Overview

The purpose of this project was to develop practical skills in implementing SQL within a host programming language. This was achieved by creating a system capable of handling and storing data related to a social-network platform. This project is a python command line application that interacts with an SQLite database utilizing the sqlite3 package. This program allows both registered and unregistered users to interact with a social feed, search and compose tweets, find and follow users, and manage followers. After logging in users can view and interact with tweets, explore user profiles and participate in the social aspects of the platform like retweeting and using hashtags.

The database schema for this project is defined as:

- users(usr , pwd, name, email, city, timezone)
- follows(flwer, flwee, start_date)
- tweets(tid ,writer ,tdate ,text , replyto)
- hashtags(term);
- mentions(tid, term)
- retweets(usr, tid, rdate)
- lists(lname ,owner)
- includes(lname, member)

User Guide

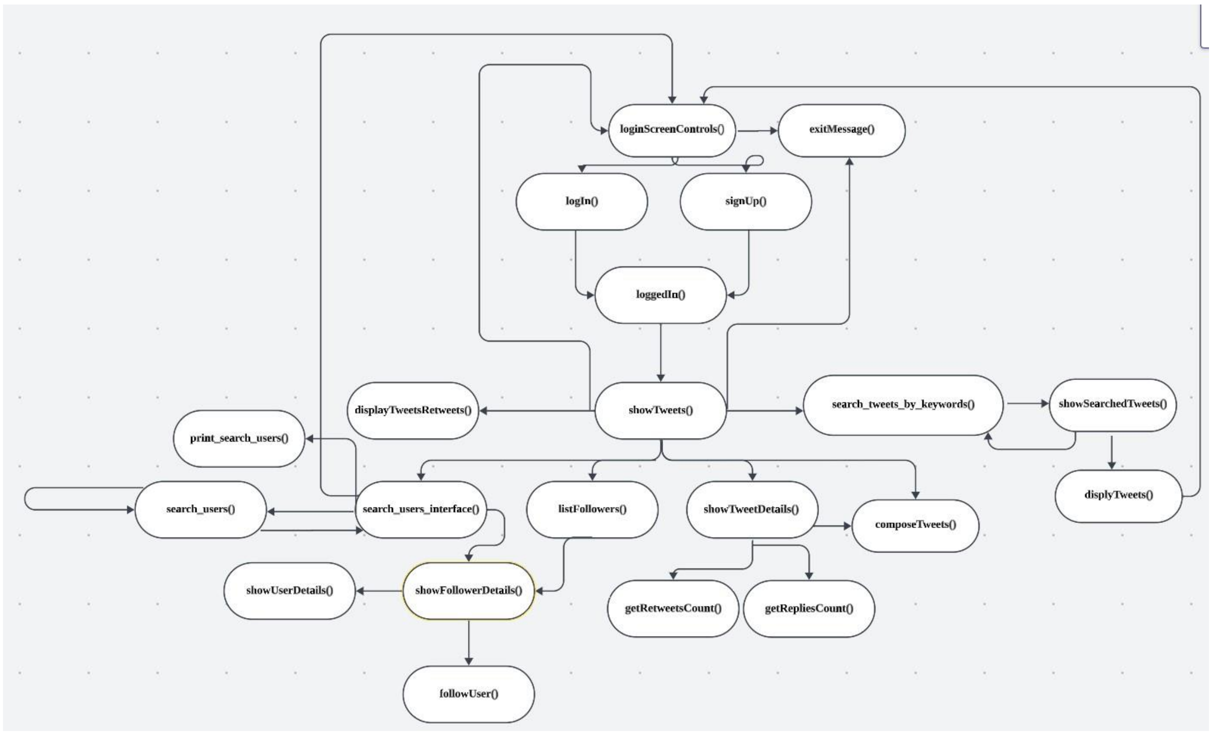
To run the program, first make a codespace on github and enter `python3 project.py` in the terminal. To enhance user experience and to easily navigate through the database, users are given every viable option at every single step of the process therefore enhancing user experience. Once the application starts up, the user is asked for an input from the given options 1, 2 or 3 with each input redirecting either to sign-up, sign-in or exit the application. After valid inputs registered users are able to view their tweets or retweets from other users. Registering users are able to sign-up by choosing their unique username and valid password and by providing other information such as email, city and timezone. For further engagement, users are able to select either tweets and see descriptions like retweets and number of replies. To create a chatting(communicative) experience users can further create a reply to a tweet and engage with other users globally. Once logged in, a user is also given the option to check his own followers. After selecting the correct option for List Followers, a list appears for all the followers of a user and for each follower a user can see further details like number of tweets, the number of users being followed, the number of followers and up to 3 most recent tweets. A user is also given the option to follow a selected user or see more tweets.

Detailed Design

1. Function: loginScreenControls() :
    a. Entry point when the application starts. It manages the initial user input and navigates to different functionalities based on the user's choice.

   b. Redirects to:
      i. logIn(): If the user chooses to log in.
      ii. signUp(): If the user chooses to sign up for a new account.
      iii. exitMessage(): If the user chooses to exit the application.
2. Function: loginScreen():
   a. Presents the user with the initial login/signup screen and captures the user's choice, returning it to loginScreenControls() for further processing.
3. Function: logIn()
   a. Handles the login process by prompting the user for their username and password.
   b. Upon successful validation, it proceeds to the main interface.
   c. Calls :
      i. loggedIn(userId) to access the main interface (upon successful login)
      ii. exitMessage() or loginScreenControls() if asked exit or go back
4. Function: signUp():
   a. Manages the creation of new user accounts.
   b. Collects required user information and saves it to the database.
   c. After account creation, it logs the user in automatically.
   d. On completion calls :
      i. loggedIn(userId) to proceed to the main interface.
5. Function: loggedIn(usr)
   a. Acts as the main interface for the user after logging in.
   b. Displays the user's feed of tweets and retweets.
   c. Uses :
      i. showTweets(usr, data) to display the feed.
6. Function: showTweets(usr, data):
   a. Displays tweets and provides interaction options.
   b. Based on user input it can navigate to functions for various actions such as composing tweets, viewing followers, searching, and more.
   c. Calls other functions such as:
      i. composetweet(usr, null): To compose a new tweet. ii. listFollowers(usr): To display the list of followers. iii. search_users_interface(usr): To search for other users. iv. search_tweets_by_keywords(usr): To search for tweets by keywords.
      v. showTweetDetails(usr, userInput, data): To display details of a specific tweet.
7. Function: composetweet(usr, replyTo) :

    a. Allows users to compose and post a new tweet or reply.
    b. It handles database insertion and ensures data integrity by committing the transaction or displaying errors
8. Function: showTweetDetails(usr, userInput, data):
    a. Provides detailed view for a specific tweet and allows the user to perform actions like replying or retweeting.
9. Function: search_users_interface: takes
    a. Has a function showFollowersDetails that then calls function followUsers which takes care of the following process



Testing Strategy

1. Approach:
    a. We engaged ChatGPT to create a dataset aligning with the schemas provided, tailored to the specific requirements of a twitter replica.
    b. Our testing strategy was comprehensive, aiming for extensive coverage to validate all functional controls as outlined in the system specifications.
2. Scenarios and Coverage:
    a. The scenarios included unit tests which covered the testing of every possible functionality(function)
    b. Full-path coverage meaning all pathways of using certain functionality were testing
    c. Boundary conditions and miss parameters were tested to see how the program responds
3. Tools Used:

    a. DBBrowser and SQLite: These tools were instrumental for monitoring real-time updates to the database, verifying that each function was interacting with the database accurately.

4. Outcomes:
    a. Multiple errors were discovered during the testing phase. These were categorized by logical errors, interface mismatches, and database update inconsistencies.
    b. Each identified bug triggered a code review and appropriate revisions to fix the fault.
5. Results:
    a. The iterative testing and debugging process ensured that the final iteration of our codebase was resilient and functioned in accordance with the programs requirements.
    b. The exact number of bugs were not documented but can be gathered from version control