Agregasi Pencarian Situs Manga dengan Asyncio

Kelompok Manga Asing

1.	Robert Antonius	2125250057
2.	Christian Richie Wijaya	2125250007
3.	Daniel	2125250028
4.	Andreas Saputra	2125250037
5.	Achmad Rizky Zulkarnain	2125250045
6.	Rivaldo	2125250065

UNIVERSITAS MULTI DATA PALEMBANG
FAKULTAS ILMU KOMPUTER DAN REKAYASA
PROGRAM STUDI INFORMATIKA
TAHUN 2024

Bab 1. Pendahuluan

A. Latar Belakang

Manga, berasal dari Jepang, adalah salah satu media komik yang populer di seluruh belahan dunia dalam bentuk gambaran pada halaman terserialisasi untuk menyampaikan cerita yang panjang (Su et al., 2021). Walaupun *manga* merupakan bentuk komik yang telah mendunia, kebanyakan *manga* yang ada hanya tersedia untuk pasar domestik Jepang karena mahalnya biaya untuk mentranslasi *manga* (Hinami et al., 2021). *Manga* namun nyatanya tetap dapat mendunia dan menyebar ke negara lain seperti Indonesia walau tidak banyak lisensi dan translasi sebuah *manga* untuk Indonesia, dan ini terjadi karena sebuah praktek dalam komunitas pembaca *manga* yang disebut *scanlation*.

Scanlation adalah gabungan dari kata *scan* dan *translation*, dan merujuk kepada aktivitas memindai, menerjemah, dan mendistribusi komik (umumnya *manga*) yang dilakukan oleh kelompok penggemar lintas negara yang berkolaborasi lewat jaringan internet (Valero-Porras & Cassany, 2015). *Scanlation* umumnya dilakukan oleh penggemar dari sebuah cerita grafik tertentu yang bekerja dalam sebuah kelompok bernama *scanlation group* (Fabretti, 2016). Setiap *scanlation group* umumnya memiliki sebuah situs web sendiri, atau menerbitkan hasil terjemahan mereka ke situs komunitas seperti MangaDex.

Scanlation bersifat sukarela, sehingga tidak jarang sebuah *scanlation group* berpisah, berubah nama, dan terbentuk kembali menyebabkan banyaknya *scanlation group* berbeda-beda (Fabretti, 2016). Dengan banyaknya *scanlation group* ini, banyak pula situs-situs terjemahan berbeda yang ada di internet tanpa metode yang dapat menyambungkan situs tersebut. Scanlation bersifat tidak resmi, sehingga walau ada usaha untuk mensentralisasi pencarian hasil terjemahan, sentralisasi ini umumnya tidak mencakup lengkap seluruh komunitas scanlation, seperti yang dilakukan oleh MangaDex yang bersifat sukarela, atau akhirnya ditutup karena mencakup wilayah yang terlalu besar, seperti yang dilakukan oleh Tachiyomi yang telah ditutup akibat surat *cease and desist* dari sebuah penerbit cerita grafik.

Untuk mencari hasil terjemahan sebuah *manga*, pembaca umumnya perlu mengunjungi satu persatu situs *scanlation group* dan mencari apakah *manga* tersebut ada pada katalog mereka, yang dapat memakan waktu dan tenaga signifikan karena suatu aksi berulang. Aplikasi ini dibuat dengan tujuan untuk memitigasi ketidaknyamanan ini dengan mengagregasi hasil pencarian dari banyak situs tersebut secara sekaligus. Komputasi paralel melalui metode

asynchronous digunakan untuk mempercepat proses agregasi tersebut dengan melakukan *scraping* situs secara paralel, sehingga aplikasi yang dibuat dinamakan “Manga Asing” karena mencari komik asing (*manga*) secara *async* (asing).

B. Rumusan Masalah

Masalah yang diatasi adalah memudahkan pengguna dalam proses pencarian hasil penerjemahan *manga* oleh berbagai *scanlation group.* Ada beberapa tujuan yang perlu dicapai untuk menyelesaikan masalah ini, antara lain:

1. Menarik hasil pencarian terjemahan dari beberapa situs *scanlation*.
2. Mencegah waktu pencarian bertambah karena mencari dari banyak situs.
3. Membuat solusi yang dapat digunakan oleh pengguna tanpa pengetahuan komputer mendalam.

C. Solusi

Solusi yang diusulkan adalah dengan mengagregasi hasil pencarian dalam situs-situs menjadi satu daftar interaktif yang dapat membuka laman situs terjemahannya pada *browser* pengguna. Ada beberapa tujuan yang dicapai untuk menyelesaikan masalah ini, antara lain:

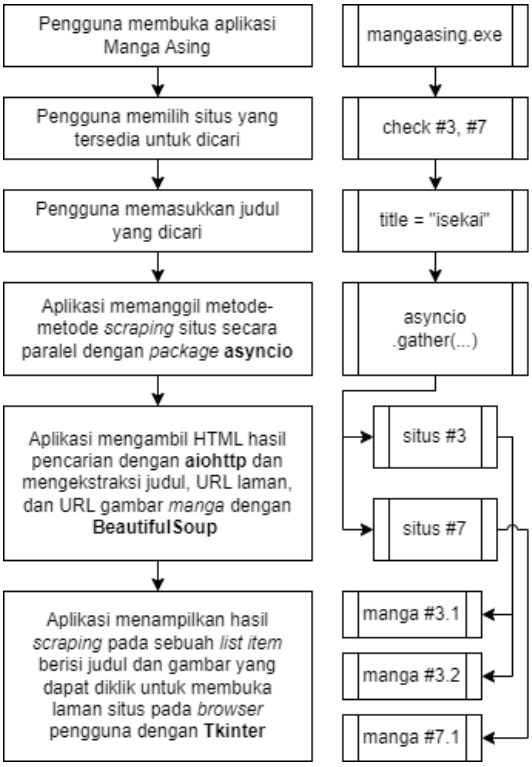
1. Melakukan *scraping* untuk mengagregasi hasil pencarian dari beberapa situs *scanlation*.
2. Melalukan *scraping* secara paralel supaya waktu pencarian tidak berkembang pesat semakin banyak ditambahkan situs *scanlation*.
3. Membuat tampilan aplikasi untuk perantara naskah *scraping* yang dapat digunakan pengguna untuk menavigasi ke laman situs *scanlation* pada *browser* pengguna.

D. Ruang Lingkup

Permasalahan yang ingin diselesaikan mencakup melakukan *scraping* pada 8 situs manga, antara lain MangaDex, MangaPill, AsuraToons, FlameComics, MangaPark, MangaFire, MangaReader, dan MangaFreak. *Scraping* dilakukan secara paralel dengan *package* asyncio, lalu ditampilkan pada UI yang dibuat dengan *package* Tkinter.

Bab 2. Perancangan dan Cara Kerja

Gambar 1 menunjukkan proses jalan kerja aplikasi Manga Asing yang telah dirancang dimulai dari saat pengguna membuka aplikasi sampai pengguna mendapat hasil pencarian *manga* yang telah di-*scrape*.



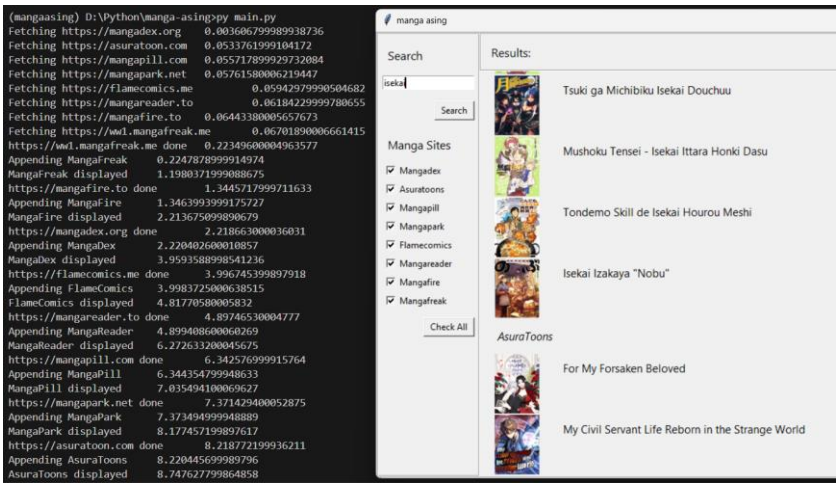
Gambar 1. Flowchart Proses Kerja Aplikasi Manga Asing

Aplikasi pertama meminta pengguna memilih situs apa saja yang ingin digunakan sebagai target *scraping* dari daftar situs-situs yang tersedia untuk *scraping* pada aplikasi. Pengguna lalu memasukkan judul *manga* yang ingin dicari lalu mengklik tombol cari. Aplikasi lalu memulai pencarian dengan membagi tugas menjadi *coroutine* yang meng-scrape hasil pencarian dari masing-masing situs dengan *package* asyncio. Setelah *coroutine* mendapat HTML hasil pencariannya melalui *package* aiohttp, *coroutine* mengekstraksi judul *manga*, URL *manga*, dan URL gambar halaman depan *manga* dari 5 hasil pencarian teratas dari masing-masing situs dengan *package* BeautifulSoup. Coroutine lalu menampilkan hasil pencarian beserta situs asalnya pada aplikasi tampilan yang dibangun dengan *package* Tkinter.

Bab 3. Hasil dan Analisis

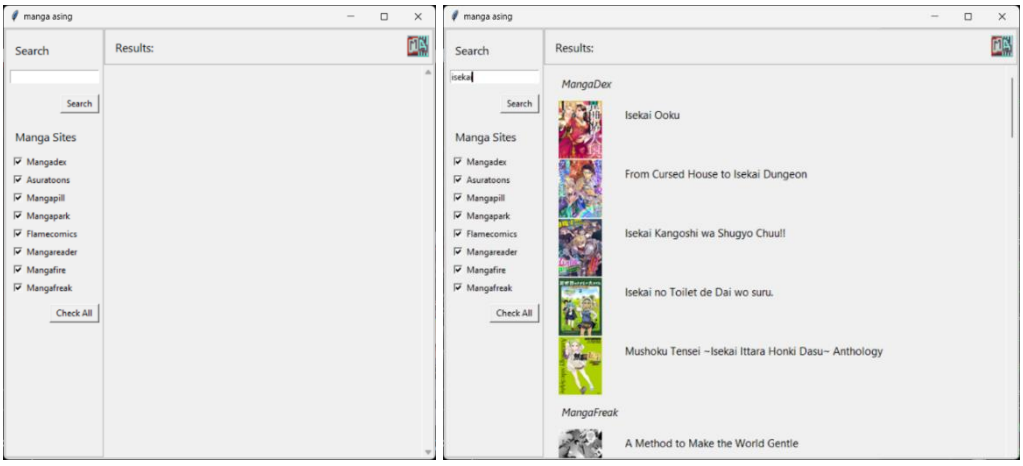
Aplikasi yang dirancangan berhasil melalukan *scraping* hasil pencarian *manga* secara paralel. Gambar 2 menunjukkan hasil proses *scraping* situs-situs *scanlation*. "*Fetching*",

"*done*", dan "*timeout*" menandakan proses scrape hasil pencarian dari situs. "*Appending*" dan "*displayed*" menandakan proses penampilan hasil pada UI.



Gambar 2. Hasil Pencarian Aplikasi Manga Asing

Gambar 3 menunjukkan tampilan aplikasi (a) sebelum melakukan pencarian dan (b) tampilan aplikasi setelah melakukan pencarian. Pencarian dilakukan dengan memasukkan judul pada kolom *Search*, memilih situs yang ingin dicari, lalu mengklik tombol *Search*. Hasil pencarian akan ditampilkan pada kolom disebelah kolom pencarian.



Gambar 3. Tampilan Aplikasi Sebelum dan Sesudah Mendapat Pencarian

Dalam pengembangan aplikasi Manga Asing ditemukan ada dua *bottleneck* yang melambatkan proses kerja aplikasi. *Bottleneck* pertama adalah pada agregasi pencarian, dimana karena aplikasi menunggu sampai seluruh *request* selesai diterima, dapat terjadi kasus dimana aplikasi lama menunggu satu situs yang sedang tersangkut. Masalah ditunjukkan pada proses jalannya aplikasi pada Gambar 4.

```
Fetching https://mangapark.net   0.10412409994751215
Fetching https://flamecomics.me          0.10585699998773634
Fetching https://mangareader.to          0.10773520008660853
Fetching https://mangafire.to    0.10987700009718537
Fetching https://ww1.mangafreak.me       0.11200960003770888
https://ww1.mangafreak.me done   0.45532980002462864
https://mangadex.org done        0.4720469999592751
https://mangafire.to done        1.4519265000708401
https://mangareader.to done      1.7398188000079244
https://mangapill.com done       1.9552621999755502
https://flamecomics.me done      2.846384499920532
https://mangapark.net done       18.806822000071406
https://asuratoon.com done       65.62716139992699
Appending MangaDex       65.62981630000286
MangaDex displayed       66.95909860008396
Appending AsuraToons     66.96075180009939
AsuraToons displayed     70.4010540000163
```

Gambar 4. Aplikasi Menunggu Salah Satu Situs selama 65 Detik

Untuk memitigasi masalah ini, digunakan kelas SessionTimeout untuk menyetel batas waktu pelaksanaan untuk setiap *request* situs supaya tidak melewati batasan yang ditentukan, yaitu 5 detik. Gambar 5 menunjukkan implementasi sistem *timeout* yang dirancang.

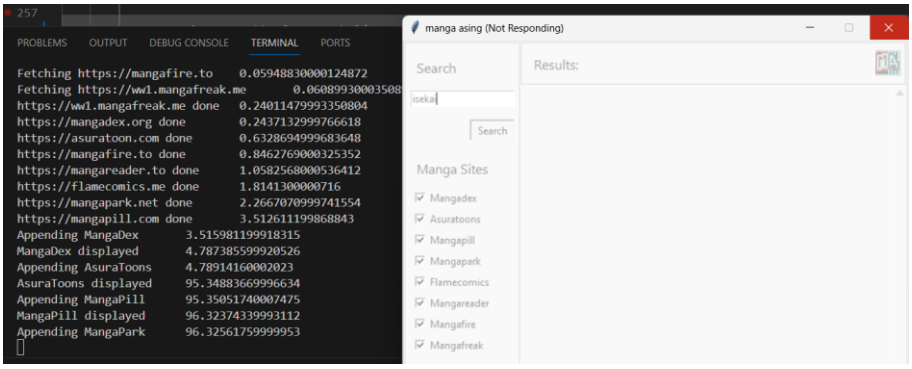
```python
async def aggregate(title):
    session_timeout = aiohttp.ClientTimeout(total=None,sock_connect=timeout,sock_read=timeout)
    async with aiohttp.ClientSession(timeout=session_timeout) as session:
        # code
        return

async def manga(title, session):
    try:
        async with session.get(f"site/search?word={title_url(title)}") as response:
            # code
            return {} #results
    except asyncio.exceptions.TimeoutError as e:
        return None
```

Gambar 5. Implementasi *Timeout* untuk *Request* yang Terlalu Lama Merespon

Bottleneck kedua adalah pada penampilan hasil pencarian yang telah digunakan. Tkinter mendukung fungsi *asynchronous* karena Tkinter memang dibangun untuk beroperasi secara *asynchronous*, namun karena aplikasi menambahkan elemen pada satu tree yang sama, elemen-elemen tidak bisa ditambahkan secara serentak dan harus ditambahkan secara sekuensial, menyebabkan aplikasi berhenti sampai proses selesai. Gambar 6 menunjukkan *bottleneck* yang terjadi pada penampilan hasil pencarian aplikasi dan Gambar 7 menunjukkan aplikasi dalam keadaan *not responding*.

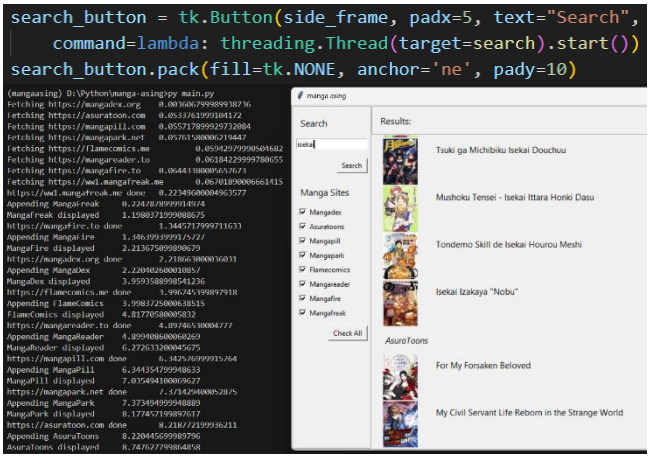
```
https://flamecomics.me done      1.8141300000716
https://mangapark.net done       2.2667070999741554
https://mangapill.com done       3.512611199868843
Appending MangaDex       3.515981199918315
MangaDex displayed       4.787385599920526
Appending AsuraToons     4.78914160002023
AsuraToons displayed     95.34883669996634
Appending MangaPill      95.35051740007475
MangaPill displayed      96.32374339993112
```

Gambar 6. Aplikasi Menunggu Tampilan Terdahulu Sebelum Menambah *Item* Baru



Gambar 7. Aplikasi *Not Responding* sampai Proses Selesai Dijalankan

Untuk mengatasi *bottleneck* ini, program menggunakan *multithreading* dan memindahkan proses-proses tersebut kepada *thread* terpisah dari proses UI Tkinter yang berada pada *thread* utama. Solusi ini sendiri tidak menyelesaikan *bottleneck* pada tampilan yang masih memerlukan waktu beberapa detik untuk memuat semua *item*, namun menyelesaikan masalah dimana aplikasi *not responding* sampai proses pencarian selesai. Implementasi solusi dan hasil *runtime* dilampirkan pada Gambar 8.



Gambar 8. Implementasi dan Hasil Penggunaan *Multithreading*

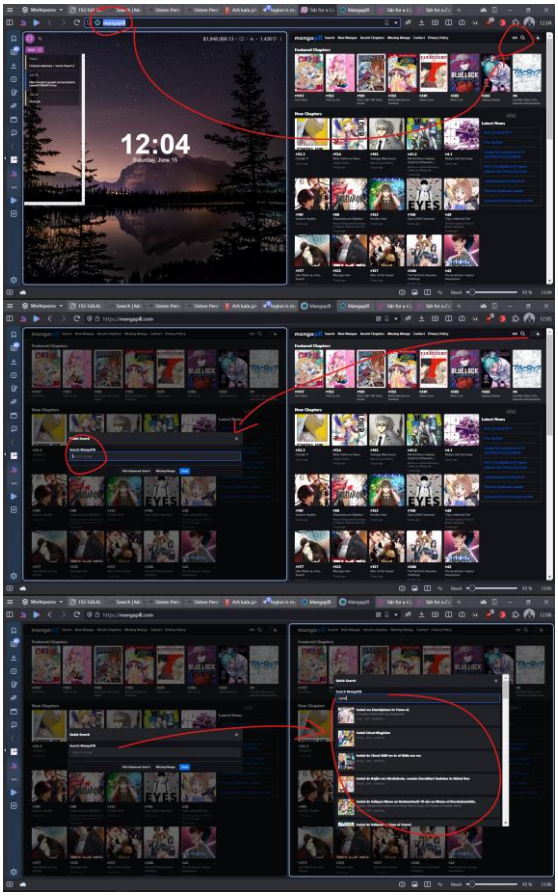
Aplikasi memiliki tiga keunggulan diatas mencari situs secara satu persatu. Keunggulan pertama adalah menggunakan aplikasi menghemat waktu pengguna dengan mencari situs secara sekaligus dengan komputasi paralel sehingga pengguna tidak perlu menunggu hasil dari satu situs sebelum pindah ke situs lain. Gambar 9 mengilustrasikan perbandingan proses *scraping* secara sekuential dan secara paralel, dimana proses sekuential akan menunggu sampai proses sebelumnya selesai sebelum memanggil situs lain, sedangkan proses paralel memanggil situs-situs secara sekaligus.

```
Running scraping sequentially
Fetching https://mangadex.org     0.005828300025314093
https://mangadex.org done         0.4219382998999208
Fetching https://asuratoon.com    0.4234790999908
https://asuratoon.com done        8.488025199854746
Fetching https://mangapill.com    8.4895424998831
https://mangapill.com done        10.217331500025466
Fetching https://mangapark.net    10.218909400049597
https://mangapark.net done        12.516588900005445
Fetching https://flamecomics.me          12.51864010002464
https://flamecomics.me done       14.238736799918115
Fetching https://mangareader.to          14.240177300060168
https://mangareader.to done       15.307039499981329
Fetching https://mangafire.to     15.308826399967074
https://mangafire.to done         16.387816000031307
Fetching https://ww1.mangafreak.me       16.388954299967736
https://ww1.mangafreak.me done    18.333360699936748
```

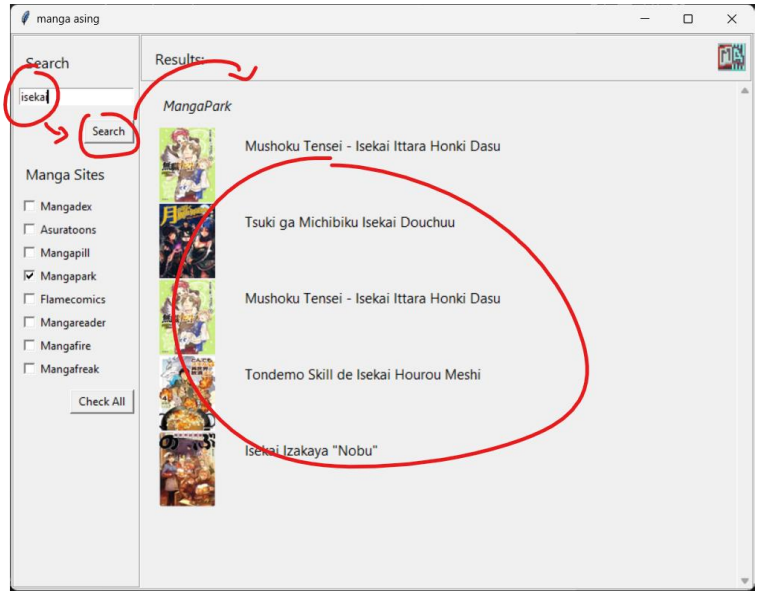
```
Running scraping asynchronously
Fetching https://mangadex.org     0.001806299900636077
Fetching https://asuratoon.com    0.0039037999231368303
Fetching https://mangapill.com    0.006453200010582805
Fetching https://mangapark.net    0.00809079990722239
Fetching https://flamecomics.me          0.010092899901792407
Fetching https://mangareader.to          0.01214609993621707
Fetching https://mangafire.to     0.013896999880671501
Fetching https://ww1.mangafreak.me       0.015695699956268072
https://ww1.mangafreak.me done    0.20652160001918674
https://mangadex.org done         0.210291099967435
https://mangafire.to done         0.8329845999833196
https://mangareader.to done       1.4765677999239415
https://flamecomics.me done       1.5613690000027418
https://mangapill.com done        1.6559041999280453
https://mangapark.net done        2.2648308998905122
https://asuratoon.com timed out          5.12939319992438
```

Gambar 9. Hasil Proses *Scraping* Sekuential dan Paralel

Keunggulan kedua adalah aplikasi mempersingkat langkah pencarian pengguna yang sebelumnya harus mengikuti tampilan berbeda dari masing-masing situs, sekarang dapat dilakukan secara tersentralisasi melalui satu aplikasi. Gambar 10 mendemonstrasikan proses pencarian *manga* pada salah satu situs yang tersedia pada aplikasi, sedangkan Gambar 11 menunjukkan proses pencarian pada aplikasi.

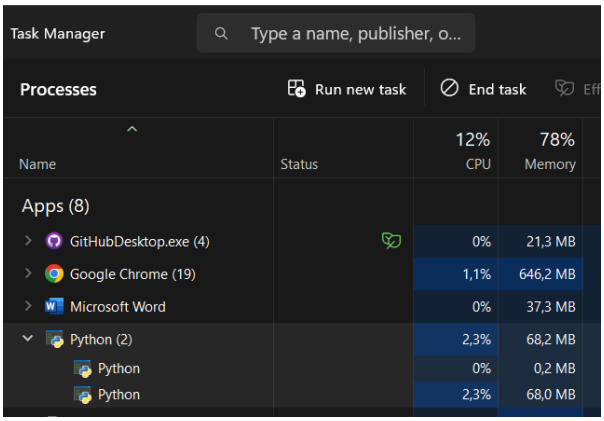


Gambar 10. Proses Pencarian pada Situs Manga Pill



Gambar 11. Proses Pencarian pada Aplikasi Manga Asing

Keunggulan ketiga adalah mencari secara manual dengan membuka setiap situs pada *browser* akan memakan banyak memori, sedangkan dengan menggunakan aplikasi akan lebih sedikit memori yang diperlukan. Gambar 12 menunjukkan bahwa aplikasi menggunakan lebih sedikit memori dibanding membuka 8 situs pada Google Chrome. Memori yang dihemat ini dapat digunakan untuk menjalankan proses lain selagi mencari *manga*, sehingga dapat mempercepat pekerjaan pengguna.



Gambar 12. Penggunaan Memori Memuat 8 Situs dan Aplikasi Manga Asing

Bab 4. Kesimpulan

Aplikasi yang dibangun berhasil menjadi solusi untuk menarik hasil pencarian dari beberapa situs *scanlation* secara paralel. Aplikasi menerapkan konsep *asynchronous* untuk menambahkan hasil pencarian yang pertama kali diterima ke dalam daftar hasil selagi menunggu hasil lainnya, dan juga menerapkan konsep *multithreading* untuk menjalankan dua

thread program secara sekaligus untuk UI dan untuk pencarian *manga*. Aplikasi menjadi alternatif bagus bagi pengguna yang ingin mencari *manga* tertentu saat pengguna telah mengetahui judul *manga* tersebut, namun tidak tahu dimana dapat menemukan terjemahan *manga*.

Saran untuk proyek pembuatan aplikasi *scraping* selanjutnya adalah menggunakan *package* atau bahasa pemrograman yang lebih mendukung proses *asynchronous*, sehingga dapat menambahkan daftar *item* secara paralel. Kelemahan dari aplikasi ini adalah kebanyakan *bottleneck* terjadi karena *package* Tkinter tidak dapat menyusun *widget* secara paralel, sehingga dapat menjadi titik *speed up* yang baik untuk iterasi proyek selanjutnya.

Referensi

Fabretti, M. (2016). The Use of Translation Notes in Manga Scanlation. TranscUlturAl, 8(2), 84–106.

Hinami, R., Ishiwatari, S., Yasuda, K., & Matsui, Y. (2021). Towards Fully Automated Manga Translation. Proceedings of the AAAI Conference on Artificial Intelligence, 35(14), 12998–13008. https://doi.org/10.1609/aaai.v35i14.17537

Su, H., Niu, J., Liu, X., Li, Q., Cui, J., & Wan, J. (2021). MangaGAN: Unpaired Photo-to-Manga Translation Based on The Methodology of Manga Drawing. Proceedings of the AAAI Conference on Artificial Intelligence, 35(3), 2611–2619. https://doi.org/10.1609/aaai.v35i3.16364

Valero-Porras, M.-J., & Cassany, D. (2015). Multimodality and Language Learning in a Scanlation Community. Procedia - Social and Behavioral Sciences, 212, 9–15. https://doi.org/https://doi.org/10.1016/j.sbspro.2015.11.291