CMPUT 291 Mini Project 2 Report

Cynthia Li (sqli) Michael Kwok (mkwok1)

November 2020

1 Project Overview

This program utilizes an online MongoDB NoSQL database to store data for posts, votes, and tags. The user interacts with the system by selecting options displayed on screen in each state.

Upon start up, the user may choose to enter a numeric user id. If the user id is valid (contains only numbers), a user report is generated which outlines the number of questions and answers the given user has posted, along with their average scores, then displays the total votes the user has made. Otherwise if a user id was not provided or if the user id is invalid, the program proceeds to the main menu.

In the main menu, the user has the option to post a question, search for questions, or exit the program. In posting a question, the user is asked to enter a title, body, and tags for the post. Upon finishing, the user is taken to view the newly posted question with all its newly generated data that is stored in the database. From there, several options are presented to the user: answer the question, list all answers to the question, vote on the question, and return to the main menu. Answering the question will have an extra line appear to prompt the user to write an answer to the question, to which once the user is finished, will take the user to view the answer inidivudally. The answer view shows the original question of which the answer is responding to, along with the answer itself as well as two actions the user can take: vote or return to main menu.

When the user selects list answers from the question screen, a new table will be displayed on a new screen that shows the accepted answer first (if any exists), and then other answers to the question. The user can select any of the answers listed by their number and go to view the answer individually.

When the user selects the vote action on a question or answer in focus, the score of the post wil increment by 1 if the user is logged in and has not previously voted on the post already. Users who did not provide a user id at the login screen are free to vote without constraint on the post.

If the user chooses to search questions, they will be first prompted to enter keywords delimited by spaces. Upon pressing enter, a table of search results will be displayed in pages of 5. The user can select any of the questions by their numbers in the results or select the see more option to display the next page of search results. The user may also choose to return to menu on the search results page.

2 Running Instructions

To install the dependencies of this program, run the following:
`pip3 install -r requirements.txt`
Start MongoDB running with:
`mongod --port [port number] --dbpath [path]`
To run the program, the following commands can be used:
`python3.5 phase1.py [port number]`
`python3.5 main.py [port number]`
Port number is optional. A default of 27017 is assumed.
Main menu will appear, and user can choose to input their IDs or otherwise.

When creating a new post, each tag must be seperated by a space character. Duplicate tags will be handled by the system.

3 Design

The first phase of the project is written as a simple python script as it's main job is to import the data and no processing. Titles, bodies and tags of posts were indexed as text to improve search speeds at the 2nd phase. Each collection was built in a different thread, as we found that this improved speed by about 20% with no changes to code.

In the 2nd phase, the user interface and database access functions are split into two files. The main file performs user input handling and displaying the user interface, also navigation between the menus. We rely on `blessed`, a curses library for python to handle clearing the terminal, positioning cursors and adding menu titles. We also use `PrettyTable` to display data. User interface flow is handled by a simple switch/case instead of a heavyweight state machine as we did not require tracking of previous state or anything complicated, and just a list of screens to show. Each state calls into the `Database` class to perform actions as requested by the user.

As previously mentioned, we use a second file. In this second file, we have the `Database` class and related functions such as a function that generates a date string that follows the required format and another that converts a list of tags into the string as stored in the database. The `PyMongo` library was chosen to access the database as it is the official library provided by MongoDB. Both normal queries and aggregation were used in conjunction as required to keep actions performant. As we were not as familiar with NoSQL as we are with SQL, the format of the queries were not consistent (e.g. implicit vs explicit `"$and"` queries), however most of these inconsistencies should be purely cosmetic and should not affect program performance.

4 Testing Strategy

The entire program was tested manually by following the assignment specification and grading rubric, checking to see what was missing in each state and implementing missing functionality as found.

Most initial bugs found were due to lack of errors in NoSQL, where queries don't fail but simply return no results instead. Subsequent bugs were mostly interface issues as getting a command line interfaces right isn't something that we have much experience with.

As clarifications about performance were released, each part was tested with a stopwatch on our local machines to estimate the amount of time it took, and we found that slower parts were mostly due to our inexperience with NoSQL, and we improved our queries to speed up execution greatly.

5 Groupwork Breakdown

The group coordinated through private messaging, and code was hosted in a private GitHub repository.

- Cynthia
  - Time estimate: 10 hours
  - Designed user flow
  - Implemented most states
  - Created user interface
  - Co-authored the report document
- Michael
  - Time Estimate: 12 hours
  - Wrote most of `database.py`
    - `visit_question()` was written by Cynthia
    - `vote()` was completed by Cynthia
    - `get_report()` was initially written by Cynthia then improved by me
  - Implemented the list answers and some of search states
  - Did initial implementation of `PrettyTable` printing
  - Co-authored the report document