Team Members:

1. Sanchit Goel
2. Rohan Behera
3. Ishika Rakesh
4. Raghav Gupta
5. Benson Wan

CS172 - Information Retrieval
5 June 2022

CS172 Project Phase 2 Report

Collaboration Details:

- **Sanchit Goel:** Worked on the LuceneSearchWebController.java and connected it to the LucenesearchApplication.java . Also worked on merging the tweets collected across different sources. The LucenesearchApplication is a wrapper class that serves as a placeholder class for the Spring boot application to run. The LuceneSearchWebController is the main class that handles routing for the entire web application. It contains one route which is "/home" with POST and GET request handlers for the route. The GET handler returns the home page with the search feature, while the POST handler forwards the client's search query to the backend (specifically Lucene) and returns a page with the search results. Merging the tweets from different sources required a python script which aggregated the tweets into files limited by the number of tweets per file. The script also removed duplicate tweets (which could occur if the crawler was run by more than one person at the same time) by keeping a set of all unique tweet IDs.

- **Rohan Behera:** Wrote the LuceneFileIndexBuilder class. In this class I used the StandardAnalyzer. The cleanupFileIndexFolder function does as its name suggests, deletes indexed files every time the program is rerun to reduce memory usage. The createFileIndex takes in the JSON files, analyzes them and adds them to the indexFiles folder. In the forEachTweetFile function the JSON files are parsed using a simple JSON library using a for loop and a try-catch exception inside of it. There's a retrieveFullDocument class so we don't have to manually change the paths to where the input and output is stored on our computer everytime we merge our commits. The addDocuments function adds the file name and array index to the documents. The convertJsonObjToDoc adds the tweet id, user id , device name, date/time the tweet was crawled, and the text to the document. Also implemented the ability to search by device such as Twitter for iPhone, Android, Web App, etc.

- **Ishika Rakesh:** Worked on the LuceneSearchWebController.java and LuceneIndexSearcherImpl.java and modified functions in order to help with the search and Implemented the extra credit feature of searching by id field. There is an int maxResults variable to return the top results and set it to a value of 10. There's setter and getter functions for setting and getting the max results as well as a function for getting the indexed document. Implemented the wildcard search strategy as well. We take in a query and calculate the time taken from when the spring boot started searching for the query and the time when it found the result. Then we take the difference between the two. Worked on the recording alongside Raghav Gupta.

- **Raghav Gupta:** Built the frontend for the project. This includes HTML, CSS, and Javascript that combines to render the user homepage, search result page, and 403 and 404 error pages. We also implemented a feature which allows the page to rerender with restarting the application. Before, we had to relaunch Spring Boot every time we had to make a new change which was inefficient and super slow. This functionality helped drastically reduce development time and made frontend development much easier. Also made sure to tie certain parts of the backend and frontend together to make sure the information was correctly being stored when the user submitted a query. Implemented the extra credit feature of searching by device within the same search text field alongside Rohan Behera. Worked on the recording of the project as well as the collection of JSON tweets and compressing them alongside Rohan Behera and Sanchit Goel. In partnership with them, we helped streamline and improve the current codebase by removing redundant code and simplifying functions and files.

- **Benson Wan:** Continued crawling for more tweets in relation to marvel during phase 2 to expand our collection of json documents to query from. In addition to this, I researched the BM25 ranking model for our program and for the purpose of expanding the explanation for it in our report. I also implemented the tweet.java class with the setters and getters to acquire and manipulate the tweets. To be completely honest, I didn't do much for phase 2 of this project in comparison to the rest of the group and they all definitely worked very hard to put this program together. I was constantly occupied/bombarded with personal things as well as tasks from other classes that I had trouble balancing and I honestly didn't contribute as much as I should have to this phase. In light of these things, I feel like I should receive less points than the rest of my group and am completely okay with receiving less credit because it's what I deserve.

Index Structures

Clarify which analyzer we are using and why:

- We are using the standard analyzer because it is the most sophisticated analyzer.

- It takes into account searching with certain token types such as setting and getting the maximum token length.
- It also takes into account lowercase letters, removing stop words such as *the, of, and*.

Which fields are adding to our document:

- The fields that are being added to the document are the tweet ID, user ID, the date and time the tweet was created, the device the tweet was sent from, and the text of the tweet itself.

Which fields are being stored in the index or not:

Being stored:

- Created_at, device

Not be stored (but still indexed):

- Tweet ID, User ID, likes, retweets, text

Which analyzer is being used for each field:

- Standard analyzer is being used for each field.

While searching which fields are being searched and which are not etc:

- Text, device, and ID are being searched. Geolocation, likes, and retweets are not

Ranking Algorithm

Which ranking algorithm is used:

After spending some time researching, we decided to use the BM25 ranking algorithm.

Explain why it's the best approach:

We decided to use the BM25 ranking model due to the fact that it's one of the most successful ranking algorithms in its ability to approximate the relevance of documents within a collection to a given search query.

If you are using the default one, explain how the algorithm works:

We aren't using the default ranking algorithm.

If you are using a different ranking function, explain it also:

Upon choosing BM25 as our ranking model, we were tasked with implementing the bag of words retrieval algorithm for our twitter web search engine. Since BM25 ranks a set of documents within a collection based on the query terms contained in each document all while disregarding proximity, we were able to apply this to our json documents that contained all of our crawled twitter tweets. The ranking algorithm takes in a given query and document then proceeds to utilize a couple of variables to calculate the BM25 score of the respective document: the query frequency of a specific term, the document frequency that contains the specific query term, the term frequency within a specific document, and the total number of documents within the collection. These values combined with a few variables that are set empirically allowed us to calculate the relevance upon which tweets within our json files we wanted to return for the user to view.

Architecture

Explain your web architecture:

How you are using the backend

- We are using the backend to parse the JSON files from phase 1 and insert them into Lucene, taking into consideration the tweetID, userID, device, text of the tweet, date/time it was crawled.

How you are creating the frontend:

- The frontend was created with pure HTML, CSS, and spring-boot's templating engine: Thymeleaf (which passes in data from the backend into our rendered page).
- How backend and frontend are being linked together
- The frontend is queried via a GET request, and when a user makes a query, the frontend executes a POST request to the backend (handled by the LuceneWebSearchController) which then parses and handles the query through Lucene. After lucene generates results, the backend then responds with an HTML page with the search results (via the Thymeleaf templating engine).

Which technologies you are using, etc:

- Java, Maven, Spring Boot Framework, Lucene, Gson, Thymeleaf

Mention the APIs you have implemented in the backend:

- Route: "/home" with GET and POST request handlers. The POST request handler parses and processes the client's search query (via Lucene) and returns a page with the search results.

Connection with Lucene backend and Local Storage

Mention how you are connecting with your database/local file system from the Lucene backend and how you are retrieving the data to show the results:

- We have a public static final String docsPath where the JSON files are taken in as input and public static final String indexPath where the indexed JSON files are stored as output. For parseInputJsonFile we have a boolean function to filter out files and accept only files with .json at the end. We parse the json files with a simple-json library known as arrayObjects. Each JSONObject contains a long tweet id, long user id, string for date/time created, string device, string text, long number of likes, long number of retweets.

Index File Size and Total Data Collected Size

Mention index file size:

The index file size for tweets is 214.4 megabytes

Mention total collected data size:

The total collected data size for tweets is 722.2 megabytes.

Extra Credit

- Implemented search by device: The way we were able to implement search by device is by having a deviceStr as input and we have a for loop traversing through the length of the scoreDocs and we have ScoreDoc sd, float score, int docID, Document d, and strings for device and text. And then we return the score, device, and text. In the homeSubmit function in lucenesearchwebcontroller.java we set topdocs equal to indexsearcher.searchByText(tweet.getDevice()).
- Implemented search by ID: The way we were able to implement search by ID is by modifying the files LuceneSearchWebController.java and LuceneIndexSearcherImpl.java. In the first file, there is a function, homeSubmit() where when a request is submitted on the homepage it calls the post method which is the homeSubmit function in the web controller. In this function we added a few extra lines of code where if search does not return any result based on search on the device or text field, then it searches data in the ID field if it doesn't find anything else in the other two fields. This happens when the variable, totalHitsVal ==0, then the index searcher searches by the search in the Id field. This is where we take a look at the builder file in which we implement search by id by using long point and newRangeQuery since the id is long and search on long field types works with a special query which is the new range query that is being used to query the id.

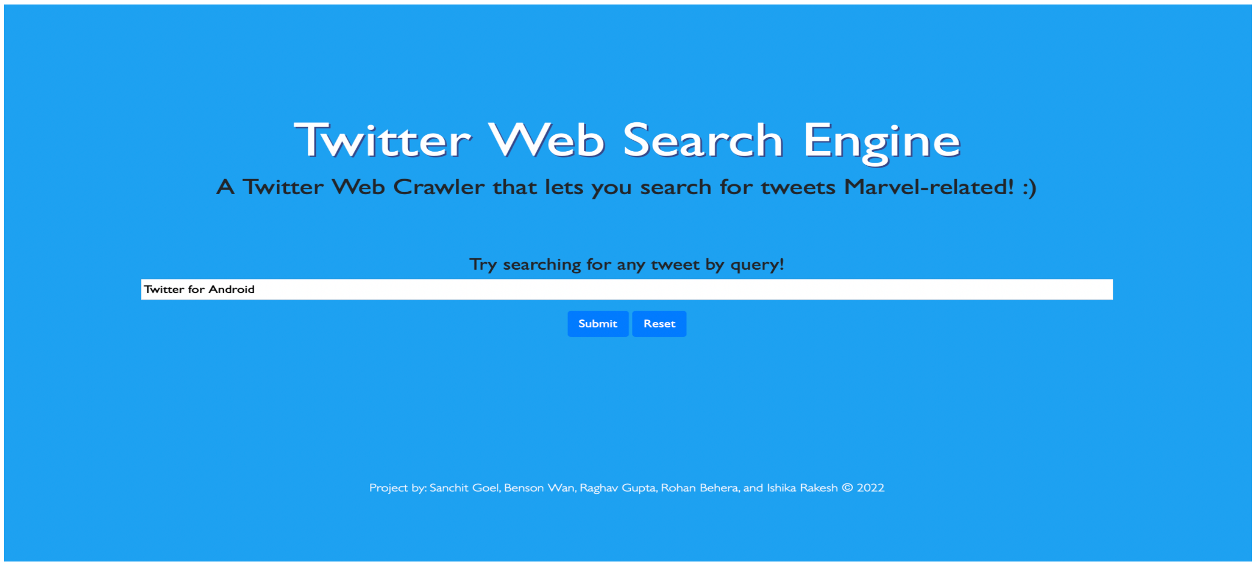
Instructions for installation

1. Requirements: Java SE Development Kit 8u333, Apache Maven, Visual Studio or other IDE
2. Download the zip file, unzip and open the folder in your IDE
3. In your terminal window press cd, enter and then vim .bash_profile. Make sure the paths to maven_home and java_home are set correctly. Then close out of vim and run source .bash_profile
4. Run the command mvn

5. cd into the folder where the source code is and run mvn clean install
6. Then run the command mvn spring:boot run
7. Open up a new tab in your browser and type in localhost:9090
8. Search for tweets!

Screenshots of User Interface (UI)

Home Page



Twitter Web Search Engine

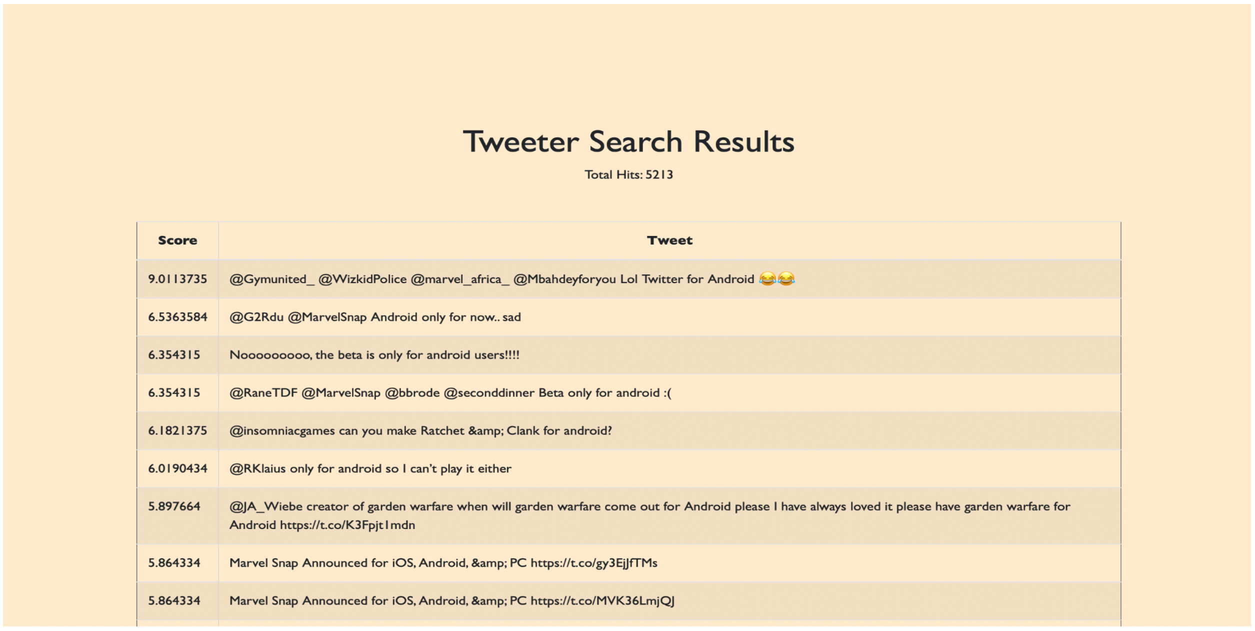
A Twitter Web Crawler that lets you search for tweets Marvel-related! :)

Try searching for any tweet by query!

Twitter for Android

Project by: Sanchit Goel, Benson Wan, Raghav Gupta, Rohan Behera, and Ishika Rakesh © 2022

Search Query Page

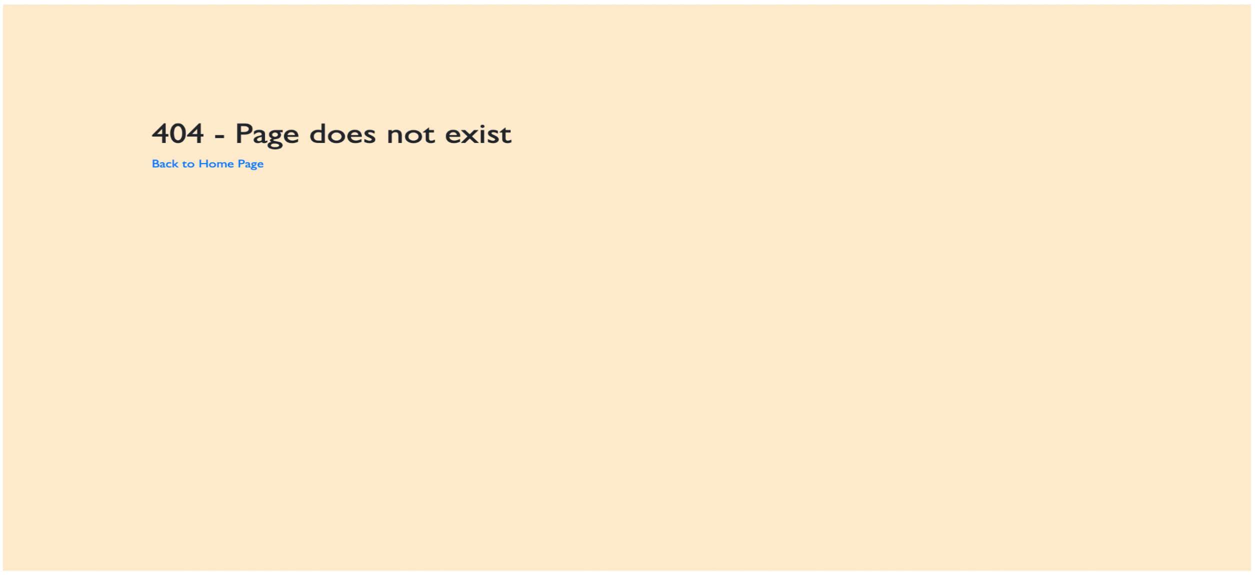


Tweeter Search Results

Total Hits: 5213

Score	Tweet
9.0113735	@Gymunited_ @WizkidPolice @marvel_africa_ @Mbahdeyforyou Lol Twitter for Android 😂😂
6.5363584	@G2Rdu @MarvelSnap Android only for now.. sad
6.354315	Nooooooooo, the beta is only for android users!!!!
6.354315	@RaneTDF @MarvelSnap @bbrode @seconddinner Beta only for android :(
6.1821375	@insomniacgames can you make Ratchet &amp; Clank for android?
6.0190434	@RKlaius only for android so I can't play it either
5.897664	@JA_Wiebe creator of garden warfare when will garden warfare come out for Android please I have always loved it please have garden warfare for Android https://t.co/K3Fpjt1mdn
5.864334	Marvel Snap Announced for iOS, Android, &amp; PC https://t.co/gy3EjJfTMs
5.864334	Marvel Snap Announced for iOS, Android, &amp; PC https://t.co/MVK36LmjQJ

404 Page


404 - Page does not exist

Back to Home Page

Demo of Application

***** Demo Video Link *****