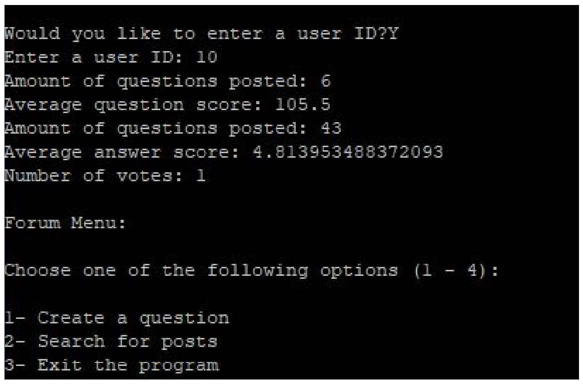
General overview of the system and user guide:

The program we created is a simple ask question, answer, search questions, and vote forum implemented in Mongodb. Users can provide a user id or not. The program is subdivided into 2 parts, the phase 1 section and phase 2 section.

The phase 1 section is related to creation of 3 collections (Posts, Votes, and Tags) from Posts.json, Votes.json, and Tags.json. Phase 1 will take in a port number where the mongodb server is being served and it will either create a database “291db” if it does not exist or use “291db” if it does exist. It will then create the collections Posts, Votes, and Tags and create a Terms array and build an index based on that Terms array for searching done in phase 2.

Phase 2 much like phase 1 takes a port number where the mongodb server is being served. Once phase 2 starts the user is prompted to either enter a user id or no user id. If the user selects Y then Yes the user wants to enter a user id which the program will ask. There is no restriction on the user id that the user can choose so they can pick anything so long as it is a numeric field (ex: 10). The user can also select no user id by typing N for “no”. If the user chooses to put a user id then a report is shown showing the amount of questions posted, average question score, amount of questions posted, average answer score, and the number of votes. If no user id is given the no report is shown. This is shown below as an example.

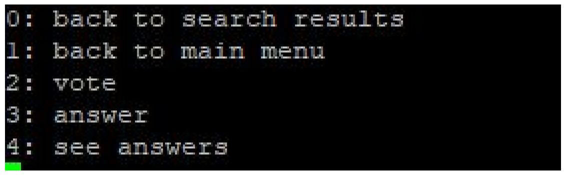

```
Would you like to enter a user ID?Y
Enter a user ID: 10
Amount of questions posted: 6
Average question score: 105.5
Amount of questions posted: 43
Average answer score: 4.813953488372093
Number of votes: 1

Forum Menu:

Choose one of the following options (1 - 4):

1- Create a question
2- Search for posts
3- Exit the program
```

From the above figure we can see that there are 3 main options in the main menu. The first is create a question. This will prompt the user to enter a title for the question, body, and (optional) tag(s). Once the user creates a question they will be put back to the main menu where they can proceed to the other options. The next option is searching, where they can search by entering space separated keywords like “macbook intel” which will return all the search results containing at least one of those keywords. The user can then select a question from the list. Once the user selects a question from the list the user is prompted with a set of actions.


```
0: back to search results
1: back to main menu
2: vote
3: answer
4: see answers
```

The above figure shows the set of actions. When a user chooses to vote they will increase the vote count on the question, if a user id is provided the user can only vote on a question and answer once, otherwise they can vote as many times as they wish. If a user chooses answer option then they can answer the question, and lastly the other option is see answers where the user can see the answers made on the question and see the accepted answer if there is an accepted answer indicated by a * symbol and listed at the very top of the list. The user can select an answer from the list and vote on it also increasing its score count. The user at each point of the menu screen can choose to go back to the previous screen, go back to the menu and at the main menu they can choose to exit the program by choosing option 3 as seen in the first figure up top.

Software Design and architecture:

The high level summary of the program is we divided the program into 2 phases and the phases are explained in the previous section. The overall design of phase 2 follows 4 major components: menu functionality, post actions, searching for posts, and making posts. Posts is defined as a question or an answer. The menu functionality is in the starting point of the program and where the menustack.py is called is the phase2_OperateStore.py this is the main menu screen where everything will start and end. From here the user can select either to post a question or search for a question and whichever choice the user makes it will keep track of the state of the menu in a stack. The post actions are located in postActions.py which contains the post actions: vote,answer, list answer, and answer a question the functionalities are implemented using both python and mongodb queries. The searching for posts functionality is located in the search post and works by taking in an input of keywords from the user. Once the user inputs a keyword the program checks if the keyword inputted is 3 characters long and if it is a mongodb query is executed utilizing the Terms array index built in phase 1. If the keyword is less than 3 characters then a simple find and regex match query is executed that will search through the title and body for such a match. Lastly a separate python file was created for making a post which will take in a title, body, and tag. Separate mongodb queries were made in that file corresponding to if the user id is given and or if a tag is given.

<u>Testing strategy:</u>

We used multiple testing strategies to ensure that our program runs smoothly and to meet the objectives of this project.

- We started by testing the code we wrote to ensure that the program doesn’t have any errors when compiled. We followed correct syntax and also made the code readable so that when we test the functionality of the program, it would be easier to find and fix the errors we encounter. It also helped us as a group to find errors faster. We also tested the connections to the mongoDB server to ensure that our programs don’t quit on us because of an error of connection. We made the port number as an argument to make it variable for anyone to use.
- We then tested the user inputs and ensured that it doesn’t produce any incorrect result. We wanted the user to enter any string format and still be fine. We used safe input for most of the inputs of the program. This makes it so that we don’t stop asking the user until we get the desired input we need. It also makes the program dynamic enough so that the user can enter anything into the program. It also reduces our testing time for these special cases. We also tested and made sure that the terms of each question is correct, and doesn’t contain any weird characters like ?@*(!)~&<>. In addition, we tested the searching portion of the program. We wanted to make sure that every search output is correct and doesn’t contain any additional questions. Examples of inputs we used: ppc , 4k, ergonomics.
- Lastly, we wanted to make sure that our program loads the json files and create the terms in less than 5 mins. We used a couple of ways to insert into the database, but we found that inset_many() is the most efficient way to insert into the db.

<u>What each group member did in this project:</u>

- Mohammed Alzafarani
  - I implemented the following parts:
    - Post a question (~2 hours)
    - Question action-Answer. (~1.5 hours)
    - Question / Answer action-Vote (~40 mins)
    - The report of the user (after login) (~1.2 hours)
  - I helped in implementing the following parts:
    - Question action-List answers (~1 hour)
    - Phase 1 implementation (~30 mins)
    - Testing strategy and methodology (~2.2 hours)
    - Helped in the creation of the report (~1 hour)

- Faisal Redwan
  - I implemented the following parts:
    - Class framework (~ 3 hours)
    - Searching functionality (~ 5 hours)
    - Post displaying for questions and answers(~ 3 hours)
  - Helped in commenting (~ 15 mins)
  - Testing strategy and methodology (~ 1 hour)

- Justin Diala
  - I implemented the following parts:
    - Phase 1 implementation (~ 20 hrs)
    - Portions of the searching queries (~0.5hrs)
    - Portions of the answer queries (~0.5hrs)
    - Portions of the tag queries (~0.5hrs)
    - Portions of the voting queries (~0.5hrs)
  - Helped in the creation of the report (~0.5hrs)
  - Helped in coordinating a plan for the group and what tasks need to be done and what deadlines and goals need to be met on each day (~0.5hrs)
  - Testing strategy and methodology (~3hrs)

Project breakdown:

- The project was broken down into small parts, we knew we had to implement a menu stack interface or some sort of menu tracking system, voting action, posting action, answering action, list all the answer action, searching action, and a phase 1 task that tackled the creation of the db and collections. These were the main “features” of the project. We primarily used AGILE terminology to maintain a steady coding velocity and at the same time provide good output.
- We agreed on a language that we were all comfortable with which is python. Once we decide on a programming language we proceed to gauge our skills in programming and where each person is more comfortable with. Since we have worked before as a team we were all quite familiar with each other's strengths and weaknesses. We just trusted each other to take a task that they felt that they can achieve in a timely manner. We just allowed our team members to pick a task and ask for help when they can. We kept each task in a bucket of tasks and via discord calls we would have mini scrum sessions indicating what we did and what still needed to be done and what I am stuck on. Also we implemented iterative testing doing mini tests and reporting issues if we find any.
- So in summary we created a bucket of tasks and posted them in our chat. We then just allowed any of us to just grab whatever task needed to be done. We used Agile to hold mini call/chatting sessions

each day to keep track of progress done on that day and if anyone needed help we just ask one another.