Project1 Task2

Nathaniel Leake
424003778
Time to complete Task: **5hrs**

Part A:

How to run the code:
To run the prototype from terminal, enter `python msg_board_v2.py`

The commands work the way they are specified in the Message Board PDF; enter `stop` to exit listening mode. One of the barriers I ran into when designing this was the program pausing when waiting for user input. Since I am new to Python, I had to do a lot of research to understand python multithreading so that I could have a separate thread waiting on user input "stop" to exit listening mode (while still listening for new messages). Another concern was real-time updating. I could not find a way to check for new messages without constantly querying the server, but the query I do for new messages is very simple and fast, and should scale properly for a large database of messages. I also set up several message boards on the MongoDB database, which are listed on start up and can be selected from the program menu. Also, it is important to note that I did not see a need to use Redis since MongoDB can supply all of the functionality I need and should remain fairly fast for larger databases. For my Mongo database, each document contains at least three properties: the message, the user who sent it, and a timestamp. The time stamp is used to sort the database so that other users can look at the top of the database and find new messages while listening. When a user enters a message, the user's name, message, and timestamp are sent to the server. This system isn't 100% secure because a user could theoretically send an invalid time stamp, meaning users who are in listen mode won't receive it. A solution to this would be to have the server assign a timestamp as the data is received, but I did not find a simple, fast way to do this while working on the project. Note: if anything is still unclear, the program itself should offer usage explanation on startup.

Note:
I have implemented thorough error checking in my program, along with informative error messages. Feel free to test the application to the limits and try to break it.

Github: https://github.tamu.edu/nateleake/489-17-c

Part B:

Consistency is roughly equitable to redundancy—the more copies of data you maintain, the less likely for it to all be corrupted. This can limit availability (performance) if several databases need to be updated at the same time for each new query. However, one solution that is actually used by Facebook is to update databases asynchronously, meaning there is still a level of redundancy but not all of the databases may match at a given moment. Thus, they have both availability and consistency, but have to continually check to ensure the databases are in sync. For the AggieFit program, this should not be a major concern. Currently, the system is running on the one (provided) database, and if at some point there is a need to upgrade to a distributed network, availability should probably be prioritized over consistency due to the need for real-time performance and continual querying.