Design Document for Mini-Project 1

Mac Malainey, Sandipan Nath, Tomas Peschke

User Guide:

Setup Instructions:

Run setup.py using the python runtime you wish to use. This will install PyInquirer, a necessary package to run the program
Example command:
python3 setup.py

Running instructions:

To run the application run main.py using python with the database path as an argument
Example command:
python3 main.py data.db

Note 1: When given a list of input, use arrow keys to navigate list
Note 2: When assigning a badge, instead of providing the name of the badge, the user is given a list of badge names (pulled from the database) to choose from

Program flow:

1. First the user will be asked to login or register.
2. Once successfully logged in (or registered) the user will be brought to the main menu where they can do the following:
    a. Create a post
    b. Search for posts
    c. Logout
    d. Exit
3. If the user creates a post they will be prompted for a title and then the body
4. If the user searches for a post they will be prompted to list keywords (seperated by ';'), at which point if the search comes back successful, they can select a post
5. When a post is selected the user will be directed to a new menu where they can perform the following:
    a. Upvote
    b. Answer (if post is question)
    c. Mark accepted answer (if post is answer and user is part of privileged group)
    d. Add tags (if user is part of privileged group)
    e. Give poster a badge (if user is part of privileged group)
    f. Edit post (if user is part of privileged group)
    g. Return to main menu
6. After an action is completed the user is returned to the main menu

General Overview of System:

The system allows users to post questions, respond to answers, and vote on posts. Users that are deemed as “privileged” are allowed more functionality such as marking an answer as the accepted question answer, adding tags to a post, rewarding badges, and editing posts.

The system works by first connecting to the provided database. The system is set up to autocommit everything. If no database is found at the given path, one will be created and initialized. Then the system will allow the user to login or register a new account. Once the login or registration process is completed, all actions from then on are done as the user that just logged in (until the application is quit or the user logs out). Once logged in the user has the ability to ask a question, search for posts, logout or exit. The system keeps track of the user by holding onto a “session” (see components breakdown). The session also caches whether or not the user belongs to the privileged group. The system then passes both the database and session over to the main menu.

When the program hits the main menu it gets put in a loop until the application is either exited or the user logs out. When a user posts a question they are given a simple CLI form to fill out. The information grabbed from any cli interface is passed back to the main menu which processes it further, and then passes any relevant information onto the database. When a user searches for a question, the user can input multiple terms to search by and it will search the database for posts with matching terms. If any matches are found the user can then select a post and perform the following operations on it:

- Upvote
- Answer (if post is question)
- Mark accepted answer (if post is answer and user is part of privileged group)
- Add tags (if user is part of privileged group)
- Give poster a badge (if user is part of privileged group)
- Edit post (if user is part of privileged group)
- Return to main menu

Post and vote IDs are created in order starting from the maximal value in the database (see components breakdown)

Component Breakdown:

Components involved: Entrypoint (main.py), UserSession class, Database class, menus.py, PyInquirer.

There are four levels of code: Codebase is in main.py, menus.py, database.py, and cli.py. Main.py has three tasks. Firstly, main.py initializes the database class, using a file that is passed into the arguments. Next it uses the cli functions to walk the user through logging in or

registering a new user. On a successful login the user session is stored and then passes it, and the database object, along to the main menu (in menus.py).

The way that menus.py is setup is that it is responsible for program flow and responding to menu choices. At each menu the program prompts the user for an action, if the action requires input it will prompt the user for more input, process the input (if any), call the appropriate database functions, notify the user, and then move on to the next place in the code. There are two levels to the menu, the main menu, and then a secondary menu which can only be accessed if the user selects a post. This secondary menu shows different options depending on whether or not the user is privileged and the type of post (question or answer).

The file cli.py handles prompting the user for input and retrieving input. It accomplishes this by heavily relying on the PyInquirer module. For each set of prompts (a form) a list of dictionaries is stored to be called. These values are what PyInquirer takes in order to prompt the user for input. Each prompt has its own form, some forms ask for multiple inputs, some do not. Cli.py then provides helper functions that automatically handle the prompting (passing to PyInquirer) of a form and doing basic unpacking of the input. Most questions that are asked to the user remain constant throughout the whole program, so the functions that use these forms just return the result from the prompt after being unpacked. However, there are some forms that depend on information about the user. For example, when selecting a post from those that match a given keyword, the function that uses this form modifies that form on the fly to include the posts that were searched. The general principle behind a function in cli.py is that it prompts the user with a form, and unpacks and returns the result.

Database.py has code for two classes. The first class “Database” is a wrapper class for the sqlite3 database. This class exposes helper functions that can be used to query and insert various information. All the database accesses, SQL scripting, query forming, and basic unpacking of the data is handled by this class. In order for the methods to work, an instance of the database object must be initialized first on a given path to a file. We initialized the database such that it will autocommit changes. If the file does not exist a file will be created and populated with tables using prj-tables.sql. Database.py also defines the “UserSession” class, which models connection a user currently has with the database. A UserSession object is created and returned by the database when the open_session or register methods are given valid data. A session object keeps track of whether a user has logged out and if that user is part of the privileged group.

Testing:

For testing we created a secondary script (admin.py) that allowed us to alter the database for other testing purposes. This script’s functions included adding members to the privileged group, removing members from the privileged group, and adding or removing badges. Testing was not done on a component level, but was done system wide. We then came up with various test scenarios to validate the system. The following test scenarios were run:

Login / Registration Testing:

- Login with incorrect information (both bad, pass good, username good)

- Login with correct information
- Register with duplicate PID
- Register with unique PID

Post Creation and Search:

- Add post from question screen
- Add post from answers screen
- Verify they are searchable from title and body, and are proper type
- Add tags and verify they are searchable via tags

User commands:

- Verified that all normal user commands on a post work
- Verified that all privileged user commands on a post work and only show if user is privileged

SQL Injection:

- For each user defined input (post creation, tag creation, searching, etc) ensure that SQL injection is not possible by attempting to perform SQL injection

PID and VNO generation:

- Ensure that PID and VNO values are always generated appropriately, especially if there are:
  - No prior votes or posts
  - Prior votes and posts
  - Post ids remain in bounds (4 characters)

Work Breakdown:

We used discord to communicate. Before we started the project, we brainstormed some ideas as to how the architecture of our code would look like. After settling upon the basic framework, we divvied up the tasks and got to work. We used github so that we could work simultaneously. This allowed each member to keep track of other's progress by seeing what they were pushing and how much work was done. Everyone was contributing at a reasonable rate, so apart from written reminders in the discord chat, we didn't need a sophisticated system to keep each member accountable.

Person	Progress and Time	Items completed
Mackenzie Malainey	35% - 5 hrs	Login operations, Database initialization, Privileged User Functions, PID generation, UserSession
Sandipan Nath	40% - 6 hrs	Search Post, Select Post, String matching, Creating the Menus, Post input
Tomas Peschke	25% - 4.5 hrs	Docstrings, comments, design doc, code cleanup, bug fixing, testing