designDOC Mp2

(a) a general overview of your system with a small user guide:

- This is a two-piece program. You must first load your data-set into an open MongoDB server using load-json.py.
- Ensure your .json is in the same directory as load-json.py, and have the port of your server on standby.
- Run load-json.py and provide the name of your .json and the port of your server as requested by the program. Load-json.py will notify you of any issues with the provided port or .json.
- After loading your data into your server, use phase2.py to navigate the data.
- The main-menu for phase2.py is navigated using indices for each available action - simply follow as requested by phase2.py. The given options are as follows:

1. Search for tweets
2. Search for users
3. List top tweets
4. List top users
5. Compose a tweet
6. Exit

(and a bonus action number 7 - Look at own tweets), which lets the user see their own tweets.

(b) details of your algorithms:

- For load-json.py, the program inserts the data into the server in batches of 125-1000 to account for limited memory space by incrementally filling an array

of documents until its capacity is at 125-1000, before flushing all documents in this array into the server.

- Search-tweets.py uses MongoDB's regex functionality and word-normalization in python to search for keyword matches while accounting for non-alphanumeric characters. It splits a given sentence into a list of words separated by spaces and punctuation, where each word is normalized by stripping it of all non-alphanumeric characters. Single keyword sentences are treated the same as multi-keyword sentences.
- The search_users() function utilizes MongoDB's $regex and $group operators to search for and group unique users based on matching keywords in displayname or location. Aggregation pipelines ensure duplicate elimination, and structured formatting is applied for detailed user profiles.
- The list_top_tweets() function queries the MongoDB database with sorting by specified fields (retweetCount, likeCount, quoteCount) using .sort() and .limit(). Results are displayed incrementally, and full tweet details are recursively formatted to handle nested fields.
- list_top_users.py lists the top n users from a MongoDB collection based on followersCount with n entered by user. It begins by inspecting a sample of the data to ensure the structure is valid and then uses a MongoDB aggregation pipeline to group documents by the username, extract the user's displayname, and determine their highest followersCount. The results are sorted in descending order of followers and limited to the top n users. Each user has the following details displayed: username, displayname, and followersCount with no duplicates. Additionally, it provides an interactive interface, allowing users to select a specific user for more details, such as their description, location, and other profile statistics. Error handling is included to manage unexpected issues during the process.
- The compose_tweet function allows a user to post a tweet by generating a unique tweet ID using uuid.uuid4() and recording the current timestamp with datetime.now().isformat().

  The tweet content, along with basic user information (a hardcoded username, "291user"), is structured into a dictionary and inserted into the

tweets_collection in the MongoDB database. The function also includes error handling to manage issues during insertion, ensuring robustness.

The uuid function is used to generate a unique tweetID, and the user is hardcoded as “291user” as required.

A helper function list_my_tweets was created as well to verify that tweets have been posted.

(c) your testing strategy

Most functions were tested on the 10.json file, as it contained enough users and tweets and was not too large like the 100.json file was. We verified whether the outputs of functions were as expected, and that they’re displayed in a user-friendly manner. We also considered invalid inputs in our functions and provided error messages for these cases. Additionally, we created fake-data to insert into 291db to ensure that our program works correctly.

Bugs encountered:

- encountered formatting issues with output for the details of tweets in list_top_tweets() function and users in search_users() functions (all details appearing on one line with minimal spacing - not user_friendly) and so we fixed this by impelmenting a a **recursive helper function** (display_full_info()) to handle structured and nested data (dictionaries and lists). The function ensures that each field is displayed line-by-line with proper indentation to reflect hierarchy.
- In compose_tweet, we needed a reliable way to make sure each tweet had a unique Id and hence imported the uuid module which would handle that. Each time a tweet is composed, the uuid.uuid4() handles the above requirement.
- load-json.py would crash after trying to extract all data from given .json file at once - fixed by extracting in chunks of 125
- search-tweets not accounting for keywords with punctuation. For example, running a search for tweets with “cant” and not returning tweets with can’t. Fixed by ignoring all non-alphanumerics (except spaces) inside each given keyword.

(d) your source code quality:

The source code is filled with comments explaining what certain parts of each function does.

The groupwork breakdown strategy included each one of us focussing on a couple functions at a time; Ali working on phase 1 and part 1 of phase 2, Jacob working on parts 2 and 3 of phase 2, and Adnan working on parts 4 and 5 of phase 2.

Each one of us worked for around 6 hours on our functions taking breaks every 2 hours to congregate and make sure all functions work together in our main file phase2.py, and that all merge conflicts are handled before working further.