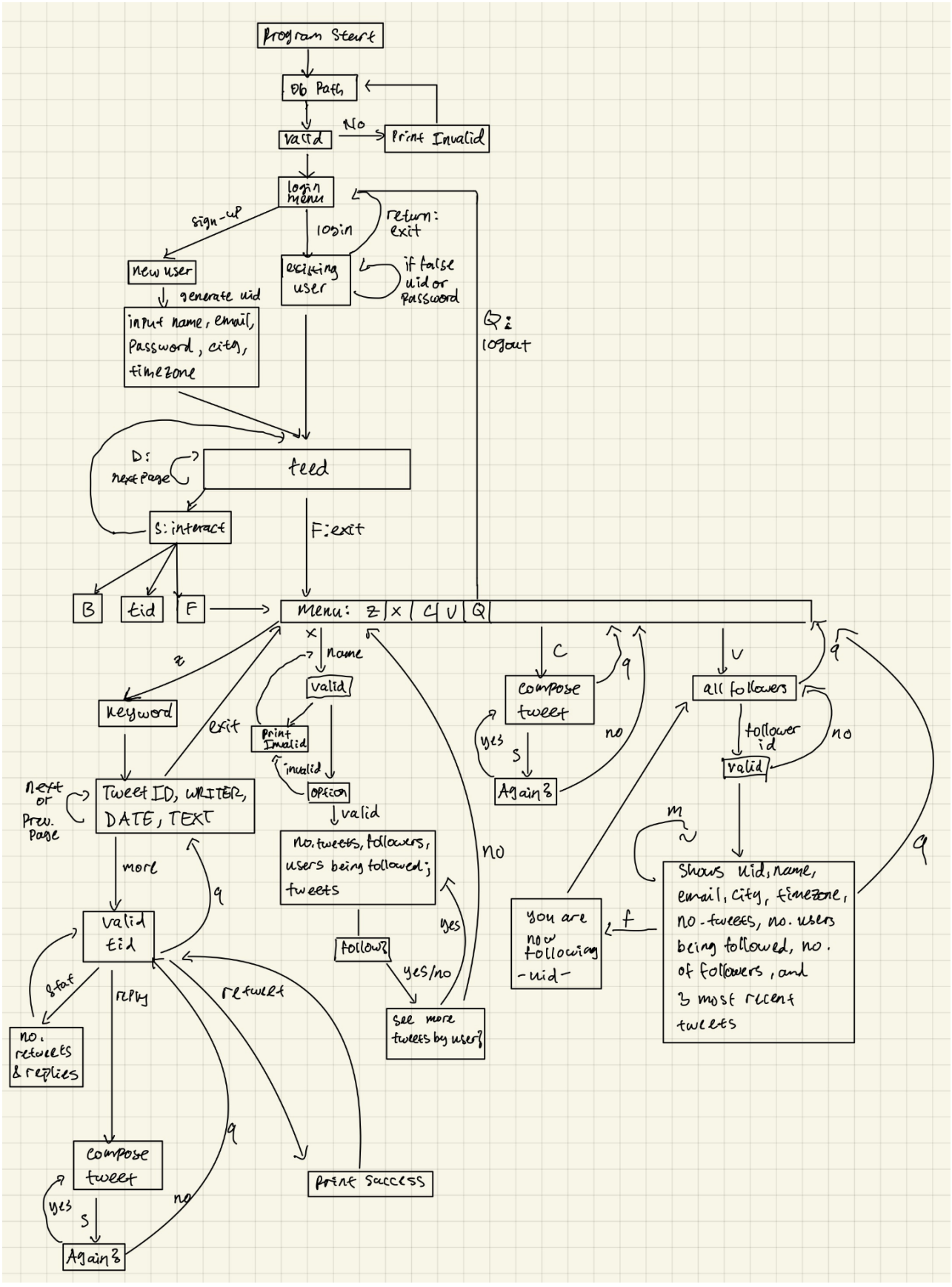
The specifications of the instructed functions were delegated to the group members and implemented using different files that are all controlled by the main file. Within the repository directory, there includes: *ComposeTweet.py, feed.py, list_followers.py, LLM.md, login.py, main.py, mp1.db, README.md, search_tweets.py, search_users.py, test1.db, test2.db, and test3.db*. These files are named respectively for their purposes to ease readers when debugging/reading functions implemented in the interface. Within the main.py, lies the implementation of the interface with regards to all the files combined, simply run *main.py* and input necessary inputs through stdin (case-insensitive).

Within the directory, the files contain the necessary functions to correctly implement the program. *ComposeTweet.py* contains a *ctmain* function that composes a tweet after a user has logged in and typed *c*, this function inserts hashtags into the mentions table and if not yet, the hashtags table. *LLM.md* will contain all information regarding the use of AI tools and other possible secondary source contributors. *Mp1.db, test1.db, test2.db,* and *test3.db* are databases taken from Assignment 2, *mp1.db* is an inspired database that also contains the group's own database.

The contributor to *main.py*, *login.py*, and *feed.py* is Daniel Cui, he spent 20 hours on this project. The contributor to *search_users.py* is Anshul Verma, he spent 20 hours on this project. The contributor to *ComposeTweet.py* and *search_tweets.py* and this design document as well as its drawn diagram is Peter Davidson, he spent 16 hours on this project. The contributor to *search_tweets.py and list_followers.py* is Nick Shan, he spent 16 hours on this project. We kept our coordination on the project by grouping together at a certain time and location and doing it together simultaneously with the discussion of solutions. When facing obstacles of any kind, we help each other out thus utilizing teamwork and not individual delegation more.

A quick guide for users would be to run *main.py* and follow instructions suggested by the stdout, the following diagram gives a general overview of the program:



Search_users.py finds all associated users based on the inputted keyword, displays the search results based on the option numbers, user id, and user name. This file also enables the user to traverse through matched users 5 at a time, select a user, present their information, and the option to follow them and see more of their tweets if there exists any.

README.md contains the contributors' information and statement of academic integrity. *List_followers.py* takes in the current user id, finds all their followers, an option to enter a user id and reveal personal information about said user, follow them, see their tweets, and quit. *Search_tweets.py* contains functions that are able to return a list of tweets associated with a keyword, retweets and replies of a tweet, and the ability to compose a reply.

Main.py controls the interface, it connects to a given database, and controls the general functionality of the program. After a successful connection, it will bring the user to the login menu, where the user has the options of logging in with a registered user id and password, or signing up as a new user, or exiting the program. After a successful login, if the user is a already registered user, it will show the feed menu, where the user is able to interact with the feed of tweets/retweets from the people they follow. After exiting the feed menu, the user is brought to the main menu where they are able to access the core 5 functionality of, search tweets, compose tweets, search users, list users, or log out. Upon logout, it is brought back to the login menu.

Feed.py finds all the tweets that a user follows, displaying it in table-form, and able to view at most 5 items per page, they can interact with the tweets displayed, an interaction leads to the exposure of the statistics of said tweet, to reply to it, or retweet it. *Login.py* processes the log in and sign up process of the user. With protection against SQL injection attacks, and hidden password input, the security is assured. If the user is logging in as a pre registered user, it will get and process the input of information required to log in and return the user id. If the user is a new user signing up, it will handle all the inputs necessary for creating a new user.

Our general strategy for testing is to brainstorm outputs for certain inputs and tested edge cases brought up by students in the 291 forums after testing simple-pseudo-code with respect to our database. Some bugs that we ran along the way was that *ComposeTweets.py* had to consider multiple edge cases where hashtags could make some issues such as: #f becomes #f, this is not actually the case, however sql viewer causes this error as it is truly unaltered; c #f becomes c #f, this is also not the case, it is an sql viewer error, it is truly unaltered in the program; c becomes c, this is also not the case, it is an sql viewer error, it is truly unaltered in the program; c

#F#f will record 1 #f in mentions and hashtags if hasn't existed yet, all hashtags will be lowered so that it becomes case insensitive, this causes #f = #F; c #F#C will be treated as c #f #c, we treat combined hashtags as different hashtags; when discarding a tweet, all records of said tweet is deleted from tweets and mentions table (if exists), if a hashtag is introduced by said tweet, we discard it too as no other tweets use said hashtag; # records #[space] in the mentions and hashtags table provided it is new, we consider #[whitespace(s)] to be valid, #[whitespace(s)] = #[whitespace], and multiple other edge cases where things should be case-insensitive. An issue that arose in *feed.py* was that when the people that the user follows tweets/retweets less than 5 times combined and created an index out of bounds array when it attempted to display 5 tweets at a time. When implementing search_users.py, we discovered an edge case where-in the user was able to follow themselves since it didn't violate the primary key constraint that was handled through exception handling. In practice, a user shouldn't be able to follow themselves and hence we decided to check for this particular condition when inserting into the follows table and print the desired message. As for *search_tweets.py,* we ran into an issue where tweets containing mentions would be selected twice if keywords in the mention also happened to be inside the primary text. User 29 has a lot of tweets, this user is used to check for errors when testing for *list_followers.py* and *feed.py* where we want to display 5 tweets at a time, this user is also used to debug errors within ComposeTweet.py where edge cases that revolve around the terms may be handled. User 97 is also used to login and debug most of the group's code; user "John Doe" is also used to handle *search_users.py* where displaying a certain number of users at a time and making it go to the next page and previous.