General Overview

The purpose of this document is to outline the functionalities of this project and give users a guide as to how to use it.

This project is split into two programs: 1. Building the document store 2. Operating that document store. The first program will first take in a json file and construct a MongoDB collection. The program will input (into the command line) a json file name and the port number under which the MongoDB server is being run. It will create a database named 291db that will have the collection of tweets.

The second part of this program will take the input of a port number under which the MongoDB server is being run, and will connect to the database 291db. It will then direct the user to a menu and give options to operate within the program. The user will be able to: **1.** Search for tweets, **2.** Search for users, **3.** List top tweets based on *retweetCount*, *likeCount*, or *quoteCount*, **4.** List top users based on *followersCount*, **5.** Compose a tweet. **6.** exit

User Guide

- 1. Search for tweets:
    - User inputs in a keyword or multiple keywords and displays a list of tweets with user id along with their user info whose tweets contain one or more of the keywords being searched. The user is then prompted to input a user id to view their data by displaying the fields of the selected user. The user can quit by inputting ‘q’ instead of a user id.
- 2. Search for users
    - User inputs in a keyword or multiple keywords and displays a list of users whose name contains one or more of the keywords being searched. The info displayed are the user’s username, displayname, and location. The user is then prompted to input a username to view their data by displaying the fields of the selected user. The user can quit by inputting ‘q’ instead of a username.

- 3. List top tweets based on *retweetCount, likeCount, or quoteCount*
    - The user is first prompted to select what field they want the top tweets to be based on. The user can input 1 for *retweetCount*, 2 for *likeCount*, and 3 for *quoteCount*. After a field has been chosen the user is prompted for the amount of tweets they would like to display. The tweets Id, Date, Content, username, and amount for the selected field are displayed.
- 4. List top users based on *followersCount*
    - The user is prompted to enter a number *n*. The top *n* accounts are then displayed based on *followersCount*. The user is then prompted to enter the username of one of the displayed accounts. The full information of the user is then displayed. The user can quit by inputting ‘q’ instead of a number or a username.
- 5. Compose tweet
    - The user is prompted to write the content for a tweet. The user can quit by inputting ‘q’ instead of a tweet.
- 6. Logout
    - The user will be exited form the program
- Main_menu: the user is supposed to type in a number to select the desired system functions
    - Once the user selects a desired function, they will be instructed to enter certain details to progress.

Design

As per the project, the user will first be prompted to load a json file where they have to input the arguments which are the json-file and the port number. The user will then run the main interface where they will input the port number as the single argument.

PHASE 1:

- load-json: A program that needs to be executed before main_interface.py can be run properly. Load-json.py will create a database within the MongoDB server called 291db with a collection named tweets. The tweets collection will be populated with documents from a json file and each document will correspond to a user, their tweets and other data. Existing documents in the tweets collection are dropped for each execution of load-json.py and the new documents are loaded from the respected json file.

PHASE 2:

- Functionalities Menu: The Functionalities Menu will be shown after the user has connected to the database 291db and will allow users to access six functionalities. Search for tweets, search for users, list top tweets, list top users, compose a tweet, and logout. After each functionality has passed, the user will be sent back to the menu
- Search for Tweets: Searching for tweets allows the user to input one or more key words and all tweets containing the word or words will be provided, the user can then choose to reply to the selected tweet or retweet it themselves.
- Search for Users: Search for users allows the user to provide a keyword and find all users whose display name or location contains the keyword. Each found user will then list the username, displayname, and location. The user will then be able to select a user and see full information about the user.
- List top tweets: A user selects a field 1. *retweetCount,* 2. *likeCount,* 3. *quoteCount* to display the top n tweets. The value of n will also be decided by the user. The resulting list will display, in descending order of the selected field, the persons who have matching tweet, display id, date, content, and username of the person who posted it. The fields of a person can also be viewed by a user after the user selects the desired person they want to view.
- List top users: A user is able to input a value to represent the top n users based on *followersCount*. The resulting list will have no duplicates and display each user with their username, displayname, and followersCount. The user can then select one of the displayed accounts to see that account's full user information.
- Compose a tweet: Prompts the user to enter the tweet content. The username will be automatically set to 291user,and date to the current date. All other fields will be turned to zero or null.
- Logout: The current user will be exited from the system.

Testing Strategy

Our general strategy for testing for any bugs would be to constantly go through the code and test different cases with the inputs to make sure all the functionalities were working properly. We set breakpoints between functions that were called within one another to determine how values that were passed were altered in order for proper output to be provided. Here are some of the common bugs we ran into:

- **Input bugs**: A few testing strategies to test for bugs in any of the inputs of the program, would be to try many different characters. Characters that were capitalized, numbers, special characters, etc. In many cases when inserting data into the database, certain inputs would not work.
- **Formatting bugs:** If the retrieved information isn't shown properly, change the formatting of the information
- **Proper function calls**: Some functions would return to different sub functions instead of returning to the proper one.
- **Try and except clauses:** For certain inputs, if it doesn't follow the needed input exactly, try and except until the correct input was given.
- **Putting code together (variable bugs):** When putting together each of our parts, some of the variables were different. There were some implementation issues that caused bugs when consolidating code.
- **Manipulating field values:** changing field values to test for edge cases especially with followcounts.

- **Using multiple iterations of from selection:** Concerned with list top tweets, a selected tweet was not shown. Needed to create list to allow the selected data to be iterated through multiple times.

Group work strategy

The group work for the project was split up between the functionalities while everyone worked on Phase one themselves as every member should understand how to load a json file into the database. One member would handle search for tweets, another would handle search for users, another would handle list top tweets, another would handle list top users and compose a tweet.

Khym Nad:

Responsible for: Search for users, load-json.py, documentation, testing
Time allocated: 10 hours

Akila edirisinghe

Responsible for: Search for tweets, load-json.py, testing, documentation
Time allocated: 10 hours

Samuel Chan:

Responsible for: List top tweets, load-json.py, documentation, testing
Time allocated: 10 hours

Andrew Zhang:

Responsible for: List top users and compose tweet, load-json.py, documentation, testing
Time allocated: 10 hours

Our main method of coordination was to meet up on campus and share the work we had done for each of our responsibilities. As well as that, we would fix each other's code to make sure the functions we were responsible for would work with one another. Communication was mainly through discord group chat where we displayed our progress once in a while.