# CMPUT 291 Project 2

**Overview**: The application consists of 2 components. The first component allows the user to parse raw data in json format into a MongoDB database in a single collection called "tweets". The second component is a program that allows users to perform operations on the generated database such as searching for tweets with keywords, searching for users based on keyword matching display name, listing top n tweets based on a certain field's values, listing top n users based on followers count and composing a tweet to insert in the database.

**User Guide**: For the load-json.py file, user must first ensure that the MongoDB server is up and running like expected. After starting the program, the user must provide a valid file path to the data file and valid number for port to run the database in.For the main program, user must first provide a valid port for the mongodb server. The user is then presented with a menu with 6 different options, user must enter number ranging from 1-6 to correspond with one of these commands. For search for users and search for tweets function, users must enter valid keywords to search for matches, after all the matches are returned, user has a choice to select a match by assigned index to see more details or leave back to the main menu. In list top tweets and list top users, user has to provide with a valid n number to return n results and in list top tweets, user must select a field to rank the tweets on. Just like the others, user has a choice to select a result by index to see more details. In compose tweets, user only has to enter the new content of the new tweet and the system will insert into database with the correct values for the required fields.

**Algorithms Details**: We will discuss the details of the application in 2 parts each corresponding to one phase of the project.

- Phase 1:
  - We completed phase 1 of the project within 1 file named load-json.py. We first ask user to provide relative path of the raw data file in json format to the command line as well as the port number we want to run the mongodb server on
  - We then try to open the provided json file by accessing its relative file path. Using a separate function called connect_mongodb(), we set up a connection to the mongodb server using the number of ports provided from the user in the previous step. After the local connection with mongodb server is established, we choose to use a database named "291db" and check whether the collection "tweets" is already in the database. We remove and drop the existing collection if it is and create a new collection with the same name.
  - After establishing a variable storing the values for local mongodb "tweets" collection, we then forward that variable to the insert_data() function to parse every single entry from the file into the program, then convert the entry into json format and insert into the collection "tweets" in mongodb database with 1000 entries at a time until the end of the provided file.
- Phase 2:
  - We decide to break the project down into different components handling different functionalities of the program. Every core functionalities are stored inside the Controllers folders, the UI commands are stored inside Commands and mongodb queries are stored in MongoDB folder. We initialized the program using the main.py file inside the root directory.
  - In main.py, we present the user with 6 different choices corresponding to 5 core functionalities of the program and option to exit. We note that all UI commands presented in the terminal mostly come from the Commands folder. If user select a number corresponding to a specific functionality, the main.py program will call the corresponding functions in each of the files in Controllers
  - In search_for_tweets.py, we first ask user to provide the keywords by typing them into the command line, we then preprocess the keywords in keywords_preprocessing() function to output only keywords in alphanumeric forms in a list with all punctuation and space removed. We then

search for the tweets content in the database that have the content including those keywords provided. Since the tweets returned using the mongodb query search may have the keywords a substring of a word in the contents, we have to further filter the tweets by preprocessing the words in the contents of the tweet into alphanumeric words without any punctuations. We then test to see if there are exact matches between the words in the tweets contents with the keywords, only then we return the tweet as a valid tweet that satisfies the matching requirements. We then use the print_tweets to output the resulting tweets and we print every field and its value of the chosen tweet from the user to give the user a more detailed look of individual tweet.

- In search_for_users.py, we also first ask for user input of the keyword in the terminal, we then search for the users using the distinct() function in pymongo and the returned list contains every unique id of users that have their name or location containing the keyword, since the name and location of the user returned may contain keywords as substrings, not matching them as whole words, like the previous steps, we need to do further filtering to find the user that have their name or location matching the keywords as whole word. We then ask the user to input the option to see more details about a specific user or go back, which we print all fields and their respective values in case the user chooses to see more details about a user.
- In list_top_tweets.py, we first ask user to pick an integer n to list n tweets as a result, we then give the user options to rank the tweets based on repliesCount, quoteCount or retweetCount. Once we obtain values for n and the choice of field to rank the tweets, we then use aggregate function in mongodb to rank the tweets based on the chosen field in descending order and limit the result to n tweets. Using the print_tweets() function in search_for_tweets.py, we can output the ranked tweets to the terminal.
- In list_top_users.py, we first ask the user to provide the integer n to output n number of users based on followersCount. We then use the aggregate function in mongodb to search group all users with the same id values together, designating the "_id" field as the user's id, username, displayname and the "followersCount" as the maximum value available for the field, and limiting the returned list to only n users. We then print the details of the users on the terminal, after which we present the user with the choice to see more about a user or go back to the main menu. We display every field and its respected values for the user details if the user wants to see more about a user.
- In compose_tweets.py, we first ask the user to enter the content of the tweet, after that, we use mongodb queries insert_one() to insert the newly created tweets with "content" field being the user input text and all the other necessary fields populated as specified in the instructions.

**Testing Strategy**:

- We decided to upload and create a MongoDB database with the raw data in json format provided and use them as our main source for testing. Using the MongoDBCompass application, we monitor the changes and accuracy of the queries in the application in obtaining or inserting the right data to the database collections.
- We tested our program as we finish any modules or functionalities as we recognize a huge potential for bugs pile-up if we leave our testing phases till we have finished everything. We also test the functions independently using unit tests as there are parts of the code where one function may depend on the output of the others.
- We rigorously test the integration of all our work in the end to check the dependencies between modules and functions and to recognize if there are any inconsistencies between our code.
- We also come up with edge cases and uncommon inputs to check the integrity and error handling of the program

**Group work break-down**: Our **main method of communication** includes setting up a group chat on a messaging platform to exchange and discuss the progress of each other's work on the project. We also occasionally met together in an online call to further help with task assigning and problem solving purposes. The tasks were evenly divided for each member of the group and can be seen in the provided table. We split the repository into 3 different branches to work on each of the tasks that we are assigned with, and in the end integrated everything into the main branch, though we decided to remove all other branches other than main to prevent further confusions or potential mistakes in using our incomplete code in our other branches.

| **An Huynh** | **Thasanka Kandage** | **Joshua Chui** |
|---|---|---|
| - Worked on the load-json.py to load the data into database<br>- Worked on search for tweets and search for users functions in the second component of the project<br>- Spent approximately 14 hours on the project | - Worked on the listing of top n users and list top n tweets in the main functionalities of the second component<br>- Monitored and managed user interface and commands displayed to the user of the program<br>- Spent approximately 13 hours on the project | - Responsible for finishing the composing tweets function of the program in the second part of the project<br>- Assisted on the layout of the project and helped to plan multiple query commands and controller files<br>- Spent approximately 13 hours on the project |

**Assumptions and potential issues**

AS-1:The raw data provided in JSON format is error free and matches the expected structure
AS-2: The MongoDB database is correctly configured and running without an issue for the user
AS-3: MongoDB Compass or MongoDB shell is available for the user to allow for monitoring and debugging.
AS-4: The user is assumed to have a working knowledge of MongoDB to access the database and use MongoDBCompass or MongoDB shell
AS-5: All dependencies and necessary libraries are installed with the latest and compatible version before running the program.
AS-6: Application returns all relevant data if user doesn't enter anything for the keywords.

PI-1: Extremely large datasets may impact the performance of the application
PI-2: The user may experience issues using the application if using different versions of MongoDB
PI-3: User might not be able to search for newly composed tweets and users created in the application due to id field set to null