General Overview

This project is a social media application implemented in Python using MongoDB. It allows users to perform various social media interactions such as composing tweets, listing top tweets and users, as well as searching for users and tweets based on specific keywords. The project creates a MongoDB collection, builds a document store, and then operates on it in a way which involves database management, user interactions, and data retrieval, providing a basic social media experience within a command-line interface.

User Guide

Upon entering the program, the user will be prompted to supply the json file name and the mongo port number. From here the user is presented with the menu as follows.

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

The user will be prompted to enter the corresponding number for the functionality they wish to perform, for example, the user should enter 1 if they would like to search for tweets. Within each functionality, the user will be guided to type specific keys (e.g 0 to go back). The messages to specify what key to press in each scenario will be clearly presented to the user.

Algorithmic Details

The program includes the following files: Main.py, Load_json.py, Search_tweets.py, Search_users.py, Compose_tweet.py, List_top_tweets.py, and Top_users.py.

Main.py acts as the entry position for the program. It starts by building a document store. It prompts the user to enter the name of the json file and the mongo port number and then calls load_json(json_name, mongo_port). This is designed to efficiently load a large JSON file into MongoDB by using parallel processing and optimizing the use of the insert_many method. The function establishes a connection to the MongoDB database and drops and recreates the 'tweets' collection to ensure a clean slate. We are also using multithreading to read the JSON file concurrently. The json_file_batcher function reads the file in batches, and the insert_batch function inserts each batch into the MongoDB collection using the insert_many method. This happens all the same time by concurrently processing and inserting data, and the batch processing minimizes the number of database insert operations. The user can adjust parameters such as batch_size and num_threads to adapt the loading process based on system resources and requirements. This will create a database named 291db, drops the current tweet collection if it exists, and creates a new tweets collection. Once the json file is loaded, the user will be presented with the main menu.

1. Search for tweets

The program prompts the user to enter space-separated keywords and will call the function search_users(keyword, mongo_port), which will first create case-insensitive indices for the display name and location field. It creates an index on the 'content' field to optimize keyword searches. If matching tweets are found (using boolean AND semantics), it displays relevant information for each tweet (ID, date, content, user who posted). The user can select a tweet by entering its number and view all fields of the selected tweet. The program then asks if the user wants to perform another search, and the process continues until the user decides to exit. The script ensures input validity with error checks throughout the user interactions.

2. Search for users

The program prompts the user to enter a keyword and will call the function *search_users(keyword, mongo_port)*, which will first create case-insensitive indices for the display name, location field and the followers count. With the line " If matching users are found, it displays relevant information for each user (username, display name and location). The user can select a user by entering its number and view all fields of the user, if they do not want to view any user then they can simply input 'exit' and will be returned to the main menu. After viewing the information of a certain user, the program also then returns to the main menu. The script ensures input validity with error checks throughout the user interactions.

3. Compose a tweet

This will prompt the user to enter the text they want to tweet and then will call the function *compose_tweet(mongo_port, user_id, text)* with the user_id set to "user291". This will insert a new tweet into the MongoDB collection using *insert_one*, constructing the tweet document with specific keys like 'url', 'writer', 'tdate', 'text', 'username' and sets None for other fields. An index on the 'url' field is created to improve search performance; this provides flexibility for additional indexes based on specific query patterns. The function is designed to handle scalability by incorporating MongoDB's efficient *insert_one* operation. The function is structured to balance simplicity, efficiency, and scalability for tweet composition in a MongoDB context.

4. List tweets

This will call the function *list_top_tweets(mongo_port)*, which will get the user's input for sorting and limiting tweets. The user can choose to sort tweets based on retweetCount, likeCount, or quoteCount. The function will then retrieve the top tweets based on the user-entered criterion and will display them. The user will then have the option to select a tweet to see its full information, which the function will then retrieve and display. In order to search for top n tweets I used this query, queried the collection with *'tweets_collection.find().sort([(field, -1)]).limit(int(n))'*, this sorts by descending order and only collects the top n tweets, hence the limit. This works well for databases of all sizes as it sorts the collection based on the specified field but only prints based on the limit the user has chosen. Furthermore indexes for the three fields were added which allows for speeding up searches.

5. List users

Selecting this option will call the function *top_users(mongo_port)*. This will use a set to keep track of unique usernames and will fetch and display the top users without duplicates. The user is then able to select a user and see their full information, which the function will retrieve and display. In order to speed up searches indexes for *followerCount* and username were added, this allows for a quicker search of the relevant data to the functionality.

6. Exit

Calling on *Exit* will simply break the *while* loop, allowing the user to exit the program. This is the only way to exit the program.

Testing Strategy

Our testing strategy involved various different methods. We each conducted individual tests on our respective functions to test base functionality. This negan with ensuring we would get results returned and invalid inputs would be disregarded. Further we tested against the 10.json file from the google drive. We cross checked against the json file to ensure our outputs were as expected. For more individual functionality testing we created smaller json files, created collections and tested those. For example, for top users, we created a small collection where a user has multiple entries, each with a different followers count, then tested to ensure only one copy of that user was returned but with the max follower Count they have. Similar tests were conducted for each functionality to test its individual components. Then once we compiled each of our functionalities together we tested against the 100.json file against cross checking output with the database to ensure they were accurate. We tested various keywords, various inputs for n and composed various tweets. All which we found to work as expected. We went through the rubric and conducted tests for each function to ensure criteria was met. Such as for search for tweets we tested; one keyword, multiple keywords, existence of AND semantics in searches, checking matches were case insensitive, ensuring the id, date, content and username for matching tweets are shown, and that a user can select a tweet. We went through tests just like this for each functionality as outlined on the rubric.

Source Code Quality

In order to keep up the quality of our source code we followed different tactics. First off we ensured to add comments throughout our code to help anyone reading the code understand what is going on in the not so obvious locations. Further we kept a consistency in our functionalities, such as for search users and list top n users, the user utilizing the program will find they work quite similarly, As both give the user the same invalid entry message, and both follow a similar structure to deal with out of range values. Further, each functionality has the user using the program select the number (according to a list) of the tweet or user they want to see more information about, this is consistent throughout the program (rather than having a user select an Id in one case, a name in another and so on). Finally we made our program modular, such that each functionality is in a different file, this allows for improved clarity and readability. Further, if an issue arises we can simply go edit the file that corresponds to that specific issue/functionality rather than messing with the whole program.

Group-Work Strategy

We used a collaborative approach to stay on track. Regular discussion meetings were held, both virtually and in-person, to discuss progress, identify and address any issues, and ensure that all components were integrated well. Regular text-based communication was also utilized to provide quick feedback. The functionalities of the project were split up as follows amongst the four team members, each with the responsibility of creating their task component.

1. *Search for tweets - Francis Garcia*
   Time spent: 3-4 hours
   Progress made: The strategy I developed revolves around using the pymongo library to connect to a MongoDB database and efficiently search for tweets based on user-specified keywords using boolean AND. The initial steps involve

establishing a connection to the MongoDB instance and accessing the relevant database and collection. To enhance search performance, an index is created on the 'content' field. The code uses a continuous loop structure, asking the user to input keywords, constructing a MongoDB query for matching tweets, and then displaying the results. The code also implements error-checking to ensure valid user inputs and handles scenarios where no matching tweets are found. The code also allows users to select specific tweets of interest and view detailed information about them (shows all fields). The process concludes with an option for users to decide whether to perform additional searches or exit the program. Overall, this strategy combines efficient database querying, user-friendly interaction, and error handling to create a proper functional tweet search tool.

2. *Search for users - Cejiro Roque*
   Time spent: 3-4 hours
   Progress made: I built the code in search_users.py. The first thing I did was write the query to get users that match the keyword and order them accordingly. Then I made sure to enumerate each user, because I coded it so that all users matching the keyword are returned. Then I cleaned up the overall results by adding a blank line between each user. I then started to think about how to retrieve the information of the user. At first I was getting back all the information contained within the tweet that that user wrote. Eventually I found out that I am able to only pull the user information and print it all. I added the functionality to see user information and then added spaces for overall neatness and readability. I then finished the function by adding error checking.
3. *List Top Tweets - Ruchali Aery*
   Time spent: 2-3 hours
   Progress made: In the first roll through of working on List Top Tweets, I added the base functionality where a user can input one of the three fields (retweetCount, likeCount, quoteCount) and then input the n value for how many results they would like to see. I queried the collection with ‘tweets_collection.find().sort([(field, -1)]).limit(int(n))’, this sorts by descending order and only collects the top n tweets, hence the limit. I put the collected values into a list so that it could be enumerated, to allow users to further select specific tweets. I then went back and error checked all the inputs to ensure all values within range would be accepted and others would print an invalid message to users. So the final and current version has functionality and error checking.
4. *List Top Users - Ruchali Aery*
   Time spent: 2-3 hours
   Progress made: Initially I had List Top Users, collect the top n users based on followers count and then print out the specified fields. However this had issues with duplication. So I went back and added a uniqueUsername array to ensure no duplicates, further based on implication added to the spec, if a user had multiple entries we must return their highest followers Count. To do this I instead incorporated a max_follower_count dictionary, this would update users follower count as it iterated through the collection so each user is associated with their maximum count. After this I incorporated checks to make sure inputs entered were valid and within range. The final version of this function lists top n users based on the following count without duplicates. In order to check for returning the max follower Count I created a test collection with one user who had three tweets and three different follwerCounts and ensured the max value was returned.
5. *Compose Tweets - Ayra Qutub*
   Time spent: 3-4 hours
   Progress made: My code builds off of the original code from project 1, which was a Python script using SQLite to compose and store tweets. The revised code migrates to MongoDB, enhancing scalability and flexibility. I built the `compose_tweet` function to construct a tweet document with explicit fields like 'url', 'writer', and 'tdate'. The url was created by looking at the format of the provided tweets. I wrote a function to create unique tweet ids for each tweet. The other fields for tweets were set to None. The function explicitly includes fields with `None` values for better representation of None values in the MongoDB collection. I used an AI chat model to extract these fields from the json file.