MYER-PW-SCRAPPER

Eky Pradhana

Demo Session

https://youtu.be/W8wbVK8WG2U

Problem Statement

The myer.com.au website presents challenges for traditional web scraping methods due to its heavy use of JavaScript for rendering dynamic content. This dynamic nature makes it difficult to find specific endpoints that return the desired data through REST API.

As a result, an alternative approach is required to extract the required data effectively.

Solution Overview

To tackle the problem, I employed a combination of **Playwright** and **BeautifulSoup4** libraries.

Playwright provides a powerful toolset for automating web browsers and interacting with JavaScript-heavy websites. It allows me to navigate the website, execute JavaScript, and extract the rendered HTML content.

BeautifulSoup4, on the other hand, offers a convenient and flexible way to parse and extract data from HTML documents.

Technology / Library used

- Playwright
- BeautifulSoup 4 (BS4) and lxml
- Asyncio
- Dataclass

Implementation Details

I developed a Python script that utilizes Playwright to navigate to the myer.com.au website and extract the HTML content of the desired pages. The script leverages the parallel execution capability of asyncio to improve performance by processing multiple pages simultaneously.

For each page, the extracted HTML is then parsed using BeautifulSoup4 to extract the required data, such as product details, prices, and variants into JSON files.

Benefits of Playwright and BeautifulSoup4

- Playwright provides a full-featured browser automation solution, allowing interaction with JavaScript-driven websites, handling dynamic content, and executing JavaScript code.
- BeautifulSoup4 offers a simple and intuitive API for parsing and navigating HTML documents, making it easy to extract data using CSS selectors or XPath expressions.

Request Optimization

To ensure efficient scraping and improve the load time of the web scraping process, I implemented several optimization techniques to block unnecessary requests and reduce the overall network traffic. Such as:

- Images
- Unnecessary Domains: google, dynamicyield.com, bazaarvoice.com, truefitcorp.com, reporting.cdndex.io, bam.nr-data.net and others.
- Unnecessary JS : ips.js, I identified that the "ips.js" resource was a contributing factor to frequent timeouts during the scraping process

```python
def route_intercept(route):
    if route.request.resource_type == "image":
        # print(f"Blocking the image request to: {route.request.url}")
        return route.abort()
    if "google" in route.request.url:
        print(f"blocking {route.request.url} as it contains Google")
        return route.abort()
    if "dynamicyield" in route.request.url:
        print(f"blocking {route.request.url} as it contains dynamicyield")
        return route.abort()
    if "bazaarvoice" in route.request.url:
        print(f"blocking {route.request.url} as it contains bazaarvoice")
        return route.abort()
    if "truefitcorp.com" in route.request.url:
        print(f"blocking {route.request.url} as it contains truefitcorp.com")
        return route.abort()
    if "reporting.cdndex.io" in route.request.url:
        print(f"blocking {route.request.url} as it contains reporting.cdndex.io")
        return route.abort()
    if "bam.nr-data.net" in route.request.url:
        print(f"blocking {route.request.url} as it contains bam.nr-data.net")
        return route.abort()
    if "ips.js" in route.request.url:
        print(f"blocking {route.request.url} as it contains ips.js")
        return route.abort()
    if "https://api-online.myer.com.au/149e9513-01fa-4fb0-aad4-566afd725d1b/2d206a39-8ed
        print(f"blocking {route.request.url} as it contains /tl")
        return route.abort()
    return route.continue_()
```

Performance Optimization

This solution is prepared to enable parallel execution using asyncio.

It enables the script to process multiple pages concurrently, significantly improving overall scraping speed.

By leveraging the asynchronous nature of Playwright and the non-blocking operations of asyncio, I achieved efficient data extraction from the myer.com.au website.

```python
async def main():
    # PREPARED FOR PARALLEL BROWSERS OPENED

    # number_of_task = 1
    # number_of_page = 51
    # pages = [
    #     1
    # ]

    # tasks = []
    # for pg in pages:
    #     tasks.append(run_playwright(pg))

    tasks = []
    tasks.append(run_playwright(page_start=2,page_end=2)) # only scrap products in page 2
    # tasks.append(run_playwright(3,3))
    await asyncio.gather(*tasks)

# Run the main function
asyncio.run(main())
```

Result Sample

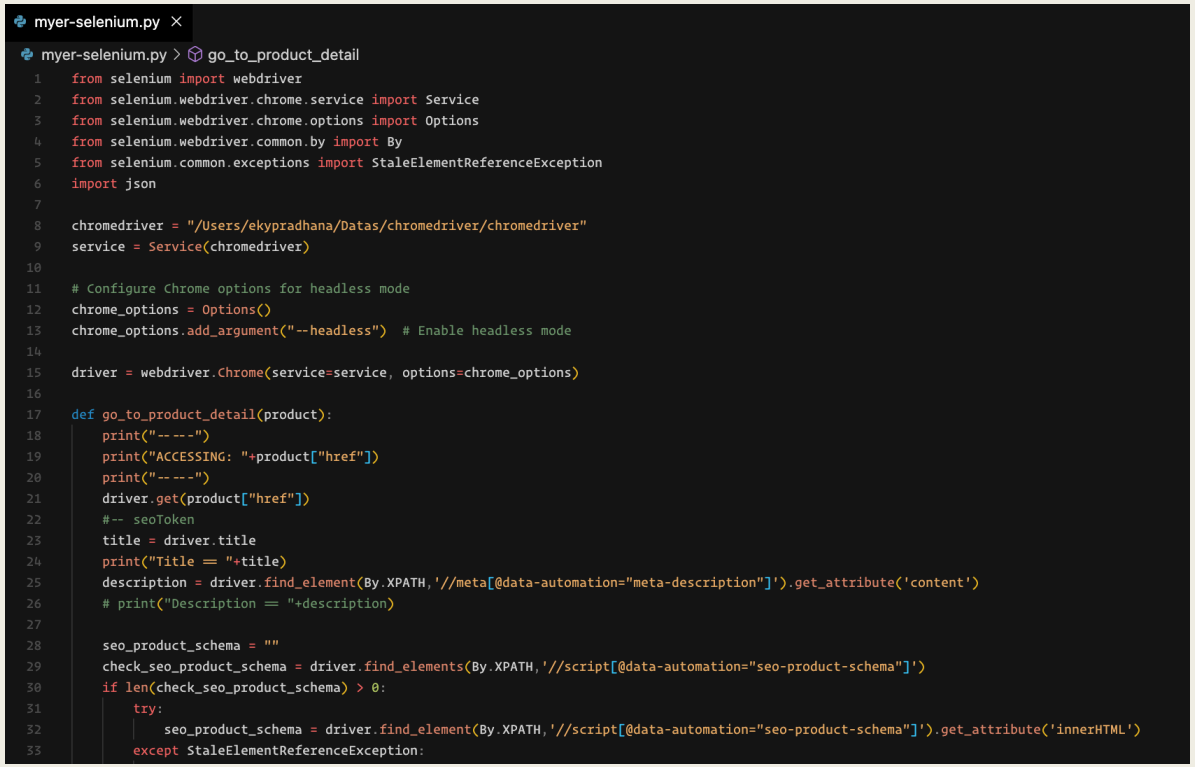

```
{
    "product_id": "956985400",
    "product_name": "Rundale Long Sleeve Check Shirt in Green",
    "product_brand": "Reserve",
    "product_detail_url": "https://www.myer.com.au/p/reserve-rundale-long-sleeve-check-shirt-in-green",
    "product_price_was": "$79.95",
    "product_price_now": "$47.97",
    "SEO": {
        "title": "Reserve Rundale Long Sleeve Check Shirt In Green | MYER",
        "description": "",
        "product_schema": "{\"@context\":\"http://schema.org/\",\"@type\":\"Product\",\"@id\":\"reserve-rundale-long-sleev
        "product_review_schema": ""
    },
    "variants": [{
        "color": "forest",
        "sizes": [{
            "size_name": "S",
            "is_available": true
        }, {
            "size_name": "M",
            "is_available": true
        }],
        "stock_indicator": "in stock",
        "url": "https://www.myer.com.au/p/reserve-rundale-long-sleeve-check-shirt-in-green"
    }, {
        "color": "navy",
        "sizes": [{
            "size_name": "S",
            "is_available": true
        }, {
            "size_name": "M",
            "is_available": true
        }],
        "stock_indicator": "in stock",
        "url": "https://www.myer.com.au/p/reserve-rundale-long-sleeve-check-shirt-in-navy"
    }]
}
```

Comparison with Selenium

During the development process, I initially experimented with Selenium.

However, I found that Selenium's performance was noticeably slower compared to Playwright when dealing with the JavaScript-heavy myer.com.au website.

Selenium's reliance on a separate WebDriver and its slower execution speed made it less suitable for efficient scraping of dynamic content.



How to Run

- Recommended to use venv
- Pip install playwright, bs4, lxml, asyncio, dataclasses
- Run : python main.py
- Step by step : https://youtu.be/W8wbVK8WG2U

Future Development

- Enable ARGS to specify number of tasks, page start and page end on initial run
- Grab other data (e.g : size guide, etc)
- Multiple output format (e.g : csv)
- Adding more logs and handling more exception

Thank You!

Eky Pradhana
eky.pradhana@gmail.com
https://www.linkedin.com/in/ekypradhana/
https://medium.com/@ekypradhana