CMPUT 291 Mini Project 1 Report

Abdul-Azeez Abass (abass) Cynthia Li (sqli) Michael Kwok (mkwok1)

November 2020

1 Project Overview

This program utilizes a local SQLite database to provide the user a question-answer forum to interact with. Interactions are done via selecting provided options in each state.

Upon startup, the user may choose to login, sign up, or exit the program. After logging in or signing up, the user will be brought to the main menu where they can create a new question post, search posts, or log out of their account (which brings them back to the first screen). Creating a new question involves setting a title and body, after which the user is taken to the post view of their question where they can make post actions.

These post actions vary depending on if the user is privileged, but all users have the option to answer the question if it is a question, and to vote the question if they have not voted it already. Privileged users have further actions of marking an answer as accepted if the post is an answer, give a badge to the user if they have not given the badge they chose to give already today, add a tag to the post, and edit the title and/or body of the post.

If the user makes a mistake in action choice or data entry, the program reminds the user that they made an error and allow a retry. When a user makes a failed attempt at login more than 3 times, the counter is reset and the user is brought back to the initial screen to login, create a new account, or log out. A new user, however, has any amount of tries to register a new account if their chosen uid is already taken.

The score is the number of votes a post as received. When entering a password during login or registration, a user will not be able to see their input but it is received nonetheless.

2 Running Instructions

This project require blessed python module: pip install blessed python main.py

3 Design

The project roughly follows the Model-View-Controller pattern, with the `Database` class doing the majority of the data structure and SQL access work.

The `Database` class handles the connection to the database file and determining whether the initial setup script `setup.sql` needs to be run. It gathers the maximum values of `pid` and `vno` every time the program starts to keep it monotonically increasing for each new post and vote. Most methods in the database work by producing the expected result as a returned value, such as `login()` returning a `User` object that should be reused. It was written with a functional style in mind, avoiding the mutation of state whenever possible unless it would make sense otherwise. The `User` and `Post` classes are the main return types used by `Database`.

`User` and `Post` contain what would be the records in the database, with relevant information included such as vote count and search ranking for Posts.

The user interface and control flow is handled by a state machine. States are defined when the program starts, with and each state having it's own output for the user interface. The state machine calls into the Database class to perform actions, fitting both the View and Controller in MVC, also allowing for cleaner seperation between parts, allowing simpler bug fixes and ensuring that bugs in one part will not affect the other.

4 Testing Strategy

Since each component of the system was developed separately and at different times with the `Database` module coming first, it has a suite of unit tests which currently has about 70% coverage of the file.

The `Database` module was mostly developed with Test Driven Development as the main process, which allowed progress to be tracked, and code to be tested as it was written. The output of functions were compared with similar but not identical SQL commands directly to the database to verify that each function ran as expected. Both failure and success situations were tested in test suite, making sure that the specification was provided was followed.

Integration testing was done manually, by testing each state and part of the interface by hand as setting up automated testing for this part would have been overkill.

Most bugs found were logic errors from processing the results from SQL, but a few were due to python's weak typing where things were not returned, or certain things were unexpectedly of the `None` type.

5 Groupwork Breakdown

The group convened through the help of a Discord group chat, and code was hosted in a private GitHub repository. Black.py was used to ensure uniform formatting of code, reducing merge conflicts.

- Azeez
  - Time Estimate: 8 hours
  - Built state machine manager interface
  - Worked on states: Login (refactored), Accept as answer
  - Wrote some test data
- Cynthia
  - Time estimate: 8 hours
  - Designed state flow of system
  - Implemented most different states
  - Created user interface
  - Co-authored the report document
- Michael
  - Time Estimate: 10 hours
  - Wrote all of `database.py`
  - Unit tests in `tests.py` and corresponding test data
  - Made the `user.py` and `post.py` 'models'
  - Co-authored the report document

A "Return to Main Menu" option was added to get out of viewing a post, which was not mentioned in the specification