DESIGN DOCUMENT - TWITTER implementation using SQLITE3 and PYTHON

GENERAL OVERVIEW OF YOUR SYSTEM WITH A SMALL USER GUIDE

The Twitter Program offers users the choice to log in, sign up, or exit. Upon signing up with valid details, a User ID is generated. Logging in needs a valid User ID and Password. Afterward, users see their 5 most recent following tweets, allowing them to choose a tweet, follow another user, navigate menus, or switch pages. Viewing a tweet displays replies, retweet count, and options to reply or retweet it. Other menu:

1) ***Search Tweets***: Allows for multiple keywords and prints 5 of the filtered tweets from latest to oldest.
2) ***Search Users***: Inputted keyword filters users and prints matching names with increasing length first and then matching cities with increasing length of city names. User is allowed to select another user and view some of their account details alongwith three of their latest tweets.
3) ***Write Tweets***: Allows the user to compose a tweet with or without #s.
4) ***Show Followers***: Allows the user to view their followers and select any follower.
5) ***Log Out***: Finally, when the user wishes to log out of their account, this allows them to do so.

DETAILED DESIGN OF OUR SOFTWARE:

Our software currently uses a variety of different functions to prevent redundancy of code along with several imported modules and globalized user defined variables.

Imported Modules

1) sqlite3: Gives us the method of connecting python to sqlite3.
2) string: Gives us the string formats to validate different user inputs.
3) re: Using re.match(), we can check the format of the email input.
4) getpass: Allows users to input passwords, WITHOUT seeing input.
5) datetime: To insert today's date into the tables which require a date.
6) sys: Helps exit the program when the user wishes to log out using sys.exit(0).
7) os: Using os.system('clear'), we can clear the output buffer.
8) time: Using time.sleep(5), the program sleeps for 5 seconds.

Global Variables

1) connection: Set to None, given a value that connects the database through **sqlite3** module.
2) cursor: Set to None, given a value that provides a cursor to execute the queries.
3) MAIN_USER: Set to None, after a user logs in/signs up, it sets the value.

Notes

- ➢ Users are almost always prompted to try inputting again if invalid input is entered.
- ➢ **os.system('clear')** and **time.sleep(5)** have been used to clear the output screen and pause accordingly to imitate website flow from one page to another.

User Defined Functions

1) main(): Connects to the inputted database using **connect(path)** function, and calls **home_page()**.

2) connect(path): Sets values of globalised connection and cursor through the imported sqlite3 functions.

3) home_page(): Entering the main home page, the user's provided with 3 choices:

- User enters 1: They're choosing to log in -> **login_site()**
- User enters 2: They're choosing to sign up -> **sign_up_user()**
- User enters 3: Exits the website.

4) sign_up_user(): This function uses imported **string module's formats** through sub-functions such as:

- generate_username(): Username inputting and validation by (check_username_or_city(user_name))
- generate_city(): City inputting and validation through (check_username_or_city(city)).
- generate_email(): Email inputting and validation through (check_email(email)) using **re.match**
- generate_password(): Password of minimum 8 characters validation by(check_password(password)).

All these validated inputs are inserted into the *users table* and a unique *usr* is generated. Finally, **view_login(user_id)** is called.

5) login_site(): SQLite query checks if inputted *usr* and *pwd* exists in the *users table*. If yes, signs in user using **view_login(*usd*)** and **getpass** module, otherwise prints an error message and offers multiple tries.

6) view_login(user_id): Sets user_id value to the global variable MAIN_USER. Logged in user's following's most recent tweets and retweets are displayed through print_tweets(rows,modulus,count). Then more choices are given. Informs users to follow other users to view their tweets.

- S/s to select a tweet -> **user_inputted_select(temp_list[0])**
- B/b to backtrack if count > 5 -> goes to previous page using **print_tweets()**
- F/f to follow selected user -> **user_inputted_follow(temp_list[1])**
- M/m to enter the menu -> **menu()** is called

7) print_tweets(rows, modulus, count): Prints the required number (modulus) of tweets. Accepts a count argument that helps traverse the list (rows) and backtrack. The list, output buffers and count are emptied or decremented as required when the user wishes to backtrack. Offers the same menu options as **view_login**. This function returns three values:

1) temp_list: valid set of tweets to select from.
2) temp1_list: valid set of users to follow.
3) count: current value of count, which is likely to be len(rows).

8) user_inputted_select(temp_list): Checks if inputted tweet ID (*tid*) is in displayed tweets (temp_list) and selects it. If invalid *tid*, user urged to re-enter. If valid *tid* (choice) inputted, **select_tweet(choice)** is called. Once it ends, the user is offered options to reply to it, **write_tweet(choice)** or retweet it **retweet(choice)**.

9) user_inputted_follow(temp1_list): Checks if inputted User ID (*usr*) is in displayed users (temp1_list) and if yes, calls **follow_user(*usr*)**. Otherwise prompts the user to try again.

10) select_tweet(choice): This function prints the tweet ID (*tid*), text (limited at 74 characters), its total number of replies and retweets, formatted.

11) menu(): Main Menu for System Functionalities. Takes user input based on the given options:
- User enters 1: To search tweets. -> **search_keywords()**
- User enters 2: To search users. -> **search_users()**
- User enters 3: To write a tweet. -> **write_tweet(None)**
- User enters 4: To show followers. -> **show_followers()**
- User enters 5: Logs user out -> **sys.exit(0)**.

12) search_keywords(): Splits inputted keywords string based on spaces. If '#' present at the start of the word, tweets (*tid*) that have the word in the *mentions* table and if '#' not present, tweets (*tid*) where the word is present are added as keys to a dictionary with values as respective dates (*tdate*). The dictionary's keys are passed as a list to the **search_tweets(*tid*s_list)**

13) search_tweets(keys): Prints the tweet for each tid from the keys (tids) list using **print_tweets**. Then:
- F/f to follow user -> **user_inputted_follow**
- S/s to select a tweet -> **user_inputted_select**
- M/m to go to main menu -> **menu()**
- B/b to go back if not on the first page of results

14) search_users(): Uses **user_keyword_validation()** to get a validated keyword. Uses the keyword to first print users that have the keyword in their name and then city in ascending order of name and city length respectively. M/m and B/b same as **search_tweets** Then:
- S/s to Select User -> **search_select** and then **print_five_users**

15) print_five_users(rows, count): Functions the same as **print_tweets** but prints users, modulus is taken as five and count is used to backtrack. Offers the same general menu options as **search_users**.

16) search_select(temp_list): Checks if inputted *usr* (choice) is among the displayed users (temp_list). If yes, calls **select_user(choice)**. Otherwise, prompts the user to try inputting again.

17) select_user(choice): Prints all the details of the user and three latest tweets/retweets of the user using **format_print_user_details(rows1)**. Offers the same general menu options as **search_tweets**

18) follow_user(choice): Calls **check_follow_possible(choice)** and if returned True, inserts a record into *follows* table using the **datetime** module.

19) check_follow_possible(choice): Checks if the logged in user follows himself or already follows the chosen user. If yes, returns False, otherwise returns True.

20) format_print_user_details(rows1): Formats **select_user** print outputs.

21) user_keyword_validation(): Checks whether a user inputted only one keyword for **search_users**.

22) write_tweet(reply_to): Allows user to write a tweet and inserts the record into *tweets* table. If this tweet is a reply to another, the original tweet ID (*tid*) is passed as argument for reply_to. Otherwise, reply_to has *NULL* value. If a tweet word has '#', calls **check_hashtag(term)**. If a term found in *hashtags* table, inserts a record only into *mentions* table. Otherwise inserts into both *hashtags* and *mentions* table.

23) retweet(choice): If call to **retweeted_already(choice)** returns False, inserts the retweeting record into *retweets* table. Otherwise, informs user that they already retweeted the same tweet.

24) retweeted_already(choice): Checks if logged in user (*usr*) already retweeted the choice (*tid*).

25) check_hashtag(term): Checks if term without the '#' exists in the *hashtags* table. If yes, returns False. If doesn't exist, returns True.

26) show_followers(): Prints all followers of the logged in user from *follows* table. If no followers, takes user back to **menu()**. Otherwise, offers to select the follower using **search_select**.

YOUR TESTING STRATEGY

We tested our code by running the program after every stage and trying to break it down to find problems. We worked on the SQL queries on the mac os terminal. Some of the errors we caught and solved are:

- ***Incorrect query outputs***: Through rigorous testing, we'd find edge cases resulting in incorrect outputs. For e.g., we had an error where our 'insert' statements weren't being reflected in the database. We later figured it was because of the absence of connection.commit() and added it.
- ***Invalid inputs***: We validated user inputs through various check functions and by printing warning statements when they input an incorrect value.
- ***Incorrect breaks in program flow***: This was because of poor program tracing that we corrected.
- ***Manual testing***: Testing by running the program repeatedly, and seeing if a bug rises through our varied inputs. We confirm database updation using the SQLite3 extension on VS Code.
- ***sys.exit(0)***: We use this to execute the logout option and to ensure our program ends successfully.

GROUP WORK BREAKDOWN STRATEGY

Firstly, we coordinated through a VS Code extension called Live Share, which works similarly to Google Docs, to edit the source code simultaneously, alongside the SQLite3 and Python extensions. Secondly, the work breakdown was decided as we worked together on this project. For example, we split work where Yaatheshini did the login and tweets composition while Vishal worked on the sign-up and showing followers implementation. However, we worked on displaying and searching tweets and users together by splitting work on building the query on SQLite3 and interpreting/formatting the results on Python. Although it was midterm season, we divided the work evenly through various contributions between ourselves. We spent roughly five days working on the project individually and together.