Design Document for Mini-Project 2

Mac Malainey, Sandipan Nath, Tomas Peschke

User Guide:

Phase 1 Running Instructions:

Run setup.py using python3. This will install PyInquirer, SimpleJson, and PyMongo packages. If a port number is provided as an argument phase 1 will run after installing the necessary packages. Otherwise run phase1.py using python3 with a port number.

Example commands:

```
python3 setup.py 20717
-- OR --
python3 setup.py
python3 phase1.py 27017
```

Phase 2 Running instructions:

Run phase2.py with a port number as an argument (after setup.py and phase 1 are complete)

Example command:

```
python3 phase2.py 27017
```

Note 1: When given a list of input, use arrow keys to navigate list

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, search questions, answer questions, list answers to a question, and vote.

The system works by first attempting to connect to a database at a given port. If no database is found, the program will exit. On a successful connection the system will allow the user to provide a user id or use the system anonymously. Next all actions from then on are done as the user that just logged in (until the application quits). After login, if a user id was provided the user will be shown a report of the user's prior activity. The user then is sent to the main menu where they can post a question, find a question, or exit. The user id is passed through the application as needed.

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 questions with matching terms. If any matches are found the user can then select a post and perform the following operations on it:

- Upvote
- Answer
- List Answers
- Return to main menu

If the user looks through the answers they can view and select answers to vote on or return to the main menu. Post and vote IDs are created in order starting from the maximal value in the database (see components breakdown)

Component Breakdown:

Components: Entrypoint (phase2.py), Database class (mdatabase.py), menus.py
Dependencies: PyInquirer, Simplejson, PyMongo

The code is separated into 2 components. Phase 1 sets up the database using provided data and does any additional processing. Included with phase 1 is a setup file that installs any necessary dependencies

Phase 2 is broken up into three sections of code: phase2.py, menus.py, database.py, and cli.py. Phase2.py is the entry point of the application. It first connects to the database then passes that on to the rest of the application.

Menus.py is responsible for program flow and responding to menu choices. The highest menu (first one shown) is the main menu, from there the user is able to select an action.

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. Some actions will lead the user to another menu such as the action list for a post.

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. 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 abstracts these away in functions that automatically handle the prompting (passing to PyInquirer) of a form and doing basic unpacking of the input. Some 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 change depending on choices made. For example: selecting a post from those that match a given keyword. The function that uses this form modifies its form 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 is an abstraction for our Mongo database. The database class is a wrapper for the PyMongo database object. This class exposes helper functions that can be used to query and insert various information. All the database accesses, query forming, and basic unpacking of the data is handled by this class.

Testing:

For testing we ran through each menu and action to verify they worked appropriately

Work Breakdown:

We used discord to communicate. We agreed to model our project structure similarly to our submission for project 1. 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% - 7 hrs	Phase 1, setup.py, 70% of cli.py, phase2.py, design doc, refactoring and optimization
Sandipan Nath	35% - 7 hrs	Docstrings, comments, optimization, database queries
Tomas Peschke	30% - 6 hrs	Creating menus, phase2 program flow, cli, database queries