# Python Web Scraper Application

Phase 2 Source

Autumn Capasso, Michael Galyen, Kacey Gambill,

Eric Viera, Anthony Washington

Department of Computer Science, University of Maryland Global Campus

CMSC-495

Professor Janak Rajani

July 26, 2022

## Table of Contents

## Phase 2 Plan

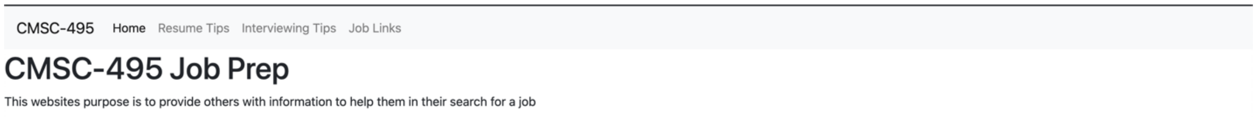
The plan for Phase 2 was to add the following functionalities:

- Performance tests for all functions.
- Migrate Unittest tests to Pytest.
- UI styling.
- Functions to scrape FreeCodeCamp for resume and interview advice (text and links).

## Additions to the Project

In this phase of the project, the team added functions to app.py to scrape FreeCodeCamp.org for resume and interview advice, started adding more styling to the user interface, and added more performance tests to the test suite. The acceptance tests in the use guide were updated to match the current UI of the application.

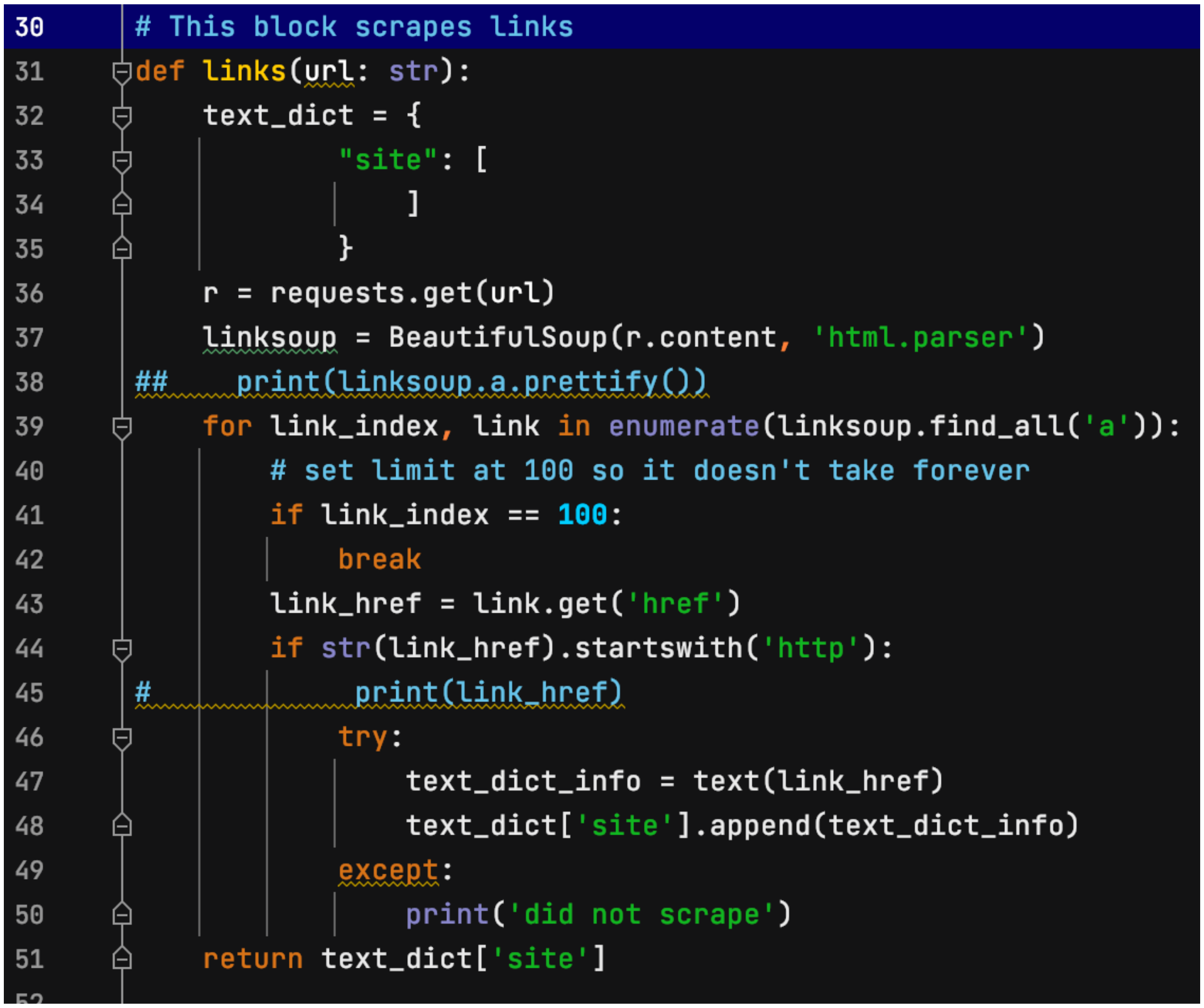
### Figure 1 image of the user interface



CMSC-495 Job Prep

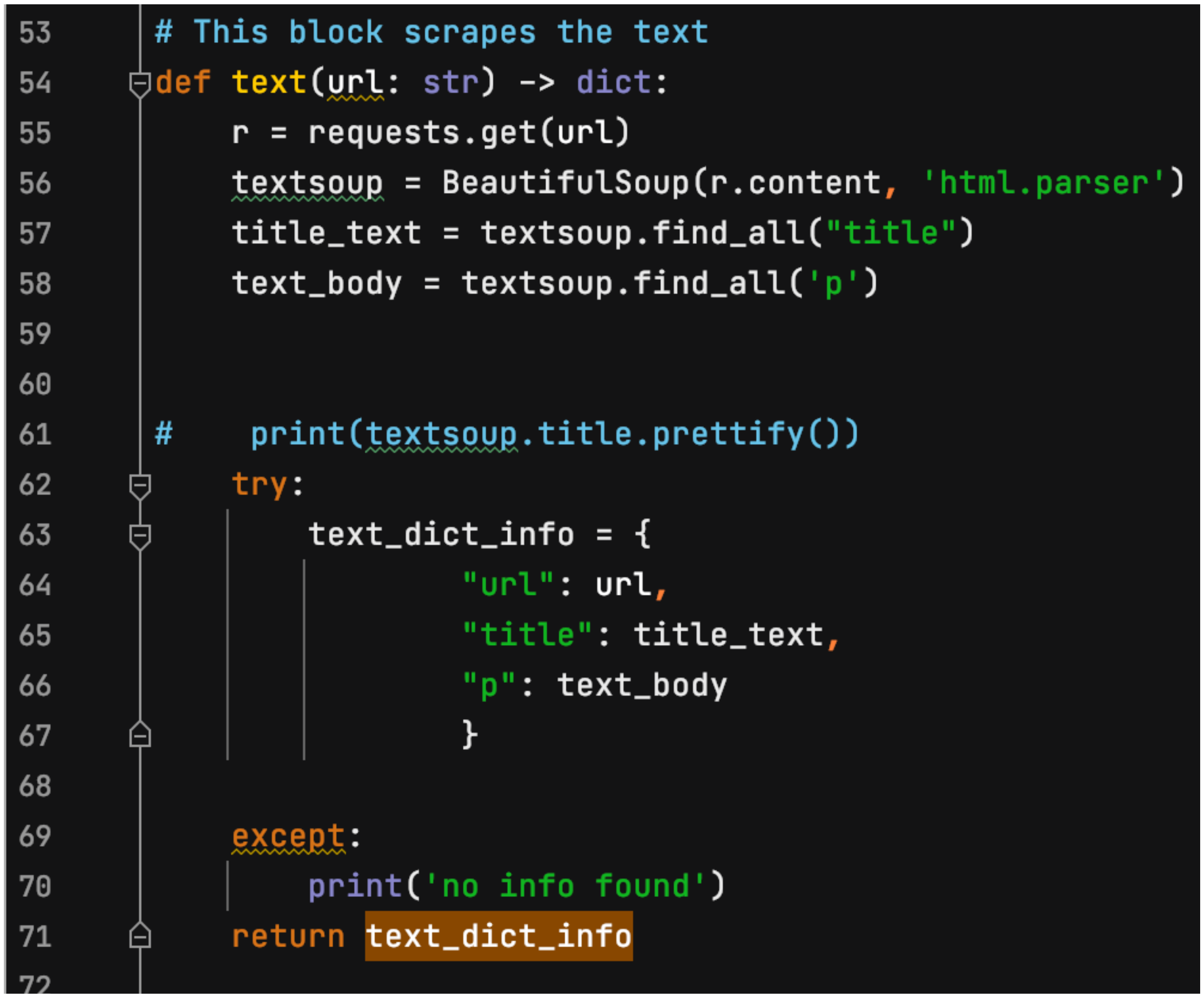
This websites purpose is to provide others with information to help them in their search for a job

Figure 2 image of the updated links function in WebScraper.py



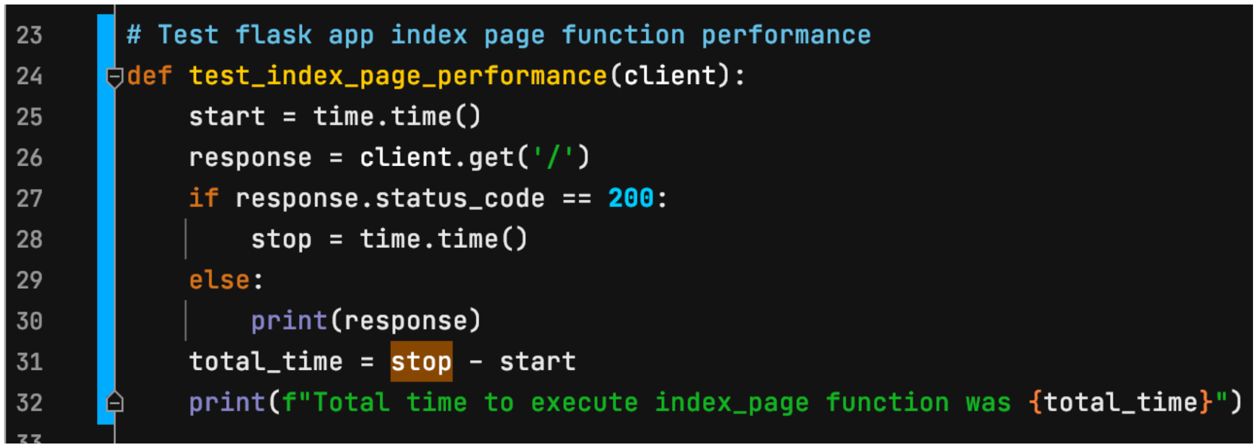
```
# This block scrapes links
def links(url: str):
    text_dict = {
            "site": [
                ]
            }
    r = requests.get(url)
    linksoup = BeautifulSoup(r.content, 'html.parser')
##     print(linksoup.a.prettify())
    for link_index, link in enumerate(linksoup.find_all('a')):
        # set limit at 100 so it doesn't take forever
        if link_index == 100:
            break
        link_href = link.get('href')
        if str(link_href).startswith('http'):
#                print(link_href)
            try:
                text_dict_info = text(link_href)
                text_dict['site'].append(text_dict_info)
            except:
                print('did not scrape')
    return text_dict['site']
```

Figure 3 image of the newWebScraper.py function, text



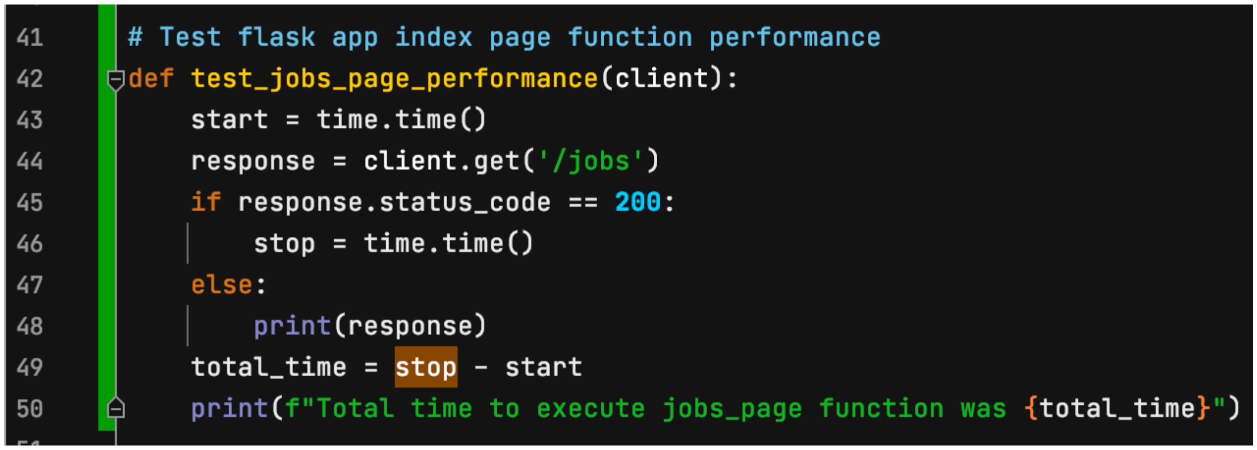
```
# This block scrapes the text
def text(url: str) -> dict:
    r = requests.get(url)
    textsoup = BeautifulSoup(r.content, 'html.parser')
    title_text = textsoup.find_all("title")
    text_body = textsoup.find_all('p')


#    print(textsoup.title.prettify())
    try:
        text_dict_info = {
                "url": url,
                "title": title_text,
                "p": text_body
                }

    except:
        print('no info found')
    return text_dict_info
```

Figure 4 image of benchmark test of the app.py index page function



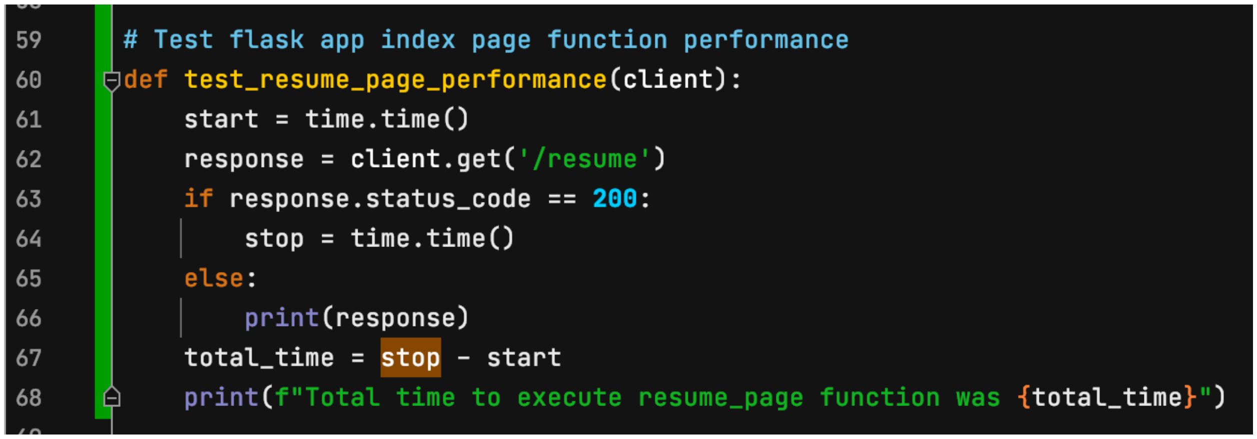
```
# Test flask app index page function performance
def test_index_page_performance(client):
    start = time.time()
    response = client.get('/')
    if response.status_code == 200:
        stop = time.time()
    else:
        print(response)
    total_time = stop - start
    print(f"Total time to execute index_page function was {total_time}")
```

Figure 5 image of benchmark test of the app.py jobs page function



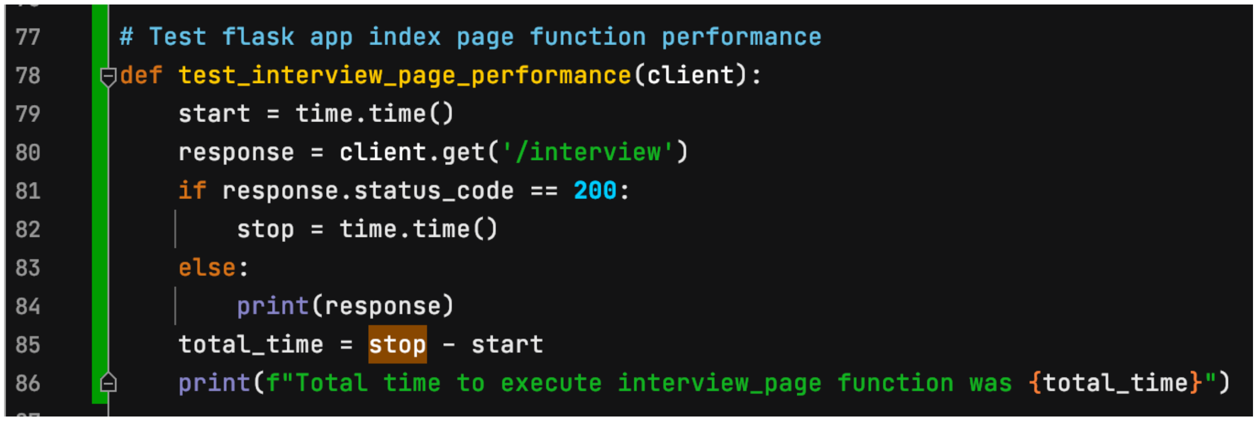
```python
# Test flask app index page function performance
def test_jobs_page_performance(client):
    start = time.time()
    response = client.get('/jobs')
    if response.status_code == 200:
        stop = time.time()
    else:
        print(response)
    total_time = stop - start
    print(f"Total time to execute jobs_page function was {total_time}")
```

Figure 6 image of benchmark test of the app.py resume page function



```python
# Test flask app index page function performance
def test_resume_page_performance(client):
    start = time.time()
    response = client.get('/resume')
    if response.status_code == 200:
        stop = time.time()
    else:
        print(response)
    total_time = stop - start
    print(f"Total time to execute resume_page function was {total_time}")
```

Figure 7 image of benchmark test of the app.py interview page function

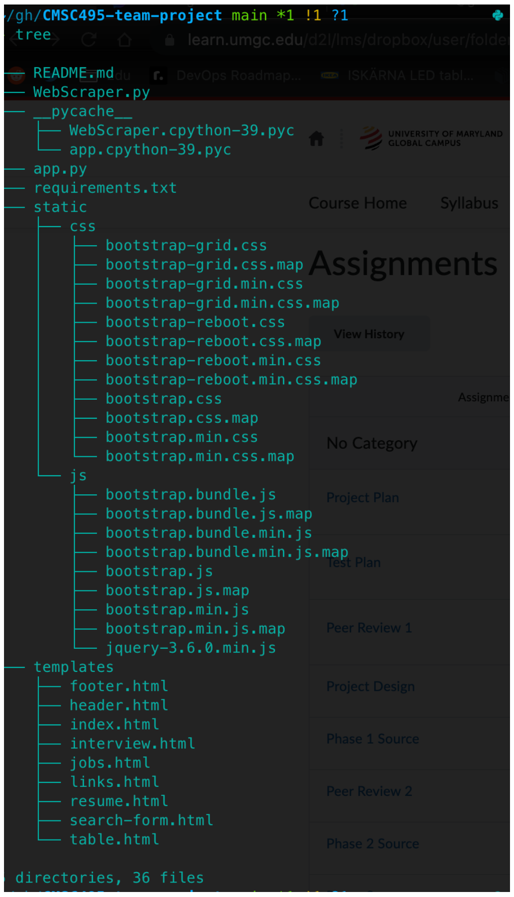


```python
# Test flask app index page function performance
def test_interview_page_performance(client):
    start = time.time()
    response = client.get('/interview')
    if response.status_code == 200:
        stop = time.time()
    else:
        print(response)
    total_time = stop - start
    print(f"Total time to execute interview_page function was {total_time}")
```

## Current Project Structure

The structure generally didn't change in a significant way from Phase 1 to Phase 2.

Figure 8 image of the project structure



```
/gh/CMSC495-team-project main *1 !1 ?1
 tree

── README.md
── WebScraper.py
── __pycache__
   ├── WebScraper.cpython-39.pyc
   └── app.cpython-39.pyc
── app.py
── requirements.txt
── static
   ├── css
   │   ├── bootstrap-grid.css
   │   ├── bootstrap-grid.css.map
   │   ├── bootstrap-grid.min.css
   │   ├── bootstrap-grid.min.css.map
   │   ├── bootstrap-reboot.css
   │   ├── bootstrap-reboot.css.map
   │   ├── bootstrap-reboot.min.css
   │   ├── bootstrap-reboot.min.css.map
   │   ├── bootstrap.css
   │   ├── bootstrap.css.map
   │   ├── bootstrap.min.css
   │   └── bootstrap.min.css.map
   └── js
       ├── bootstrap.bundle.js
       ├── bootstrap.bundle.js.map
       ├── bootstrap.bundle.min.js
       ├── bootstrap.bundle.min.js.map
       ├── bootstrap.js
       ├── bootstrap.js.map
       ├── bootstrap.min.js
       ├── bootstrap.min.js.map
       └── jquery-3.6.0.min.js
── templates
   ├── footer.html
   ├── header.html
   ├── index.html
   ├── interview.html
   ├── jobs.html
   ├── links.html
   ├── resume.html
   ├── search-form.html
   └── table.html

 directories, 36 files
```

## Challenges

Migrating tests from Unittest to Pytest and adding styles to the UI were both proposed for this week. More research and development time was needed in terms of the font end work so that will be represented better in phase 3. Migrating tests from Unittest to Pytest was an unexpected challenge so that will be pushed to phase 3 as well.

While functions were added to scrape FreeCodeCamp for links and text in the categories, resume tips and interview tips, it was discovered that these links of FreeCodeCamp are likely being generated by JavaScript so more time will be needed to figure out how to get the intended links. The text being returned by these scrapes is also a little messy with HTML tags mixed in with the text so that will also be dealt with in the next phase.

## Milestones

June 28 – Finalize project plan and specifications
July 5 – Web scraper is built and ready to test
July 5 – Test plan is finalized

July 12 – Project design is finalized
July 12 – Flask application is finalized
July 19 – Web scraper is integrated with the Flask framework
July 19 – User guide is finalized
July 19 – Alternative design write up (if needed)
July 26 – Submit test results
August 2 – Finalize history write up
August 9 – Finalize documentation

### Milestones Review

According to the milestones listed in our original project plan document, all testing with results was supposed to be finalized by this phase. More time is needed for this so it will get pushed to phase 3. The user guide was finalized so that is up to date. The web scraper has been integrated with the Flask app. The general structure and main functions of the Flask app have been finalized. The project design has been finalized. The project plan and specifications have been finalized.

## Conclusions

While this sprint was productive, some things will carry over to the next sprint. UI styling, test migration, and better parsing will all be pushed to Phase 3.