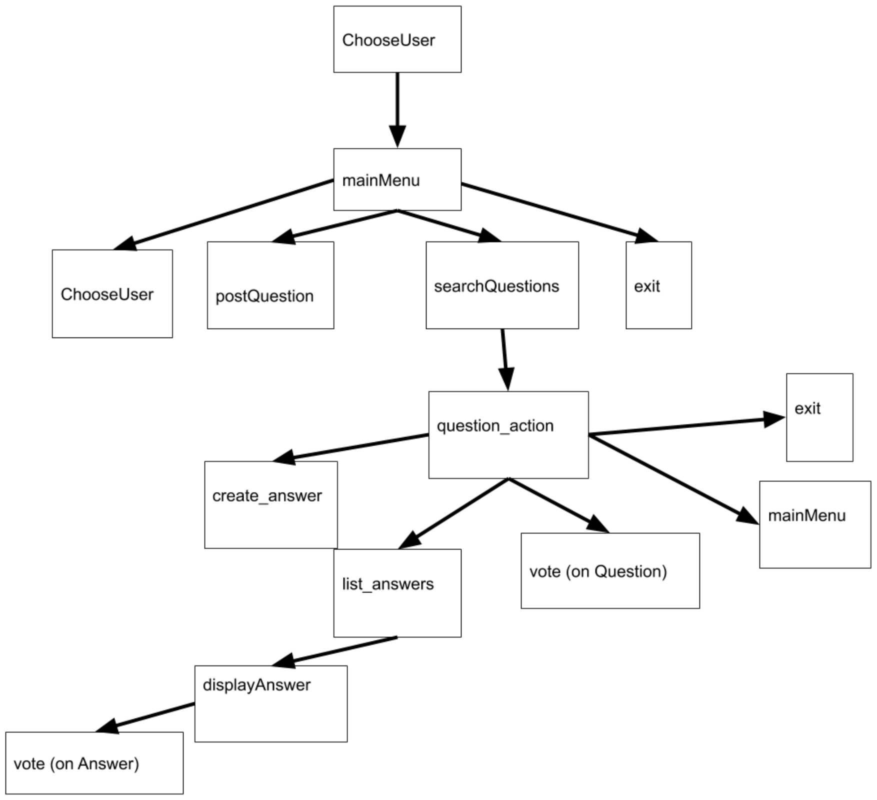
(a) In phase-1.py, the data will be loaded into the database. A message will be printed indicating that it was successful. In phase-2.py, the user will use the interface for the database to perform various actions such as searching questions, voting, etc. When phase-2 is running, messages will be displayed throughout which will guide the user.



phase-1.py will be run before phase-2.py.
In order to run the program, the following commands are used:
python3 phase-1.py <port>
python3 phase-2.py <port>
where <port> is a port number (eg. 27017)

(b) chooseUser(): The user is prompted to enter a user id (if they wish to do so). If a user id is provided, then a user report will be displayed and the user id will be returned so that it can be used in other functionalities (eg. postQuestion).

postQuestion(): The user is prompted to enter a title and body for their question, as well as zero or more tags. If the question has been posted, a message will be displayed indicating that it was successful.

searchQuestions(): The user is prompted to enter one or more keywords and the post results will be displayed. Once a question has been selected (by typing in a number), a message will be displayed to the user which indicates that the question has been selected. The unique id of the selected question will be returned.
create_answer(): The user can post an answer on the selected question.

list_answers(): The answers to the selected question will be displayed, with the accepted answer being at the top.

displayAnswer(): Based on the list of answers displayed, a user will choose one of these answers by typing in the index number as shown above each answer.

(c) At first we did unorganized testing. This simply involved as we were creating functionalities that captured the specifications of the assignment, doing our best to see if the functionalities worked on the fly. This meant feeding in test data and making sure that everything was working as expected. In certain cases this meant using the database to make sure our finds updates ect were working as expected. Once all the functionalities were implemented and brought together thus began the systematic testing. To perform the systematic testing we first went through the marking rubric. We turned the marking rubric into a series of actions to be executed and the correct results. For example, "User initial Screen- A user id can be provided (but is not mandatory), If a user id from the database is entered, a report is given (as per spec)" 1. On initialization -> User can provide user id 2. Enter in user id -> User report is given, user report is correct for said user.

We went through the rubric making sure that any functionality specified in this rubric was included in the program and then when it specified some result that should occur if we take said action in said circumstance we made sure we get into said circumstance, take said action and then make sure that our result was correct. In order to verify if the changes in the database and to our collections were done correctly a second terminal was used. This one accesses the same server as our program and then used mongo queries (not inside of another language such as python) to make sure that the correct changes were made to the database corresponding to the action we took or that a result that is returned is the correct one in regards to the database. Since not every specification was captured in the marking rubric we then tried to enter a state and take an action to capture the specification and made sure the result (and the resulting state) were as expected. It is worth noting that any time the user had to provide input we tried both upper and lower case to make sure our program is case insensitive. We also timed our Phase 1 (to make sure it was running quickly enough) and we timed every action done by the program to make sure they all ran under 3 seconds.

(d) Elena was in charge of creating the user report for phase 2 (while prompting the user for a user id), as well as working on two of the functionalities: posting and searching for questions. About 4-5 hours was spent on the user report and posting questions, and about 7-8 hours was spent on searching for questions. The limit of search results was later implemented in order to prevent the overwhelming number of results being displayed.

Richard was in charge of allowing users to post an answer, list the answers, as well as voting on questions/answers. About 2-3 hours were spent figuring out what mongodb is, how to use it, how to set up servers, clients and run mongodb inside and outside of python on the lab machines accessed through X2GO. Another hour was needed to figure out how to implement post an answer (since this was pretty easy). Listing answers was a bit harder and took 3 hours. Voting on questions and answers was somewhere in the middle of the difficulty of the other two and as a result took 2 hours. I then built much of the functions needed to integrate the pieces together (such as the main menu) and integrated my pieces into the larger project (4 hours). I also did a great deal of testing and bug fixing/making changes to reflect the specifications better (around 7 hours in total).

Isaias was in charge of creating the databases as well as extracting the terms (all of phase 1). It took 4 hours to implement all of phase 1. Extensive testing was also made on both phase 1 and phase 2, which led to refactor the entirety of phase 2 into separate module files that took 3 hours. Overall this member contributed ~10 hours of work accounting the testing and debugging hours as well.

A first meeting was to discuss how to split up the work, when we should have it done and how to keep the project on track. As a result, a Github repository was created and meetings were scheduled from time to time through Discord. Much of the time even when no meeting was specified there would be two or three people in the discord call working together on an issue or working in silence and asking for help when needed. If someone had an issue they needed help with and no one was in the call, they could post a message and since we all had alerts for discord someone (or both others) would shortly be online to help.

The report should also include any assumption you have made or any possible limitations your code may have.
In order to run efficiently, the Id for votes and posts should not be a ridiculously large range (ie 1 to 10000 is fine but as you start getting a larger range than this some functionalities may take over 10 seconds
Every post has an Id field that contains its unique Id