Kendrick Schellenberg
Nandini Parekh
Noor Kalra

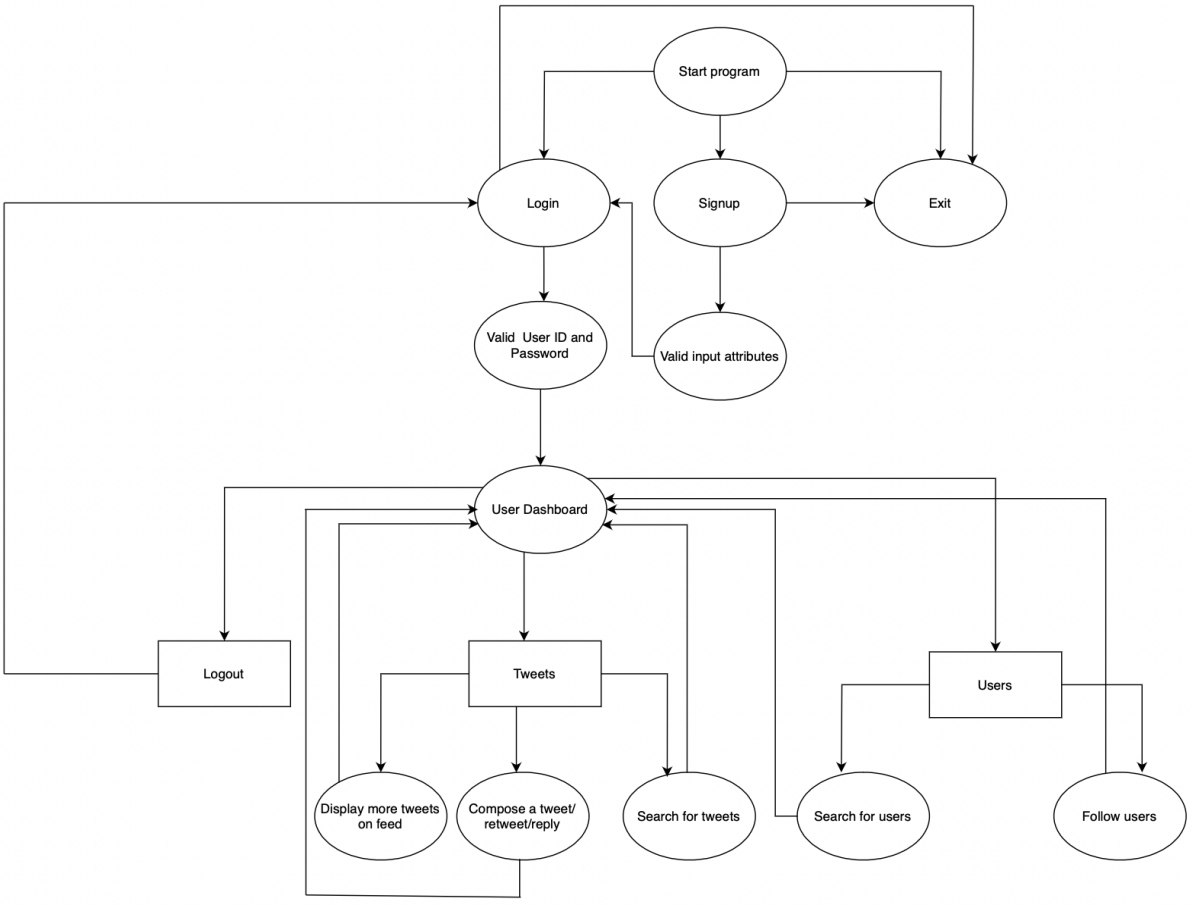
CMPUT 291 - Group Project 1 - Design Document

General Overview

SQL is a powerful programming language which can effectively manage and manipulate relational databases. However, it can have a synergistic effect when combined with a host programming language such as Python. By using both programming languages, a system can harness the database advantages of SQL while providing user services, such as that of a twitter-like system.

The system in question has a variety of features but can be distilled into five functionalities: logging in/out for registered and unregistered users, tweet searching, user searching, tweet composing, and follower listing. As seen in the diagram below, a user first has the option to login, sign up for an account, or exit the program. After a registered user logins in, they have the option to search for tweets, search for users, compose a tweet, list followers, or logout, as well as displaying tweets from followed users. All but the logging out returns the user to the dashboard to execute more commands or traverse resulting/remaining tweets.

Figure 1 - A diagram showing the control flow of the twitter-like system



Detailed Design

Functions (and structures)	Responsibilities	Relationships
main_screen() *login_check(user_id, password)* *get_and_check_entries(param)* *user_dashboard(user_id)*	• Presents user options to login, sign up or exit • Facilitates login • Successful login redirects to user dashboard	• Calls *sign_up* for unregistered users • Calls *get_and_check(param)* to validate login inputs (user_id and password) • Calls *user_dashboard(user_id)* to redirect logged in users to their feed
sign_up() *get_and_check_entries(param)*	• Prompts user for values for the input attributes (user_id, password, name, email, city, timezone) • Validates the inputs • Stores the profile onto the database	• Calls *get_and_check(param)* to validate signup inputs
exitSafely()	• Allows user to exit the program • Closes the database connection	
user_dashboard(user_id) *user_dashboard_options(user_id, tweet_list = None)*	• Displays the user feed with 5 most recent tweets along with their statistics	• Calls *user_dashboard_options(user_id, tweet_list = None)*
user_dashboard_options(user_id, tweet_list = None)	• Presents users with various system functionalities	• Calls various system functionality calls as per user choice
compose_tweet(user_id) *extract_hashtags()* *post_tweet()*	• Prompts user input for tweet content • Extracts hashtags from the tweet content • Posts the tweet and hashtags to the database	• Calls *extract_hashtags()* to extract hashtags • Calls *post_tweet()* to post tweet to the database
post_retweet()	• Posts the tweet and hashtags, and retweet info, to the database	• Returns to user_dashboard_options
post_reply()	• Prompts user input for reply content • Extracts hashtags from the reply content • Posts the reply and hashtags to the database	• Returns to user_dashboard options

search_user_keyword(keyword, user_id) *show_user_info(user_list,user_id)*	• Query users using keywords matching either their name or city • Returns users and provides options to view or follow users, or view their tweets	• Is called by *search_users(user_id)* which validates the keyword and passes it into *search_user_keyword()* • Calls *follow_user(this_user, followers_list, user_id)* to allow users to follow each other
list_followers(user_id) *show_user_info(followers_list, user_id)* *follow_user(False, followers_list, user_id)*	• Displays the user ids of users that are following the logged in user • Gives the user an option to see more details about a follower	• Is called by *user_dashboard_options(user_id)* • Calls *follow_user(flwee), show_user_info(followers_list, user_id)*

Testing Strategy

General strategy

The general strategy was to approach a form of statement and branch coverage. Neither unit tests nor other forms of testing were utilized. Rather, we identified possible variations of inputs to cause as many areas of our code to trigger as possible (statement coverage) and tested the branches of our code. Neither statement coverage nor branch coverage were difficult to attain given that most branches involved user input to select the executed branch.

Sign up • Missing inputs • Flawed inputs	*Logout* Initial tests indicated our strategy of using return to navigate back to the main screen was flawed. Thus, we switched to using a direct navigation call to the main screen.
Logging in/out • Login of registered user • Attempted login of unregistered user • Sign up of unregistered user • Attempted sign up of registered user • Exit	*Search for tweets* • No keywords • Single keyword - hashtag mention • Multiple keywords - hashtag mention • Single keyword - tweet text • Multiple keywords - tweet text • Mixed keywords - hashtag mention and tweet text
Tweet display and selection • View additional tweets • Select tweets • Reply to selected tweet	*List followers* • View user info • Follow user • Exit

• Retweet selected tweets	
Search for users • Empty keyword check • Valid keyword entered • Multiple users (>5) returned • Few users (<=5) returned	*Compose a tweet* • Tweet without hashtags Failed due to an incorrect sql call for tweet id. Adjusted code to determine next tweet id, and has functioned correctly since then. • Tweet with one hashtag • Tweet with two hashtags

* Assume all tests provided expected output unless there is a description of failure

SQL injection attacks

We attempted to input malicious sql commands that would allow access to undesired information. For example, we attempted a SQL command while inputting the user id during login, but it failed given our input constraints. Overall, we found that using ? as placeholders and providing tupled parameters was sufficient to prevent SQL injection attacks.

Group Work Break-down Strategy

The work was broken down roughly by primary functionality:

Kendrick worked on the overall structure of the code initially. This took him about 4 hours to complete. After that, he worked on ‘search for tweets’ and ‘compose a tweet’ as well as tweet display, testing, and code polishing which took an additional 12 hours, for a total of 16 hours.

Nandini primarily worked on the login component. She added functionality to enable registering new users as well as allowing users to log in. She also worked on the user dashboard which included the top 5 tweets posted by their followees. In total she took around 7 hours to complete her part.

Noor worked on the algorithm for Search users, list followers, showing user details and following a user. In total it took about 6.5 hours to complete all the functionalities. Editing and debugging the errors in the logic took about another 1 hour to complete.

All members contributed to the design document which took another few hours.

The method of coordination we used to keep the project on track was Discord. We held several meetings between Discord and in-person in order to discuss and work on the project. Github was also used as a collaborative coding tool.

We do not have any decisions that resulted in work beyond the project specification. We assumed that retweeting or responding to a tweet would cause displayed tweets to be hidden given that the stats thereof would not be updated.