Schuyler Martin
CSCI-759-01
December 2, 2016

Queuing Management System for the CS Department's Mentoring Center: Backend Final Project Report

The Make the Mentoring Center Great Again (MMCGA) is a queuing app to improve the answering question workflow in the Mentoring Center. This project is the result of multiple discussions across department faculty members about the goal of the tutoring facility. It is the hope that this app will enable the faculty to track usage statistics on the Mentoring Center as well as improve the workflow of the tutors who run it. Now if a student has a question, they can log into the Mentoring Center app to submit a question. The next available tutor is then alerted about the question and who asked it. If there are more questions then there are tutors, a student question queue and a tutor queue are maintained to maintain order in the Mentoring Center. Students can no longer abuse the failings of a paper sign-in sheet, tutors no longer have to manage the sign-in sheet themselves, and the faculty can gather statistics on the questions students are asking.

To accomplish this, a Python server program was developed to model and control the current operational state of the Mentoring Center. This server communicates directly with a RabbitMQ server that handles all the messaging between the clients and server. The Python server has virtually no knowledge of the network architecture involved; network agnostic, as it were. This layer of abstraction inherently solves concurrency and race condition messaging issues for us. The Python server does not need to worry about students asking questions at the same time nor does it have to worry about how the messages are sent. It just listens to queued messages off of the RabbitMQ server and works from their.

The Python server is also heavily object oriented by design to make it easier to maintain and provides several logical layers of abstraction. For example, there is a class to represent a generic User that the Student and Tutor classes inherit from to minimize code duplication. Students and Tutors are then placed into queue structures to control who has the next question and who can answer the next question. Another class (called the "Bunny") is responsible for tracking who is currently logged into the system and communicates directly with the database and RabbitMQ server (hence the name). At the highest level of abstraction, the QueueManager uses the Bunny instance and the two User queues to provide the commands or *verbs* necessary to control the system. The QueueManager is used by the top-level server program to handle messages that have been received by RabbitMQ.

As an example, suppose a returning user logs into the system. RabbitMQ receives this message as a JSON string of parameters (method name identifier, username, encrypted password, etc) and puts it in a message queue. When the Python server is ready, it will read it off of the RabbitMQ queue, identify it as a login request, and call the QueueManager's user login function. The login function will contact the Bunny class to lookup the user in the database. If found, the user will be deserialized, read in as a Student or Tutor object, added to the appropriate user queue, and then the Bunny class will attempt to check if a tutor can help a student, if appropriate.

Although seemingly complicated at first, these layers of abstraction make the code much easier to maintain because software concerns have been split up. For example, if the RabbitMQ code needs to be modified only the Bunny class will need to be modify. For this reason the server also defines a series of "datagram" classes designed to store pertinent clumps of data together. There is a UserStats datagram that stores information on user statistics. If more statistics are needed to be collected in the future, minimal work will be needed to track them in the server code. It is the hope that this design will easily enable this project to be expanded and maintained by future Mentoring Center tutors and other interested students. This project, along with additional documentation and design diagrams, is available on GitHub:
https://github.com/RIT-CS-Mentoring-Center-Queueing/mmcga_project