Design Document for Twitter-like CLI Application

General Overview

App Name: TwitSys CLI

We made the TwitSys CLI as a Twitter-like command-line interface application. Users can interact with a social media-like platform. Users can sign up, log in, post tweets, search for tweets and users, view followers, follow other users etc. we use SQLite for database management by using Python's sqlite3 library and Python3 codes to implement the entire program.

User Guide

Welcome to the Twitter-like CLI Application!
Our user can enter number 1. Login 2. Sign Up 3. Exit (exit the program)

- **Login:** If you are the registered users, enter your username and password to access their account. (we will check SQL injection attacks as password is invisible at the time of typing).
- **Sign Up:** New users must provide valid personal information (user name, city, password, timezone, email) and receive a unique user ID.
- **Main Menu:** After login, users can enter number or symbol *. Main Menu (back to main manu) 6. Search for tweets 7. Search for users 8. Compose a tweet 9. List followers 0. Logout (exit program)

Detailed Design

Major Functions

database.py:

connect_to_db(db_name: str) -> sqlite3.Connection
Responsibility: Establishes a connection to the SQLite database file specified by db_name.
@Param db_name (str): The name of the database file to connect to.
@Return: The database connection object or None if the connection fails.

initial_query(conn: sqlite3.Connection)
Responsibility: Executes a series of SQL queries to create the initial database schema or reset it.
@Param conn (sqlite3.Connection): The active database connection.
@Return: None.

user_operations.py

login(db_name: str) -> Tuple[int, Optional[int]]
Responsibility: Authenticates a user based on the provided username and password.
@Param db_name (str): The name of the database file to connect to for user authentication.
@Return: A tuple where the first element is the status code (0 for success, 1 for failure), and the second element is the user ID if authentication is successful.

sign_up(db_name: str) -> Tuple[int, Optional[int]]
Responsibility: Registers a new user with the provided details and inserts them into the database.
@Param db_name (str): The name of the database file to connect to for registering a new user.
@Return: A tuple where the first element is the status code (0 for success, 1 for failure), and the second element is the newly generated user ID.

check(email: str) -> bool
Responsibility: Validates the format of the provided email address using a regular expression.
@Param email (str): The email address to be validated.
@Return: True if the email address is valid, False otherwise, with an "Invalid Email" message printed to the console.

generate_user_id(db_name: str) -> int
Responsibility: Generates a unique user ID for a new user.
@Param db_name (str): The name of the database file to connect to for generating a user ID.
@Return: The new unique user ID.

logout()
Responsibility: Ends the user session and exits the application.
@Return: None, but prints a logout confirmation message and terminates the program execution with sys.exit().

search_users(keyword: str, db_name: str, userid: int) -> List[Tuple]
Responsibility: Searches for users whose names or cities contain the specified keyword, excluding the current user.
@Param keyword (str): The keyword to search for within user names and cities.
@Param db_name (str): The name of the database file to connect to for searching users.
@Param userid (int): The ID of the current user to exclude from the search results.
@Return: A list of tuples containing the details of users that match the search criteria.

display_user_details(userid: int, user_id: int, db_name: str)
Responsibility: Fetches and displays details for the specified user, including recent tweets and follow statistics.
@Param userid (int): The ID of the user requesting the details.
@Param user_id (int): The ID of the user whose details are to be displayed.
@Param db_name (str): The name of the database file to connect to for fetching user details.
@Return: None, but prompts the user for a follow action or to view more tweets.

follower_operations.py

list_followers(user_id, db_name):
Responsibilities: List the followers of the logged-in user.
@param user_id: The ID of the user whose followers are to be listed.
@param db_name: The name of the database file.
@return: None. Outputs the list of followers directly to the console.

follow_user.py

follow_user(current_user_id, user_id_follow, db_name):
Responsibility: Follow another user.
@param current_user_id: The ID of the current user who wants to follow another user.
@param user_id_follow: The ID of the user to be followed.
@param db_name: The name of the database file.
@return: None. Outputs the result of the operation directly to the console.

main.py

display_menu1():
Responsibility: Display the initial login/signup menu options to the user.
@return: The choice of the user as an integer.

display_menu2():
Responsibility: Display the main menu options to the user after login.
@return: The choice of the user as an integer or a special character.

main_menu():
Responsibility: Handle the initial menu logic including login, signup, and exit.
@return: A tuple with the user ID and an exit code.

internal_menu(len_tw=0, last=0,):
Responsibility: Handle the internal menu logic after a user is logged in.
@param len_tw: The number of tweets currently loaded.
@param last: The index of the last tweet loaded.
@return: An exit code indicating the state of the menu (continue, error, or logout).

tweet_operations.py

compose_tweet(user_id, db_name):
Responsibility: Allow a user to compose and post a tweet.
@param user_id: The ID of the user posting the tweet.
@param db_name: The name of the database file.
@return: None. The result of the tweet post is output directly to the console.

search_tweets(keywords, db_name):
Responsibility: Search for tweets matching the given keywords or hashtags.
@param keywords: A list of keywords to search for within the tweets.
@param db_name: The name of the database file to connect to for searching tweets.
@return: A tuple containing the action code (0 for no action, 1 for retweet, 2 for reply) and the tweet ID of interest, if any.

Testing Strategy

General Approach
Functional Testing
Scenario: Inserting valid and invalid follow relationships, such as a user attempting to follow themselves and duplicate follow entries.
We need to ensure the 'follows' feature correctly enforces business rules and constraints, such as not allowing self-following and preventing duplicate follows.
Tweet Insertions:
Scenario: Adding tweets with registered and unregistered hashtags, and checking if hashtags not present in the hashtags table are handled correctly. We need to ensure validates the tweet creation process, ensuring that hashtag relationships are properly managed.
Negative Testing and Non-Existing Entities:
Scenario: Attempting to insert mentions for non-existent tweets and retweets for non-existent tweets or users.
We need to ensure checks the database's referential integrity constraints to ensure that such operations are not allowed.
SQL Injection:
Scenario: Insert statements that appear to be attempts at SQL injection, which could potentially alter the database structure or data if not handled correctly, and , ensuring that all input is properly sanitized before executing SQL commands

Scenarios

1. Login/Signup: Attempt to login with incorrect credentials, signup with existing user data, and SQL injection attempts.

2. Tweet Operations: Compose tweets with and without hashtags, search tweets with multiple keywords, and attempt to reply or retweet.
3. User Operations: Search for users with partial and complete keywords, attempt following users, and test for case-insensitive matches.
4. Follower Listings: List followers, validate the selection of followers, and test the follow functionality.

Bugs and Issues solved refer to Github:
https://github.com/ualberta-cmput291/f23-proj1-the-sequel/issues?q=is%3Aissue+is%3Aclosed

Task Allocations

1. FALAK SETHI (fsethi)

Responsibilities:

Framework Design and Coordinator: Design and implementation of the origin user interface menus (login, sign-up, main menu), setting up the initial database schema, then continue to work as a group for rest of parts and fixed different issues.
User Testing: Conducting manual tests for user interface functionality and reporting bugs.
Hours Allocated: Approximately 24 hours.
Progress Tracking: Regular updates in group meetings and commits to the version control system.

2. ZHIYUAN LI (zhiyua15)

Responsibilities:

Adding follower and followed functions, check the functionalities of login and, managing connections, and optimizing queries by coming up more test cases, reporting issues and add comments. Also, I accomplished DesignDoc.pdf file.
Security: Implementing measures to prevent SQL injection and ensuring password security.
Hours Allocated: Approximately 24 hours.
Progress Tracking: Through shared documentation and code review sessions.

3. AARYAN SINGH (ajsingh)

Responsibilities:

Made key aspects of the tweet operations, including development, search functionality, and user interactions within the application. He also contributes to login processes and the resolution of program-wide issues.
Progress and Final Integration Testing: Ensuring that the most the functionalities satisfy the rubric.
Hours Allocated: Approximately 24 hours.
Progress Tracking: Version control branches for feature development.

4. SUHANSH PATEL (suhanshk)

Responsibilities:

User Operations: Implemented user search, follower listing, and follow features, improving overall code
Documentation and Code Refinement: Reviewed and enhanced code quality, addressed bugs. Assisted significantly with the followers function, main menu, and code improvement.
Hours Allocated: Approximately 24 hours.
Progress Tracking: Through documentation commits and update meetings.

Coordination Method

Weekly Meetings: To discuss progress, challenges, and re-allocate tasks if needed.
Communication Platform: Using Discord and phone call/text msg for daily check-ins.
Version Control: Using Git for code versioning with separate branches for each member's tasks.
Project Management Tool: Utilizing a tool like ChatGPT for double-check codes correctness.