Final Project Report

CS 172: Information Retrieval

Rogelio Macedo

Instructor: Mariam Salloum

Alexander Yee

Team: LightSABR

Shiyao Feng

Brett McCausland

18-March-2019

Search Engine

In completing this project, I consulted:

- Mariam Salloum during lectures/office hours
- Class channel on Campuswire

All the important code is original. Unimportant subroutines that are not completely original are sourced:

- https://sundog-education.com/elasticsearch/
- https://www.udemy.com/elasticsearch-6-and-elastic-stack-in-depth-and-hands-on/
- https://www.elastic.co/blog/how-to-find-and-remove-duplicate-documents-in-elasticsearch
- https://www.elastic.co/guide/en/elasticsearch/client/javascript-api/current/api-reference.html
- https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-match-all-query.html
- http://exploringelasticsearch.com/searching_data.html#ch-searching-data
- http://okfnlabs.org/blog/2013/07/01/elasticsearch-query-tutorial.html#query-dsl-overview
- https://www.youtube.com/watch?v=52G5ZzE0XpY#t=1471
- https://sundog-education.com/elasticsearch/
- https://towardsdatascience.com/getting-started-with-elasticsearch-in-python-c3598e718380
- https://www.elastic.co/guide/en/elasticsearch/guide/current/_talking_to_elasticsearch.html
- https://www.elastic.co/guide/en/elasticsearch/client/index.html
- https://stackoverflow.com/questions/24153996/is-there-a-limit-on-the-size-of-a-string-in-json-with-node-js
- https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-bulk.html
- https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-delete-index.html
- https://www.elastic.co/guide/en/elasticsearch/reference/6.1/_list_all_indices.html
- Tutorial on Webscriping and parsing with BeautifulSoup
- https://stackoverflow.com/questions/1936466/beautifulsoup-grab-visible-webpage-text
- https://chartio.com/resources/tutorials/how-to-install-elasticsearch-on-mac-os-x/

- https://stackoverflow.com/questions/12451997/beautifulsoup-gettext-from-between-p-not-picking-up-subsequent-paragraphs
- https://stackoverflow.com/questions/50045253/if-statement-in-ejs
- https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-match-query.html
- https://stackoverflow.com/questions/10326950/render-a-variable-as-html-in-ejs
- https://stackoverflow.com/questions/50103466/ejs-cannot-read-property-of-undefined-how-can-i-fix-this
- https://code.tutsplus.com/articles/introduction-to-parallel-and-concurrent-programming-in-python--cms-28612
- https://docs.python.org/2/library/threading.html
- https://www.oreilly.com/learning/python-cookbook-concurrency
- https://youtu.be/gnsO8-xJ8rs
- https://getbootstrap.com

Introduction

The search engine is made up of three main components: the web crawler, the indexer and the user interface. There is also an intermediate phase in which we create json objects from each of html files crawled in order to load the index. This external phase is crucial for combining the search engine's components.

The web crawler crawls a set of seed .edu urls and extracts any link with the .edu top level domain. We then iterate through all the crawled html files, parsing each one for certain fields and create its instance as a json object. This intermediate phase results in a json file containing all of the web-document objects. This file is then bulk loaded onto the Elasticsearch cluster through an HTTP protocol PUT/POST request.

The interface offers a search bar and a submit button that returns query results based on the title and paragraph fields of the html document. Our json objects have collected paragraph text that is all compiled into one "body" field. Elasticsearch returns the results based on multi-field matching, displaying the matched web-documents and italicized highlights from "title" and "body" matches.

Web Crawler

(a) Architecture

- Multi-threaded design
- Duplicate detection

(b) The Crawling or Data Collection Strategy

- Using multiple threading to get the URL from seed web, and extract links from it to other docs (URLs) without duplicates by concurrent.futures and regular expression.
- Using multiple threading to download each HTML file to a local folder by 16 multi-task.
- In the data folder, the application creates a different level folder to keep the files.
- The application allows user input the number of pages to crawl and number of levels to limit application.

- The default setting of application: if the application gets the file over 1GB in some arbitrary level K, the application will stop when it finished web crawling in level K.
- If the user wants to input the limit by page or level, the memory wouldn't be limit by 1 GB
- When web crawling reaches the max level, the application will stop until the job is at max level.
- When web crawling reaches the max pages, the application will stop at Max ~ Max + 15.

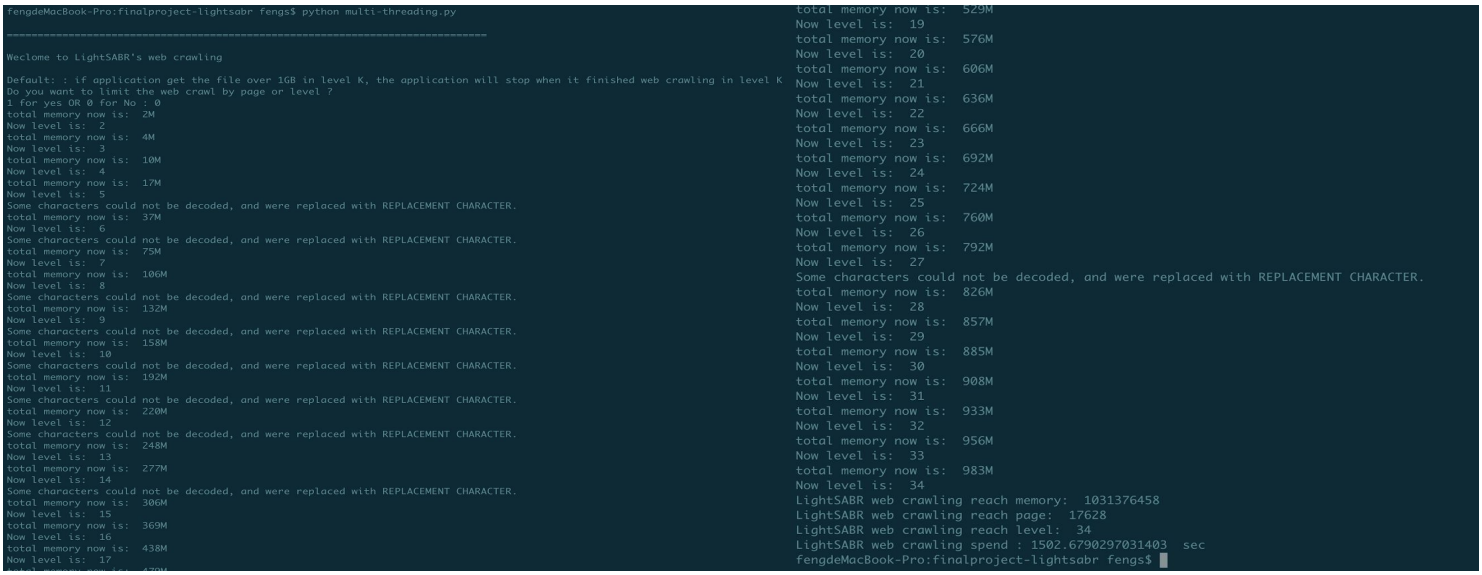

Figure 1 & 2: show the result of the web crawling application default setting.

(c) Data Structures/libraries employed

- Python lists, dictionaries, queues, time, hurry filesize(file size), re(regular expression)
- The BeautifulSoup library
- The `urllib.request` module

(d) Limitations

- the multiprocess of python is hard to control the shared variable - page_number, so the application will stop at Max ~ Max + 15. The reason why the application will stop at Max +15 is that I set 16 tasks to run the process.

(e) Structure HTML Data

- Parse .html and convert to .json bulk-load format

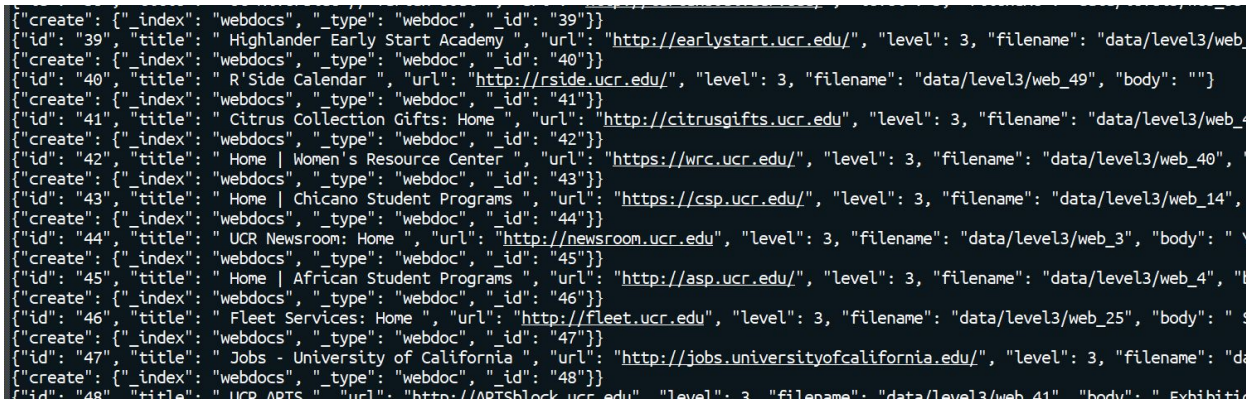

Figure 1: html web-documents readily available to load into an Elasticsearch Lucene Index.

Indexer

(a) Architecture: Ubuntu Virtual Machine on Virtualbox

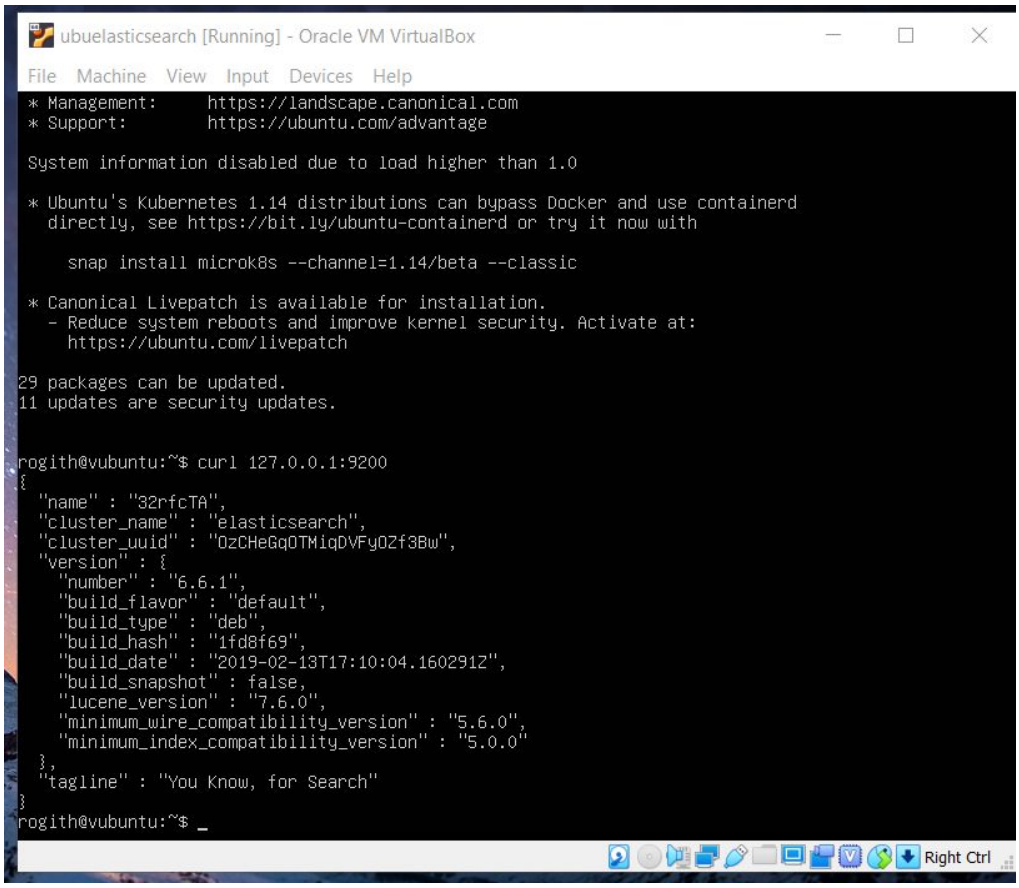

Figure 1: Elasticsearch instance on the virtual Ubuntu server/cluster.

(b) Index Structure: JSON Mapping Structure of a Web-document

```
{
    id: number,
    title: string,
    url: string,
    level: string,
    filename: string,
    body: string
}
```

(c) Search Algorithm

- Pose queries through RESTful API over HTTP requests (HEAD, GET, POST, etc.)
- Retrieve results based on multi-match of web-document fields: title, body

(d) Limitations

- Due to virtual cluster, only the host machine of the cluster may successfully make HTTP requests and thus run the application

Extension : Web Interface Front-End

(a) Frameworks used: Express and Node.js

(b) Ping the cluster when executing the application

(c) Allow user queries through a search bar and submit button

(d) Display ranked list of .edu web page documents

(e) Limitations

- We wish to modify the Embedded JavaScript file (.ejs) to search for the matched words in a query and only highlight those-
- The only way to access the virtual cluster is directly through the host running the virtual machine

```
rogith@DESKTOP-1Q8NDL3:~/finalproject-lightsabr/FrontEnd-master$ node app.js
body-parser deprecated undefined extended: provide extended option app.js:14:2
0
body-parser deprecated undefined extended: provide extended option app.js:32:2
0
server is on port 3000
all is well
```

Figure 1: Ping the Elasticsearch cluster when executing application (HEAD request).

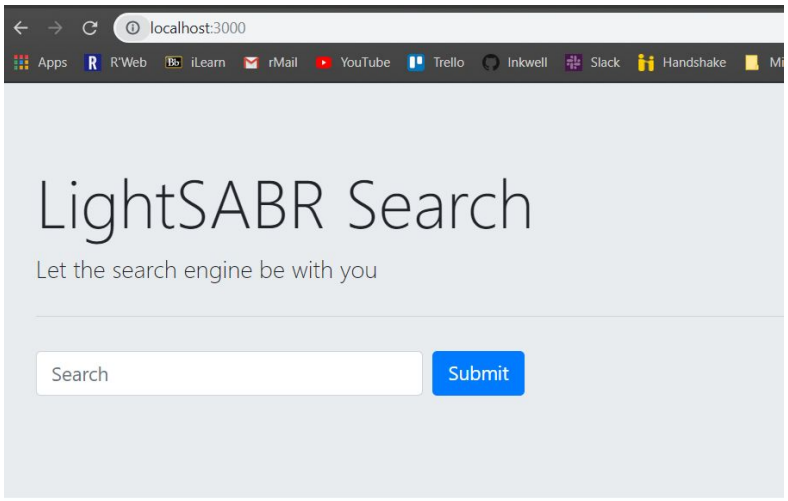

Figure 2: User interface.

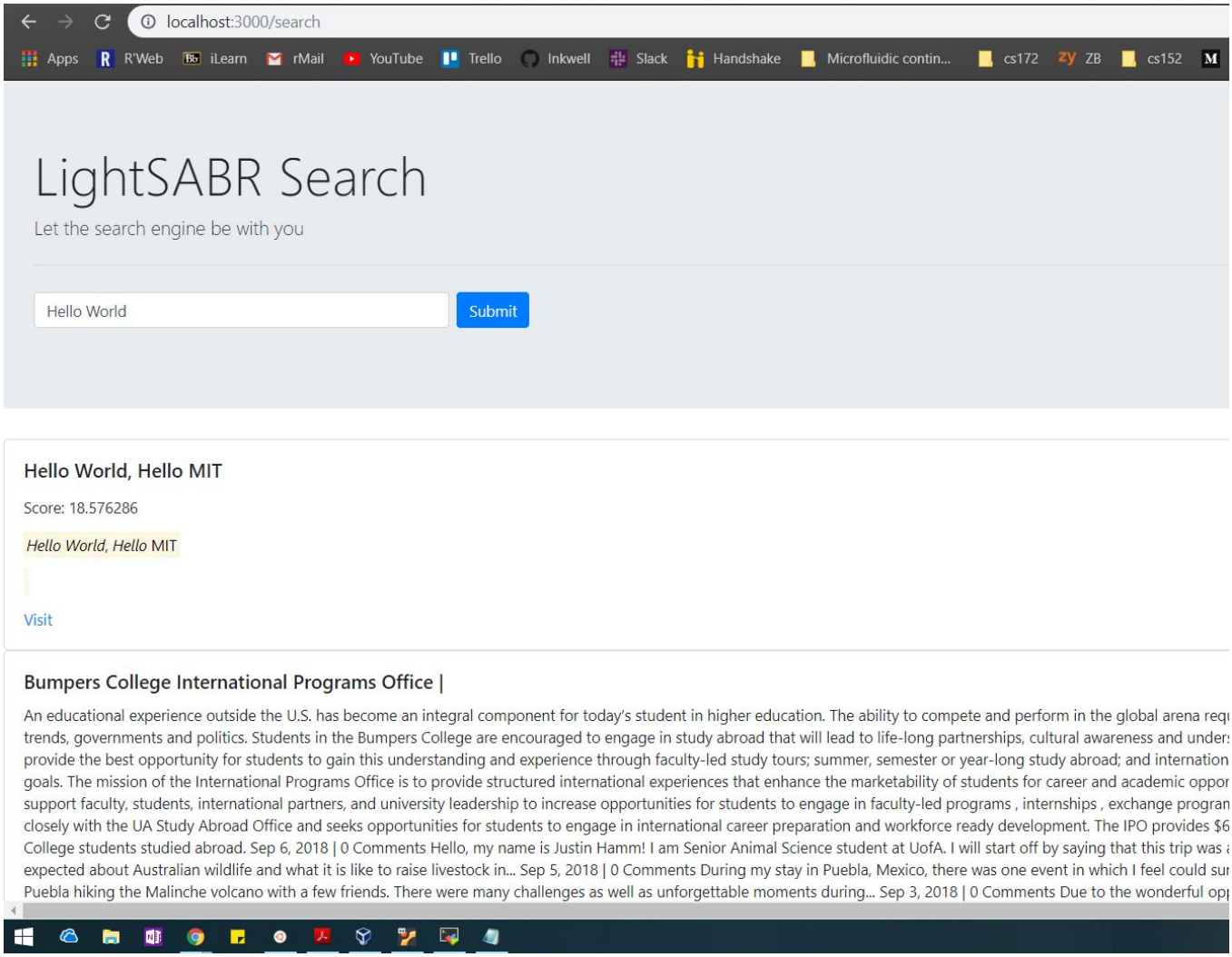

Figure 3: Search results in action.

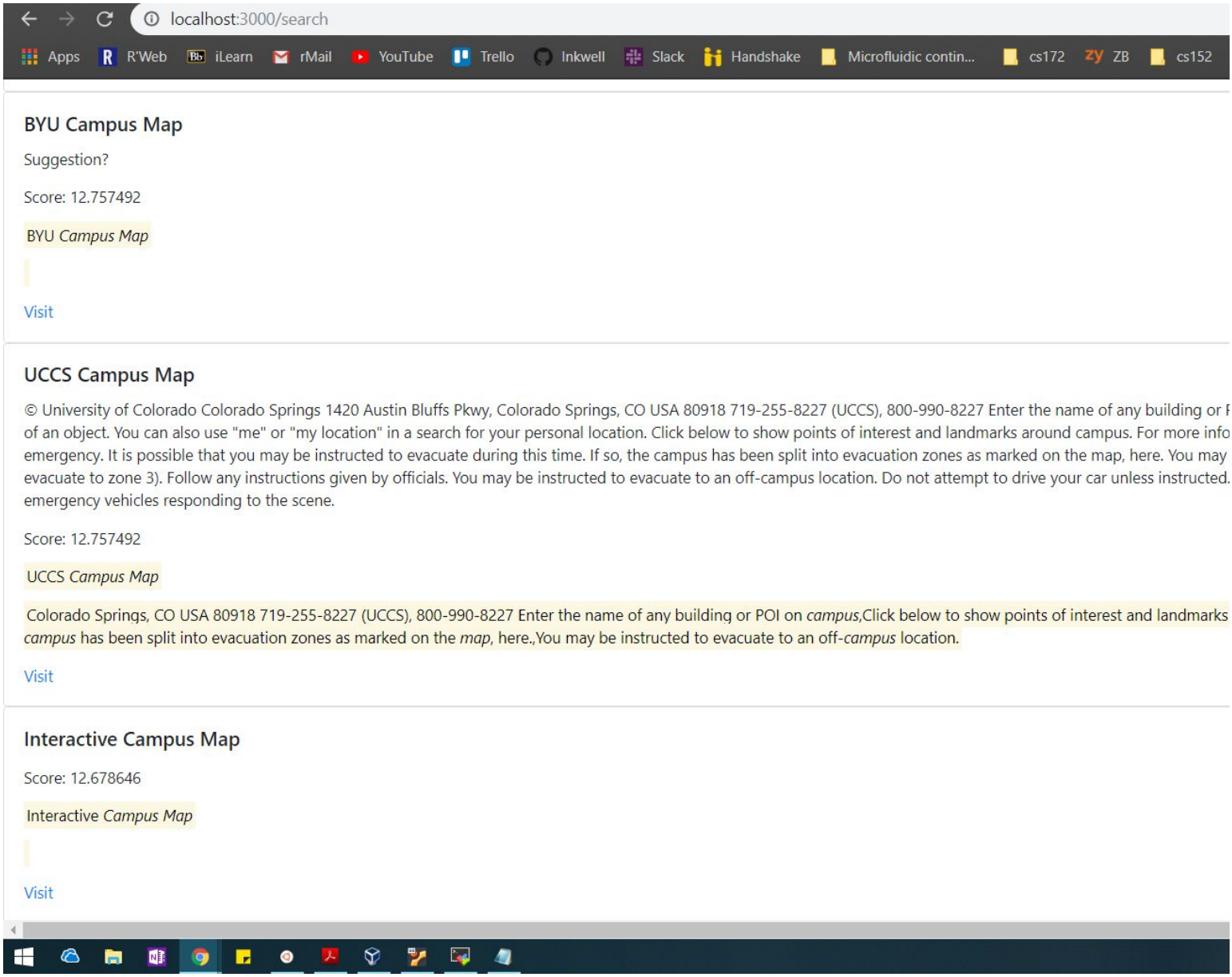

Figure 4: Highlights displayed from title and body matches (query shown: “campus map”).

Sample Test Cases and Data

```
curl -XPUT 127.0.0.1:9200/_bulk -d '

{ "create" : { "_index" : "movies", "_type" : "movie", "_id" : "135569" } }
{ "id": "135569", "title" : "Star Trek Beyond", "year":2016 , "genre":["Action", "Adventure", "Sci-Fi"] }
{ "create" : { "_index" : "movies", "_type" : "movie", "_id" : "122886" } }
{ "id": "122886", "title" : "Star Wars: Episode VII - The Force Awakens", "year":2015 , "genre":["Action", "Adventure", "Fantasy", "Sci-Fi", "IMAX"] }
{ "create" : { "_index" : "movies", "_type" : "movie", "_id" : "109487" } }
{ "id": "109487", "title" : "Interstellar", "year":2014 , "genre":["Sci-Fi", "IMAX"] }
{ "create" : { "_index" : "movies", "_type" : "movie", "_id" : "58559" } }
{ "id": "58559", "title" : "Dark Knight, The", "year":2008 , "genre":["Action", "Crime", "Drama", "IMAX"] }
{ "create" : { "_index" : "movies", "_type" : "movie", "_id" : "1924" } }
{ "id": "1924", "title" : "Plan 9 from Outer Space", "year":1959 , "genre":["Horror", "Sci-Fi"] }
'
```

Figure 1: Test data that was first loaded to the Elasticsearch cluster to render results onto the interface. (Can be found from the Udemy sources above).

Trello Board Project Organizer

For a comprehensive look at the history of our development, please take a look at our public team Trello board: https://trello.com/b/lz8MPUvc/cs172-final

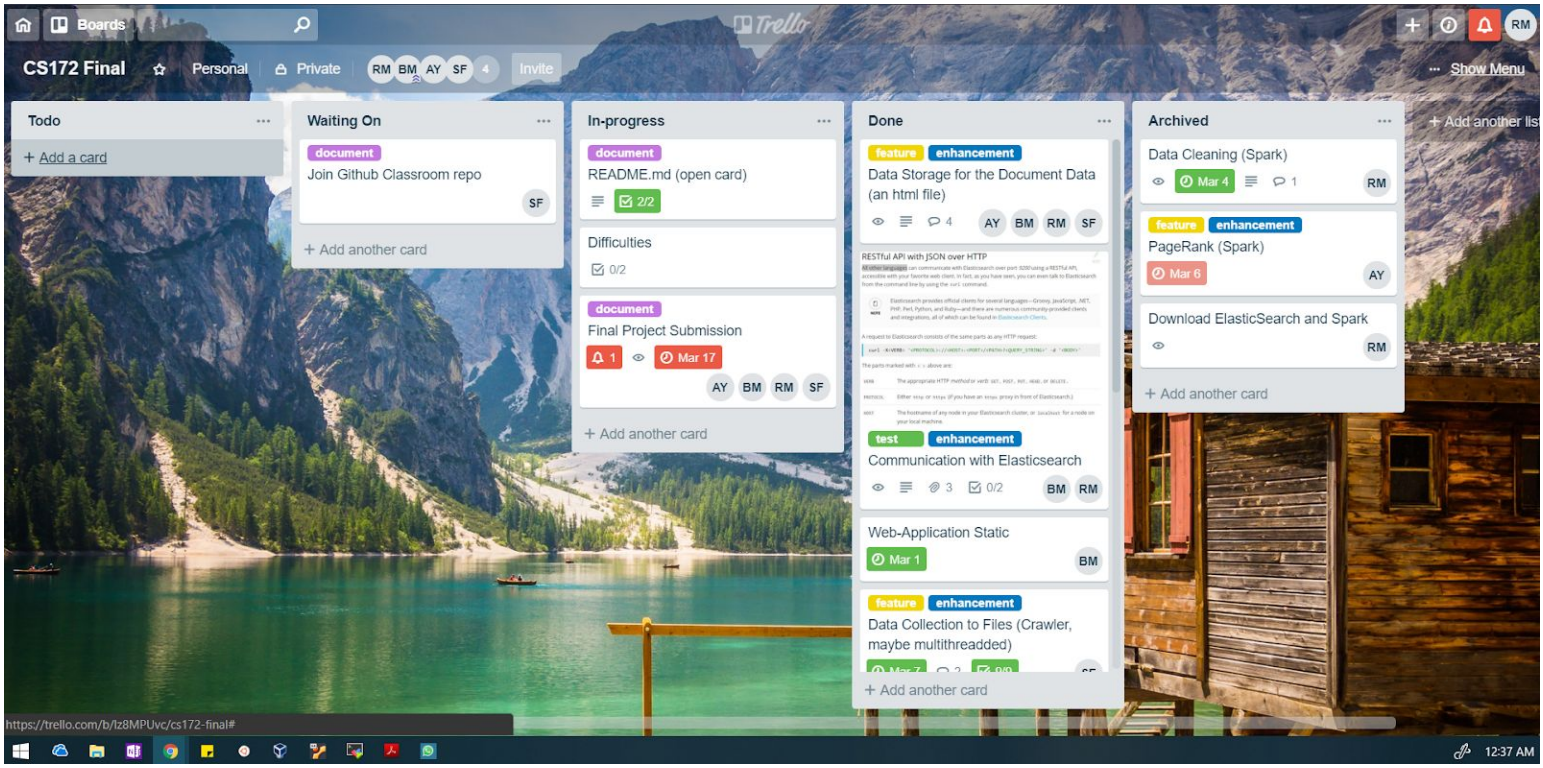

Figure 1: Team LightSABR Trello Board.

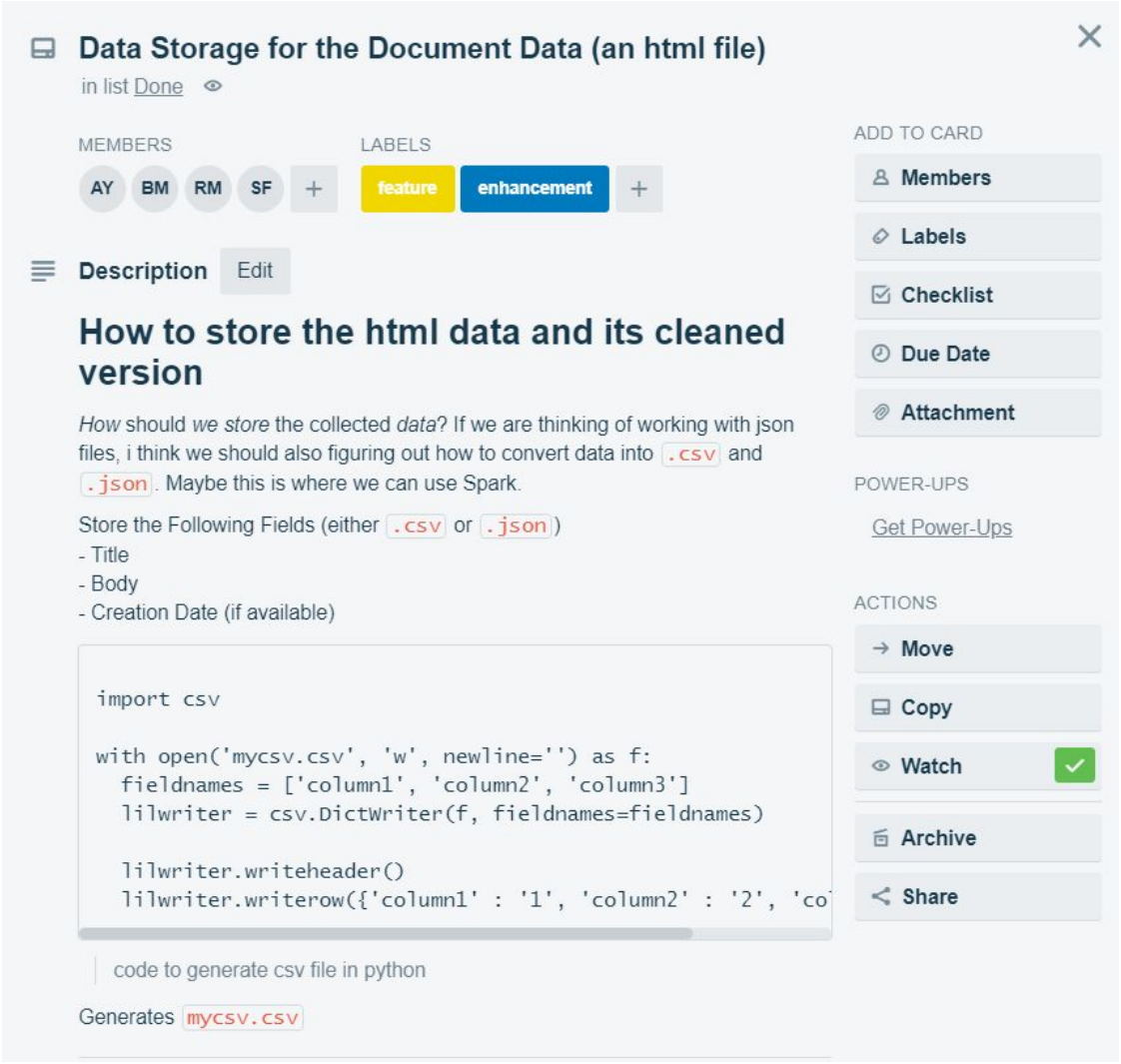

Figure 2: A completed Trello card that led us towards converting html data into json format: Data Storage for the Document Data (an html file).

Conclusion

At first glance, although the final project seemed very intimidating because of how many critical components there are, our group believes that producing a working search engine provides us with valuable knowledge that helps us when working in the industry. This knowledge includes how to build a web crawler, experience using an index API (Elasticsearch and Lucene), experience with working with html and json data, implementing a front-end user interface and overall how to efficiently work with big data.

A difficult challenge our team faced while working on the final project together was integrating all of our work into a finished final product. The fear of integration influenced us to use a scrum-like system to prioritize crucial components for functionality to optimize work efficiency. We implemented a Trello board to easily partition and prioritize each individual task and also use the board to archive a list of tasks that are complete.

Some of the further build-upons are mentioned in each limitations section above.