        time.sleep(4)
```

Gambar 18. Fungsi Mengembalikan HTML Source dari Halaman URL

Gambar 18 merupakan fungsi yang digunakan untuk mengembalikan HTML *source* dari situs yang diakses berdasarkan *parameter* URL. Terdapat *filtering* untuk menghapus semua *comment*dan hanya mengambil *tag body* untuk mengurangi pengambilan *source* yang tidak diperlukan. Setelah pengambilan, sistem mengeluarkan *webdriver* menggunakan fungsi 'quit()' dan memberi jeda 4 detik untuk digunakan kembali.

```
temp_result = [[] for _ in template.tag_data]
while collected_data < limit_data:
    current_url_pattern = []
    for pattern in request.url_pattern:
        if pattern.get('is_page', False):
            current_url_pattern.append({
                "identifier": pattern['identifier'],
                "form_id": str(page),
                "is_page": True
            })
        else:
            current_url_pattern.append(pattern)
    url = self.create_site_url(CreateSiteUrlDto(
        request.site_url,
        current_url_pattern,
        request.space_rule
    ))
    print(url)
    soup = self.get_html_source(url)
    if not soup:
        print(f"Soup '{soup}' not found.")
        break
    container = []
    if template.is_class:
        container = soup.find_all(template.container_tag, class_=template.container)
    elif template.is_id:
        container = soup.find_all(template.container_tag, id=template.container)
    if not container:
        print(f"Container '{template.container}' not found.")
        break
    for item in container:
        for idx, tag in enumerate(template.tag_data):
            try:
                if tag['type'] == "":
                    elements = item.find(tag['tag'])
                else:
                    elements = item.find(tag['tag'], attrs={tag['type']: tag['identifier']})

                temp_result[idx].append(elements)

            except (AttributeError, KeyError, TypeError):
                print(f"Error processing tag: {tag}")
                continue
    collected_data = len(temp_result[0])
    page += 1
```

Gambar 19. Potongan Fungsi Mengambil Container Tiap Data Berdasarkan HTML Tag

Gambar 19 merupakan potongan dari fungsi melakukan *web scraping* dengan mengekstrak data dari halaman yang dibentuk berdasarkan URL *pattern* tertentu. Data yang dihasilkan elemen disimpan dalam *list* sementara. Proses *scraping* dilakukan secara iteratif hingga jumlah data mencapai batas dari *parameter* 'limit_data'. Pada setiap iterasi, URL dinamis

akan terus dihasilkan dengan nomor halaman yang diperbarui, mengambil sumber HTML, lalu mencari elemen dalam *container* yang sesuai berdasarkan *tag* dan atribut dalam' template.tag_data'. Jika ditemukan, data disimpan ke dalam 'temp_result' atau *list* sementara. Jika tidak, proses berhenti dengan menampilkan pesan kesalahan.

4) Transformasi Web Data: Setelah *list* dari *container* tiap data sudah disimpan, selanjutnya untuk setiap data pada satu *container* akan diekstrak data-data yang ingin di-*scraping* melalui pendefinisian atribut dari *tag* data yang bersangkutan. Setiap data yang diekstrak akan disimpan pada *database* untuk diproses lebih lanjut pada fitur lainnya.

```
for i in range(len(temp_result)):
    temp_result[i] = temp_result[i][:limit_data]

num_items = len(temp_result[0])
for i in range(num_items):
    item_data = {
        "index": i,
        "is_favourite": False,
        "note": ""
    }
    for idx, tag in enumerate(template.tag_data):
        element = temp_result[idx][i] if i < len(temp_result[idx]) else None
        if ',' in tag['title']:
            list_title = [title.strip() for title in tag["title"].split(",")]
            for title in list_title:
                field_key = title.lower().replace(" ", "_")
                item_data[field_key] = "-"
        else:
            field_key = tag['title'].lower().replace(" ", "_")
            item_data[field_key] = "-"
        if element:
            if tag.get("is_container", False):
                if tag["child_type"] == "":
                    list_element = element.find_all(tag['child_tag'])
                else:
                    list_element = element.find_all(tag['child_tag'], attrs={tag['child_type']: tag['child_identifier']})
                if tag["child_tag"].lower() == "img":
                    item_data[field_key] = [elem.get('src', "-") for elem in list_element]
                elif tag["child_tag"].lower() == "a":
                    hrefs = [elem.get('href', "-") for elem in list_element]
                    for href in hrefs:
                        if request.site_url[12:] not in href:
                            item_data[field_key] = f"{request.site_url}{href}"
                        else:
                            item_data[field_key] = href
                else:
                    list_title = [title.strip() for title in tag["title"].split(",")]
                    for j, elem in enumerate(list_element):
                        if j < len(list_title):
                            item_data[list_title[j].lower().replace(" ", "_")] = elem.get_text(strip=True, separator=" ").strip() or "-"
            else:
                if tag["tag"].lower() == "img":
                    item_data[field_key] = element.get('src', "-")
                elif tag["tag"].lower() == "a":
                    href = element.get('href', "-")
                    if request.site_url[12:] not in href:
                        item_data[field_key] = f"{request.site_url}{href}"
                    else:
                        item_data[field_key] = href
                else:
                    item_data[field_key] = element.get_text(strip=True, separator=" ").strip() or "-"
        else:
            item_data[field_key] = "-"
    scraped_data.append(item_data)
```

Gambar 20. Potongan Fungsi Mengambil Web Data Relevan pada Container

Gambar 20 merupakan potongan fungsi lanjutan dari potongan fungsi pada Gambar 19, dimana untuk setiap data *container* yang disimpan, akan diambil seluruh *web* data yang relevan di dalam sebuah *container* tersebut. Untuk mendefinisikan *web* data yang diambil, akan dilakukan pencocokan melalui jenis atribut atau nama atribut yang sudah disimpan pada data 'Template'. Untuk *web* data yang sudah diambil, akan ditransformasikan menjadi bentuk *list of object* yang disimpan dalam *database*. Proses ini juga dilakukan secara iterative untuk keseluruhan data *container* yang sudah diambil supaya keseluruhan *web* data dapat diambil dan disimpan pada sistem.

V. PENGUJIAN

A. Pengujian Black Box

Pengujian *black box* dilakukan untuk menguji keberhasilan tiap fitur dilihat dari sisi *input* dan *output* oleh *user*. Terdapat beberapa segemen pengujian untuk tiap fitur yang terpisah.

TABEL II
HASIL PENGUJIAN BLACK BOX SISTEM WEB SCRAPER

Fitur	Skenario Pengujian	Harapan Hasil	Hasil
Login	*Login* dengan kredensial yang terdaftar	Berhasil terautentikasi dan masuk ke halaman *dashboard*	Berhasil
	Login dengan *email* tidak valid	Pesan error: *- Email not valid*	Berhasil
	Login dengan *email* atau *password* tidak terdaftar	Pesan error: *- Incorrect email or password*	Berhasil
Register	*Register* dengan data lengkap dan valid	Data 'Account' berhasil ditambahkan dan kembali ke halaman *login*	Berhasil
	Register dengan *email* tidak valid	Pesan *error*: *- Email not valid*	Berhasil
	Register dengan *email* yang sudah terdaftar	Pesan *error*: *- Account already exist*	Berhasil
	Password dan *confirm password* tidak sama	Pesan *error*: *- Password not match*	Berhasil
Manajemen Data Site	Menampilkan seluruh data 'Site'	Seluruh data 'Site' berhasil ditampilkan	Berhasil
	Menambahkan satu data 'Site' dengan data lengkap dan valid	Data 'Site' baru berhasil ditambahkan	Berhasil
	Merubah satu data 'Site' dengan data lengkap dan valid	Data 'Site' terpilih berhasil diubah	Berhasil
	Memilih satu data 'Site' dan menghapusnya	Data 'Site' terpilih berhasil dihapus	Berhasil
Manajemen Data Site Template	Menampilkan satu Data 'Template' yang dipilih	Seluruh data 'Template' berhasil ditampilkan	Berhasil
	Menambahkan satu data 'Template' dengan data lengkap dan valid	Data 'Template' baru berhasil ditambahkan	Berhasil
	Merubah satu data 'Template' dengan data lengkap dan valid	Data 'Template' terpilih berhasil diubah	Berhasil
Manajemen Data Site Request	Menampilkan seluruh data 'Site Request'	Seluruh data 'Site Request' berhasil ditampilkan	Berhasil
	Menampilkan seluruh data 'Site Request' berdasarkan akun	Seluruh data 'Site Request' berhasil ditampilkan	Berhasil
	Menampilkan detail dari satu data 'Site Request'	Data detail dari 'Site Request' yang dipilih ditampilkan	Berhasil
	Menambahkan satu data 'Site Request' dengan data lengkap dan valid	Data 'Site Request' baru berhasil ditambahkan	Berhasil
	Merubah satu data 'Site Request' dengan data lengkap dan valid	Data 'Site Request' terpilih berhasil diubah	Berhasil
	Memilih satu data 'Site Request' dan menghapusnya	Data 'Site Request' terpilih berhasil dihapus	Berhasil
	Merubah status menjadi *accept* atau *done* pada 'Site Request' terpilih	Status 'Site Request' terpilih menjadi *accepted* atau *done*	Berhasil
Manajemen Data Site Request	Merubah status menjadi *decline* pada 'Site Request' terpilih dan memberikan alasan ditolak	Status 'Site Request' terpilih menjadi *declined*	Berhasil
Manajemen Data	Menampilkan seluruh data 'Account User'	Seluruh data 'Account User' berhasil ditampilkan	Berhasil
	Menambah satu data 'Account User' dengan data lengkap dan valid	Data 'Account User' berhasil ditambahkan	Berhasil

Account User	Merubah satu data 'Account User' dengan data lengkap dan valid	Data 'Account User' terpilih berhasil diubah	Berhasil
	Memilih satu data 'Account User' dan menghapusnya	Data 'Account User' terpilih berhasil dihapus	Berhasil
	Input data *email* tidak valid saat menambah atau merubah data 'Account User'	Pesan *error*: - *Email not valid*	Berhasil
	Form email diisi dengan *email* yang sudah terdaftar saat menambah atau merubah data 'Account User'	Pesan *error*: - *Account already exist*	Berhasil
	Password dan *confirm password* tidak sama saat menambah data 'Account User'	Pesan *error*: - *Password not match*	Berhasil
Manajemen Data Scrape	Menampilkan seluruh data *scrape history* berdasarkan akun	Seluruh data *scrape history* berhasil ditampilkan	Berhasil
	Menambahkan satu data *scrape history* menggunakan *web scraping* dengan data valid dan lengkap	Data *scrape history* berhasil ditambahkan dan menampilkan hasil data *scraping*	Berhasil
	Merubah nama stau data *scrape history*	Data *scrape history* berhasil diubah	Berhasil
	Submit data tidak valid sebelum melakukan *web scraping*	Pesan *error*: - *Please enter a number*	Berhasil
	Submit limit data melebihi *limit* sebelum melakukan *web scraping*	Pesan *error*: - *Value more than {limit}*	Berhasil
	Gagal mendapatkan halaman URL saat melakukan *scraping*	Pesan *error*: - *Failed to scrape data*	Berhasil
	Memilih satu data *scrape history* dan menghapusnya	Data *scrape history* terpilih berhasil dihapus	Berhasil
Manajemen Web Data Hasil Scraping	Menampilkan seluruh *web data* dari *scrape history* terpilih	Hasil data pada *excel* berhasil di *download*	Berhasil
	Menampilkan detail dari satu *web data*	Data detail dari *web data* terpilih berhasil ditampilkan	Berhasil
	Mengubah *note* pada satu *web data*	*Note* dari *web data* terpilih berhasil diubah	Berhasil
	Mengubah *status* favorit pada datu *web data*	*Status* favorit dari *web data* terpilih berhasil diubah	Berhasil
Export Excel	Melakukan *export excel* pada satu data 'Scrape'	Hasil data pada *excel* berhasil di *download*	Berhasil
	Melakukan penggabungan *export excel* pada beberapa data 'Scrape'	Hasil data pada *excel* berhasil di *download*	Berhasil

Tabel II merupakan hasil dari *black box testing* dari keseluruhan fitur yang ada pada sistem. Pada table tersebut terdapat 10 segemen fitur yang berbeda dengan hasil pengujian yang menunjukkan keberhasilan untuk 42 skenario pengujian.

B. Pengujian Performa Scraping Data

Pengujian performa dalam melakukan *scraping* data dilakukan untuk menguji fitur *scraping* dengan aspek yang diukur adalah: jumlah data yang diambil, waktu melakukan *scraping* keseluruhan, dan kesesuaian data yang diambil dengan aslinya, dengan *parameter* pembandingnya adalah jumlah permintaan data, jumlah data dalam satu *pagination*, dan kecepatan internet.

TABEL III
HASIL PENGUJIAN PERFORMA SCRAPING DATA

Domain	Permintaan Data	Data per Page	Hasil Data	Waktu	Kecepatan Internet	Kecocokan Data
raywhite.co.id	100	39	100	00:00:49	29.23 Mbps	100
	200	39	200	00:01:39	29.23 Mbps	200
	300	39	300	00:02:15	29.23 Mbps	300
	400	39	400	00:03:23	29.23 Mbps	400
	500	39	500	00:03:48	29.23 Mbps	500
rumah123.com	100	20	100	00:01:37	29.23 Mbps	100

	200	20	200	00:02:55	29.23 Mbps	200
	300	20	300	00:04:09	29.23 Mbps	300
	400	20	400	00:05:46	29.23 Mbps	400
	500	20	500	00:06:57	29.23 Mbps	500
belirumah.co	100	30	100	00:01:22	29.23 Mbps	100
	200	30	200	00:02:25	29.23 Mbps	200
	300	30	300	00:03:30	29.23 Mbps	300
	400	30	400	00:04:42	29.23 Mbps	400
	500	30	500	00:05:22	29.23 Mbps	500

Tabel III merupakan data hasil pengujian performa fitur *scraping data* dengan sampel pengujian menggunakan 3 situs yang berbeda. Untuk tiap situs dilakukan 5 kali percobaan dengan jumlah data yang berbeda dari rentang 100 sampai 500 data.

VI. KESIMPULAN

Berdasarkan hasil penelitian dan pengembangan, terdapat beberapa kesimpulan yang diambil. Untuk mengatasi tantangan dalam kebutuhan data produk dari situs *e-Commerce* sebagai bahan untuk analisis pasar, teknologi *web scraping* dapat digunakan sebagai solusi. Selain itu, teknologi ini berhasil diimplementasikan dalam sebuah aplikasi menggunakan *library Selenium Webdriver* dan *BeautifulSoup*, serta berhasil diuji secara fungsional dan performa untuk dapat melakukan *scraping web* data secara dinamis. Teknik dan alur kerja yang digunakan pada sistem *web scraping* juga berhasil dianalisis dengan metode akses ke URL dengan *webdriver*, melakukan HTML *parsing* dengan *BeautifulSoup*, dan data dikembalikan dengan bentuk yang telah ditransformasikan.Pengembangan ini juga masih memiliki kelemahan dari fitur yang masih terbatas dan fleksibilitas dalam merancang *template* untuk beberapa situs tertentu. Diharapkan pada penelitian dan pengembangan sejenis di masa mendatang dapat memberikan Solusi terhadap kekurangan dari penelitian ini.

DAFTAR PUSTAKA

[1] I. Onyenwe, E. Onyedinma, C. Nwafor, and O. Agbata, "Developing Products Update-Alert System for e-Commerce Websites Users Using HTML Data and Web Scraping Technique," *International Journal on Natural Language Computing*, 2021, doi: https://doi.org/10.48550/arXiv.2109.00656.

[2] S. Mehak, R. Zafar, S. Aslam, and S. M. Bhatti, "Exploiting Filtering approach with Web Scrapping for Smart Online Shopping : Penny Wise: A wise Tool for Online Shopping," in *2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, IEEE, Jan. 2019, pp. 1–5. doi: 10.1109/ICOMET.2019.8673399.

[3] A. Z. Rizquina and C. I. Ratnasari, "Implementasi Web Scraping untuk Pengambilan Data Pada Website E-Commerce," *Jurnal Teknologi Dan Sistem Informasi Bisnis*, vol. 5, no. 4, pp. 377–383, Oct. 2023, doi: 10.47233/jteksis.v5i4.913.

[4] T. Jach, "Web Scraping Methods Used in Predicting Real Estate Prices," in *Advances in Computational Collective Intelligence* , K. Wojtkiewicz, J. Treur, E. Pimenidis, and M. Maleszka, Eds., Cham: Springer International Publishing, 2021, pp. 375–387. doi: 10.1007/978-3-030-88113-9_30.

[5] J. M. A. Santos and P. G. F. A. Nogueira, "Real Estate Market Data Scraping and Analysis for Financial Investments," 2018. Accessed: Dec. 16, 2024. [Online]. Available: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://repositorio-aberto.up.pt/bitstream/10216/116510/2/296684.pdf

[6] B. Zhao, "Web Scraping," in *Encyclopedia of Big Data*, Cham: Springer International Publishing, 2022, pp. 951–953. doi: 10.1007/978-3-319-32010-6_483.

[7] L. Tiara, H. Syaputra, W. Cholil, and A. H. Mirza, "Web Scraper Dan Graphql API Untuk Data Perguruan Tinggi Di Indonesia Berdasarkan Website Kementerian Ristekdikti (Studi Kasus: Website Kementerian Ristekdikti)," *Jurnal Nasional Ilmu Komputer*, vol. 2, no. 3, pp. 193–212, Nov. 2021, doi: 10.47747/jurnalnik.v2i3.533.

[8] M. Khder, "Web Scraping or Web Crawling: State of Art, Techniques, Approaches and Application," *International Journal of Advances in Soft Computing and its Applications*, vol. 13, no. 3, pp. 145–168, Dec. 2021, doi: 10.15849/IJASCA.211128.11.

[9] A. Abodayeh, R. Hejazi, W. Najjar, L. Shihadeh, and R. Latif, "Web Scraping for Data Analytics: A BeautifulSoup Implementation," in *2023 Sixth International Conference of Women in Data Science at Prince Sultan University (WiDS PSU)*, IEEE, Mar. 2023, pp. 65–69. doi: 10.1109/WiDS-PSU57071.2023.00025.

[10] R. Patil and R. Temkar, "Intelligent Testing Tool: Selenium Web Driver," *International Research Journal of Engineering and Technology*, 2017, [Online]. Available: www.irjet.net

[11] B. García, M. Gallego, F. Gortázar, and M. Munoz-Organero, "A Survey of the Selenium Ecosystem," *Electronics (Basel)*, vol. 9, no. 7, p. 1067, Jun. 2020, doi: 10.3390/electronics9071067.

[12] B. García, M. Munoz-Organero, C. Alario-Hoyos, and C. D. Kloos, "Automated driver management for Selenium WebDriver," *Empir Softw Eng*, vol. 26, no. 5, p. 107, Sep. 2021, doi: 10.1007/s10664-021-09975-3.

[13] A. Z. Ablahd and S. A. Dawwod, "Using Flask for SQLIA Detection and Protection," *Tikrit Journal of Engineering Sciences*, vol. 27, no. 2, pp. 1–14, May 2020, doi: 10.25130/tjes.27.2.01.

[14] D. F. Ningtyas and N. Setiyawati, "Implementasi Flask Framework pada Pembangunan Aplikasi Purchasing Approval Request," *Jurnal Janitra Informatika dan Sistem Informasi*, vol. 1, no. 1, pp. 19–34, Apr. 2021, doi: 10.25008/janitra.v1i1.120.

[15] A. Yani, B. Saputra, and R. T. Jurnal, "Rancang Bangun Sistem Informasi Evaluasi Siswa Dan Kehadiran Guru Berbasis Web," *PETIR*, vol. 11, no. 2, pp. 107–124, Sep. 2018, doi: 10.33322/petir.v11i2.344.

[16] P. S. Maratkar and P. Adkar, "React JS-An Emerging Frontend Javascript Library," *IRE Journals*, vol. 4, no. 12, pp. 99–102, 2021, [Online]. Available: https://nodejs.org/en/download/

[17] F. Rifandi, Tri Viqi Adriansyah, and Rina Kurniawati, "Website Gallery Development Using Tailwind CSS Framework," *Jurnal E-Komtek (Elektro-Komputer-Teknik)*, vol. 6, no. 2, pp. 205–214, Dec. 2022, doi: 10.37339/e-komtek.v6i2.937.
[18] A. Boicea, F. Radulescu, and L. I. Agapin, "MongoDB vs Oracle -- Database Comparison," in *2012 Third International Conference on Emerging Intelligent Data and Web Technologies*, IEEE, Sep. 2012, pp. 330–335. doi: 10.1109/EIDWT.2012.32.
[19] M. F. A. Muri, H. S. Utomo, and R. Sayyidati, "Search Engine Get Application Programming Interface," *Jurnal Sains dan Informatika*, vol. 5, no. 2, pp. 88–97, Dec. 2019, doi: 10.34128/jsi.v5i2.175.
[20] B. Adi Pranata, A. Hijriani, and A. Junaidi, "Perancangan Application Programming Interface (Api) Berbasis Web Menggunakan Gaya Arsitektur Representational State Transfer (Rest) Untuk Pengembangan Sistem Informasi Administrasi Pasien Klinik Perawatan Kulit," *Jurnal Komputasi*, vol. 6, no. 1, pp. 33–42, Apr. 2018, doi: 10.23960/komputasi.v6i1.1554.