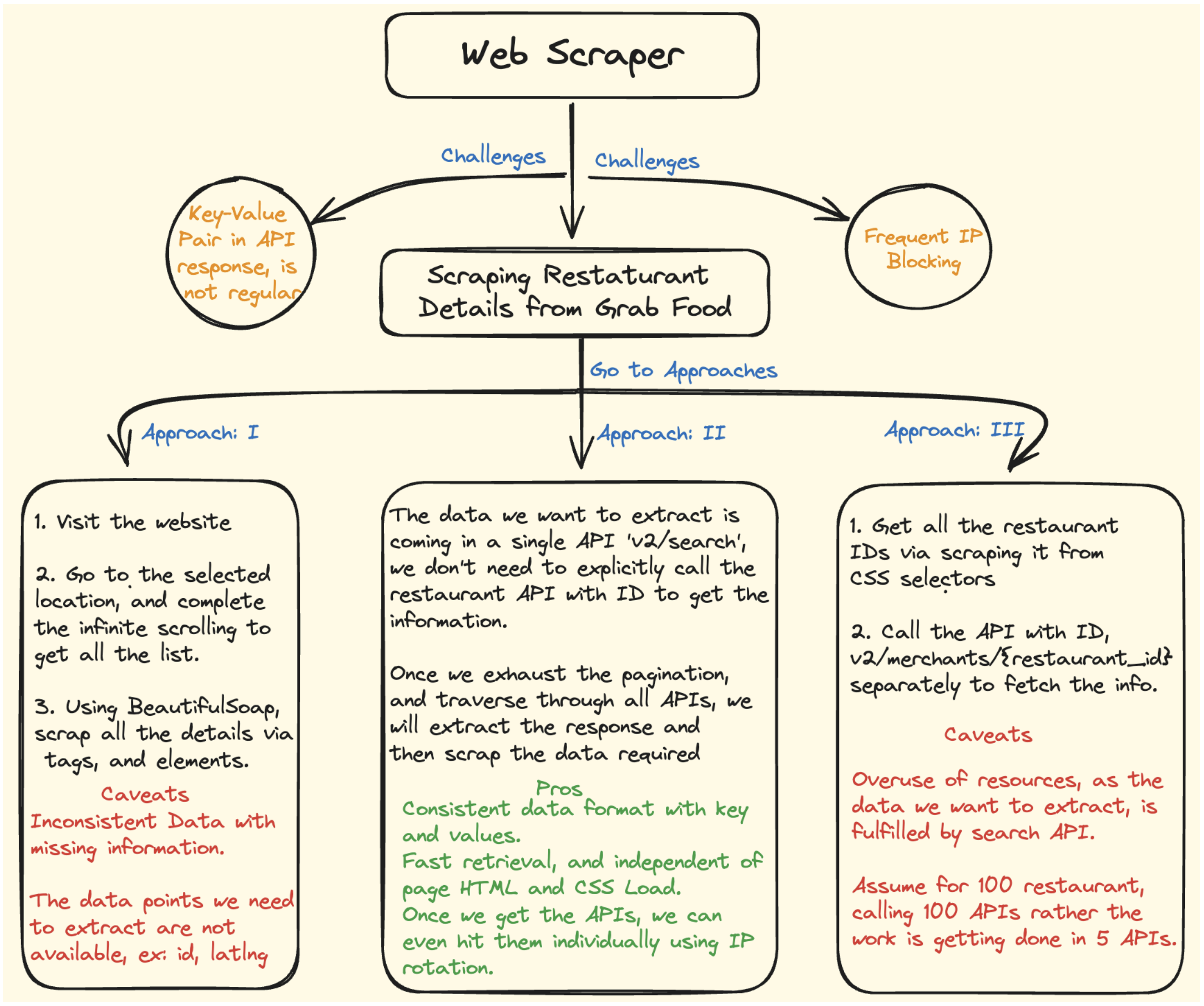
Grab Food Delivery Web Scraper

This project aims to develop a web scraper to extract specific information from the Grab Food Delivery platform.

One View: Objective, Approach, Challenges, and Go-to Strategy Look (Finalized Approach-II)



1. Visit the website

2. Go to the selected location, and complete the infinite scrolling to get all the list.

3. Using BeautifulSoap, scrap all the details via tags, and elements.

Caveats

Inconsistent Data with missing information.

The data points we need to extract are not available, ex: id, latlng

The data we want to extract is coming in a single API 'v2/search', we don't need to explicitly call the restaurant API with ID to get the information.

Once we exhaust the pagination, and traverse through all APIs, we will extract the response and then scrap the data required

Pros

Consistent data format with key and values.

Fast retrieval, and independent of page HTML and CSS Load.

Once we get the APIs, we can even hit them individually using IP rotation.

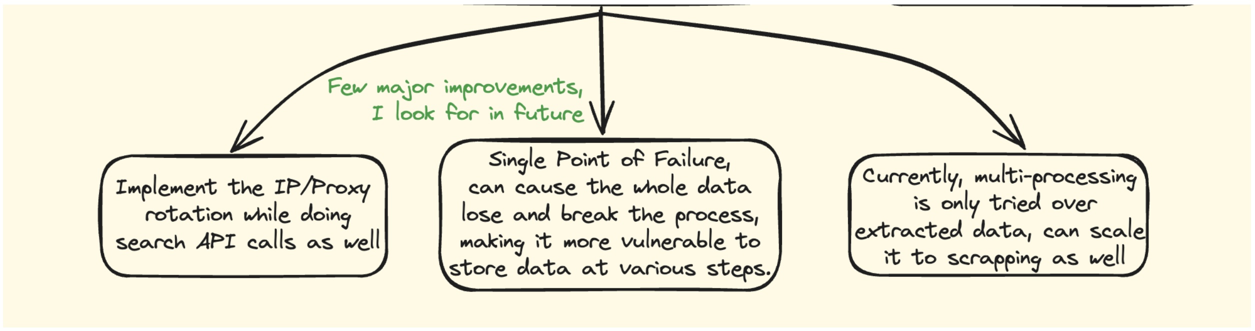
1. Get all the restaurant IDs via scraping it from CSS selectors

2. Call the API with ID, v2/merchants/{restaurant_id} separately to fetch the info.

Caveats

Overuse of resources, as the data we want to extract, is fulfilled by search API.

Assume for 100 restaurant, calling 100 APIs rather the work is getting done in 5 APIs.



Implement the IP/Proxy rotation while doing search API calls as well

Single Point of Failure, can cause the whole data lose and break the process, making it more vulnerable to store data at various steps.

Currently, multi-processing is only tried over extracted data, can scale it to scrapping as well

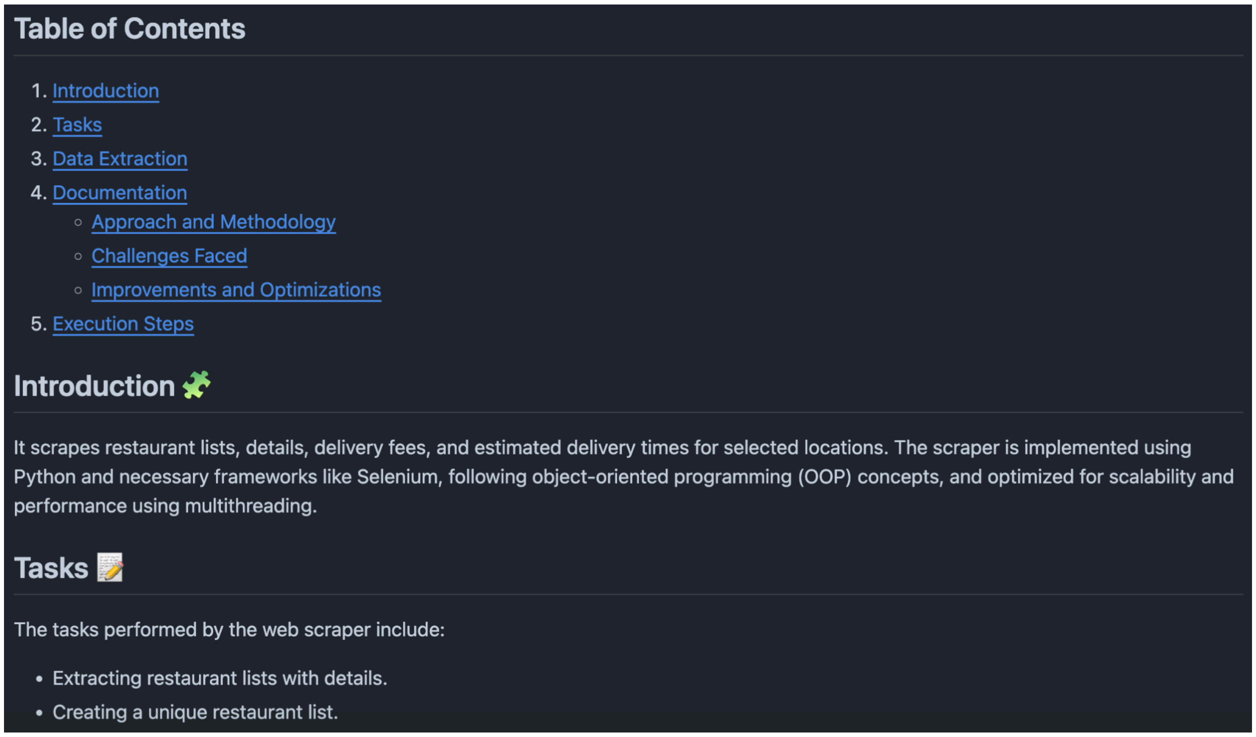


Table of Contents

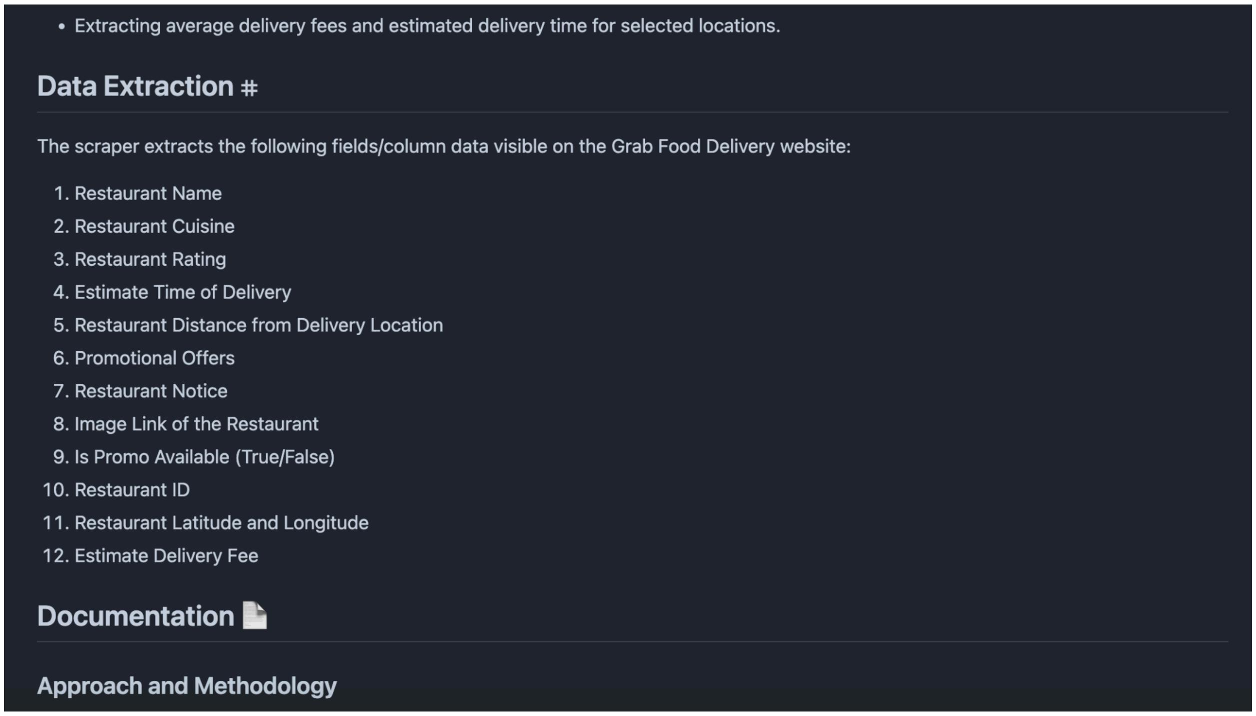
Introduction 🧩

It scrapes restaurant lists, details, delivery fees, and estimated delivery times for selected locations. The scraper is implemented using Python and necessary frameworks like Selenium, following object-oriented programming (OOP) concepts, and optimized for scalability and performance using multithreading.

Tasks 📝

The tasks performed by the web scraper include:

- Extracting restaurant lists with details.
- Creating a unique restaurant list.



- Extracting average delivery fees and estimated delivery time for selected locations.

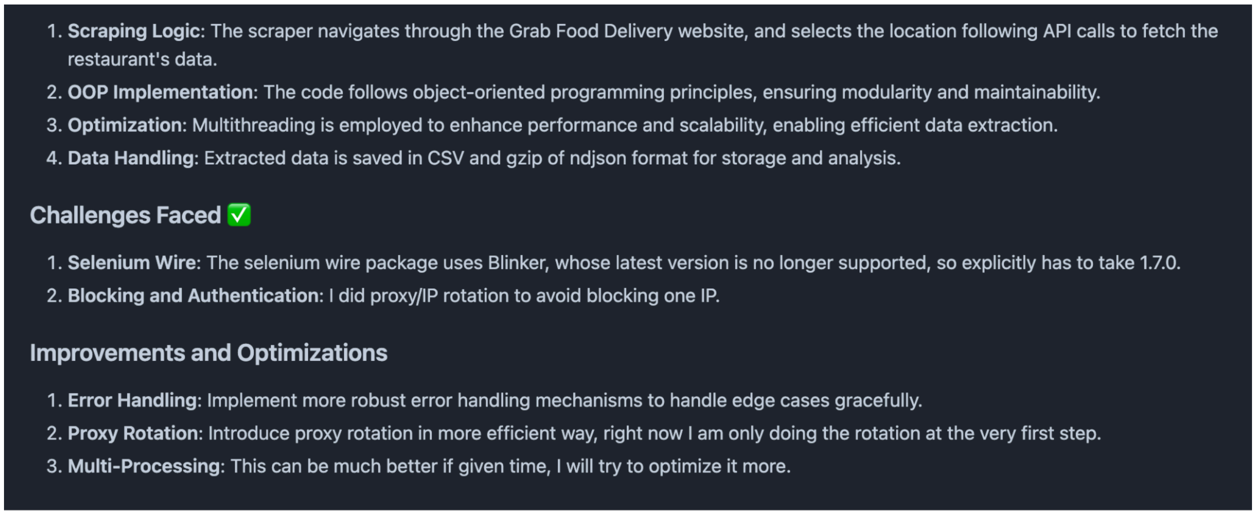
Data Extraction

The scraper extracts the following fields/column data visible on the Grab Food Delivery website:

1. Restaurant Name
2. Restaurant Cuisine
3. Restaurant Rating
4. Estimate Time of Delivery
5. Restaurant Distance from Delivery Location
6. Promotional Offers
7. Restaurant Notice
8. Image Link of the Restaurant
9. Is Promo Available (True/False)
10. Restaurant ID
11. Restaurant Latitude and Longitude
12. Estimate Delivery Fee

Documentation 📄

Approach and Methodology



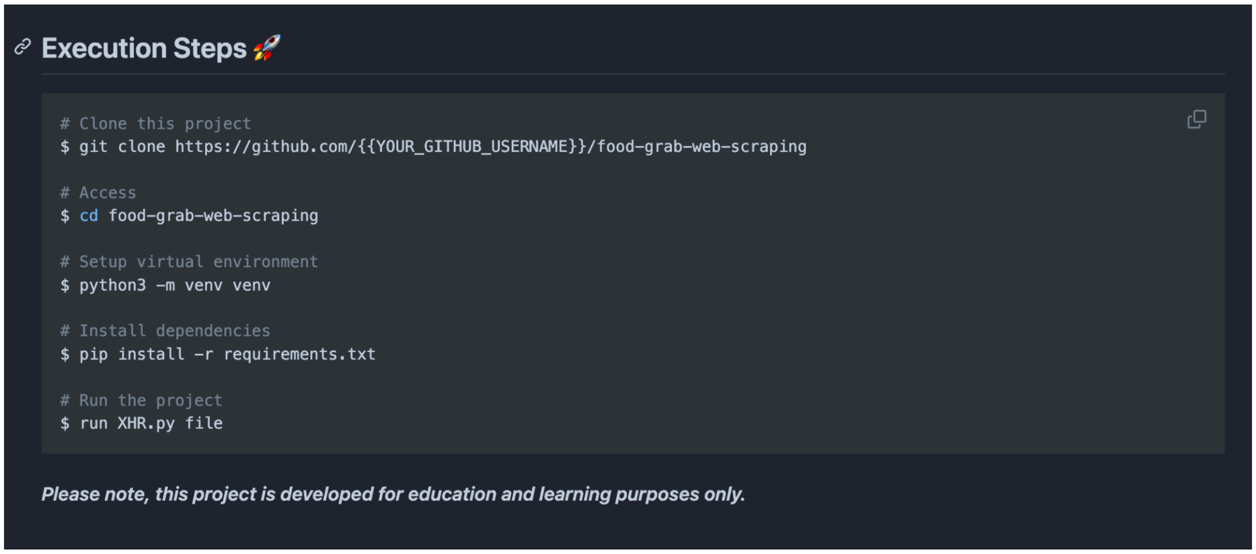
1. **Scraping Logic**: The scraper navigates through the Grab Food Delivery website, and selects the location following API calls to fetch the restaurant's data.
2. **OOP Implementation**: The code follows object-oriented programming principles, ensuring modularity and maintainability.
3. **Optimization**: Multithreading is employed to enhance performance and scalability, enabling efficient data extraction.
4. **Data Handling**: Extracted data is saved in CSV and gzip of ndjson format for storage and analysis.

Challenges Faced ✅

1. **Selenium Wire**: The selenium wire package uses Blinker, whose latest version is no longer supported, so explicitly has to take 1.7.0.
2. **Blocking and Authentication**: I did proxy/IP rotation to avoid blocking one IP.

Improvements and Optimizations

1. **Error Handling**: Implement more robust error handling mechanisms to handle edge cases gracefully.
2. **Proxy Rotation**: Introduce proxy rotation in more efficient way, right now I am only doing the rotation at the very first step.
3. **Multi-Processing**: This can be much better if given time, I will try to optimize it more.



Execution Steps 🚀

```
# Clone this project
$ git clone https://github.com/{{YOUR_GITHUB_USERNAME}}/food-grab-web-scraping

# Access
$ cd food-grab-web-scraping

# Setup virtual environment
$ python3 -m venv venv

# Install dependencies
$ pip install -r requirements.txt

# Run the project
$ run XHR.py file
```

Please note, this project is developed for education and learning purposes only.