**Task #1 Gab Data Scraping using Python**

Given a website to work with, picking the right tool for web scraping depends mainly on what you’re trying to achieve, what kind of website you’re scraping, and what to do with the data. For this specific task, we are asked to extract the following information to monitor users from the social media platform Gab:

* Date joined
* User name
* User image
* Cover photo
* About
* Number of Gabs
* Number of followers
* Number of following
* Last 50 posts (including media in case it exists)
* Average engagement of the posts

Python is our main tool for accomplishing this task, with the limitation that we are not allowed to use any platform related third party code. This also includes the prohibition of any official platform API. With these in mind, we first look at the structure of the website. This was done using the Chrome browser with its built-in Developer tools.

At first glance, the user page for the Gab website doesn’t require you to login to see the content. This saves us the time to consider or give up credentials to access the site. Upon inspection using the Developer tools, it can be seen that the classes of the elements are unusual or very hard to comprehend. For example, we have classes with values likethose shown in the right side of Figure 1. Furthermore, by navigating to the Fetch/XHR filter from the network tab, we can see all the requests that use the Fetch API as shown in Figure 2. This allows to make HTTP request in JavaScript.

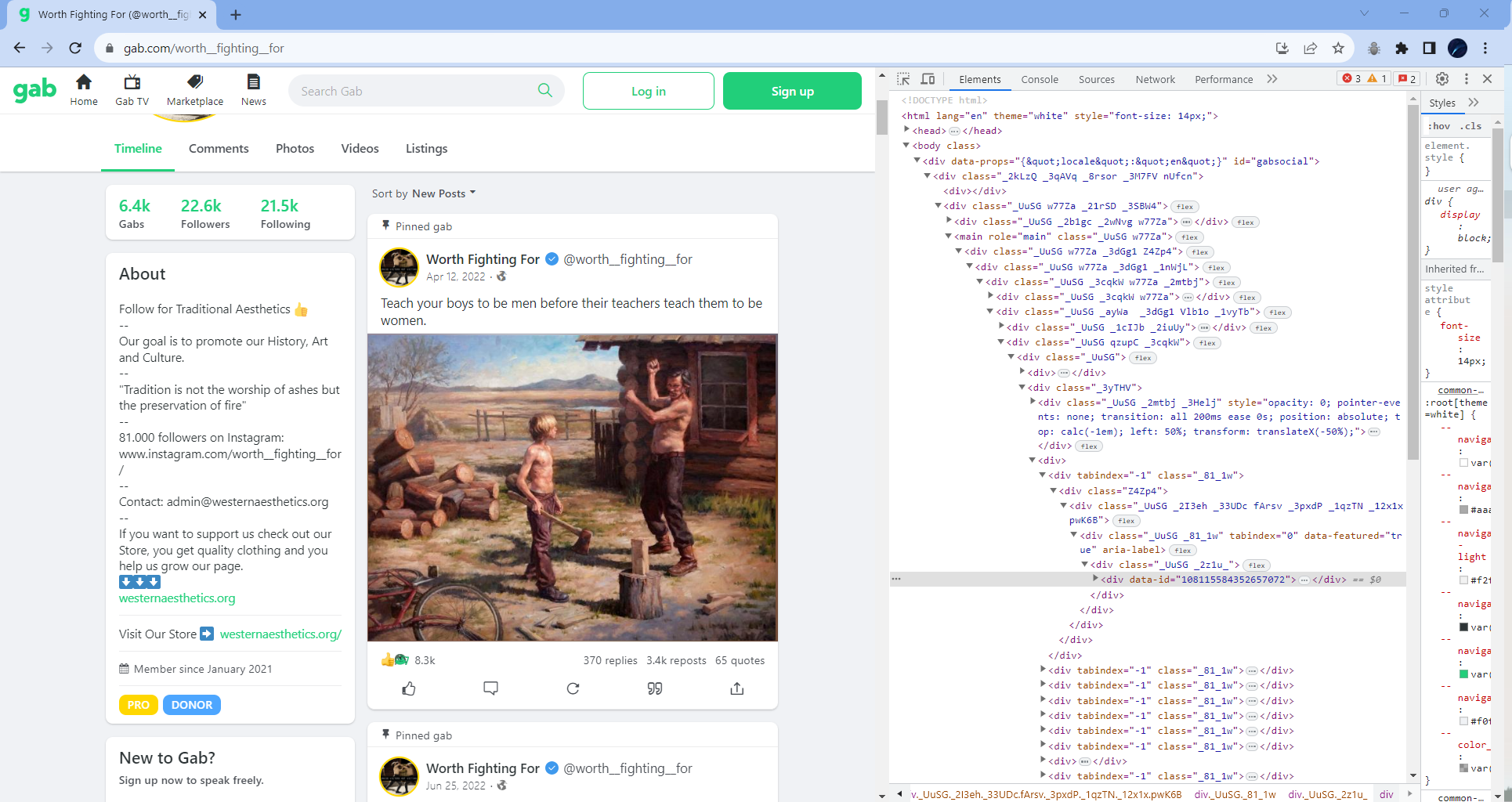


Figure 1. HTML structure of the profile page of a Gab user

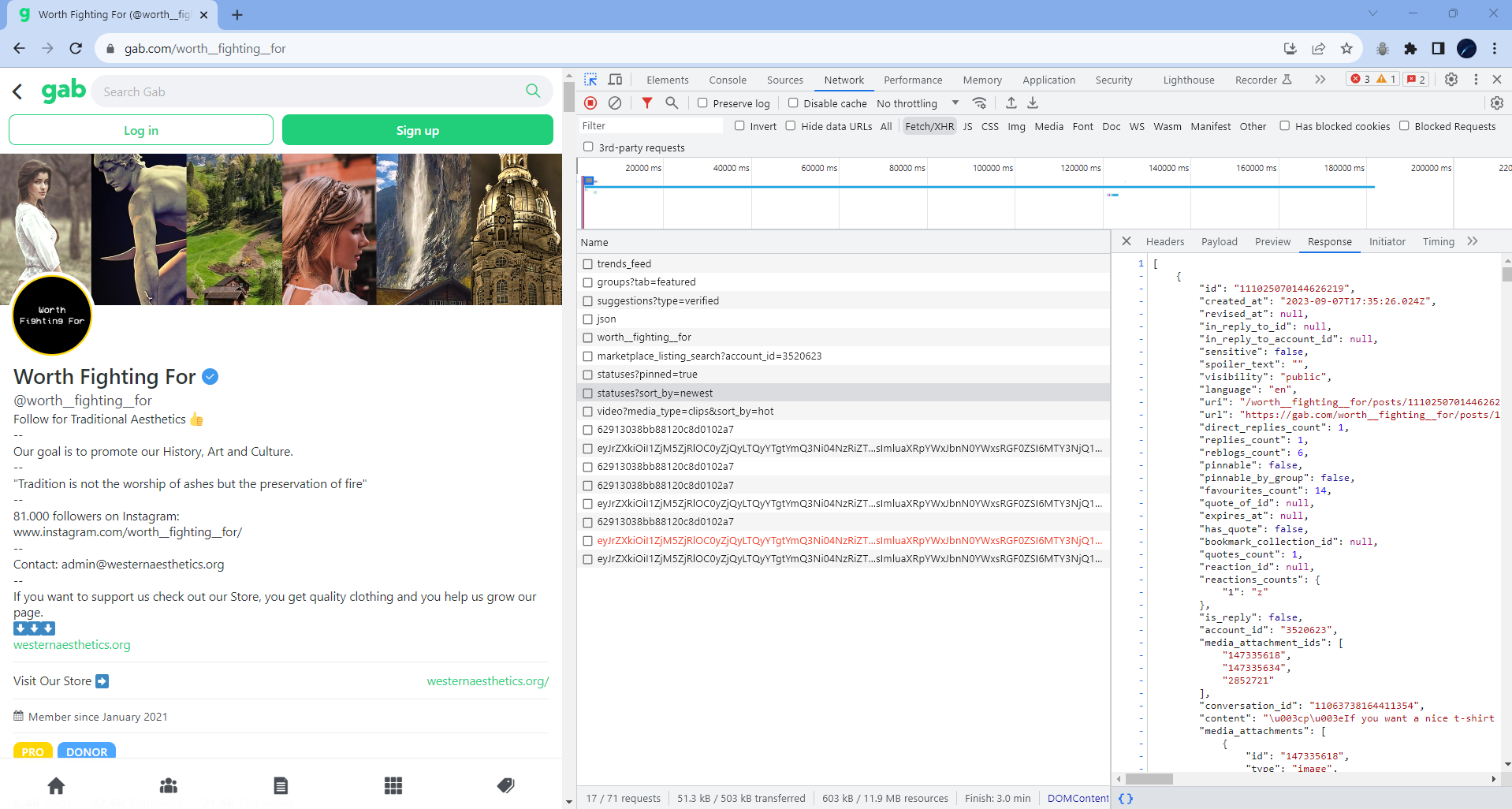


Figure 2. Filtered network responses of the website upon inspecting with Developer tools

The inspection made tells us that website is modern in nature allowing its data to be rendered heavily through JavaScript. This means that the data extraction from the front-end would be tricky. Generally, it would be easier to access the information from the back-end or through APIs.

Since we must build our own scraper in order to understand how data is structured in the platform, we ought to find another way. In this task, we used the selenium library in Python. The website was fully requested with a successful response using selenium, as opposed to when using BeautifulSoup or Scrapy, which may require more advanced stuff to perform.

First, we created a virtual environment to help keep the dependencies separated. VS Code was used as the IDE. Then, different modules/libraries were installed including selenium, pandas, and time. The list of dependencies are listed in the *requirements.txt* file. During the coding part, these modules were first imported and configured. Taking into account that approximately 100 users per day would be monitored, a user list was made in order to make the code scalable. The website was then inspected to see where the needed data can be located.

Data extraction was primarily done through CSS selectors. It was made sure that each selector is unique to get data accurately. Exception handling was also implemented to users and posts with missing data. This include users that are relatively new with no posts yet and posts with no engagements. For such cases, no values are explicitly stated.

The initial output generated by the program is a list object (*users\_info*) containing the info for each input user. Each user info is contained in a dictionary (*info*). This is similar to JSON in format so it can be viewed as in Figure 3 using a JSON viewer (). The final output is a csv file named *gab\_data.csv*.

The code and output can be accessed from my GitHub profile as indicated below.

https://github.com/benicastro/platform-research/tree/main/scrape\_gab/selenium



Figure 3. Structure of the output in JSON format

**Task #2 Discord Data Scraping using Python**