Internship Web Crawler

University of California, Riverside

Github Repo Link: https://github.com/CS-UCR/final-project-nougat

Authors

Name	Github Profile
Sydney Pun	https://github.com/spun001
Neal Goyal	https://github.com/nealgoyal
Emma Rivera	https://github.com/emmabernicerivera

Table of Contents

Overview

This internship web crawler seeks to simplify the internship searching process. With this web application, gone are the days in which students spend hours scouring the internet for internships. Instead, the searching process will be expedited because our application allows the user to simply upload their PDF resume and a ranked list of internship opportunities will be a click away.

Detailed Description

Crawling

We will build a web crawler in Python. We plan to crawl company pages in search of their internship opportunity listings. The libraries that we have considered are Scrapy and BeautifulSoup. Although Scrapy is a powerful web framework that allows for the extraction, processing, and storing of web data, it has a larger learning curve than BeautifulSoup which in comparison, is slower unless multiprocessing is used. As a result, we will consider testing both libraries to see which one is the best fit for our crawler.

Retrieval

We reviewed the TA's documentation on both indexing libraries. We also saw his examples from discussion on using Lucene for his project on CA DMV. We decided to go with Lucene because of the TA's guidance on the benefits of Lucene. We will be using a Java Lucene wrapper called PyLucene to index our internship web crawler. Our MVP will be querying major, GPA, skills to match opportunities. The trickier part will be to evaluate experience level. In our second iteration, we will increase our scope to parse through experience and curate a more personalized internship list matching the individual's experience level.

Extension

We will extend this web crawler by transforming it into a web application. Our web application will be developed with the web framework Django because it is a Python framework that allows for the integration of retrieval tools such as ElasticSearch and Lucene. With this web application, a front-end user-interface will allow the user to upload their PDF resume. We will then proceed to use page rank to return a ranked list of internship opportunities. The page rank algorithm will be constructed by identifying the similarities between the queried GPA, skill sets, major, etc. from the resume and the indexed web pages that were crawled. In addition, the application deadline for the internship opportunity will also be considered in the ranking process.

Contributions

Emma - Crawler

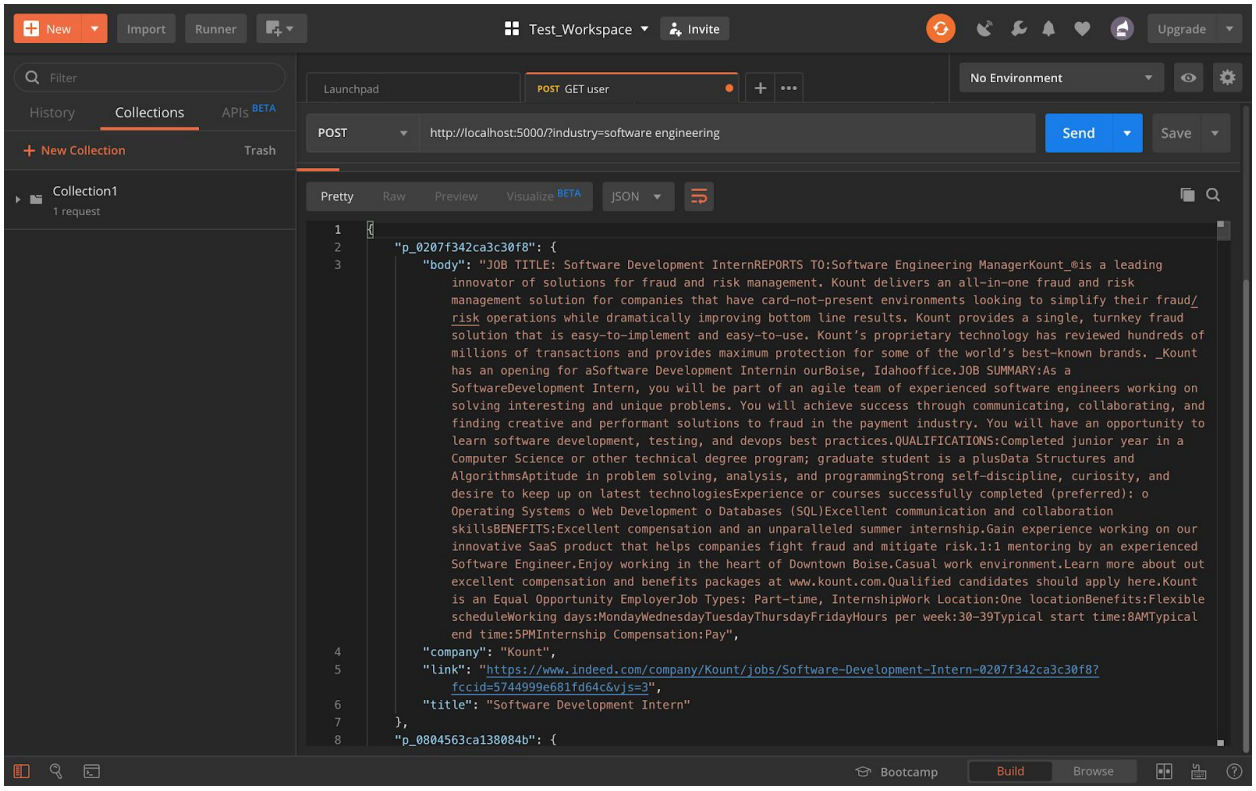
Description and Overview

I am using BeautifulSoup to crawl Indeed.com for internships. We allow the user to pass an industry to the API which then constructs the URL accordingly and parses the returned pages. If the industry provided is invalid then we return invalid industry. We are able to detect this because Indeed.com shows a “bad_query” if the industry is an invalid one.

I used Flask to build the API so we could pass an industry to query.

The crawler parses, after having constructed the URL from the given industry, and collects data from each job returned for the first 5 pages. It collects the job_id, job_title, company, url, and body/description of the job. Each of these elements is stored in a hashmap with the job_id being the key and the rest of the data, values.

Example returned result shown on Postman using GET request for software engineering roles:



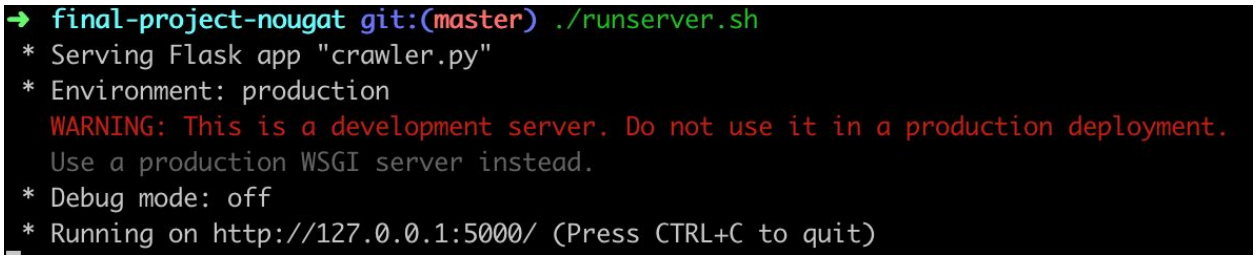
Limitations

The crawler itself takes a while to run. This is because we are only making one request. If we were to use Scrapy or multiprocessing, a python package that supports spawning multiple processes at once using an API, then we would have been able to achieve a faster crawler.

Another limitation is that we are only limited to the industries and internships returned and supported by Indeed.com. If we reach an invalid industry then we return "Invalid Industry". This, however, doesn't mean that this is an invalid industry but rather one that we cannot query on Indeed.

How to Run the Server

To run the server I made a bash script called runserver.sh. Simply run ./runserver.sh in the terminal to start the server. Make sure to set executable permissions for the file using: chmod +x runserver.sh.



Neal - Indexer

Description

I am using Elasticsearch to index the internships json file. The json file contains the company name, job title, link, and description. I am creating a collection and then using Kibana to visualize the indexes.

Alongside building the retrieval model, I built the connector that allowed the crawler and resume parser to communicate with one another. I am using TF-IDF on the descriptions to create weights for the terms. Then, I will be using the terms from query_skills.txt file to skill match. The highest frequency the term pops up will give the job a higher rating. The top 5 highest ratings will be displayed on the front end.

How To Run The Indexer

```
# Running indexer
python indexer.py

# Running elasticsearch
python elasticapp.py
```

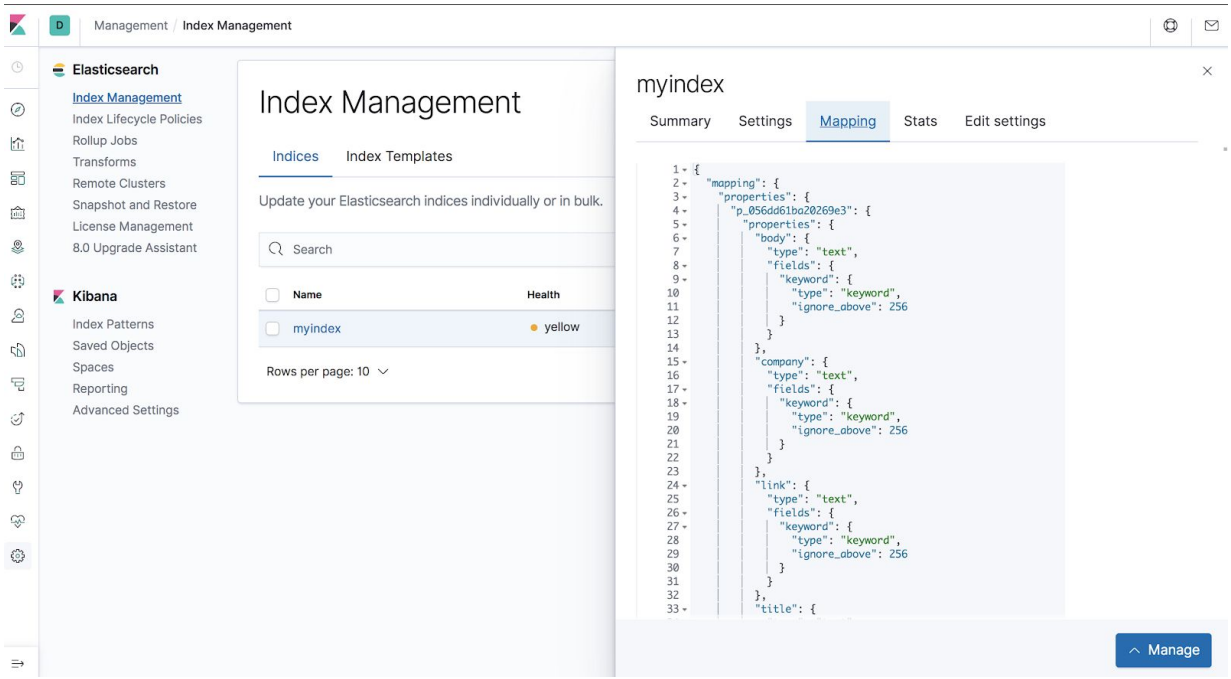
Limitations

I found that it was very difficult to get both entities to communicate with one another via indexing with Elasticsearch. This mainly came with difficulty understanding how to use Elasticsearch for the first time. Therefore for completion, I used a data structure that queries at ~ 4 seconds.

Usage

Elasticsearch running

Index shown on Kibana



Sydney - Extension

How To Run The Web Application:

```
cd web_application
mkvirtualenv proj_env
workon proj_env
pip install -r requirements.txt
python manage.py makemigrations
python manage.py migrate
python manage.py runserver
View project on: http://127.0.0.1:8000/
```

How To Run The Resume Parser:

```
cd resume_parser
python resume_parser.py
```

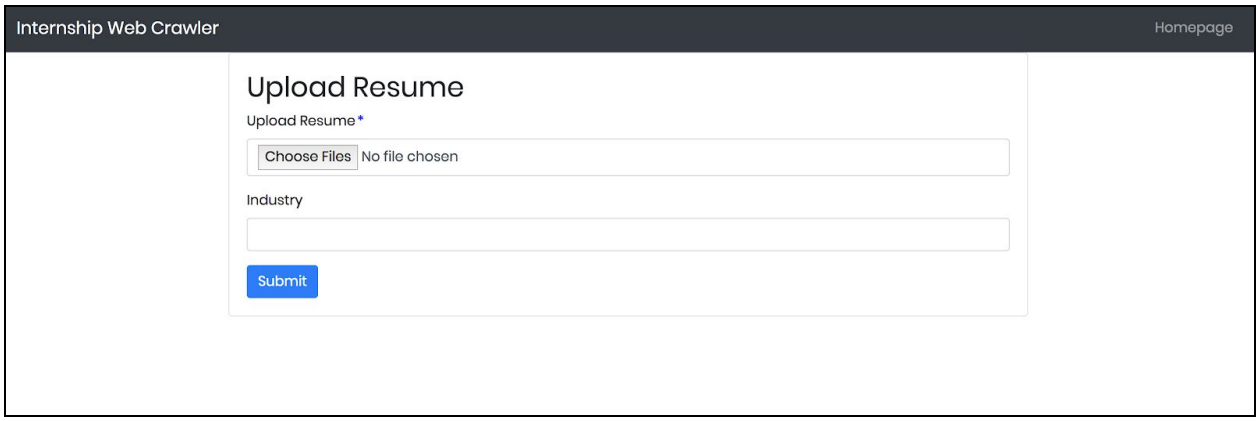
The web crawler is extended by transforming it into a web application. Our web application is developed in Django because it is a Python framework that allows for the integration of retrieval tools such as ElasticSearch and Lucene. With this web application, a front-end user interface will allow the user to upload their resume and specify the industry.

I used the "pyresparser" library which was developed by Mr. Omkar Pathak to parse a PDF resume. As indicated by the resume.txt outputs and the query_skills.txt outputs, the parser does work. However, an error in which I encountered is that the file paths have to be hard-coded in order for the library to recognize which file to parse. As a result, in order to use this resume parser, we will need to specify the file path and name of the file, and then run the script manually by entering the path and running "python resume_parser.py" to extract resume contents to begin the indexing. This is definitely something I will consider fixing by allowing for any PDF to be parsed and for the parser to run automatically with a task manager such as Celery in Django if I decide to build upon this project in the future.

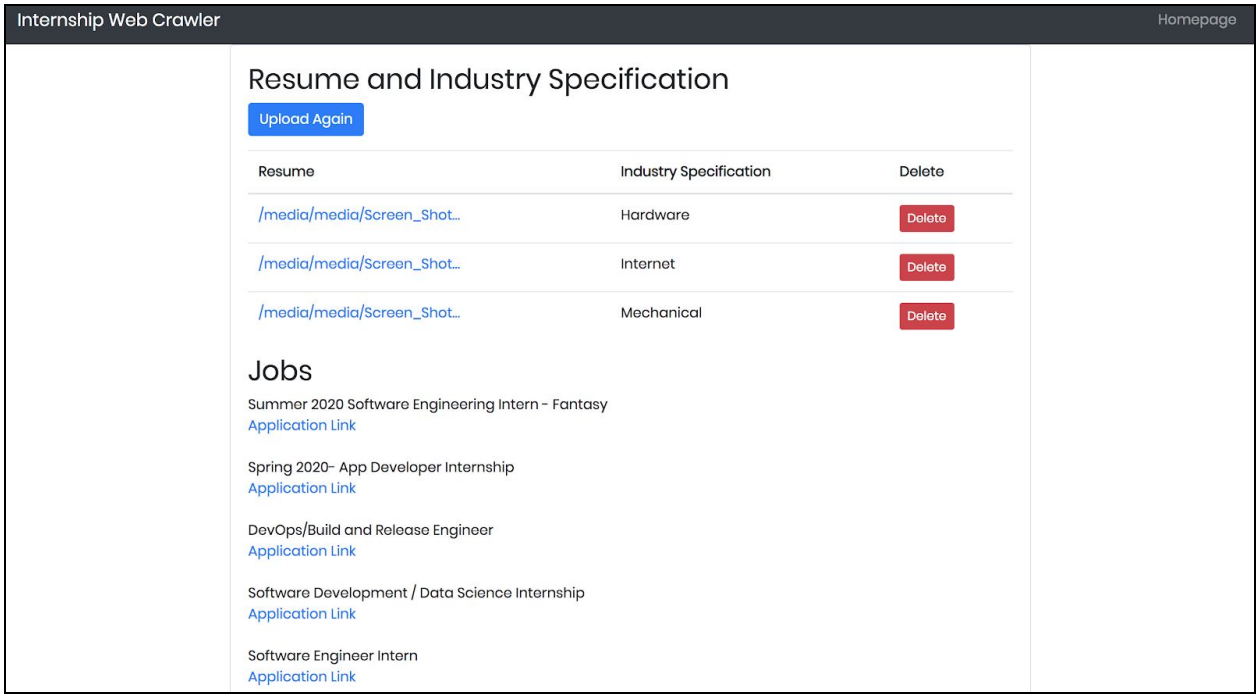
Another limitation is that the industry input is not a drop-down menu and as a result, there is no restriction as to what the user can enter.

Final Design

Home Page



Results Page



Reflection

Summary: This project was complex, but had some great learning outcomes.

Everyone in the group tackled a new technology during this project. A lot of firsts! Emma creating an API and using BeautifulSoup. Neal using Elasticsearch and Kibana. Sydney using a resume parser. This was a great opportunity to challenge ourselves by tackling a complex problem most students face - finding the perfect job tailored toward them.

Initially, we thought this project was going to run smoothly. The idea seemed straightforward and we had the entire plan laid out. However, we quickly faced many hurdles from the start.

Some challenges included:

- Issues with latency with crawling Indeed.com using third party library BeautifulSoup
- Many job sites block web crawlers from scraping their website (i.e. Chegg Internships)
- Front end had several issues for running on other team members computers due to environmental issues, which caused a delay in testing
- Balancing other CS finals (i.e. CS 161, CS 150) and this project was tough because everyone had finals at different times. Budgeting time toward this project and meeting up became increasingly difficult as Week 10 arrived. This led us to spending multiple all nighters debugging code.

If we were given more time, we would:

- Create an ML Model using sentiment analysis to take the entire contents of the resume and get a more accurate, tailored job results. Right now, we are focusing on skills for our MVP.
- We would improve the UI by providing the user with a dropdown menu for the different industries.
- Improve team management. We were mainly working offline. With additional weeks, we could plan weekly stand ups to go over the components we worked on and track spill over easily and reassign work.

We found the experience rewarding and are excited to demo our final product on Friday to everyone!