Design Document for Mini Project One

By Ishaan Chandel, Ananmaya Mongia, Nick Maodus, and Aaron Chiu

Overview, Design, and Strategy

General Overview and User Guide:

Upon running the program by executing main.py in python3 with the database intended for use passed as a command line argument, the user will be prompted with three options: login, signup, and exit. Each option will be numbered when presented, and the user is prompted to select an option by typing the number associated with their desired task (henceforth referred to as the option index) and pressing enter. Once the option is selected, they will be displayed information and prompted input associated with the selected task. For the login option, the user will be prompted to enter their user ID and password. For the sign-up option, the user will have to fill in personal details in order to create their profile. Once the user is logged in, they will be taken to a different screen where action options such as writing tweets, searching for tweets, searching for other users, listing followers, and logging out will be available to interact with.

General descriptions for some of the options are listed:

a) **Write Tweets:** If the user wishes to write a tweet they will be prompted to enter text. If text has the prefix # it will be saved as a hashtag.

b) **Search for Users:** The user can write a keyword and the program will list five matched users with an option to see more. The program will then list five more users whose city matches the keyword. The user logged in can then select a listed user to be taken to their user profile where they will be able to see their number of tweets, number of users they are following, number of followers, and 3 recent tweets with an option to see more. The user will also be presented with an option to follow the selected user given they have not already and are not looking at their own profile.

c) **Search for Tweets:** The user can search for hashtags using the prefix # and enter keywords without the prefix and the program will search for the five most recent tweets that either use the hashtag searched with or mention the non-hashtag keyword in their tweet text. The user can select a tweet to view more information about the tweet, as well as be given the option to reply to the tweet or retweet the tweet.

d) **List Followers:** If the user chooses this option they will be able to view all of their followers. They will also be able to view a follower’s profile or back out. If the user interacts with a follower they will be taken to the follower’s profile where they will see the same information and options as mentioned in option b) search for users.

e) **Logout:** If the user wishes to log out they will be returned to the entry page where the user will be prompted with three options: login, signup, and exit (Indicating they have logged out).

Detailed Design and Major Functions

The implementation of our program is effectively a state machine, wherein a user will be brought to a new state of the user interface depending on which core option they choose. Using inheritance and superclasses, this state machine (each state referred to as frames within the code commenting and other documentation) simplified the process of adding and removing static and dynamic options the user would have the ability to interact with. This resulted in a fully immersive and realistic framework for what a fully-fledged software akin to the guidelines of this project would be based on.

User options are split into two types, static options and dynamic options. Options that are fixed and will always be available on a given screen are stored as static options in a dictionary within their respective frames. Examples of static options include log in, write tweet, and search for tweets. Dynamic options, sometimes referred to as dynamic events, are options that are unfixed and are dependent on the number of interactable tuples returned by a given query. Dynamic options examples include tweet selection for interaction, user selection for interaction, as well as the next and previous page options. Back and log-out options were also stored as dynamic options for ease of user maneuverability.

Class: Database

The Database class provides a basic framework for setting up and interacting with the SQLite3 database. It defines the necessary tables under the define_tables function. This class is used to establish a connection to the database.

Class: LoggedInFrame

The LoggedInFrame class serves as a user interface for logged-in users. It retrieves the most recent 5 tweets of the logged-in user. The class uses static options to offer the user with various actions. The class handles dynamic events such as viewing more information about selected tweets, navigating between pages, and logging out. Upon the selection of one of the various actions, a certain function will direct the user to the frame class corresponding to the option they chose.

Class: EntryFrame

This class serves as an entry point for users, offering options such as logging in, signing up, and exiting the program. The sign-up process guides the user through the registration process and once the information is gathered an INSERT statement is executed on the database in order to store the user data. The ? placeholders in the query are replaced with actual values from user_data. The login option allows users to enter their user ID and password. A SELECT statement is executed to the database in order to return the number of users that match the user ID and password. If the amount is zero then no such users exist and the program enters a loop until the user enters valid credentials. After logging in the user is directed to the LoggedInFrame.

Class: Frame

This class contains various methods and attributes for handling and rendering options within a frame. The __init__ method initializes the frame with reference to the FrameManager and initializes dictionaries to manage static and dynamic options. The render method displays the static options if any exist. The handle_event method processes user input and triggers handlers for static and dynamic options. Dynamic options can be added with the add_dynamic_render method, which increments the dynamic option count and displays new options.

Class: ListFollowersFrame

This class is designed to display and interact with the logged-in user. The __init__ method initializes the frame, and the render function retrieves all followers after executing a SELECT statement and stores them in an array. The handle_dynamic_event function enables users to interact with their followers or return to the previous frame. If the user selects to interact with a follower, then the user is directed to the UserProfileFrame.

Class: ComposeTweetFrame

The ComposeTweetFrame class is responsible for composing and posting tweets. The render method displays the tweet content and presents the user with options to confirm or cancel the tweet. The handle_delete function allows the user to discard the tweet and return to the LoggedInFrame. If the user decides to confirm the tweet the __handle_compose method will handle generating a unique tweet ID, adding the tweet to the database, and extracting and inserting hashtags.

Class: SearchForTweetFrame

The SearchForTweet class is designed for searching and displaying tweet search results based on keywords entered by the user. The render method displays the tweet search results, allowing users to navigate through pages and view individual tweets. The class constructs a dynamic SQL query based on the provided keywords and retrieves relevant tweets from the database, ordered by tweet date. The handle_dynamic_event class allows users to interact with tweets, view specific details regarding the tweet, and navigate through the results. Additionally, users can return to the LoggedInFrame.

Class: ViewTweetFrame

The purpose of this class is to display detailed information about a specific tweet. The render method retrieves details about the tweet, such as its text, date, writer, and the number of replies and retweets it has. It displays this information and options for composing a reply, retweeting the tweet, or returning to the previous frame. Users can also compose a reply, and if the logged-in user has not retweeted the tweet, they can choose to retweet it. The handle_dynamic_event function manages user interaction with the tweet. When a user responds to the displayed options, this function takes appropriate actions. If the user selects back, it navigates back to the previous frame, either the search results frame or the main logged-in frame, depending on whether the keyword attribute is set. When the response is a reply from the user, the function prompts the user to input text for a reply and then directs them to the ComposeTweetFrame to compose the reply with the provided text. In the case of retweets, the function checks whether the logged-in user has already retweeted the tweet. If it has not, it records the retweet in the database and displays a confirmation message, allowing the user to see the retweet on the same tweet details page.

Class: UserProfileFrame

The UserProfileFrame is responsible for displaying and managing the user profile information of the logged-in user. The render function displays the user's profile information and their recent three tweets with an option to show more. The handle_dynamic_event is in charge of handling user interactions. If the user selects back, they are directed to the previous frame. If next or prev is chosen, the function will navigate to the next or previous page of the user's tweets. The user can also follow the displayed user, and the function will record the new relationship in the database.

Class: UserSearchFrame

The UserSearchFrame class is in charge of searching and displaying user profiles based on the entered keyword by the user. The render method displays search results for users whose names or cities match the keyword. The user can select next to see more results. The handle_dynamic_event function handles user interactions. If the user selects a result, it directs them to UserProfileFrame. This function also handles navigation between pages and returning to the previous frame.

Testing Strategy:

General Testing Strategy: Each group member was responsible for testing their own frame and devising their own test cases. If the group member did not have data stored in the database during testing they would interact directly with SQLite3 and insert data into the database. Corrections, if needed, were mostly done by each group member on their own frame but input was taken from other group members in the case of severe problems. The majority of bugs and issues we ran into were caused by SQL Query logic problems and were time-consuming to fix.

Testing Scenarios: Included but not limited to entering two hashtags, checking if tweets are listed in the specific order, making sure terms are stored as lowercase in the mentions table, formatting for information needing to be displayed, query structure, string formatting, the functionality of interactive options, making sure ids for followers are not displayed as a tuple, set different user has more than one tweet, set one user to follow more than one id.

Group Work Strategy:

The general strategy had each member pick one system functionality and implement it. Each member was responsible for string matching, SQL injection attacks, and error checking (troubleshooting) while implementing their part of the program. In some instances, group members provided input to other members on their part of the program. The design document was a team effort. Our method of coordination was through Discord, and we would work simultaneously on our parts during a group call. Oftentimes we would help each other if needed and dedicate multiple people to combatting a certain bug or helping others understand our frame design philosophy.

Additional Functionality added:

On certain screens, we added page traversal functionality, which allowed users to go back to the previous sub-menu instead of being returned to the main home page after performing an action. When asked for the input to compose a tweet, the input is displayed to the user again, asking for confirmation if the user does want to compose the tweet. In certain situations, such as following a user and retweeting a tweet, instead of returning the user to the home page, we refreshed the current page to display the tweet details or user profile again.

Breakdown of Frames/Functions Worked on Alongside Estimated Time Spent:

maodus (ETS: 40 hours): EntryFrame, LoggedInFrame, Frame Manager, Frame class, UserSearchFrame, UserProfileFrame, ComposeTweetFrame, EntryFrame, Documentation

ichandel (ETS: 40 hours): LoggedInFrame, SearchForTweetFrame, ViewTweetFrame, ComposeTweetFrame, ListFollowerFrame, Documentation, QA, user output display readability

ananmaya (ETS: 30 hours): LoggedInFrame, ListFollowerFrame, Documentation

achiu3 (ETS: 30 hours): LoggedInFrame, ComposeTweetFrame, SearchForTweetFrame, UserSearchFrame