Project Report

General Overview

Phase 1: Building a Document Store

- Begin by configuring the MongoDB port. Then, execute the load-json.py script using the prescribed command format:
  load-json.py <json_file> <mongodb_port>
- Upon completion of data loading, users may proceed to phase 2.

Phase 2: Operating on the Document Store

1. Starting the Program:
   - Run the program in the command line (main.py).
   - Enter the port number for the MongoDB server when prompted.
2. Main Menu:
   - Upon launching the program, the main menu will be displayed.
   - Users can perform various operations by typing the number corresponding to the desired action.
3. Choosing Options:
   - Each option in the menu is represented by a number.
   - To select an operation, type the number and press Enter.
4. Available Operations:
   a. Search for Tweets: Enter the keywords separated by spaces or punctuation. The system will retrieve tweets matching all provided keywords.
   b. Search for Users: Enter a keyword related to username, display name, or location. Users matching the keyword will be displayed without duplicates.
   c. List Top Tweets: Choose the field (retweetCount, likeCount, quoteCount) to sort by. Enter the number of top tweets (n) to display.
   d. List Top Users: Enter the number of top users (n) based on followersCount.
   e. Compose a Tweet: Input the content for a new tweet. The system will insert the tweet with system date and username "291user".
   f. Return to Main Menu: After completing an action, users can return to the main menu for further operations.
5. Viewing Results: Upon execution of an operation, the system will render relevant data in a numbered list based on the user's input. Users have the option to access comprehensive information for specific tweets or users by selecting the corresponding number in the list, allowing them to view all associated fields or complete data.
6. Exiting the Program: Choose the exit option from the main menu to close the program.

Algorithm

1. Loading json:
   The process involves connecting to the MongoDB server, creating a database and collection, and subsequently loading JSON data. This is achieved by opening the specified JSON file and parsing it line by line, treating each line as a tweet in JSON format. For each successfully parsed tweet, it's added to a batch list. When the batch reaches a preset size (e.g., 1000 tweets), it's inserted into the MongoDB collection using insert_many, and the batch is reset. Any remaining tweets, forming a smaller batch than the threshold, are then inserted into the collection to ensure the complete transfer of JSON data into MongoDB.
2. Searching for tweets:
   The search_tweets function performs tweet searches within a MongoDB collection based on provided keywords. Initially, it accesses the 'tweets' collection within the MongoDB database. It constructs an updated regex pattern that incorporates word boundaries for precise matching of keywords within the 'content' field. This regex pattern

is applied to the 'content' field using a case-insensitive search. After executing the query and retrieving matching documents, the function returns a list containing the results, including all fields from each matching tweet document. In case of any exceptions during execution, the function handles errors by clearing the console, displaying an error message, prompting for user interaction, and ultimately returning an empty list. This function provides an improved approach for searching tweets within the MongoDB collection, ensuring accurate retrieval based on specified keywords with word boundary considerations for better precision.

3. Searching for users:
   The search_users function facilitates user searches within a MongoDB collection based on provided keywords. Initially, it splits the input keywords string and isolates the first keyword for search focus. It then constructs an updated regex pattern, ensuring partial matching with word boundaries for precise search results. This pattern is applied to fields user.displayname and user.location within the MongoDB collection. After retrieving matching documents, the function processes the results, extracting user details and organizing them into a dictionary to maintain uniqueness based on usernames. Finally, it returns a list containing the unique user data based on their usernames, ensuring accurate user retrieval. In case of any exceptions during execution, the function handles errors by clearing the console, displaying an error message, prompting for user interaction, and ultimately returning an empty list. This function provides an enhanced approach for accurate and targeted user searches within the MongoDB collection based on specified keywords.
4. Searching top tweets:
   The search_top_tweets function aims to retrieve the top 'n' tweets from a MongoDB collection based on a specified field for sorting. It begins by accessing the 'tweets' collection within the MongoDB database. Using an empty query (to match all tweets), the function sorts the tweets in descending order based on the provided field. It then limits the results to the first 'n' tweets after sorting and returns these tweets as a list. In case of any exceptions during the process, the function handles errors by clearing the console, displaying an error message, prompting for user interaction, and ultimately returning an empty list. This function provides a streamlined way to fetch and present the top 'n' tweets based on a specified field from the MongoDB collection.
5. Searching top users:
   The search_top_users function retrieves the top 'n' users with the highest followers count from a MongoDB collection. Initially, it accesses the 'tweets' collection within the database. Using an aggregation pipeline, it groups tweets by the 'username' field, preserving the user details. Then, it sorts these grouped users in descending order based on their 'followersCount' and limits the results to the first 'n' users. After executing the aggregation pipeline, the function extracts and constructs a list containing the user details of the top users based on their followers count. In case of any exceptions during the execution, the function handles errors by clearing the console, displaying an error message, prompting for user interaction, and ultimately returning an empty list. This function provides an efficient way to fetch and present the top 'n' users with the highest followers count from the MongoDB collection.
6. Compose a tweet:
   The compose_tweet function allows users to create and insert a new tweet into a MongoDB collection. Initially, the function prepares a tweet document containing various fields such as content, date, user details (with predefined '291user' username), and other optional tweet-related fields set to null. Using the MongoDB Python driver, the function accesses the 'tweets' collection within the specified database. It inserts the prepared tweet document into the collection using insert_one. Upon insertion, it checks the result to determine if the tweet was successfully added to the collection. If the insertion succeeds, it displays a success message; otherwise, it prints a failure message. In case of any exceptions during the tweet composition or insertion process, the function handles errors by clearing the console, displaying an error message, and prompting user interaction for acknowledgment. This function offers a straightforward method to compose and add new tweets into the MongoDB collection, enabling the insertion of user-generated content.

Testing Strategy

The testing strategy involves a systematic approach to validate different functionalities of the system across various scenarios and database sizes, focusing on precision, performance, and diverse input conditions. Here's an outline for testing based on the mentioned features and scenarios:

1. Testing Process:
   Conduct initial testing on a small database to verify basic functionality and precision of outputs. Scale up to a larger database size to assess system performance in handling increased data volume. Record and analyze time processing for each feature under varying database sizes.
2. Testing Scenarios for Different Features:
   - Searching for Tweets: Test on single and multiple keywords for precise retrieval. Validate search functionality with single and multiple hashtags. Test combinations of hashtags and keywords to ensure accurate results.
   - Searching for Users: Test user search by display name and city for accuracy. Verify if the output contains duplicate entries or handles duplication correctly.
   - Top Tweets Search: Test for different fields (retweetCount, likeCount, etc.) with varying output sizes. Validate precision for few output entries as well as larger output sets.
   - Top Users Search: Gradually increase the number of users to display and ensure accurate retrieval. Validate precision and performance when displaying varying user counts.
3. Metrics and Tracking:
   Record time processing for each feature across different database sizes. Monitor and track the precision of output for search functionalities.
4. Error and Exception Handling:
   Test edge cases and unexpected inputs to verify proper error handling. Ensure the system responds appropriately to errors and exceptions.

Source Code Quality

1. Readability and Maintainability: The code is easy to understand and follows a consistent naming convention for variables, functions, and classes. The code includes descriptive comments and documentation explaining complex logic and algorithms.
2. Coding Standards: The code follows a consistent coding style and indentation throughout. Each function or feature of the program is written in a separate method and placed in the appropriate file.
3. Error Handling and Robustness: The code includes proper error handling mechanisms to manage unexpected scenarios gracefully. Inputs are validated to prevent potential vulnerabilities or errors.
4. Testing and Quality Assurance: We evaluate the extent of test coverage to ensure the correctness and reliability of the codebase. We have the testing strategy to identify bugs and validate functionalities.
5. Efficiency and Performance: We ensure optimal performance for algorithms and methods.
6. Version Control and Collaboration: We use GitHub for collaboration and tracking changes. The code is reviewed carefully when a member finishes their task.
7. Error Handling: We ensure the system responds appropriately to errors and exceptions.