General Overview

This project is a social media application implemented in Python and SQLite. It allows users to perform various social media interactions such as composing tweets, following other users, displaying tweets, retweets, and user information, as well as searching for users and tweets based on specific keywords. The project involves database management, user interactions, and data retrieval, providing a basic social media experience within a command-line interface.

User Guide

When you launch the system, you will be presented with a login screen that provides options for both registered and unregistered users. Registered users can log in using their user ID and password while unregistered users can sign up by providing their name, email, city, timezone, and a password. The system will generate a unique user ID for them. Once logged in, you will be prompted to either view the tweets or retweets of the users you follow. If there are more than 5 tweets, only the latest 5 are shown, and you can choose to view more, 5 at a time and you can select a tweet to view the number of retweets and replies.You can also compose a reply to a tweet or retweet it to share it with your followers. You then can return to the main menu which consists of 5 options. You can view the tweets and retweets again, search for tweets, search for users, write a tweet and view a list of your followers.

Pressing 2 will allow a user to search for tweets by entering one or more keywords (ordered by date). After that you can select a tweet to view more information about it. After some prompted key inputs, you will be returned to the main menu. Pressing 3 will let you search for other users by entering a keyword that matches their name or city. If there are more than 5 matching users, you can choose to view more, 5 at a time. You can then select a user to view more information about them (number of tweets, the number of users they are following, the number of followers, and their three most recent tweets). After that you can follow the selected user or choose to see three more tweets from them. After some prompted key inputs, you will be returned to the main menu.

Pressing 4 will let you compose and post your tweets. Your tweets can be viewed and searched by your followers. Once the tweet is posted, you will go back to the main menu. Pressing 5 will show a list of all users who follow you and from which you can select a follower to view more information about them. After that you can follow the selected user or choose to see all their tweets. After some prompted key inputs, you will return to the main menu; from there, you can logout by pressing 6, returning you to the login screen. Pressing 3 on the login screen will shut down the program.

Detailed Design

The program includes the following files: Main.py, Search_tweets.py, Search_users.py, Compose_tweet.py, List_followers.py, Display_tweets.py , User.py.

We have also included a file first_version.py which shows our initial code implementation. In this version, we built a “driver code” which shows a rough outline of all the functionalities required in the project. However, each function (functionality) does not meet all requirements (i.e. list_followers, only listed the followers and did not allow us to see their tweets). From this first version, we modularized the code, creating specific files and functions for each functionality as indicated by the file names above. This allowed editing each functionality to be easier without breaking other functions. Further, we developed main.py which directs the use of each of these functions, by creating the menu and allowing the user to choose what functionality they want to see.

In our final version, main.py, acts as the entry position for the program. It starts by initializing the SQLite database connection and allows the user to enter a database. It then enters the main menu, from which the user can choose to login, register an account, or exit the program.

If the user chooses to login or register an account, the function calls login() or register(), respectively, both of which are located in user.py.

- login(conn)
  - This function asks for user input for the username and password. It then checks to ensure that these are valid.
- register_user(conn)
  - This function asks users to enter their information and enters it into the table users.

Once the user is logged in, they are prompted with a question asking if they would like to see the tweets or retweets of the people they follow, or return to the main menu which is where they will be given the following options, which they can select by entering the corresponding digit.

1. Back to Display tweets or retweets

2. Search for tweets
3. Search for users
4. Compose a tweet
5. List followers
6. Logout

1. Display tweets or retweets
   - This will call the function '*display_tweets_and_retweets(conn,user_id)'* (located in display_tweets.py). This function will then call either *display_tweets(conn, user_id)* or *display_retweets(conn,user_id)*, both of which are located in the same file.
     - *display_tweets(conn, user_id)*
       - This function will enter a loop, which continues until the user no longer wants to see any more tweets. Within this loop, the function executes a SQL query to fetch tweets. This retrieves the tweet information, selects tweets where the writer is in the list of users followed by the current user, and orders them by their date in descending order, displaying 5 at a time. If no tweets are retrieved, the function prints "No tweets to display". Further for tweets the user has the ability to select a tweet based on its ID from any tweet written by a user they follow, they will also be able to reply or retweet the selected tweet. If they choose to reply, this will then call the *compose_tweet(conn, user_id, tweet_text, replyto)* located in compose_tweet.py.
     - *display_retweets(conn,user_id)*
       - This does much the same as display_tweets() but for retweets instead of tweets. It selects retweets where the user who made the retweet is in the list of users followed by the current user.
2. Search for tweets
   - This will prompt the user to enter keywords and then will call the function *search_tweets(conn, user_id, keywords)*, which is located in search_tweets.py. The function executes an SQL query to search for tweets. The query orders the matching tweets by their tweet date in descending order. If matching tweets are found, the function '*display_tweet_list(conn, user_id, tweets)'* is called, (located in search_tweets.py). This allows the user to view the search results.
     - *display_tweet_list(conn, user_id, tweets)*
       - Within the loop, the function iterates through the tweets list, starting from the count index and going up to a maximum of 5 tweets or until there are no more tweets left to display. After each set of 5 tweets, it prompts the user to see more, go back, or see information about a tweet in which case the function *display_tweet_info(conn, user_id, tweet)* is called.
     - *display_tweet_info(conn, user_id, tweet)*
       - This code prints detailed information about the selected tweet, including its ID, writer, date, text, reply-to reference, retweet count, and reply count.
3. Search for users
   - This will prompt the user to enter a keyword and then will call the function *search_users(conn, keyword, user_id)*, which is located in search_users.py. The code executes an SQL query to search for users whose names or cities contain the specified keyword. It uses the LIKE operator in the SQL query to perform a partial string match, and it orders the results based on two criteria: First, it orders by matched name, in descending order of name length. Second, it orders by the matched city, again in descending order of name length. It displays these 5 at a time, with the option for the user to see more. If the user selects a specific user by entering a number, the code attempts to fetch more details about the selected user using the *get_user_info(conn, user_id)* function and then displays it with the *display_user_info(user_info)* function.
     - *get_user_info(conn, user_id)*
       - The function uses SQL queries to retrieve information about the user, including their tweet count, following count, followers count, and their recent tweets.
     - *display_user_info(user_info)*
       - This function takes the output from the last function as an argument and displays it.

     The user can then choose to follow the selected person, upon which the code then runs an sql query to insert into the "follows" tables. They can also choose to view more tweets, and display it using the *print_more_tweets(user_info,count)* function.
     - *print_more_tweets(user_info,count)*

- This function takes the output from the get_user_info function as an argument and prints the next 3 tweets

4. Compose Tweets
   - This will prompt the user to compose a tweet and then call *compose_tweet(conn, user_id, tweet_text, replyto)* with replyto set to null (in compose_tweet.py). This first finds any hashtag terms present in the tweet. It stores the tweet itself into the tweet table; its tweet id is found through the *get_unique_tweet_id(cursor)* function, which is also located in compose_tweet.py. If the hashtag is new, it is stored into the hashtag table. Tweets with hashtags are stored into the mentions table.
5. List_followers
   - This will call the function *list_followers(conn, user_id)* which is located in the list_followers.py file. This function will first list the id of the users the logged in user follows,then the user can select one of these followers to see more information, including name, number of followers, number of tweets, and the three most recent tweets, which will call *get_follower_info(conn,user_to_check)*, also located in list_followers.py. The user then has the choice to follow this follower (if they already follow them and they select this choice they will be told they already follow), or see more tweets which will display every tweet the specified user has beyond the 3 most recent, using the function *print_tweets(conn, user_to_check)*, again located in list_followers.py.

Testing Strategy

We executed a detailed testing plan. The first thing we did is outline what we needed to check, that being how we handled invalid user inputs, ensuring we return what we expect based on our test database, and ensuring we meet all the requirements described by the rubric and functionality description on eclass. To build the test database, we first created tables using the schema given on eclass, then to populate data we asked ChatGPT for various input statements, to fill in values in the table. This condensed the time needed to populate tables, and so when testing we cross referenced the database to ensure the output was as expected. To start we created a checklist that had all the points from the rubric grouped by the functionality they were for, and from that added on additional requirements based on the eclass spec. We then proceeded to test various cases, and took down notes for each test to ensure we could review it and capture anything we missed. On our initial test we conducted 25 test situations. This concluded the test cases we did as a group. Prior to that each individual ran various test cases against their own functionality, to handle errors, and confirm output is correct.

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 project's codebase was managed through Google Collab for real-time updates.

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: ~5-6 hours

   Progress made: I built the code 'search_tweets.py'. Initially, I developed the queries needed to fulfill the given specification on eclass (with agent). In the first version of the program, I made functions for searching the tweets by creating a SQL statement that selects distinct tweets that contain the keywords passed by the user input. I added error protection code such as when the tweet number selected is invalid (when the input is a string) or out of range. By looking at the given specification, I also added a functionality where we display the tweet information. Inside this 'display_tweet_info()' function, I have implemented the functionality of giving the option of retweeting the selected tweet or composing a reply. This is done by calling the 'compose_tweet()' function developed by Ayra to create replies for a tweet (a reply is a tweet). The only difference between a reply and a composed tweet is that in my code, the tweet id is set to the tweet id of the selected tweet whereas in composing a tweet, the replyTo would be Null as the composed tweet is not a reply. Then, the 'retweet()' function made by Ruchali is called when the user decides to retweet the selected tweet.

2. Search for users - Cejiro Roque
   Time spent: ~4-5 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. I then figured how to only print 5 at a time. Then I added more functionality to the code, to be able to get more information on users in the search results, and further allowing the user to choose to follow the person or see more tweets. I then went back and made sure that all cases of input are accounted for, which means incorrect inputs are properly handled.

3. Compose tweets - Ayra Qutub
   Time spent: ~2-3 hours
   Progress made: Built the code compose_tweets.py. Connected to the database, found hashtags, and inserted the tweet and hashtags to the appropriate tables. After it became clear that this function would be useful to call in search_tweets in order to reply to a tweet, this was edited to take in the replyto tweet id instead of automatically setting it to null. After testing, the next change was made to create a function to get the tweet id. Originally, tweet id was found through tweet_id = cursor.lastrowid. This was changed to the current version so that the tweet id's increased in order.

4. List followers - Ruchali Aery
   Time spent: ~2-3 hours
   Progress made: I built the code in list_followers.py, using based code written by Francis, which simply listed the followers the logged-in user had. After that I added more functionality to the code, to be able to get more information on followers, and further choose to follow the follower or see more tweets as specified in the guidelines. Further, I tested the functionality by adding test data to a test database that would check different parts of the code, i.e. a user with no followers, a follower with no tweets, a user attempting to follow a follower they already follow, and other base-level tests.

5. Login Screen - Francis Garcia and Ruchali Aery
   Francis:
   Time spent: ~4 hours
   Progress Made: By referring to the given relational schema and specification, I was able to develop a login screen that can implement different functionalities. I made the code so that it can be easily modified and account possibilities for new functions. When running the main code, it should prompt the users the correct flow of the program (entering the database name, login, main menu).
   Ruchali:
   Time spent: ~4 hours
   Progress Made: After Francis had developed the menus for the main screen, I went in to update the display of the tweets. After my updates, the user can choose to go to the Display Tweets or Retweets from the menu and from there can choose to see tweets or retweets. For tweets, the user can see (5 at a time) the users of the people they follow and can choose a tweet to see info (reply and retweet count), the user can also go and see (5 at the time) retweets retweeted by the users they follow. I also updated the user registration to create a unique user Id for new users. Further, I allowed users to choose to reply to or retweet a selected tweet, I incorporated the compose_tweets in this functionality.

In the main function (main.py) which was primarily designed by Francis, I made the program modular so that my group mates and I should be able to create our assigned functionalities without clashing our code. The main code should import the different modules we created and call the functionalities in the main code. Ruchali added the functionality for being able to read in a file, rather than hardcoding, and also ensuring the file exists in the path.

The functionalities were then edited in a way that they could be called independently. This included fixing parameters and ensuring that the cursor was initialized within the function. Then, we wrote the detailed design of our project. This was done by Ayra and took 2-3 hours. We then came up with a basic testing strategy which included the creation of a database, made by Cejiro with the aid of a chat agent; this took 1 hour. We robustly tested every aspect of the code together to ensure the system worked correctly; we spent approximately 12 hours on testing.