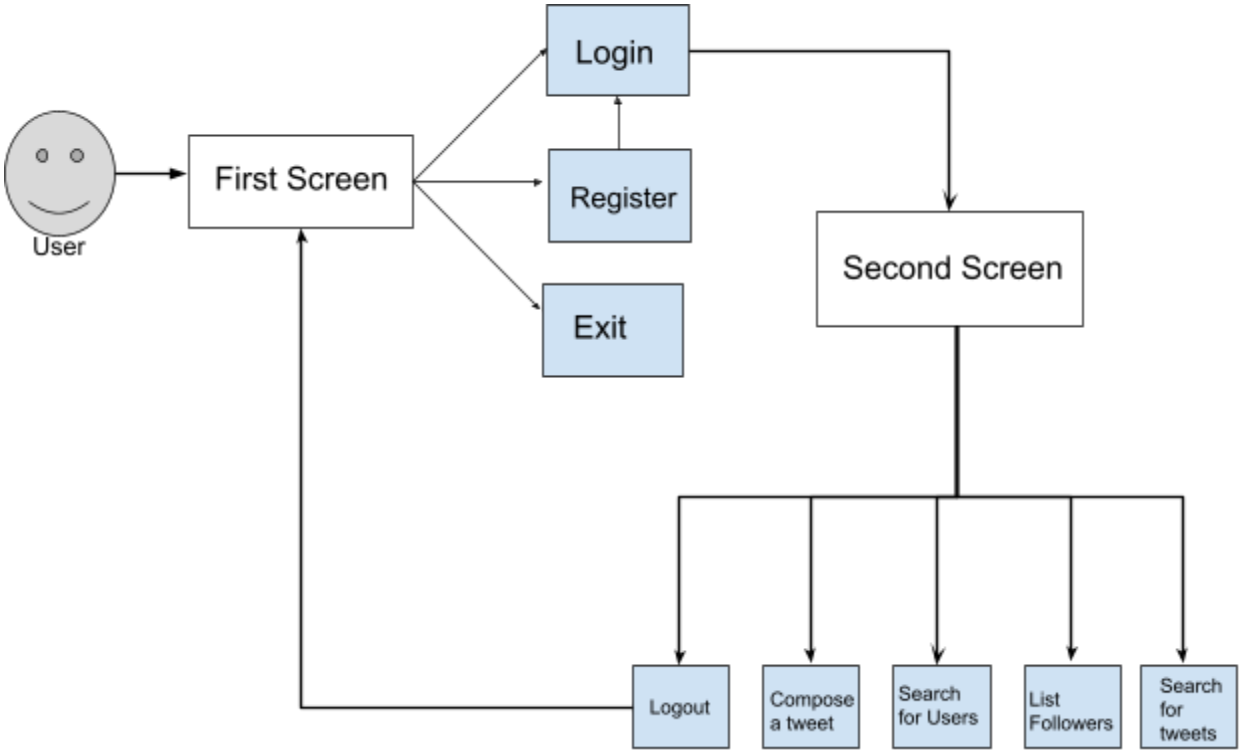
General Overview

The proposed social media platform comprises a user-friendly system that caters to both registered and unregistered users. Upon login, registered users can access their feeds, which display tweets from users they follow, sorted by date. Unregistered users can easily sign up by providing essential details, including a unique user id generated by the system. Post-login, users can perform various key functionalities, such as searching for tweets based on keywords, exploring user profiles, composing and sharing tweets with hashtags, viewing their list of followers, and securely logging out. The system ensures case-insensitive string matching, counters SQL injection attacks, and maintains password confidentiality during entry, while also implementing error checks to ensure a smooth user experience.



User Guide

1. A user can login to their existing account or create a new account. To create a new account, the user will have to enter name, email, city, timezone and set a password.

2. Upon login/sign-in, the user will be prompted to the homepage which will show a list of tweets and retweets of the users followed by them, 5 at a time. The user can select any tweet that is displayed and get more information about the tweet.
3. The homepage also displays 5 other options which include:
    a. Search for tweets: The user can enter one or more keywords and will get a list of every tweet that matches at least one of the keywords. The user can also reply to a tweet from the search results and display more information about that particular tweet.
    b. Search for users: The user can enter a keyword and will get all the users whose names or cities contain the keyword. The user can also select a profile from the search results and get more information about that particular profile.
    c. Compose tweet: The user can write a tweet and post it.
    d. List followers: The user can see all the followers that follow them. They can also view the profiles of the users that follow them and follow them back as well.
    e. Logout: The user can logout of the system.

Software Design

1. **connect(filename: str)**

   Purpose: Connects to an SQLite database and enables foreign key constraints.

   Interface: Accepts a filename as input and establishes a database connection.

2. **first_screen()**

   Purpose: Displays the first screen of the application, allowing the user to log in, register, or exit the app.

   Interface: No arguments, returns the user's choice.

3. **second_screen()**

Purpose: Displays the home screen of the application, showing tweets and retweets from user's followers, and provides options for actions like searching tweets, searching users, composing tweets, listing followers, and logging out.

Interface: No arguments, returns the user's choice or a selected tweet.

4. **login()**

   Purpose: Handles user login, checking for valid credentials and preventing SQL injection attacks.

   Interface: No arguments, prompts the user for login details.

5. **register()**

   Purpose: Handles user registration, ensuring that the provided user details are valid, and stores them in the database.

   Interface: No arguments, guides the user through the registration process.

6. **compose_tweet()**

   Purpose: Allows the user to compose a new tweet or reply to an existing tweet, including handling hashtags and storing the tweet in the database.

   Interface: No arguments, interacts with the user to compose and save the tweet.

7. **listFollowers()**

   Purpose: Lists the followers of the user, providing details about each follower, their tweets, and options to follow them back.

   Interface: No arguments, displays follower information and user interactions.

8. **retweet_tweet(tweet_id)**

   Purpose: Retweets a tweet with the given tweet_id, and records the retweet action in the database. It checks if the user has already retweeted the tweet to avoid duplication.

Interface: tweet_id (int): The tweet id of the tweet to be retweeted.
Returns None.
Prints a message to indicate whether the tweet was retweeted successfully or if it has already been retweeted.

9. **get_follower_tweets(id)**

   Purpose: Retrieves tweets and retweets from the accounts that the user follows. It fetches information about the tweet's writer, text, date, retweet user (if it's a retweet), and retweet date (if applicable) for display.

   Interface: id (int): User ID for which to fetch follower tweets.
   Returns a list of tuples, each containing information about a tweet or retweet.

10. **follow_user(flwer, flwee)**

    Purpose: Creates an entry in the "follows" table to indicate that the user with ID flwer follows the user with ID flwee. It checks if the user is already following the specified account to avoid duplication.

    Interface: flwer (int): User ID of the follower.
    flwee (int): User ID of the followee.
    Returns None.
    Prints a message to indicate whether the user was followed successfully or if the user is already followed.

11. **get_tweets_flwer_flwee(id)**

    Purpose: Provides information about a user, including the number of tweets they've posted, the number of followers they have, and the number of accounts they are following.

    Interface: id (int): User ID for which to retrieve user information.
    Returns a list containing a single tuple with the user's tweet count, follower count, and following count.

12. **clear_screen()**

    Purpose: Clears the screen with a pause of 0.5 seconds. It is used to provide a cleaner console interface.

Interface: No arguments and returns None.

13. **get_time()**

    Purpose: Gets the time of the active user, incorporating the user's timezone, which is used for timestamping actions.

    Interface: No arguments and returns a DateTime object.

14. **print_table(rows, headers=None)**

    Purpose: Displays information in a tabular format. It is a utility function to print data in a structured table.

    Interface: rows (tuple of tuples): Information to be displayed, typically the output of a database query.
    headers (list of strings, optional): Table headers for the displayed data. If not provided, the table will not have headers.
    Returns None.

15. **get_tweet_info(tweet_id)**

    Purpose: Retrieves information about a specific tweet, including the tweet's text, the number of retweets, and the number of replies.

    Interface: tweet_id (int): The tweet id for which to fetch information.
    Returns a list of tuples containing information about the tweet, including the tweet text, the number of retweets, and the number of replies.

16. **search_tweet()**
    Purpose: Searches for tweets in the database using any number of keywords and returns tweets that match at least one keyword or if the keyword starts with ‘#’ then it searches for the mentions table instead (search is case-insensitive)

    Interface: No arguments, prompts the user to enter space-separated keywords and returns the tweets in a table (5 at a time), the user can then select a tweet and see some statistics about the tweet such as number of replies and retweets. The user can then reply to this tweet or retweet it to his followers.

17. **search_users()**

    Purpose: The search_users() function facilitates the searching of users in a database based on a keyword input from the user. It implements pagination to allow the user to browse through the search results in manageable sets. The function provides the ability to sort the results by the length of the names or cities that match the keyword and gives the user options to delve deeper into a user's profile, load more results, or exit the search.

    Interface: This function does not take any arguments. It interacts with the user via the console, accepting input and providing output directly. The user inputs are used to search the database and to navigate through the search results and options presented.

18. **view_user_details(user_id)**

    Purpose: The view_user_details() function is tasked with retrieving and displaying detailed information about a specific user, identified by their user ID. It provides a comprehensive view, including the user's contact information, location, social metrics (followers and following), and recent activity (tweets). The function also offers interaction options to follow the user, view more of their tweets, or return to the search results.

    Interface: This function takes a single argument, user_id, which is used to query the database for the user's details. It interacts with the user through the console for displaying the information and handling user responses for further actions.

19. **view_more_tweets(user_id, limit=5)**

    Purpose: The view_more_tweets() function's primary purpose is to present additional tweets from a user's profile. It allows the user to view more of the selected user's tweets beyond what was initially displayed, up to a specified limit. This function enhances the user's engagement with the content of the user they are interested in.

    Interface: The function accepts two arguments: user_id to specify whose tweets to retrieve, and an optional limit parameter to define how many tweets to display. It interacts with the user through the console, showing the additional tweets and handling any further commands or navigation.

Testing Strategy

The general test strategy was creating entries for each of the major function and testing them out. The testing strategy for each of the functions are discussed below:

1. Search for Tweets

General Testing Strategy:

- Test the search function with various input types to ensure it retrieves the correct tweet data.
- Validate that the search handles edge cases, such as empty queries or special characters.

Scenarios Being Tested:

- Searching with exact tweet content.
- Searching with a partial match.
- Searching with no matches expected.
- Searching with special characters and numbers.
- Case insensitive searching.
- Handling of empty search queries.

Coverage Goals:

- Aim to cover all code paths within the search function, including error handling and edge cases.

2. Search for Users

General Testing Strategy:

- Ensure the user search functionality accurately finds users based on the provided criteria.
- Test for proper handling of non-existing users or incorrect input formats.

Scenarios Being Tested:

- Searching with a complete username.
- Partial username matches.
- Search with no expected results.
- Searching with mixed-case input to test case insensitivity.
- Handling of special characters in search queries.
- Empty search input scenario.

Coverage Goals:

Target complete coverage of the search algorithm, including validation and normalization of input.

3. Compose a Tweet

General Testing Strategy:

Test the composition of tweets, ensuring correct handling of input and storage in the database.

Check for proper character limits and handling of any metadata (e.g., timestamps).

Scenarios Being Tested:

Composing a tweet with valid content.

Handling tweets that exceed the character limit.

Composing an empty tweet.

Including special characters and emojis in tweets.

Timestamp and user association correctness.

Coverage Goals:

Full coverage of tweet validation, creation, and database insertion processes.

4. List Followers

General Testing Strategy:

Verify that the list of followers for a user is accurate and up to date.

Ensure proper handling of users with no followers.

Scenarios Being Tested:

User with a variety of followers.

User with no followers.

Newly added followers are listed correctly.

Unfollowing should update the list accordingly.

Coverage Goals:

Aim to test all aspects of the follower retrieval and display logic, including database queries.

5. Logout

General Testing Strategy:

Confirm that the logout functionality securely logs a user out and clears any session data.

Scenarios Being Tested:

Standard logout procedure.

Attempt to use the application after logging out to ensure session has ended.

Handling of logout with unsaved changes or pending actions.

Coverage Goals:

Test every part of the logout process, including session termination and user feedback.

Group Work Strategy

Method of Coordination:

To coordinate this project we used GitHub and to do this we had to learn some git basics to work on this project smoothly and without any development conflicts. Collaboration between group members was mostly virtual via calls and texts and also in-person meetings. Breaking up of work was based on each member's expertise. Members with proficient database skills handled the database aspect of the project and certain members handled the integration and programming part. Quality assurance and testing was also assigned to members who are familiar with data validity checks. Finally, the design documentation was a collaborative effort with each member of the team filling their parts.

Task Progress:

1. **Search for tweets:**
   Responsibility of: Mazen Khafagy
   Time allocated: 10-15 hours
   Progress made: Finished SQL statements for searching for the tweets, removed duplicate tweets and ordered them based on date. Used Raj's functions for selecting tweets and getting statistics for them and the compose_tweet function.

2. **Search for users:**
   Responsibility of: Ishfaq Mostain
   Time allocated: 12 -15 hours
   Progress made: finished up the code and used Raj's helper function for adding new user

3. **Compose a tweet:**

Responsibility of: Raj Hazarika
Time allocated: 2 - 3 hours
Progress made: created the functionality to compose a tweet, this required handling insert statements and user inputs.

4. **List followers:**
   Responsibility of: Raj Hazarika
   Time allocated: 2 - 3 hours
   Progress made: created the functionality to view all followers, this function calls follow_user().

5. **Login:**
   Responsibility of: Joshua Jimmy
   Time allocated: 8 - 10 hours
   Progress made: created the login screen using first_screen(), login(), register()

6. **Homepage:**
   Responsibility of: Raj Hazarika
   Time allocated: 2 - 3 hours
   Progress made: created the homepage using the prints.py file for formatting. Most of the logic in the function revolves around handling input and assertions.

7. **Testing functionalities:**
   Responsibility of: Mazen Khafagy and Joshua Jimmy
   Time allocated: 4 hours
   Progress made: Created a few databases for testing purposes. Tested the Login screen. Tested the search_tweet function.

Work Breakdown:

1. **Raj Hazarika:** worked on homepage, list followers, compose a tweet and logout. Created multiple helper functions: retweet_tweet(), get_follower_tweets(), follow_user(), get_tweets_flwer_flwee(), clear_screen(), get_time(), print_table(), get_tweet_info().
2. **Joshua Jimmy:** worked on the login screen: first_screen(), login(), and register(), code quality, and worked with Mazen Khafagy to create test cases.
3. **Mazen Khafagy:** Did the search_tweet function and created databases with Joshua Jimmy for testing purposes

4. **Ishfaq Mostain:** created the search_users() , view_user_details(user_id) and view_more_tweets(user_id, limit=5). Also added a couple of test cases.