Natnail Ghebresilasie, Afaq Nabi, Krutik Soni

Mini-Project 1 Report

General Overview:

When first opening the program they're greeted by a login screen that allows them the options to login, sign up or exit the program. After successfully logging in or signing up, regular users are granted the opportunity to post a question, search for a post and post an answer/or vote for the selected post from the searched options. Privileged users are different and cannot be signed up into the program, they are initially made, think of them as like moderators of a website. They, additionally to the tasks regular users have, can also do more such as mark answers as the accepted answer to a question, give badges to users, edit the title and or body of a post, and add tags to posts. Users are prompted to keep doing one of the tasks until they either choose to logout or exit the program. For a representation of the flow of the program, look at figure 1on the last page of this document.

Design:

The main function is where the login process and signing up takes place. It allows the program to continuously run until the user chooses to exit the program completely in which case the loops would terminate. Within the loops is where the functions after_login get where that function handles the actions that a user is able to do such as post, search, login, and exit. Function make_post allows the user to post a question, in which case they will input a title and body and the proper parameters such as date and the ID of the user will be automatically added into the database along with the inputted title and body. The main function return_post returns posts takes in the parameter keyword and returns the posts that the user has entered when initially searching for a post. The program sorts the results based off of the highest number of keywords matching the post. This is done by looping through the results and only looking at the unique keywords and putting them into a dictionary based on the words. The results are taken from the dictionary into an array that is returned by return_post in the ordered method Late. The results are displayed by the following functions print_column_headers where the results are neatly shown under the headers post ID, post date, post title, post body, user ID, tags, number of votes, and number of answers by the function print_selected_post. Search_post is also in charge of allowing the user to enter > to see 5 more results of their searched keywords if the results are greater than 5 or they can choose to enter a uppercase or lowercase p to enter a pid of a post to view it in which case they are allowed to enter in actions that are recorded by post_search. This function allows regular users after selecting a post to either answer it if it's a question, vote on it or if it's an answer just to vote on it. However, for privileged users, they along with the actions regular users have can also edit titles and or body, add tags, add badges to the poster, mark a post if it's an answer as the accepted answer and more. The functions are called based on what the

privileged user selects. Function vote takes in parameters user ID, post ID, vote number, and asks the user if they'd like to vote on a post. This vote is recorded into the database if they choose to do so. Votes can be on answers or questions, any sort of post can be voted on. The privileged user can mark an answer of a question as the accepted answer and replace it as well, this is done in the mark function, the function receives the post selected by the user and then marks the answer as accepted or asks if they'd like to replace the answer. This is recorded in the database afterwards if they replace or mark an answer. Privilege users ability to add a tag is done after taking the parameters into the add_tag function which is the post ID. Users can add a tag but not a duplicate add that is checked by the sql statement in the source code. The function edit receives the pid from the function that handles post selection and then allows a privileged user to edit the body and or title of the post, and the last function badge allows a privileged user to add a badge to a user. A user can only receive a badge once a day as it is a unique constraint. All of these 4 functions receive information by the function searchPostControlFlow, this function calls the necessary functions based on the action the user has decided to take and the options provided to the user based on access level such as privileged or not.

Testing:

The testing strategy our group decided on was that we wrote the sql queries in sql first to make sure that the proper rows are returned and then we made the sql queries into python as we were sure they would return the same thing since all we did was access the database with the cursor. The results returned what we expected and we made a mock database filled with new users and privileged users where we performed the tasks that each user is allowed to partake in. There were many bugs, unfortunately we don't have statistical information as we didn't record it but the main bugs were the search posts. Many times it would fail by making the count of the number of keywords in a post higher than it should be due to duplicates. A minor fix as we implemented a dictionary and used multiple for loops to increment through the results and only select information that is unique with a correct count. The badges table proved to be somewhat challenging as it resulted in many unique constraint errors, however when adding badges into a user twice a day it fails as badges of the same type cannot be given to a user twice in one day. Adding in an OR IGNORE into the query fixes that error. Our main testing strategy was writing these queries in sql first and making sure that the right results were returned by finding the right query needed. Majority of the bugs we found ended up being in the way we pass in a keyword to a search post. Positional arguments in search posts failed a lot due to the fact that when passing in the keyword it needed to be wrapped in % at the beginning and end inside of a list so that it returns the right results without failing as until we figured out this methodology we were getting many positional argument errors.

Group Work Break-down:

We decided to work on this project by first coding it in an IDE that allows us to see each other's code by someone live sharing, due to the nature of none of us being able to physically meet up right now. Natnail was in charge of the privileged users, while Krutik implemented the flow of the program and search posts, and Afaq created the login screen and the actions that a regular user can do. However, this doesn't mean that we never worked together, we helped each other with every single function and fixed all errors together so that everyone learns how to fix any errors that show up. The time spent on the project is estimated to be roughly 60 hours total and it took roughly 20 hours for each person to implement their features and fix any errors or bugs that appeared. We decided to meet up almost every night around 7 pm Mountain Time and work on it together for roughly 5 hours a day. Each person was required to make the functions into the live shared code and was encouraged to get help from partners when they were stuck on implementation.
\

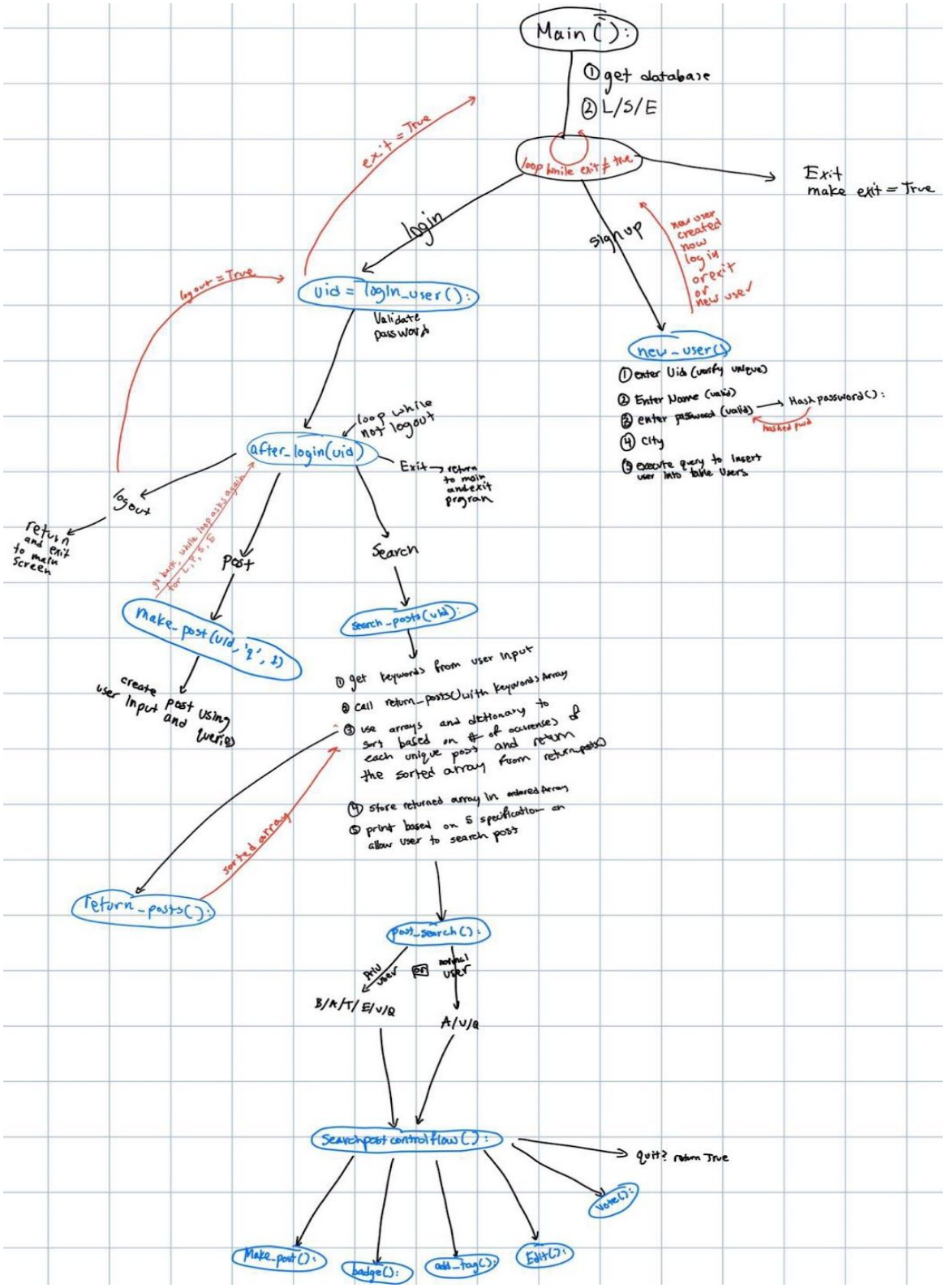

Figure 1: Flow of data between components