Time to Complete Task

12 hours

Architecture

The architecture of the prototype is fairly simple, it makes use of two databases, one which stores metadata (name and topics) about each message board and one which stores the actual messages sent to the board. The boards are accessed through a command line interface.

MongoDB was selected to store the metadata. MongoDB was chosen because the good people at AggieFit expressed a desire to be able to categorize the message boards by topic. MongoDB's query power allows it to perform a find query based on topics. Although this functionality wasn't required to be implemented in the first prototype, the choice to use MongoDB in this prototype was made with an eye towards ease of implementation in the future.

Redis was selected to store the actual messages. Each message board is stored in the Redis database with the message board acting as the key, and the messages being stored as a simple list. Redis was selected to store the messages for two reasons. First, it's PubSub functionality made it the obvious choice for implementing the listen functionality since MongoDB does not have a similar functionality. Second, by storing the messages as a list, new message could be added to the list by simply pushing them onto the right side of the list as they were sent. This maintained the correct order that the message were sent without the need to develop a novel ordering system.

When the select command is issued, the MongoDB is queried for a message board with the same name that the user specifies using the find() query. If none are found, an error message is printed letting the user know that the message board doesn't exist. Otherwise, the current message board is set to the user's selection.

When the read command is used, an LRANGE query is sent to the Redis database with the given range being (0, -1) to return all items in the list with the key of the message board the user has selected. The messages are then printed using a for loop.

When the write command is used, an RPUSH query is sent to the Redis database for the message board key and the new message that the user is sending. Next, the same message is publish to the Redis pubsub channel for that message board. If there is an error at either of these steps, a message is printed. Otherwise, a message is outputted letting the user know their message was sent successfully.

When the listen command is used, the user subscribes to the Redis pubsub channel for their selected board. The client cleans the message before it is outputted, so that the user only sees the text of the message that was sent.

Prototype and How To Run

The prototype is fairly user-friendly, but requires some setup before it will run. The client requires a Redis server to be running locally and a MongoDB database to be running at 34.233.78.56. The main message board client is the file messageBoard.py, which is written in python 2.7 and requires the pymongo and redis-py libraries in order to run. The client also requires two other files, constants.py and mongoConnect.py, both included in the submission, to be in the same directory in order to run correctly.

Once the setup is complete, a client can be started with the command "python messageBoard.py". The client opens with a welcome message listing the available commands. It also includes error handling and input validation for all commands. If the user attempts to enter a command that is not on the list, they'll be asked to use a valid command. It the user attempts to use a valid command incorrectly, they'll be asked to re-enter their command in the proper format. The user can exit the chat gracefully by typing the command "exit".