CMPUT 291 Mini-Project 1 Documentation

Group: Crystal-methadata | November 7th, 2023

General Overview

Overview

Crystal-methadata will have people addicted to Shell Twitter, our app that connects the whole world through a Command Line Interface. Our app allows users to login using their user ID and password or register by simply entering their name, email, city, timezone, and a password. Our program will generate a user ID that can be used to login. Once logged in, users will be shown a feed of recent activity from the users they are following. At any time, the user can also search for more tweets using any number of keywords. The user can reply to, retweet, or view more information about the displayed tweets in the feed/search results. At any time, the user can also search for users or get a list of their followers and learn more about a user by selecting their profile. Selecting a user will display their user information as well as their most recent tweets. Users can interact with these tweets as well. The user can also follow a user that is on their followers list or the search results. If the user so chooses, they will be able to compose a new tweet at any time. The user can clear the screen to reduce clutter (note that this will also end any activities the user is currently viewing). The user can also view a list of valid commands using the help keyword. When the user has finished their session, they may choose to logout or exit the app entirely.

Global Commands

^Login: allows the user to enter their user id and password to login
^Register: allows the user to create a new account by entering their user information. A unique user id will be given to the user upon successful registration.
*Feed: shows the user the recent tweets, replies and retweets that their followees have posted.
*Followers: lists everyone that follows the current user and allows the user to select users from that list.
*Searchtweets: allows the user to search for tweets using up to multiple keywords/hashtags.
*Searchusers: allows the user to search for users using a single keyword.
*Compose: allows the user to compose a tweet.
*Logout: signs out the current user.
Help: displays a list of commands that are in scope.
Clear: clears the screen.
Exit: exits the application.

^ indicates the command only works when not logged in
* indicates the command only works when logged in

User-specific Commands:

Select <index>: Allows the user to select a user by the list index and view more information about them
Follow <index>: Allows the user to follow a user by using the index indicated on the list that displays users
Scrollup: Allows the user to display the next page of users (up to 5 users per page)
Scrolldown: Allows the user to display the previous page of users (up to 5 users per page)

Tweet-specific Commands:

Reply <index>: Allows the user to compose a reply to the tweet at specified index
Retweet <index>: Allows users to retweet the tweet at specified index
Viewinfo <index>: Allows the user to view more information about the tweet at specified index
Scrollup: Allows the user to display the next page of tweets
Scrolldown: Allows the user to display the previous page of tweets

Software Design

Main:

- Main is responsible for taking the database as an input when running the main program and using the database for the rest of the program. The main function will then print available commands available and call Shell.main_menu_do(cmd) to process user input such as logging in or composing a tweet. This will also connect to the database which can be our test database or the database given. For both user and developer convenience, we decided to allow certain command-line arguments when launching the program: namely, specifying the database file, being able to wipe the database and start fresh, and being able to test the program with some fixed mock data.

Login:

- The Login class's main responsibility is to control user sessions by handling user authentication and managing user registrations. It keeps track of the currently logged-in user's ID and name as class-level attributes, which can be accessed across the application to maintain the state of the user session.

Shell:

- The main responsibility of the Shell class is to act as the user interface layer for the application. It is in charge of presenting the menu options to the user, handling user commands, and invoking the appropriate functionality based on the user's input. It also provides utilities to manipulate the command line interface, like clearing the screen.
- `main_menu_do(cmd, additional_options:[str]=[])`: Executes actions based on the command (cmd) provided by the user. Performs actions such as login, registration, displaying the feed, searching for tweets, composing tweets, searching for users, and more. Additionally we gave the user options to use ‘clear’ and ‘help’ commands at any point, which clears the terminal thus far and lists all possible commands available respectively.

Feed:

- The responsibility of the Feed class is to find all the actions of the users being followed by the currently logged in user. Actions include tweets, replies, and retweets. After finding all activity by querying the database, the activity within the Feed class will display all the actions (with the most recent first) to the user by passing it to the Search class, which lists out the tweets and presents a phony shell, allowing the user to interact with the displayed tweets. Although the feed is displayed automatically after login, the user can also manually view it by executing the feed keyword in the shell.

Search:

- The Search class is responsible for searching and retrieving different types of data from a database, including tweets, users, followers, and additional information on the tweet and users. It interacts with the database to execute search queries and provides mechanisms for the user to interact with the search results.
- This class allows the users to search for users (using a single keyword) or for their followers (using the command ‘searchusers’ or ‘followers’ respectively); the user can select and/or follow one of the returned users (using the command ‘select’ or ‘follow’ respectively). To streamline the user experience, we implemented the ‘follow’ command in a way that allows the logged in user to follow the users in the returned list directly without needing to select them first.
- This class also allows the user to search for tweets using multiple keywords/hashtags (using the command ‘searchtweets’). Once the search is submitted, the class submits the tweets to the def interact(...) method, which takes care of displaying the tweets and letting the user interact with the tweet.
- The `interact(...)` is the main UI that lets the user interact with either the tweet and/or user profiles. If it is a tweet, the options available for the user are reply <index>, retweet <index>, viewinfo <index>, scrollup, scrolldown. The <index> for these are as specified above. When viewing tweets from a keyword searched, 5 tweets that are displayed at a time and the user has the option to scrollup and scroll down to see different pages of tweets, 5 max. When the tweets are viewed as part of a user profile, the pages display 3 tweets at a time. If this function is called with a user object, the options available for the user are select <index>, follow <index>, scrollup, scrolldown. The <index> is the position in which the user appears on the displayed list. Additionally, we made the design decision of always allowing the user to enter a global keyword, regardless of the current activity. This means that the user never has to go “back” between activities; they can simply start a new one at any time.

- The `search_for_tweets(...)` is the function that takes user input for keyword[s] and/or hashtags[s] and will query that database to return all matching tweets, sorted such that the most recent tweets display first; the user will then be able to interact with the returned tweets. As an extension, we specified whether a tweet was a retweet or reply to let the user know exactly which tweet they are seeing, this method improves UI like regular twitter.
- The `search_for_users(...)` is the function that takes user input for a single keyword and will query that database to return users who match the given keyword, sorted such that the users whose name match the keyword are returned first (in order of ascending name length) followed by users whose city match the given keyword (in order of ascending city name length); the logged in user can then interact with the users in the returned list.
- The `search_for_followers(...)` is the function that will return all the followers of the current user and let the user interact with them.

ComposeTweet:

- This class allows the user to compose new tweets and reply and/or retweet existing tweets. The two main methods `createTweet(replyTo: int = None)` and `createRetweet(tid:int)` can be used to create a tweet or reply to an existing tweet, and create a retweet for an existing tweet respectively. The createTweet gets the highest tid yet and adds 1 as the new tid for tweets.
- This class also implements the functionality of parsing through hashtags in a given tweet

Follow:

- This class provides the functionality of getting the name of a user given the username. Within this class, the `follow(...)` method implements the ability for the currently logged in user to follow another user; if the logged in user is already following the user they are attempting to follow, this function will notify the them and not execute the creation of a follows relationship

Connection:

- This class is responsible for creating and closing connections with the database, checking that a connection exists. It also provides a helper function to determine if a query result is empty or not. The `connect(path: str)` connects to the given path's database and `close()` safely closes the connection.

Testing Strategy

Our testing methodology mainly consists of testing the functionality of each module to ensure that the modules and the functions work as intended. We created our own mock database, Test.py, which is initialized in the main of each module to be able to test the queries and outputs. We created the database and made it to account for different cases and functionality of our program. We would run each module to test the functionality of individual modules, ensuring that each function was being called and the edge cases were being considered. We wanted to get as close to full coverage as possible by testing all valid and different invalid inputs from the user.

After writing the code to use sqlite's execute() method with parameters instead of string formatting, we also tested for SQL injections by typing arbitrary SQL statements as input (after closing strings) and checked whether the database was affected in any unexpected or adverse way. We used Visual Studio Code's sqlite extension to view the database while running the program, and were able to check that our database wasn't impacted by injection attack attempts.

We also had to do a lot of integration testing as we were all working on different functions and modules. To test the integration of the different modules and functions interacting with each other, we reconfigured the main functions of the modules to test the integration for each module and then the interaction between modules. To test the integration of the whole system, we tested main.py and ensured that the functionality of each module was covered and all valid, invalid and edge cases were covered by our program.

We did come across a few bugs, but there were two big ones in particular that took a while for us to debug. One was our call stack for Search.interact calling itself again when searching for tweets of a selected user. When a user would input "Help", it would show the wrong list of commands to the user. This was confusing to debug as it deals with nested call stacks but by having a return statement after select, we were able to fix this issue. Another bug that we encountered was

our database connection not committing after inserting into the follows table, which caused the data to not be inserted into the table. We used breakpoints to debug and find the source of this error and inputted a Connection.connection.commit() at the end of the insert query to add the follower to the database.

Group-work Strategy

There were 6 main tasks that we considered when working on the project–the UI, logging in and main commands, searching and selecting users, searching and selecting tweets, composing tweets/replies/retweets, and searching for followers. While each individual group member often worked on implementing single functions on their own, we often had to work in pairs for closely-integrated functions, and spent lots of time on call discussing our intended design.

The breakdown of the work and the approximate times spent are as follows:
Tawfeeq (13-15 hours):

- I worked on organizing the structure of the classes in our source code, developing the login/registration and feed activities, and creating the UI that our shell app would present. Much of my job about laying out solid groundwork (eg. Connection wrapper class, Setup) so that every class could be integrated easily, as well as cleaning up loose ends when we finished the project.

Parshva (15 hours):

- I worked extensively on the Followers class in which I implemented the functionality of listing the followers of the user, and the ability to follow/select any of the listed users. I later pair-programmed with Samin to help refactor the Followers class entirely into the Search class, removing the need for a Followers class. I also created the Login and Follows class and implemented the functionality of logging in and following a user.

Aakash (14-15 hours):

- I worked on the Search class and Shell class extensively. Allowed the shell to process user commands and print menu options. Specifically I allowed the user to type global commands at any time, which integrated the Search and Shell classes. Within Search, I created methods to search for tweets based on keywords, and mentions and then allow the user to interact with a tweet/user type items and display these results on the terminal.

Samin (13-14 hours):

- I worked extensively on the Search class; I implemented the functionality of being able to search for users using a given keyword and letting the user follow a returned user and/or select and view more information about any of the users. I pair-programmed with Parshva to help refactor the Followers class entirely into the Search class. I wrote and changed functions to help test the application.

In addition to the individual contributions as listed above, each member was also heavily involved in code review for pull requests. We established a de facto rule that all pull requests must be approved by a member who didn't author the code. This had an added benefit that members were accountable for writing their code in a timely manner, as often we had to merge and close one pull request before we could effectively start writing code for other sections. If one member didn't update their branches regularly, other sections of the code would end up at a standstill since we wouldn't know how to integrate it. For very large merges, we had the full team on call during code review, with the purpose of discussing the changes, resolving conflicts with our individual branches, and testing the PR thoroughly for bugs.