**Mini Forum Program Design Report**

1. User guide page2
2. Design detailed implementation page3
3. Testing strategy page3

4. Group-work breakdown --------------------------------------------------

page4

# Preface (overview)

This is a mini background data view programme having searching and posting as main functionality, offering the manager a place to manage and administer problems and questions. With extra features such as searching, posting and so on for managers, we aim to create a relatively concise, user-friendly and easy-performed platform for our clients to use.

# User guide

1. At first, it will ask you the port number, you can just enter 27017, then it will ask you whether you want to provide a user id or not, if you want, you can just enter y and input the user id, then it will show 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. Users may also use the system without providing a user id, in which case no information is displayed.
2. The initial interface manifests itself in a concise way giving you 5 options, you can perform following actions:

By entering 1: post a question to the forum, you will be providing a title and body for your question, and it will ask you if you want to post current user id or add any tags.

By entering 2: you can search for existing questions with the key words you provided. Note that you can give more than 1 keyword and result will be based on most relevant posts to least relevant. After displaying, you are able to choose one question to check the complete information. Then it will ask you can choose to answer the question. After that it will ask you if want to see all the answers of the question and choose one to see full information. Then it will ask you if want to vote for the question or not.

By entering 3: you can answer a question by giving a post id and your answer text.

By entering 4: you can list answers for a question, then you can select an answer to check the full information and vote for it.

By entering 0: you exit the program

Other input will be regarded as invalid for the basic user on this page and will have no further effect.

# Detailed Implementation for Design

For this program, we select python as main language and mongodb as our embedded database system. Our core idea is to chop the whole program into functions and to run the program by function calling in main(). As to user interact interface, simple os print is used to show the menu/output. The functionality of the program is separated into 2 .py documents which own 11 huge function and stored in a dictionary so that the calling process can be mapped easily. By doing so function calling is easier and implementation between our group members can be more consistent. Most of queries are trivial and can be found back in assignment2. What worth mentioning is the implementation of search function and post function. In order to count the total occurrence times of keyword provided by user, we implement a dictionary( key:pid, value: occurrence of keywords) for each pid of post. Each time a keyword occurs the value corresponding to the key get incremented. After iterating over all user input keywords, we simply sort the dictionary based on values and obtain the ordered tuple. This way we can print out the result in descending order. In the function post\_qn and answer\_post, we exploited function attribute to track the valid post and to form the unique pid. Each time a post is formed, this attribute called counter get incremented and we set it as our unique pid. Finally, in the main function, while loop is used to keep the programming running without breaking. Exceptions are handled with the python built in try – except clause so as to user wont be able to break the flow by unexpected input.

# Test Strategy

There are 2 testing methods exploited in this program. Unit test and real-time test. For each function, we used a specific written test-function to test under several cases, this unit test approach makes sure that each small part of the program works properly, each function is doing its own job. Before we integrate all the functions together, each function went through unit test so no trivial errors are produced later on. After we glued everything together, unit test does not work effectively as it was so we switched to real-time supervising. We run our code at the same time supervising the change into the database. Each time we make a change/perform a user action, we will open our database see if the correct data is put into the right table. If not, we just look into the part with respect to this functionality and see where goes wrong. Cases we used in tests are general cases and special cases. By doing those we guaranteed the program running properly under majority of circumstances.