SECTION A - GENERAL OVERVIEW

OVERVIEW

The program created provides a python-based command line interface to interact with a SQLITE3 database. The system is designed to mimic that of a basic question forum.

LOGIN SCREEN

Here the user can either sign in or register to enter the system. Both options offer instructions on how to

USER TYPES

There are two types of users, regular and privileged. Only database administrative personnel can make a user privileged. This will become relevant further below.

MAIN MENU

Upon successful login, users are prompted with the main menu. Here the user can select from the prompted options by entering the corresponding number beside the option.

POSTING QUESTIONS

The user is prompted to input both a title and text body for their question.

SEARCHING POSTS

This interface allows users search through posts using provided keywords. The user can enter as many keywords as they like. Each keyword should be entered individually. Results are limited to 5 per page. If there are no results, the user is returned to the main menu.

SELECTING A POST (POST SEARCHING CONT.)

The user can select a post ID by terminating their page search and entering a post ID provided in their search. Users can also return to the main menu by entering “return”.

POST ACTION MENU

This menu provides the actions available on the selected post. Actions may be limited based on the instance’s circumstances (SEE BELOW)

POSTING ANSWERS (POST MUST BE QUESTION)

The user is prompted to input both a title and text body for their answer which will correspond to their selected question.

CASTING VOTES (ALL USERS)

Upon selecting this option, the user automatically upvotes the selected post. It is to be noted the user can only vote on a post once.

MARKING ACCEPTED ANSWERS (POST MUST BE ANSWER & PRIVILEGED USER ONLY)

Selecting this option assigns the selected post ID as the official answer to the question it refers to.

GIVING BADGES (PRIVILEGED USER ONLY)

The user will be able to assign a badge to the poster of the selected post. The user will be prompted to enter the badge name they wish to assign to the poster.

ADDING TAGS (PRIVILEGED USER ONLY)

Selecting this option allows the user to add tags to the selected post. The user can enter as many unique tags as they please. After a tag has been entered, the user is asked if they wish to enter more. “Y” for yes, “N” for no. Each tag should be entered individually.

EDITING POSTS (PRIVILEGED USER ONLY)

Selecting this option allows the user to alter the selected post’s title and/or body. The user can choose to edit one,all or none of the provided fields.

LOGGING OUT

Selecting the option to logout will return the user back to the login screen. Here the user can sign back in using their username and password or register as a new user.

QUITTING

Selecting the option to quit will safely save and shutdown the database connection, thus terminating the program.

SECTION B - FUNCTION DESIGN

main()

Starts by gathering database information via GetDB(). Continually loops LoginSignUp() for a valid response. Upon validation, LoginSignUp() returns a userID which is then fed to MainMenu(). MainMenu() is looped until the user wishes to quit the program. Upon quitting ExitProgram() is called to safely save and close the database.

LoginSignUp()

Determines whether the user is existing or new through a simple y/n/q prompt. If the user is existing, they are prompted for their credentials which are then verified within the database. If incorrect, the user is brought back to the initial login screen where they can try again. If the user is new, they are prompted to enter their information (uid, name, pwd, city). Error checking is in place to ensure none of the fields are left empty and that the UID is not in use within the database via isUniqueID(). All user-inputted values and the system's date are inserted to the *users* table. A successful login sends the user to MainMenu().

MainMenu()

Prompts the user with the option to post a question, search for a post, logout or quit. The user will have to enter the numerical value corresponding to the action. Choosing an action will call its corresponding function. Ex: 'Post a question' will call PostQuestion().

PostQuestion()

Prompts the user for both the post title and body. It ensures that the neither are empty before gathering the system's date and calling makePostID() for a unique post identifier. All of these values are inserted into the posts table. The pid is then inserted into the *questions* table with the answer ID left as NULL.

SearchPost()

The user is prompted to enter a search term, they can enter more search terms by using the y/n prompts. The values are appended into a list and then used to search the database for all applicable posts. The top 5 most relevant posts will be displayed to the user, each post will also display the corresponding number of votes and answers they have. If the user wishes to see the next 5 posts, they can accept the prompt asking if they wish to go to the next page of results. "Y" for yes, "N" for no. If no results are found the user is prompted with "No relevant posts" and returned to the main menu by calling MainMenu(). If the user is satisfied with the given page of results, entering "N" will stop the next page prompts, allowing the user to either enter a valid post ID or return to the main menu via entering "return". The following input is checked to ensure it is within the list of post ids gathered in the search, or that it is a direct match to "return" (case insensitive). Upon entering a successful post ID, the postID is passed along to PostActionMenu() to carry forth further operations.

PostActionMenu()

Users are prompted to enter a numerical value corresponding to the choices printed on their screen. Choices include: Posting an answer, voting on the selected post, returning to main menu, logout, quit. If

the user is privileged, they will also have the options for marking an answer as accepted, giving badges, adding tags, and editing the post. This function will check if the user is privileged before prompting options, ensuring un-privileged users do not see options they cannot use. Input restrictions are put in place to ensure that the user can only choose from what is available to them. This is done via a string OptionList that is altered based on whether the user is privileged or not. Selecting an option will call the corresponding function and pass it the current user's ID and the previously selected postID.

AnswerPost()

Only if the selected post is a question will this function continue to operate. If not, the user is returned back to PostActionMenu(). This is done by verifying the postID exists in the questions table. The user will be prompted to enter a title and text body, both of which will be checked to ensure they are not blank. A date will be generated based on the current system date, and a unique post ID will be generated via makePostID(). All values will be inserted into the *posts* table. The questions's ID and current postID will be inserted into the *answers* table. The user will be returned back to PostActionMenu() after failure OR success.

VotePost()

The function will first check if the current userID has already voted on the selected post. This will be done by validVote(), which will check the SQL database for an entry containing the postID and userID. If the post is deemed valid, a vote number is generated by makeVno(), which scans the post id for the highest vote number and returns a number one higher. The postID, votenumber, current system date and current userID is inserted into the *votes* table. The user will be returned back to PostActionMenu() after failure OR success.

MarkAnswer()

The function first checks to see if the selected post is an answer by referencing the *answers* table. It then gets the question ID the answer is referencing via getQuestionID(). With this quesitonID, it checks whether it has an allocated answer via hasAnswer(). If it does not have an allocated answer, updateAnswer() is called, updating *theaid* in the *questions* table. If there is already an allocated answer, the user is prompted whether they would like to overwrite it or not. If so, updateAnswer() is called to enact the changes. The user will be returned back to PostActionMenu() after failure OR success.

GiveBadge()

The function first gets the poster's Id from the given postID by calling getPoster(), which references the *posts* table to acquire the data. The user is prompted to enter the name of the badge they'd like to give. The input is cross referenced with all badge names via ValidBadge(). To ensure uniformity, getRealBadge() is called to get the properly written badge name from the database. This avoids any case-sensitive errors. The badge name, posterID and system date are inserted into the *ubadges* table. The user will be returned back to PostActionMenu() after failure OR success.

AddTags()

The user is prompted to enter a tag they'd like to add to the selected post. Their input is validated by validTag() to ensure it is currently not used by the postID. They are then prompted if they'd like to add another tag. This continues until "N" is selected. All tags are appended to a list of new tags. The system then loops through the list and inserts each tag into the *tags* table using the selected postID. The user will be returned back to PostActionMenu() after failure OR success.

EditAPost()

The user is individually prompted whether they'd like to edit the title and the body of the given post. If the user selects yes after a prompt, they are prompted to input the replacement text. The *posts* table is

updated accordingly. The user can choose to edit one, all or none of the given options. The user will be returned back to PostActionMenu() after failure OR success.

SECTION C - TESTING STRATEGY

Comparison of SQL query results before and after a program function were essential in confirming the database was being updated properly. This took place on nearly all python functions that altered a table within the database. All queries within the python code were checked to ensure they were gathering the correct data and that their corresponding functions were reacting properly to this data. On top of functionality, case sensitivity and bad inputs were a big concern. Many changes took place to ensure improperly written inputs would not break the program. This covered all user prompts and sql queries, excluding anything password related.
SQL data used was referenced in the README.txt. It was edited by Tyler Bach to adequately suit the changes in this project's database.

SECTION D - BREAK-DOWN STRATEGY

Communicated with each other via email and Social Networks when any questions, concerns or ideas came about. Clear and consistent line of communication was maintained throughout the project.
Main idea was that project members worked on separate portions (I.e. Python backbone, SQL data, SQL queries, Report etc.) and towards the last week, bring it all together and merge our work whilst communicating with each other. Files were kept on group google drive which included a to-do list.

--SANAD--
Implemented Entirely: Near entire program backbone with pseudo code for unimplemented portions. SearchPosts()
Partial Implementation: Code Commenting
Tweaked/Polished: main(), main_menu(), answerPost(), VotePost(), AddTags(),GiveBadge(), MarkAnswer(), EditAPost()
Time: 1.5 Days of work spread over 1 week.
--TYLER--
Implemented Entirely: CheckPrivilege(), validVote(), CheckInput(), ExitProgram(), makeVno(), makePostInfo(), makePostID(), hasAnswer(), updateAnswer(), relevantAnswers(), getRealBadge(), validBadge(), validTag(), isUniqueID(), getPoster(), PostActionMenu(), getQuestionID()
Finalized Implementation: PostQuestion(), answerPost(), VotePost(), AddTags(), postIDVerification(), questionIDVerification(), answerIDVerification(), GetDB(), GiveBadge(), MarkAnswer(), EditAPost(), LoginSignup(), SearchPost()
Tweaked/Polished: main(), main_menu(), lab machine testing
Time: ~28 hours
--IVAN--
Implemented Entirely: Complete rework of LoginSignUp(), uniqueIDVerification(), passwordVerification(), validMatch(), GetDB(), Code Commenting.
Finalized Implementation: Complete debugging and testing of code using old backbone (before we changed the backbone of program).
Tweaked/Polished: main()
Time: ~30 hours