

# Pulling The Trigger With Internaloha: The New Opportunities For Internship

Shinya Saito

Collaborative Software Development Laboratory  
Department of Information and Computer Sciences  
University of Hawaii

## ABSTRACT

In the last few decades, the importance of internships for a career development has grown. However, the difficulty to search through possible internships has not decreased. Undergraduate students are forced to spend long hours and use multiple job searching websites for internships. Furthermore, these websites do not provide them with internships that they fully qualify for. Thus, the Internaloha project provides an alternative "shortcut" for students seeking internships. The approach of Internaloha is creating a scraper to extract Computer Science internships from the different job websites and have them integrated into RadGrad's internship explorer page. The goal for Internaloha Version 2 is to recreate them using TypeScript and clean it up to look more efficient.

Keywords: Software Engineering, Web Scraping, Internships

## 1 INTRODUCTION

From day 1, most undergraduate students are under the pressure of employment post-graduation. The task of job hunting is easier said than done as it is a war zone. Thus, student applicants need to build an outstanding resume filled with experience. While it is also good to have a high-grade point average (GPA), one of the other great ways for students to gain experience is to participate in internships. According to a survey done in June 2020 with undergraduate students, 90% of them intended to get an internship before graduation [1]. However, this is another challenging task. To begin with, there exist multiple job search websites such as Indeed, Monster, Glassdoor, etc making internship search a challenging task as each site has its ways of filtering. From the same survey, 44% of respondents agreed that they had a difficult time on the search experience. Thus, this is where RadGrad's Internaloha comes in. The main purpose of RadGrad is to raise student's retention and one of the best ways to do so is through participation in internships. Internaloha, a tool created in 2020 to improve RadGrad's internship section [2]. Before the creation, each internship was manually uploaded by either a faculty member or administrator. However, with this limited ability, it did not meet student expectations. Thus, to change the Internaloha scraper was created to scrape internship information from multiple websites with a single command. In other words, this tool reduces the tedious time it takes to type and search through multiple websites. Despite Internaloha being able to extract internships there were some areas of improvements that needed to be made such as the code set up.

## 2 CONCEPT OF SCRAPERS

According to an article from Imperva, a Web Scraper is a process of extracting certain content or information from websites [3]. While the task can be done manually, most of the time it is done automatically through the usage of bots. The basic two parts of a web scraper is web crawler and a web scraper. Zyte explains as a web crawler as a horse and a scraper as a chariot [4]. The general process of how a scraper works is the following:

1. Identify the target website through a URL
2. Collect the list of URL of the pages where the data is extracted from
3. Make a request to these URLs to get the HTML of the page
4. Use locators to find the data in the HTML
5. Write and save the target data in JSON, CSV, or an intended file format

Overall, the scraping tool makes HTTP request to the target website for the data. While the steps seems easy there are couple of challenges associated with maintaining scrapers. For example, the layout for the certain website changes or the website has an

antibot process. According to Web Harvy, the top three uses of web scraping are marketing such as price comparison, real estate, and academic research [5]. From a ranking provided by Hevo Data the top 3 scrapers as of February 2021 are ParseHub, Scrapy, and OctoParse [6].

### 3 CONCEPT OF INTERNALOHA

Initially, University of Hawaii Computer Science undergraduates were able to find internships through the previous version internship page of RadGrad. However, since this version needed each internship to be manually entered. In order to spread internship awareness efficiently, Internaloha was created. Internaloha is an automated scraper system of collecting internships from a variety of job searching websites and having them displayed on the RadGrad internship explorer page. Version 1 uses Javascript and the tool Puppeteer. A node.js library which uses Chromium dev tool, Puppeteer runs headless on default. Each scraper goes to a page and then manually searches through the site for internships. Eventually data is stored inside a JSON file. Finally, the data on the JSON file is manually transferred to RadGrad's internship explorer page.

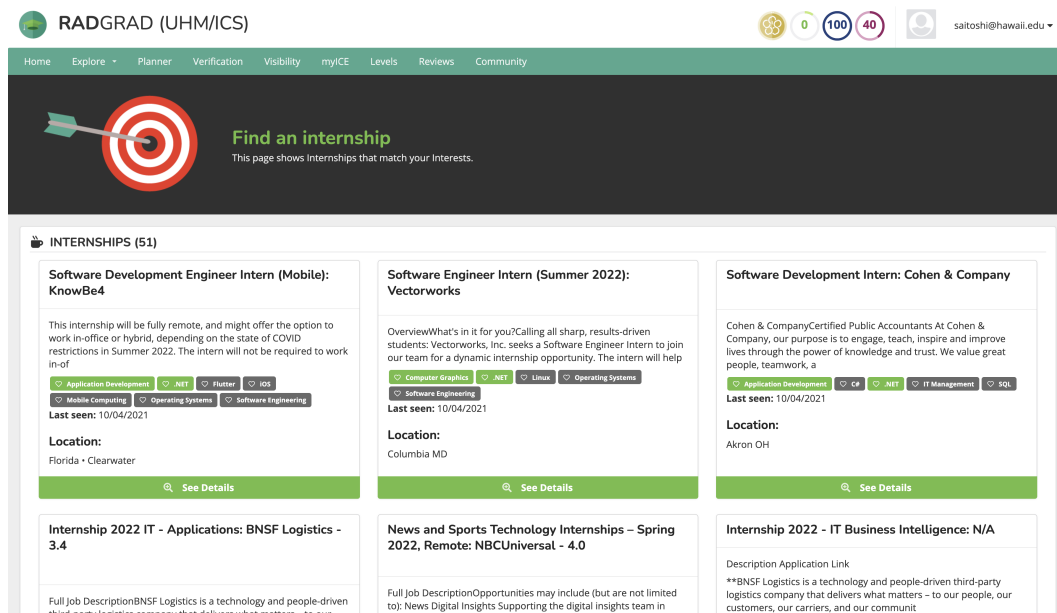


Figure 1: A screenshot of the RadGrad Internship explorer page.

One of the significant characteristics that make Internaloha different from the Top 3 scrapers listed previously is the cost. The three scrapers ParseHub, charges users a fee for usage. However, Internaloha is an open source software and is free for students registered in University of Hawaii at Manoa.

### CHALLENGES WITH INTERNSHIPS

However, why are internships so significant? While it is important to learn in the classroom, some things learned there are not enough to prepare for a career. According to Evan Kilgore, there are four ways that students benefit from participating in internships [7].

- With internships generally being a task specific, it allows the students to gain career development such as workplace knowledge, business etiquette, and strong communication tactics.
- Internships allows for character growth as they shape the student's integrity, commitment, and self motivation.
- With most internships providing preparation for the workforce, it provides most students with a door to opportunity as most employers seek career-ready undergraduate students who are equipped with prior experience.
- Internships provide undergraduate students with a stronger connection giving them an upper hand with references.

With the benefits on the side, why is Internaloha concentrated on internships?

The three key points behind the usage of Internaloha are the following [7]:

1. Searching internships based on your skills and interests
2. Saving time by having the scraper go through dozens of sites
3. Providing better local Hawaii internships

As mentioned before, the idea of “Computer Science” is vague as it comes with multiple subfields such as Data Science, Software Engineering, Game Design, etc. Therefore, finding an internship that matches the student’s interests is essential. Not only does it give them an experience, but it also gives them a better understanding or a preview of that work field. Therefore, on the RadGrad internship explorer, it displays all the internships that best match the specific student’s interests. Furthermore, RadGrad also contains each student’s experience such as class grades, club participation, etc. The internship explorer can further incorporate this into the system giving students a baseline of internship that they qualify for.

### **Personal Experience with Scrapers**

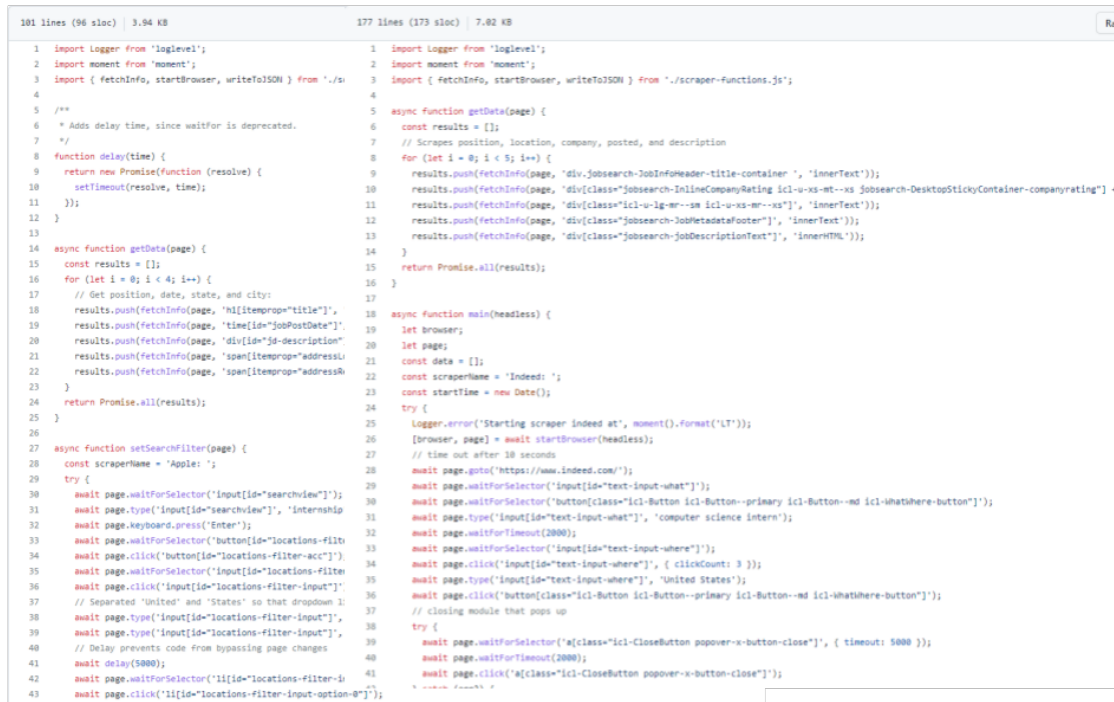
Prior to the participation of this project, my knowledge of scrapers was low. Since I participated in a summer internship at RadGrad, I was aware of the existence of Internaloha and its concept. To get a better understanding of scrapers, I talked to a previous member of the project through Discord. Additionally, to get a better understanding I took a look at a code of version 1 scrapers and made an attempt to translate it into written English. Furthermore, I took part in master scraping of the first couple of weeks to see what is the expected amount of data scraped.

When writing the individual scrapers, I followed the guidelines written by a group of WhereInChicken [8].

1. Be nice to your data sources
2. Do not interrupt the scraper
3. Write your scraper in stages
4. Grab more information you think you will need
5. Sanitize for sanity for others

## 4 ISSUES WITH INTERNALOHA VERSION 1

The purpose of the Capstone Project was to create upgrades on Internaloha. This implies that there were issues existing in the previous version. One of the noticeable issues was that the code written for each scrapers lacked uniformity. Initially, I had a difficult time understanding how scrapers were implemented as there were different structures. For example, Indeed scraper was written using 177 lines of code but Apple scraper used 101 lines of code. Besides comprehension, this made it hard to visualize and compare to see efficiency. In addition to the amount of code, Indeed filtered their search results directly within the main, but Apple had a separate filter function written. This lack of uniformity made it difficult to truly see what was going on and a great amount of time was spent just understanding the code.



```
101 lines (96 sloc) | 3.94 KB      177 lines (173 sloc) | 7.82 KB
1 import Logger from 'loglevel';      1 import Logger from 'loglevel';
2 import moment from 'moment';        2 import moment from 'moment';
3 import { fetchInfo, startBrowser, writeToJson } from './utils'; 3 import { fetchInfo, startBrowser, writeToJson } from './scraper-functions.js';
4                                     4
5 /**                                  5 async function getData(page) {
6 * Adds delay time, since waitfor is deprecated. 6 const results = [];
7 */                                  7 // Scrapes position, location, company, posted, and description
8 function delay(time) {              8 for (let i = 0; i < 5; i++) {
9   return new Promise(function (resolve) { 9 results.push(fetchInfo(page, 'div.jobsearch-JobInfoHeader-title-container ', 'innerText'));
10    setTimeout(resolve, time);         10 results.push(fetchInfo(page, 'div[class="Jobsearch-InlineCompanyRating icl-u-x-s-rt--vs Jobsearch-DesktopStickyContainer-companyrating"] + .
11  });                                  11 results.push(fetchInfo(page, 'div[class="icl-u-ig-mr-on icl-u-s-s-mr--xs"]', 'innerText'));
12 }                                   12 results.push(fetchInfo(page, 'div[class="Jobsearch-JobMetadataFooter"]', 'innerText'));
13                                     13 results.push(fetchInfo(page, 'div[class="Jobsearch-JobDescriptionText"]', 'innerHTML'));
14                                     14 }
15 async function getData(page) {      15 return Promise.all(results);
16 const results = [];                16 }
17 for (let i = 0; i < 4; i++) {        17
18   // Get position, date, state, and city: 18 async function main(headless) {
19   results.push(fetchInfo(page, 'h1[itemprop="title"]', 19 let browser;
20   results.push(fetchInfo(page, 'time[id="JobPostDate"]', 20 let page;
21   results.push(fetchInfo(page, 'div[id="jd-description"]', 21 const data = [];
22   results.push(fetchInfo(page, 'span[itemprop="address"]', 22 const scraperName = 'Indeed';
23   results.push(fetchInfo(page, 'span[itemprop="address"]', 23 const startTime = new Date();
24 }                                   24 try {
25 return Promise.all(results);        25   Logger.error('Starting scraper indeed at', moment().format('LT'));
26 }                                   26 [browser, page] = await startBrowser(headless);
27                                     27 // time out after 38 seconds
28 async function setSearchFilter(page) { 28 await page.goto('https://www.indeed.com/');
29 const scraperName = 'Apple';        29 await page.waitForSelector('input[id="text-input-what"]');
30 try {                               30 await page.waitForSelector('button[class="icl-Button icl-Button--primary icl-Button--md icl-whatsere-button"]');
31   await page.waitForSelector('input[id="searchview"]'); 31 await page.type('input[id="text-input-what"]', 'computer science intern');
32   await page.type('input[id="searchview"]', 'internship'); 32 await page.waitForTimeout(2000);
33   await page.waitForSelector('button[id="locations-filter-acc"]'); 33 await page.waitForSelector('input[id="text-input-where"]');
34   await page.click('button[id="locations-filter-acc"]'); 34 await page.click('input[id="text-input-where"]', { clickCount: 3 });
35   await page.waitForSelector('input[id="locations-filter-input"]'); 35 await page.type('input[id="text-input-where"]', 'United States');
36   await page.click('input[id="locations-filter-input"]'); 36 await page.click('button[class="icl-Button icl-Button--primary icl-Button--md icl-whatsere-button"]');
37   // Separated 'United' and 'States' so that dropdown 1: 37 // closing module that pops up
38   await page.type('input[id="locations-filter-input"]', 38 try {
39   await page.type('input[id="locations-filter-input"]', 39   await page.waitForSelector('a[class="icl-Button button-close"]', { timeout: 5000 });
40   // Delay prevents code from bypassing page changes 40   await page.waitForTimeout(2000);
41   await delay(5000);                41   await page.click('a[class="icl-Button button-close"]');
42   await page.waitForSelector('li[id="locations-filter-input-option-0"]'); 42
43   await page.click('li[id="locations-filter-input-option-0"]');
```

Figure 2: A screenshot of Apple (left) and Indeed (right) scrapers.

Another preexisting issues was segment of code that existed that fell under one of the following: redundant, meaningless, and too long. To begin with there were codes that could have been declared as a super and sub class.

Meaningless codes are the ones that serve no particular purpose. This as a result just take space and as a result, take a longer time running the scraper. For instance, this `getData()` served no purpose as it just went iterated through the same object five times.

Additionally, one of the developer tips for this specific project was to "be kind to future you" [9]. Some of the code found in the previous version lacked comments to provide clarity. An example is in version 1, the code written for Idealist lacked any comments. This made it difficult for other team members, especially new developers, to truly understand some of the code. With websites updating their set up to change their designs, it is important for the developers to comprehend what was going on and see where the error occurred.

```

1  import Logger from 'loglevel';
2  import moment from 'moment';
3  import { startBrowser, fetchInfo, writeToJSON } from './scraper-functions.js';
4
5  async function getlinks(page) {
6    return page.evaluate(
7      () => Array.from(
8        document.querySelectorAll('[data-qa-id="search-result-link"]'),
9        a => a.getAttribute('href'),
10     ),
11   );
12 }
13
14 async function getElements(page) {
15   let hasNext = true;
16   const elements = [];
17   while (hasNext === true) {
18     try {
19       await page.waitForTimeout(1000);
20       getlinks(page).then(links => {
21         elements.push(links);
22       });
23       await page.waitForTimeout(1000);
24       await page.click('button[class="Button__StyledButton-sc-1avp0bd-0 ggDAbQ Pagination__ArrowLink-nuwudv-2 eJsmUe"]:last-child');
25     } catch (e) {
26       hasNext = false;
27       Logger.trace('Reached the end of pages!');
28     }
29   }
30   return elements;
31 }

```

**Figure 3: A screenshot of Idealist Scraper lacking comments.**

## LEGAL ISSUES

Besides the challenges of creating an online scraper the other question One of the main challenges with scrapers is the distinguishing between malicious and well intended bots. According to Imperva there are two key features to distinguish between them [10].

1. Well intended bot are identified with their organization such as Google identifying themselves within the HTTP header. Additionally, they abide by a site's robot txt file which lists those pages a bot is permitted to access. Examples of these scrapers include:

- Search engine bots crawling a site
- Price comparison site bots

2. Malicious bots on the other hand, impersonate legitimate traffic by creating false HTTP user agents. Additionally, they go through the website regardless of what the site operator allows.

Examples of malicious usage of scrapers are the following:

- Price scraping with the purpose to undercut rivals and boost sales
- Content scraping with the purpose to steal information from databases

Thus, legal issues with the usage of scrapers occurs. In the past there have been several cases on the usage of scrapers. For example, in 2001 a travel agency company sued a rival company for scraping its website for prices [11]. The judge of that specific case ruled against the travel company stating that not being welcomed by the owner of the site is not a sufficient reason for unauthorized access. Most of the time the court's ruling was that stating that scrapers are not welcome in term and usage conditions was not sufficient. Furthermore, in 2019 US Court of Appeal ruled that, "showed that any data that is publicly

available and not copyrighted is fair game for web crawlers” [12].

However, despite the previous ruling, there have been situations where scraping was considered unlawful. For example, Craigslist sued a company Instamotor for web scraping and the company was ordered to pay \$31 million to Craigslist [13]. Thus, the borderline of legal and illegal scraping is complicated. The blog provides a guideline of legal and illegal scraping activities. [14]

1. Under the Computer Fraud and Abuse Act (CFAA) as long as data is not accessed in an abusive manner or for a commercial purpose then it is legally safe
2. If data is not reused for business purposes, then it is legally safe.
3. If the scraper does not enter prohibited areas or enter in a harmful way then you are legally safe.
4. If the scraper abides rules written in the Robot.txt then it is legal.
5. Since the website is made for human usage purpose, the crawl rate must remain within a reasonable range of a person.
6. Check with the Terms of Service (ToS) and seek in permissions if there are any mentions about web scraping within the ToS.
7. Define a user agent that clearly describes the purpose of the scraper and include contact information within the user agent.

## **INTERNALOHA AND LEGAL ISSUES**

Thus, how does Internaloha address the many legal issues found within the usage of scrapers? Below is a summary of rules on how Internaloha scrapers are created and used [15].

1. Websites that do not require logging in to access information such as Glassdoor, Monster, Apple, etc. are protected under “fair use”.
2. To avoid violating the “Trespass to Chattels” law, each website is scraped at a moderate “humanly” rate and not frequently scrapping the site even if the scraper is under development.
3. If the specific website’s ToS prohibits the usage of web scraping then explicit permission will be directly obtained.

Furthermore, under the CFAA, data for Internaloha is used to increase awareness of different internship opportunities. Under that description, Internaloha is legally safe as it is not used for a harmful purpose.

## 5 UPGRADING INTERNALOHA

With supervision from Professor Phillip Johnson, Internaloha was upgraded or adjusted.

- **Usage of TypeScript rather than Javascript**

One of the significant changes in Internaloha version 2 is that it was written using TypeScript, a modern-age Javascript development language that provides optional static typing, classes, and interface [16]. The advantage of using TypeScript over Javascript is the following [17]:

- TypeScript points out errors during the development stage which reduces the chances of having errors during run time
- TypeScript comes with a feature called Static typing which allows for checking type correctness at compile time
- Additionally, Typescript is believed to have better code organizing and object-arranged programming procedures

- **Usage of superclass that provides common structure**

One of the initial difficulties with the original design was that there was a variety of implementation making it difficult for a beginner to comprehend. In addition, there was little repetition such as functions that had the same code but different names. Thus superclasses were implemented as they came with a couple of advantages [18].

- Reusing code  
This will prevent the redundancy of the same code being implemented and save time during development of scrapers.
- Specialization  
In a subclass, a user can add new methods to handle cases that the superclass does not handle and also add new data items that the superclass does not need.
- Change in action  
A subclass can override a method that it inherits from a superclass.

Furthermore, an uniform set up reduces the chances for mistakes or errors to occur.

```
async scrape() {
  try {
    await this.launch();
    await this.login();
    await this.generateListings();
    await this.processListings();
  } catch (error) {
    const message = error['message'];
    this.errorMessages.push(message);
    this.log.error(message);
  } finally {
    await this.close();
    await this.writeListings();
    await this.writeStatistics();
  }
}
```

### Standard workflow of the new scrapers.

- **Usage of await super.getValues() or super.getValue()**

To further add consistency, super.getValues()/getValue() was created to extract information of positions from a certain website. In version 1, there exist multiple ways to extract information but overall, they behaved similarly. However, having one function makes tasks easier to perform and analyze when there is an error.

```
urls = urls.concat(await super.getValues('a[class="jobLink"]', 'href'));
```

### **A sample usage where getValues is being used to extract URLs**

- **Structural Support for multiple disciplines such as Computer Science and Computer Engineering**

With the default discipline being "Computer Science", the new version of Internaloha provides a discipline parameter with the other being "computer engineering" [19]. Using the discipline parameter the json files containing the scraping results are organized accordingly into a directory of each discipline.

- **Automatic generation of scraper statistics**

Each time a scraper is called, a json file is automatically written containing the time stamp of YYYY-MM-DD of the most recent run of the specific scraper. In the previous version, this was done by writing the code for each scraper. However, with this automation it reduces the chances for error and decrease the time taking to write a scraper.

- **Usage of effective comments.**

In version 1 there was few to none comments on the previous scrapers. This made it difficult for other or new members to comprehend what the code was doing. Even if there was comments it did not provide enough guidance. Thus, comments or debug statements were placed for the usefulness.

```
//retrieve the urls of page 1
urls = urls.concat(await super.getValues('a[class="jobLink"]', 'href'));
// retrieve the total number of pages
let key = await super.getValue('div[class="cell middle d-none d-md-block py-sm"]',
'innerText');
```

### **Comments found in Glassdoor scraper briefly explaining the process.**

- **Avoiding the usage of try-catch method for normal control flow**

Where version 1 had multiple usage of try-catch method, there were situations where the usage was not appropriate such as the following code

```
try {
  // Click the "Load More" button
  await this.page.click('.load_more_jobs');
} catch (err) {
  this.log.debug('--- All jobs are Listed, no "Load More" button --- ');
}
```

Since it is not an error to not have a "load more button", the usage of try-catch method is not appropriate. Rather, the usage of "if-else" or "while" is preferred. For instance, the try catch method written above could be rewritten as the following.

```
const loadJobsSelector = 'load_more_jobs';
if (await super.selectorExists(loadJobsSelector) {
  await this.page.click(`.${loadJobsSelector}`);
}
```

- **Usage of commander for top-level CLI processing.**



## 6 CREATION OF INTERNALOHA V2

To create each scraper, a basic Typescript template was given by Professor Johnson [20].

```
1 import { Scraper } from '../Scraper';
2
3 const prefix = require('loglevel-plugin-prefix');
4
5 export class TemplateScraper extends Scraper {
6   constructor() {
7     super({ name: 'template', url: 'https://testscrapersite.com' });
8   }
9
10  async launch() {
11    await super.launch();
12    prefix.apply(this.log, { nameFormatter: () => this.name.toUpperCase() });
13    this.log.warn("Launching ${this.name.toUpperCase()} scraper");
14  }
15
16  async login() {
17    await super.login();
18    // If you need to login, put that code here.
19  }
20
21  async generateListings() {
22    await super.generateListings();
23    // here is where you traverse the site and populate your this.Listings field with the listings.
24  }
25
26  async processListings() {
27    await super.processListings();
28    // here is where you do any additional processing on the raw data now available in the this.listings #field.
29  }
30
31 }
```

**Figure 4: The template of the Scraper version 2.**

The template contains the launch, login, generateListings, and processListings class. However, for most scrapers, launch() and generateListings() were frequently used. Below is a brief explanation on what each of these sub-classes did.

- launch()  
This is where the scraper is launched as it turns on Puppeteer and notifies the user which scraper they have selected to run.
- login()  
This is where the task of login occurs where the account information is extracted from the config.json file and then the scraper types it in accordingly depending on the website.
- generateListings()  
This is where the actual task of extracting the job information occurs by having the scraper go to the website URL and then follow the code written within as the steps vary from the website.  
After the information is extracted it is added to the JSON file of the appropriate website using addListings().
- processListings()  
This is where the listings that are generated are checked and to see whether the listing is in an appropriate format. If it is not then it formats them that way it does.

For the following project, I have taken part in writing Zip Recruiter, Monster, Glassdoor, and StackOverFlow scraper. Each scraper was named with the following pattern of Scraper.[nameOfSite].ts. For example, ZipRecruiter was labeled as Scraper.ziprecruiter.ts. All four of these websites did not require an account to search internships.

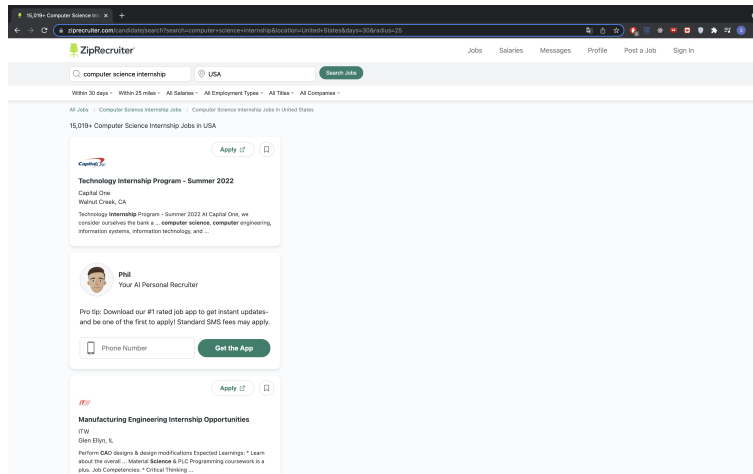
The basic steps for how each scraper was developed was the following:

1. Go to the actual website and see how the HTML and div set up works.

2. Transfer the given webscraper template and then fill in the code appropriately.
3. Update the main.ts file by adding the new scraper to the CLI.
4. Do a test run of the scraper to see if it writes the listings correctly.
5. Compare the number of internships scraped with the number listed on the website to see whether it operated correctly.
6. Finally, did a clean up on the code by adding comments or deleting unnecessary code that provided on meaning to the scraper.

Below is a further detail on the different scrapers I have written for this project.

## Creation of Zip Recruiter



**Figure 5 : A screenshot of ZipRecruiter page.**

ZipRecruiter, was the first scraper I wrote with the new format. Initially, my process was to translate the code written in version 1 of the same ZipRecruiter and then run it. However, after the first trial I noticed that that was not an option. To begin with, the HTML code that was written in version 1 did not match the current ZipRecruiter set up. Furthermore, there was a for loop within the code that did not make any sense as iterated within the same item five times.

```
async function getData(page) {
  const results = [];
  for (let i = 0; i < 5; i++) {
    results.push(fetchInfo(page, '.job_title', 'innerText'));
    results.push(fetchInfo(page, '.hiring_company_text.t_company_name', 'innerText'));
    results.push(fetchInfo(page, 'span[data-name="address"]', 'innerText'));
    results.push(fetchInfo(page, '.jobDescriptionSection', 'innerHTML'));
    results.push(fetchInfo(page, '.job_more span[class="data"]', 'innerText'));
  }
  return Promise.all(results);
}
```

### **The for loop that iterated through the same job information 5 times.**

Compared to the other scrapers, writing ZipRecruiter took a longer time to write. Furthermore, the difficulty here was having the scraper keep scrolling through the page until there are no available jobs. With the new format, ZipRecruiter was written using 48 lines of code where version 1 of the same website required 130 lines of code. Overall, working on ZipRecruiter scraper provided a basic understanding on creating scrapers on the new format.

```

130 lines (127 sloc) | 5.44 KB | 48 lines (39 sloc) | 1.93 KB
1 import logger from 'loglevel'; 1 import { Listing } from './Listing';
2 import moment from 'moment'; 2 import { Scraper } from './Scraper';
3 import { fetchInfo, startBrowse 3 import _ from 'underscore';
4 4
5 async function getData(page) 5 const prefix = require('loglevel-plugin-prefix');
6 const results = []; 6 export class ZipRecruiterScraper extends Scraper {
7 for (let i = 0; i < 5; i++) 7 constructor() {
8 results.push(fetchInfo(pa 8 super({ name: 'ziprecruiter', url: 'https://www.ziprecruiter.com/candidate/search?search=computer-science-inter
9 results.push(fetchInfo(pa 9 }
10 results.push(fetchInfo(pa 10 }
11 results.push(fetchInfo(pa 11 async launch() {
12 results.push(fetchInfo(pa 12 await super.launch();
13 } 13 prefix.apply(this.log, { nameFormatter: () => this.name.toUpperCase() });
14 return Promise.all(results); 14 this.log.warn('launching ${this.name.toUpperCase()} scraper');
15 } 15 }
16 16
17 async function main(headless) 17 async generateListings() {
18 let browser; 18 super.generateListings();
19 let page; 19 await this.page.goto(this.url);
20 const data = []; 20 // Interestingly, we must request the page again, otherwise only 20 listings. Maybe to get past popup?
21 const startTime = new Date(); 21 await this.page.goto(this.url);
22 const scraperName = 'Zip'; 22
23 try { 23 // Autoscroll to retrieve and display all of the listings on the page.
24 logger.error('Starting sc 24 await this.page.click('.load_more_jobs');
25 [browser, page] = await s 25 await super.autoScroll();
26 // await page.goto('https 26
27 await page.goto('https:// 27
28 await page.waitForSelecto 28 let url = await super.getValues('.job_link_t_job_link', 'href');
29 await page.waitForSelecto 29 url = _.uniq(url);
30 const searchQuery = 'comp 30 this.log.info('found ${url.length} listings');
31 logger.info('Inputting se 31
32 await page.type('input[id 32 const positions = await super.getValues('h[class="job_title"]', 'innerText');
33 // eslint-disable-next-li 33 const descriptions = await super.getValues('div[class="job_description_container"]', 'innerText');
34 await page.$eval('input[id 34 const companies = await super.getValues('a[class="t_org_link name"]', 'innerText');
35 await page.click('button. 35 const locationStrings = await super.getValues('a[class="t_location_link location"]', 'innerText');
36 36 const locations = [];
37 for (let i = 0; i < locationStrings.length; i++) {
38 const locPair = locationStrings[i].split(', ');
39 locations.push({ city: locPair[0], state: locPair[1], country: 'USA' });
40 }

```

Figure 6: Comparison of the Version 1 (left) and Version 2 (right).

## Development of Monster

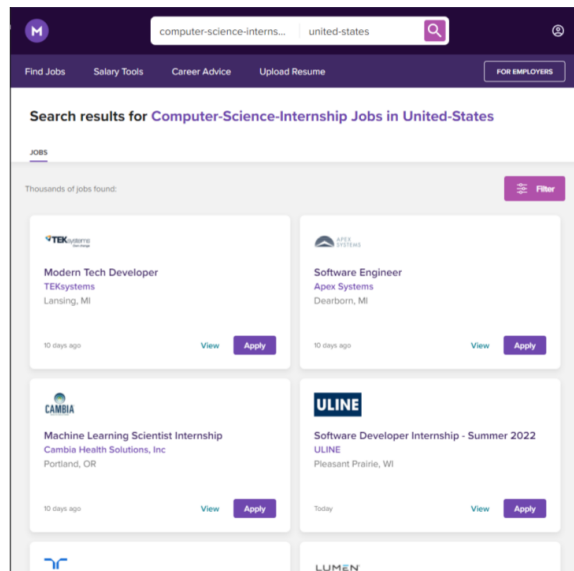


Figure 7: The sample page for Monster internship listings.

For the second scraper, I wrote a scraper for Monster. The challenging part of Monster was to have it iterate it through multiple pages through a tab. Monster was similar to ZipRecruiter where the as long as there are information, the page can keep scrolling down through the list of internships. Unlike ZipRecruiter or other job websites, there was no div section name to create a while loop for. However, when looking at the source code using Google Chrome, there existed a div button where it stored the next section of listings. Thus, the scraper will keep extracting URL as long as the button exists.

```

//while next link exists
while (await super.selectorExists(nextLink)) {
await this.page.click(nextPage);

```

The while loop that will be performed as long as the button for the next page exists.

Furthermore, the Monster page is set up where all of the necessary information is spread out. Thus, the scraper collects the URL, position title, and company name from the main page. Then, the scraper goes to the individual URL of that specific position title and extracts the description of the job and the job location. While the job location was written in the main page, it was simpler to extract it from the individual position page for an easier to readable code.

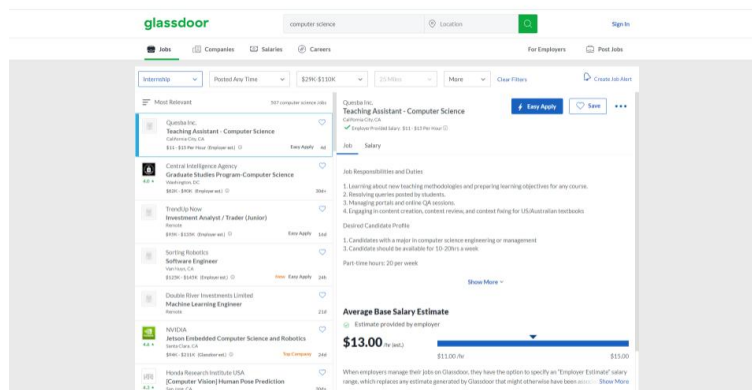
## Development of Glassdoor

After writing two scrapers I was more confident in writing scrapers. As my third scraper, I selected to write Glassdoor. The HTML setup for Glassdoor was like Monster. However, the main difference was that extracting key information besides URL was easier to do by going to the URL page itself.

```
// go to the page and retrieve its relative information
await super.goto(urls[k]);
positions.push(await super.getValue('div[class="css-17x2pwle11nt52q6"]', 'innerText'));
```

**The scraper goes to the specified URL and extracts information from there.**

The difficulty for Glassdoor was that this scraper took a while to run. In the previous version it took an estimate of 1 to 2 hours to completely scrape. However, to abide with the legal issues, the scraper was written by handling a small amount of pages rather than the whole.



**Figure 8: A listing of Glassdoor.**

Additionally, there were couple of duplicates found within Glassdoor website that they had to be filtered out using the underscore's unique function.

```
// there can be lots of duplicates in this list, so get rid of them.
urls = _.uniq(urls);
```

**The code used to filter through the available URL but reducing any duplicates.**

## Development of StackOverFlow

When writing StackOverFlow scrapers, I was at the confident level where I can do scrapers without any issues. That was the case as it did not take me any long time compared to the previous three scrapers I wrote. However, once testing I noticed that the scraper scrapped more jobs than that was listed. For instance, the page itself listed only five internships but it wrote a listing of 25 jobs on the JSON file.

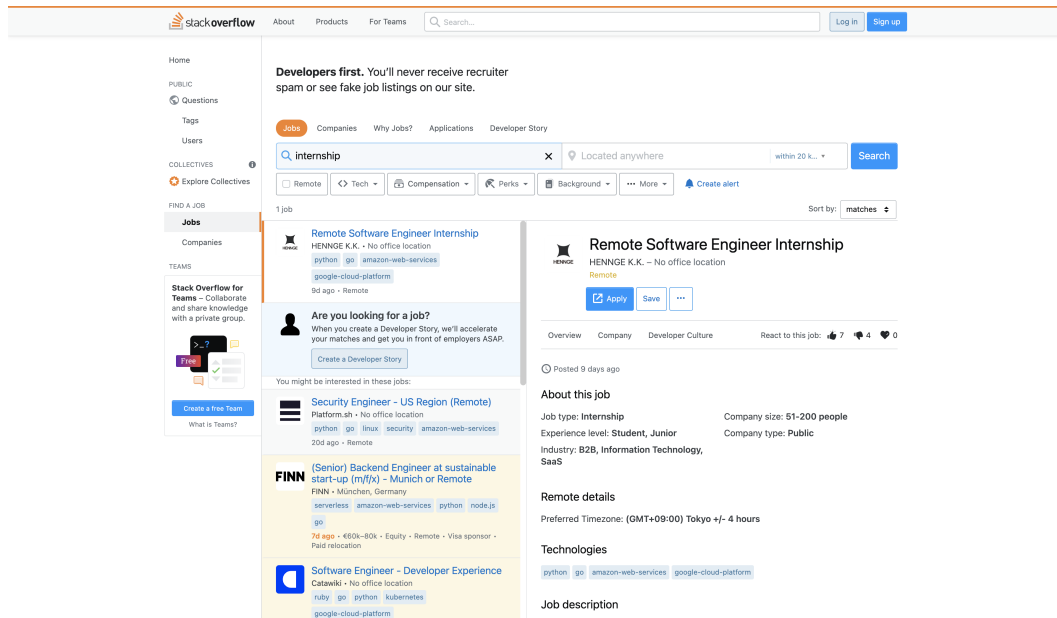


Figure 9 : A page when users search through internships on StackOverFlow.

The issue was that the scrapper would extract information of positions listed on the “You might be interested in these jobs:” column. To fix this, I have scrapped the number of internships listed. Then created a for loop such that it does not exceed the actual number of internships.

```
// extract the number of internships currently available const
key = await super.getValue('span[class="description fc-light fs-body1"]', 'innerText');
// store it called jobNumber
const jobNumber = key.match(/\d+/gm);
this.log.debug(`Amount of internships available: \n${jobNumber[0]}`);
for (let i = 0; i < jobNumber ; i++) {
```

The code written to not exceed the actual count of internships available.

## STATUS OF INTERNALOHA

Besides the four websites, Internaloha provides Internship listing from the other websites including: Apple, Chegg, Cisco, Glassdoor, NSF, SimplyHired, StackOverFlow, and Ziprecruiter. All of the free websites that do not require log in account are supported. However, unlike version 1 there is no command to run all of them at once. Multiple scraper command resulted an error as puppeteer is not "thread" safe [21]. Despite that a multiple scraper call is theoretically possible through the usage of opening multiple terminal tabs.

```
npm run scrape -- -s nsf
```

### The command to run National Science Foundation scraper.



**Figure 10: A run of multiple scrapers using multiple OS tabs.**

Each scraper will take different amount of time to scrape all the necessary information. For example, the scraper for National Science Foundation takes less than a minute, but Glassdoor takes a couple of hours. The time each scraper takes depends on the amount of available internships.

## CONCLUSION

Overall, the main purpose of this project was to revamp Internaloha. This project was interesting as it was a different project from the ones I have previously participated in as this did not start from zero. Instead, this project had a basis and I was working on a continuation. Furthermore, this project did give me a better understanding on the significance of scrapers such as from a legal perspective.

## Future Plans With Internaloha

- **Support for additional websites such as ones that require an account**  
Despite Internaloha having a new version, it still has space for future upgrades that are beneficial for students. Currently, the scrapers supported are the websites that require no account set up. By receiving consent from other websites requiring account information, Internaloha can start supporting websites such as Angellist or Student Opportunity Center.
- **Support for location filter**  
Currently, the Scrapers provide information of internships based on United States. In the future, Internaloha should provide a filter based on location such as by states.
- **Finding a way to reduce the time for certain scrapers.**  
While Chegg and Glassdoor scraper runs, they take an estimate of one to two hours for completion. Thus, it would be great for Internaloha to find a way to speed up certain scrapers that way information can be grabbed with a reasonable amount of time.

- **Automatic connection with RadGrad and Internaloha**

Additionally, data scraped using Internaloha are manually sent to the RadGrad's internship explorer page. In the future, Internaloha should be able to automatically send data to RadGrad every time a scraper runs. This way students who are using RadGrad are up to date with the possible internships they can participate in.

- **Using Secret Agent along with Puppeteer**

Currently, the version 2 scrapers are processing the information by this.page fields. However, with the usage of Secret Agent, an agent can refer all the processing.

## REFERENCES

- [1] Johnson, P., 2021. “Motivation | InternAloha,” [internaloha.github.io](https://internaloha.github.io). <https://internaloha.github.io/documentation/docs/overview/motivation> (accessed Nov. 29, 2021).
- [2] Johnson, P., 2021. InternAloha Needs Assessment | InternAloha. [online] [Internaloha.github.io](https://internaloha.github.io). Available at: <https://internaloha.github.io/documentation/docs/overview/needs-assessment/> [Accessed 21 November 2021].
- [3] “What Is Scraping | About Price & Web Scraping Tools | Imperva,” Learning Center. <https://www.imperva.com/learn/application-security/web-scraping-attack/>.
- [4] “What is web scraping and how does it work?,” Zyte (formerly Scrapinghub) #1 Web Scraping Service, 03-Nov-2021. [Online]. Available: <https://www.zyte.com/learn/what-is-web-scraping/>. [Accessed: 04-Dec-2021].
- [5] SysNucleus, “WebHarvy Web Scraper,” [www.webharvy.com](http://www.webharvy.com). <https://www.webharvy.com/articles/web-scraper-use-cases.html>.
- [6] “Top 5 Web Scraping Tools Comparison,” [www.octoparse.com](http://www.octoparse.com). <https://www.octoparse.com/blog/top-5-web-scraping-tools-comparison>.
- [7] E. Kilgore, “The Importance of Internships in College,” Colleges of Distinction. <https://collegesofdistinction.com/advice/the-importance-of-internships-in-college/>.
- [8] “Some guidelines for writing web scrapers,” Wherein The Chicken, 02-Dec-2021. [Online]. Available: <https://cushychicken.github.io/rules-for-web-scrapers/>. [Accessed: 04-Dec-2021].
- [9] “InternAloha | InternAloha,” [internaloha.github.io](https://internaloha.github.io). <https://internaloha.github.io/documentation/> [Accessed Nov. 29, 2021].
- [10] “Developer Tips | InternAloha,” [internaloha.github.io](https://internaloha.github.io). <https://internaloha.github.io/documentation/docs/developers/tips>.
- [11] “What Is Scraping | About Price & Web Scraping Tools | Imperva,” Learning Center. <https://www.imperva.com/learn/application-security/web-scraping-attack/>.
- [12] “Is Web Scraping Illegal? Depends on What the Meaning of the Word Is | Imperva,” Blog, Sep. 17, 2018. <https://www.imperva.com/blog/is-web-scraping-illegal/>.
- [13] T. Waterman, “Web scraping is now legal,” Medium, Apr. 27, 2020. <https://medium.com/@tjwaterman99/web-scraping-is-now-legal-6bf0e5730a78> (accessed Nov. 29, 2021).
- [14] J. Neuburger, “Ending Data Scraping Dispute, Craigslist Reaches \$31M Settlement with Instamotor,” The National Law Review. <https://www.natlawreview.com/article/ending-data-scraping-dispute-craigslist-reaches-31m-settlement-instamotor>.
- [15] “Is Web Scraping Legal? : The Definitive Guide [2020 update] – ProWebScraper,” Prowebscraper.com, 2020. [https://prowebscraper.com/blog/is-web-scraping-legal/#why\\_does\\_web\\_scraping\\_often\\_appear\\_offensive](https://prowebscraper.com/blog/is-web-scraping-legal/#why_does_web_scraping_often_appear_offensive) (accessed Nov. 29, 2021).
- [16] “Legal Issues | InternAloha,” [internaloha.github.io](https://internaloha.github.io). <https://internaloha.github.io/documentation/docs/developers/legal> (accessed Nov. 29, 2021).
- [17] “Typescript vs JavaScript: What’s the Difference?,” [www.guru99.com](http://www.guru99.com). <https://www.guru99.com/typescript-vs-javascript>.
- [18] “Difference between TypeScript and JavaScript,” GeeksforGeeks, Jun. 14, 2018. <https://www.geeksforgeeks.org/difference-between-typescript-and-javascript>.
- [19] “Defining a subclass,” [www.ibm.com](http://www.ibm.com). <https://www.ibm.com/docs/en/cobol-zos/6.2?topic=programs-defining-subclass> (accessed Nov. 29, 2021).
- [20] “Scrapers V2,” GitHub, Nov. 22, 2021. <https://github.com/internaloha/scrapers/blob/main/scrapers/Scraper.template.ts> (accessed Nov. 29, 2021).
- [21] “Invocation: InternAloha,” InternAloha Blog RSS. [Online]. Available: <https://internaloha.github.io/documentation/docs/developers/invocation>. [Accessed: 04-Dec-2021].