CS 172 Project Part B - 6/8/2018

James Thi
Qiwen Lyu
Patrick Aben

Twitter: Write a program that parses the JSON objects of your big files form Part A and inserts them into Lucene. Handle the fields like username, location, and so on. Create a Web-based interface.

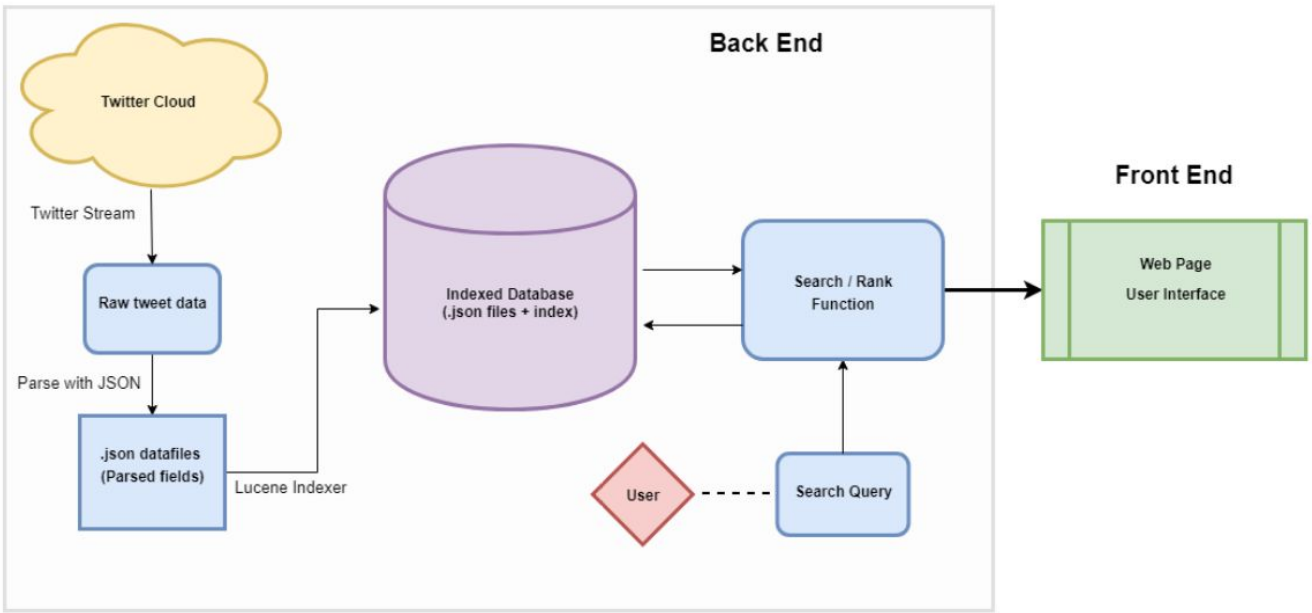
Collaborations

Nelly (Qiwen) started off by creating the project and added all necessary libraries and dependencies. She built the index for the 5gb of datafiles using Lucene. Patrick then wrote the searching and ranking portion of our project, so that given a query and the number of results it would output, in decreasing score, the relevant results. James then wrote the separate front-end project for the Web-based UI, producing a well-formatted interface for the user to interact with and submit his own queries/number of results. **Blank** created the batch executable which allows the program to be run very easily.

System Overview

In this project we successfully met all requirements given by the prompt. Using the datafiles (.json) containing tens of thousands of tweets, we indexed 5GB worth of tweets so they were properly searchable and rankable through Lucene. Then we created the web-based user interface to allow the user to easily enter a search query to view ranked results, in decreasing score. The user can also choose the number of results shown, or a default of 10 results is given. This whole part of the project was done using Lucene libraries and a separate Spring-boot project for the front-end Web UI.

Architecture



Index Structures

In the main app class we created a class “page” to store all important fields: title, latitude, longitude, source, date, tweet_urls, hashtags and text. We also created a constructor to assign these values into the page. A new function called getDoc is then used to add all the related fields listed above into the document.

After all preparation is complete, the .json files can be parsed into the main function. The first line is empty so it is skipped. Then for each line, which represents one tweet, Json parse and Bufferreader are used to read in the different string object successfully. The constructor and function are then called to push it into the index.

The size of the index is a small fraction of the data itself, but still took up almost 2GB of memory. To work within the limitations of our computers, our index is instead written onto the computer’s physical disc rather than RAM. A directory containing the index must be referenced for the ranking system to later check and use to search with query terms, and is constantly referenced whenever new searches are made. The index is comprised of the pages constructed during the documenting portion of the JSON files.

Search and Rank Algorithm

The search algorithm checks the indexed data and uses the index files to quickly find instances of the query terms and frequency per document. After finding the tweets with the highest frequency occasions of the query term appearing, the ranking algorithm lists the tweets from highest to lowest. In our searching system, we also count the number of times the query terms appear in the title of links, as well as hashtags. Because we added hashtag consideration in ranking, query terms that appear in a hashtag are counted twice, once for being in the text of the tweet, and a second for also being part of the tweet’s hashtags. This creates a bias in rank, where hashtagged terms are ranked much higher than those without. For example, a tweet that mentions a query term 5 times would be ranked lower than a tweet that simply hashtagged the same word 3 times. Or if there are two separate query terms, and a tweet that only mentions one term in a hashtag is higher ranked than a tweet that directly references both query terms. Lastly, it is also possible that the title of a link that the tweet references may have a high frequency of query terms, and not mention them at all in the text of the tweet to be higher ranked than a tweet that directly references all query terms, but fewer times. This means that our search algorithm considers all query terms to be independent of one another.

System Limitations

Our Web UI is set up so it should not produce any errors. Search queries that only contained white space or only contained 2 or less characters produced a message telling the user to enter a valid query. Entering a non-positive integer into the “Number of results” textbox still produced search results with a valid query, but at a default “number of results” of 10. From our testing there was no other way to break the system.

Ideally we would have liked to provide additional details in the results section, such as the username of the tweet's poster,the profile picture of the user, or the link to the original tweet, but unfortunately in our datafiles we never recorded the username or the link to the original tweet. We did not see that option when we were recording data and it would be much too late to gather another 5gb of data. If we had that information though it would be simple to add that detail to each result using a couple html formatting commands.

We also would have done the extra credit and Nelly worked to create the .css and .js files needed to produce the google map api. However we tried multiple times to connect the .css and .js files to the html file, but were unable to make the two work together. The html file would not show anything and could not detect the .css and .js files regardless of what folder they were in. We think it is because we used spring-boot to build the Web UI, and the spring-boot template hid some connections and prevented the html from detecting the .css and .js files. If we recreated the project or did more research we might have been able to get the map working.

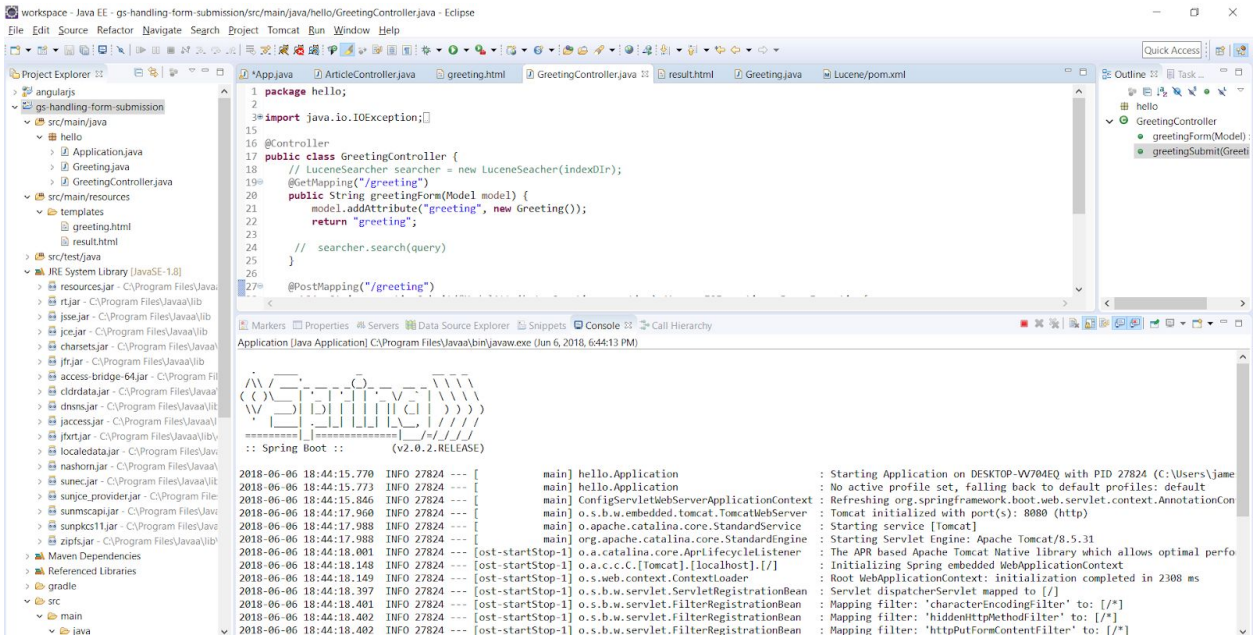
How to Deploy System

How to launch the spring-boot project

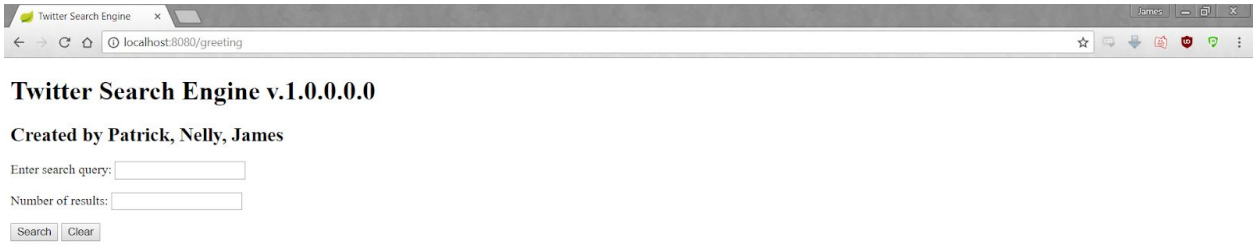
Once the Spring-boot project has been launched, the local webpage can be accessed. In a web browser head to *localhost:8080/greeting* . A welcome page will appear with two text boxes: one to enter a search query, and one to enter the number of results desired. Any search query of 3 or more non-whitespace characters should process correctly. If a non-positive integer is entered under the number of results, the system will default to a maximum of 10 outputted results. Simply fill in the two text boxes and click Search to run the search function, or Clear to clear both text boxes. From the results page there is also a link to Submit another query.

Screenshots

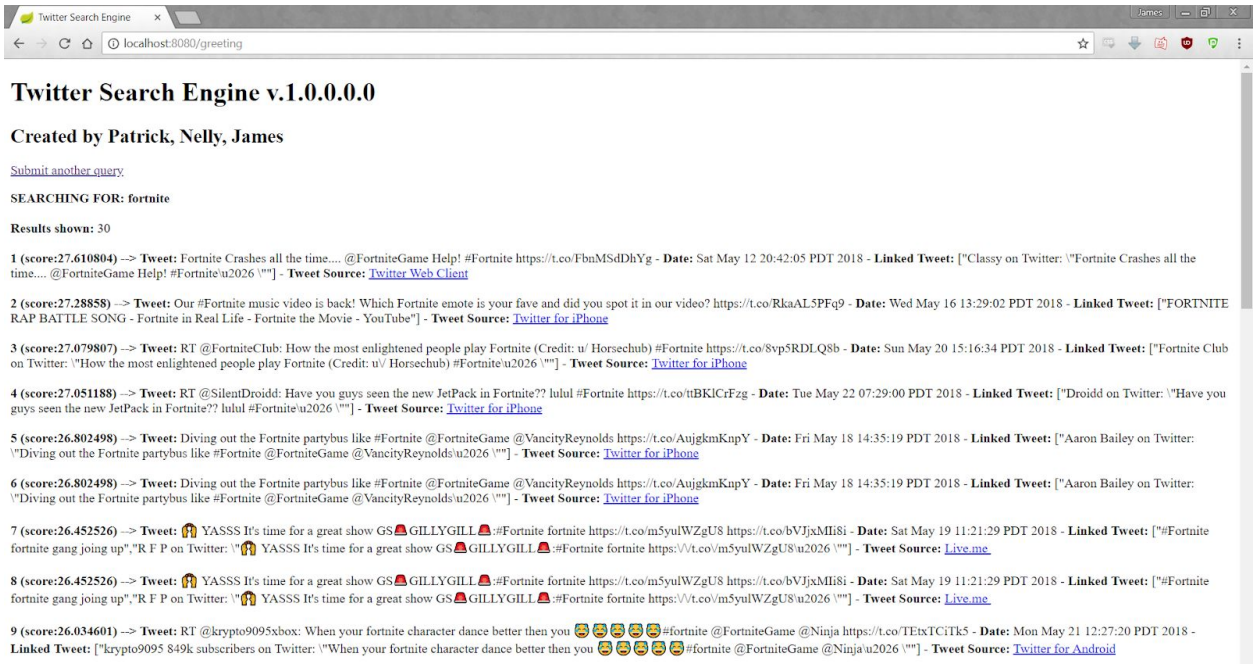
Spring-boot launched and running in Eclipse:



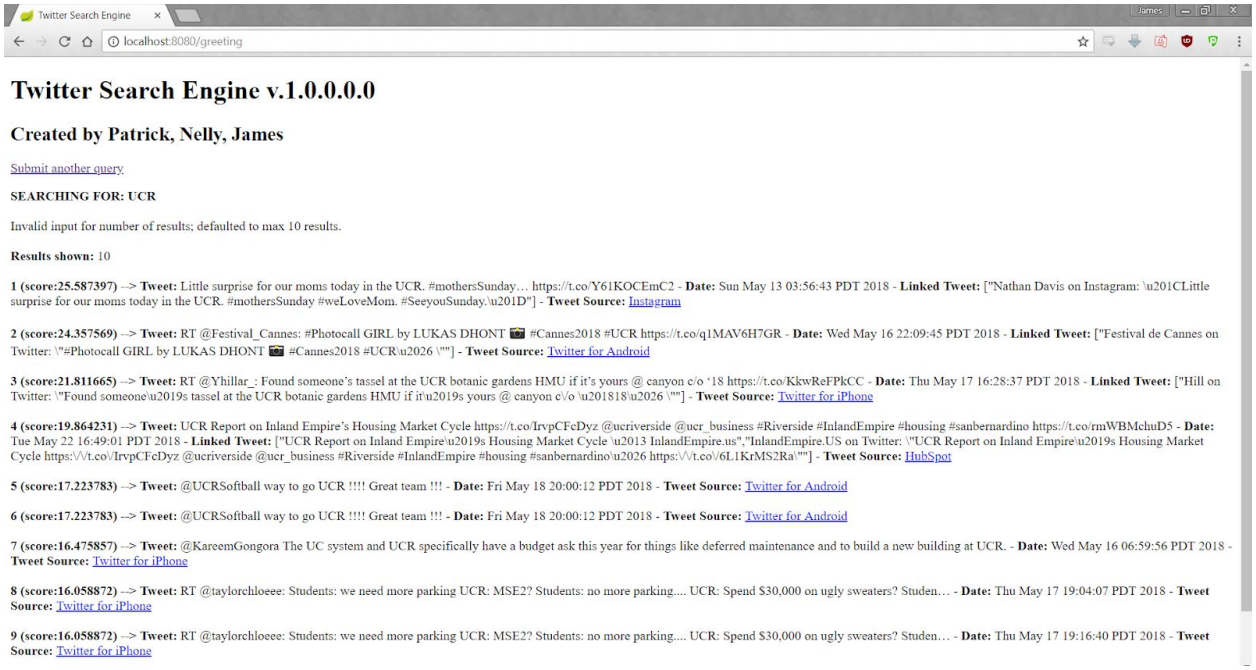
Welcome screen of the Web UI:



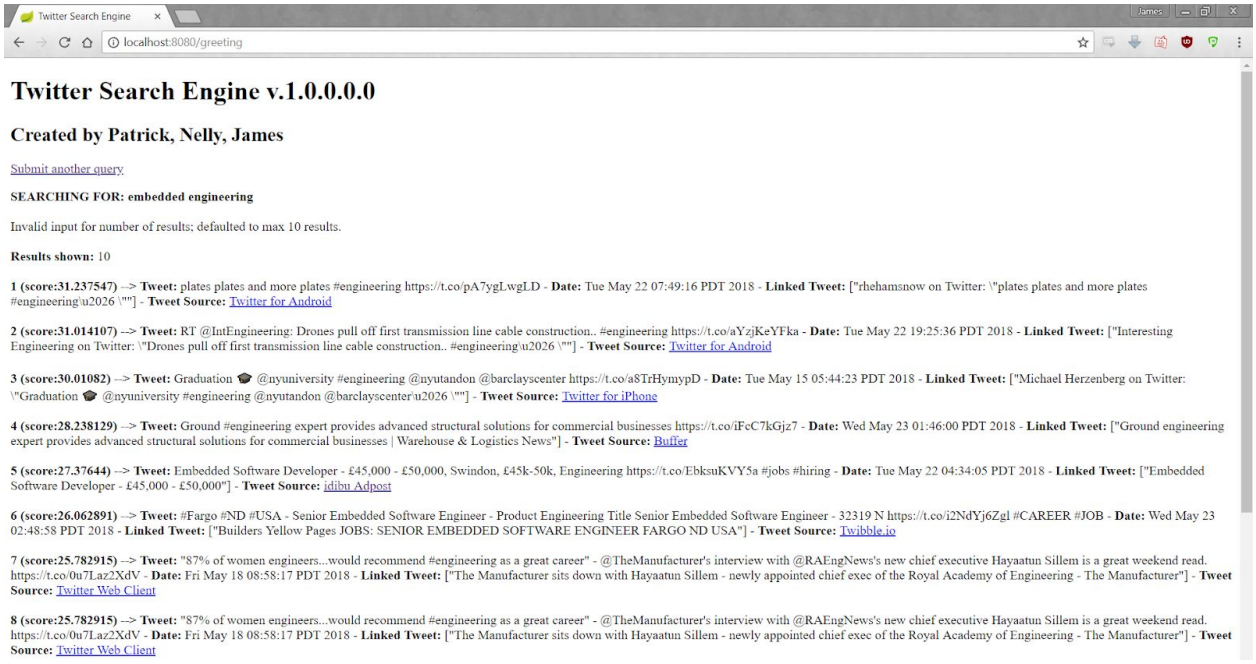
Results page with Search Query: “fortnite” and Number of Results: “30”:



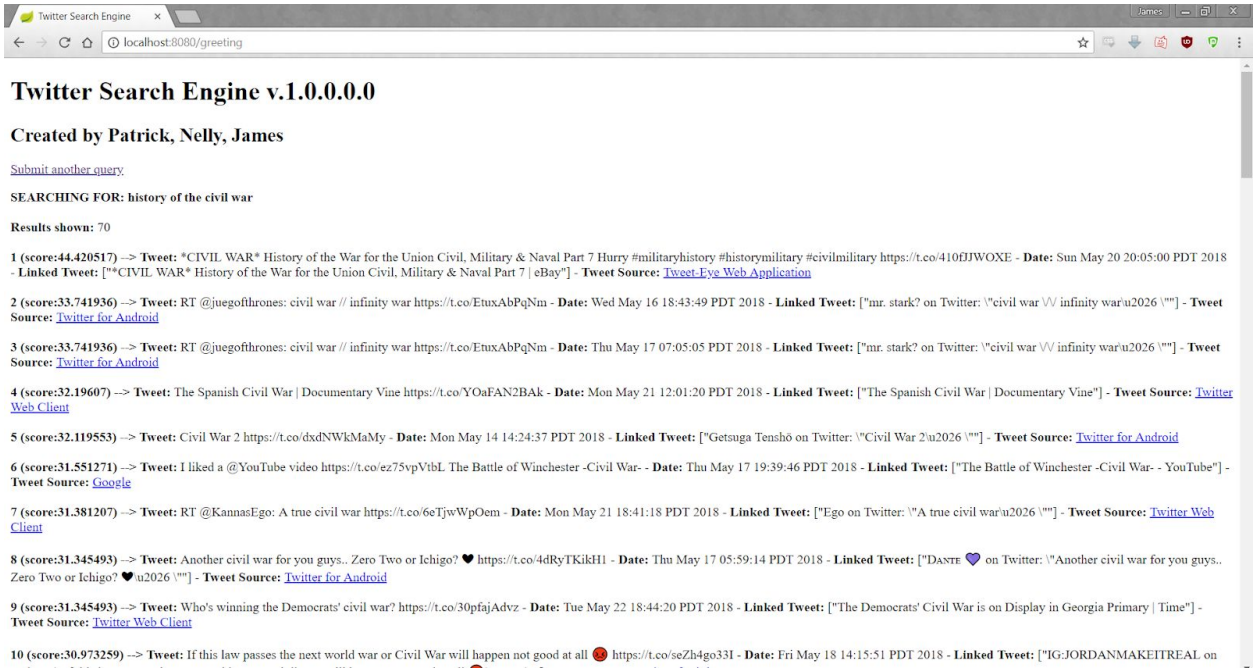
Results page with Search Query: “UCR” and Number of Results: “invalidnumber30asdf”:



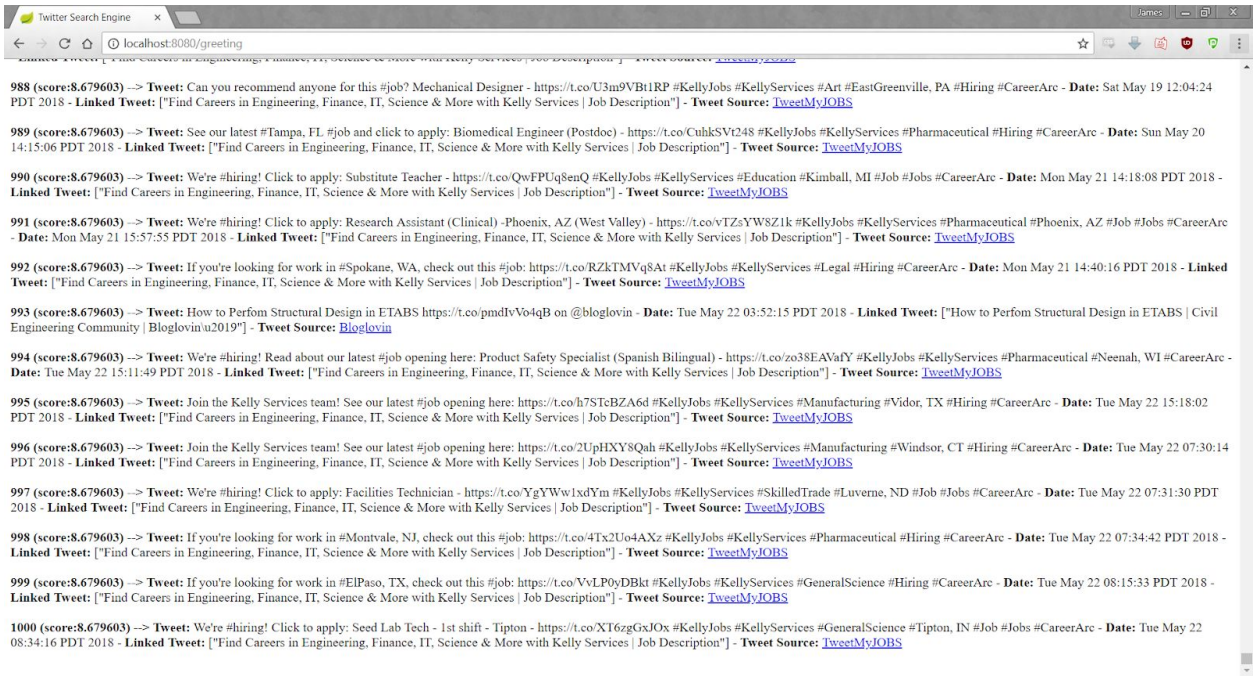
Results page with Search Query: “embedded engineering” and Number of Results: “”:



Results page with Search Query: “history of the civil war” and Number of Results: “70”:



Last part of Results page with Search Query: “engineering” and Number of Results: “1000d”:



Results page with Search Query: “” and Number of Results: “10”: