# DESIGN REPORT

Author: Le Minh Dang (mldang), Zi Xue Lim (zixue)

## Overview:

Our project is separated into many files which are all written in Python3, except for phase2Main.py, all other files contain the necessary components taken from the CMPUT 291 Specification. For this project, it requires a dependency ***pymongo***. The user can install it via ***pip3 install pymongo*** or create a virtual environment and install it in there if they do not want to interfere with the main system files.

In order to run *phase 1* for importing data to the server, users need to use the command "***python3 phase1.py dir/Posts.json dir/Tags.json dir/Votes.json"*** where ***dir*** is the directory containing the database file. Users can set the command in any order, but it requires exactly 3 JSON files. After that, it will prompt for the port so it will know where to connect to the server and import the data.

For *phase 2*, users need to use the command "***python3 phase2Main.py"***. The program will prompt for the port to connect to the server. We took some extra measurements and decided that we will create the text index for full text search instead of in phase 1 because we thought that phase 2 might connect to somewhere else instead of using the same port as phase 1. Therefore, users might need to wait a couple of seconds before entering the user ID. If a user ID is given, the program will create a report for the user about the number of questions owned and the average score for those questions, the number of answers owned and the average score for those answers, and the number of votes registered for the user. If no user ID is inputted, it will go straight to the main menu. In the menu, there are 3 options: Post Question, Search Question or Exit. For the first option, the program will prompt for body, title and the tags. For the second option, users can search for questions by prompting keywords. The software uses the full text search mechanism by MongoDB to search for posts and return in sorted order by the textScore it has. Users can choose the question by prompting number given by ***No.*** column. The program will generate a full report of that question, notice that it might not be in any order. After a question is chosen, users will have options to list answers, answer the question or vote the question. If the answer option is chosen, the software will prompt for the body of the answer, then it will generate a unique ID based on the object ID from the answer. If the list answer is chosen, users can choose one of the answers, which will also generate a report of full answer description. Similar to the question, the answer might not be in any order. In the answer screen, users can vote for it. And similar goes for voting questions.

## Detailed Description:

With the program being divided into multiple files, the components will be contained in these individual files:

1. *phase1.py*
    - This was where we read the 3 json files, Posts.json, Tags.json and Votes.json and constructed three collections from them, Posts, Tags and Votes. If the collections already existed in the database, they would be deleted, then remade again. These 3 collections will be stored in a database This part will also take in an input for the port number and will connect it to a server port where the MongoDB server is running.
2. *phase2Login.py*
    - This will prompt for the user ID. If a user ID is given, it will provide a full report based on the specs. Otherwise, it will go straight to the main menu
3. *phase2PostQuestion.py*
    - This part of the program allowed users to post a question by providing a title, body and tags. The program would give a unique id to the post that was just created and store it into the Posts and Tags collection.
4. *phase2Search.py*
    - This was where the user could search for a post by providing keyword(s) to the program to search through the Posts collection where it would match if the keyword appeared in a question where it had at least one of the following : title, body or tag field. The program would then output the results on the terminal where the user could perform a “Question Action”
5. *phase2Answer.py*
    - This part of the program allowed the user to answer a selected question from the search question. The user can provide a text which would then be stored accordingly with all the correct information.
6. *phase2ListAnswer.py*
    - Once selecting a question from the search user page, the user will be able to see the replies to the question from the *phase2Answer.py* part and the user will be able to select an answer to see all the fields. The user can also select an answer. The selected answer will be shown first in the question action list and a star will be next to it. The user can also select a reply to the question and vote on it (discussed in the following part)
7. *phase2QuestionAnswerVote.py*
    - Once selected a question(from selecting a question after searching for a question or selecting an answer to a question in the list answer part), the user can vote, where the score of the post will increase.

## Testing Strategy:

The testing we have done in this program relied on us using a test function (as seen in figure 1). The reason for this was to allow us to run only a certain part of the overall program as we had split the task up (discussed below). We did manual testing for this program by providing 3 different types of test cases, edge case, normal case and cases where it should break the program. We would check if the test cases matched what was expected to be outputted. We also relied on VSCode extensions such as pylance and MongoDB to observe and debug. The major issue we encountered was to test search. Because it returns so many results (> 10k) which is very hard to test.

To make the environment similar to the testing conditions done by the markers, we make sure to test it on lab computers (ohaton/ug33) to simulate a similar environment.


*Figure 1:* func_test() demo for QuestionAction Listing answer

## Breaking Down Strategy:

The specification on eclass helped us break down the project. We started from the first task and took the next higher task (in the specs) when we finished the previous task. We decided this would be the best as we would have a better idea of how to implement the programs together and also some task required a longer amount of time to complete the task. We used Github to collaborate with one another (at the time of writing this, the repository is in private mode, if required, we can make it public after the submit time). The following is how we distributed the task:

**Zi Xue Lim:** Question/Answer action-Vote, Post a question, Login Session, phase 1(40%)
**Le Minh Dang:** phase 1(60%), Search for questions, Question action-Answer, Question action-List answers, Main Menu.