Team 18
Varun Ganesh vganesh7@vt.edu

ECE4564, Assignment 2

Objective:

The objective of this assignment was to learn about the creating a message broker using the AMQ protocol, which is the Advanced Message Queueing Protocol. Another focus of this assignment was learning about callback routines and Data Persistence. Specific tools used to implement and understand these concepts were RabbitMQ and MongoDB.

This assignment required us to repeatedly take a user input in the form of: <p or c>:<Place>+<Subject> "<Message>". The <p or c> in the command represented what type of request was being issued to the RabbitMQ. A "p" meant it was a produce command, which would simply publish the message to the appropriate exchange and queue. A "c" meant it was a consume command, which meant it would retrieve the messages contained in the provided queue. The <place>+<subject> portion of the command indicated what exchange and queue the message would be stored in, or which exchange and queue the messages would be retrieved from. The place corresponded to the exchange where the subject corresponded to the queue. The exchanges and queues were managed by RabbitMQ. MongoDB was used to collect all the given commands and store them in a JSON format. This process runs on an endless loop until the user terminates the program.

Design:

In this project, I had to use RabbitMQ and MongoDB to store and process the commands provided by the user. Certain design choices were made in preconfiguring the appropriate queues and exchanges. All the queues and exchange were declared and bound at the very beginning of the program. Along with this the database was initialized and a small list of key value pairs was initialized to values of 0. These key value pairs were a local way of counting the number of messages currently being held in each queue.

The program then entered a never ending while loop which prompts the user to enter the given command. The inputted string is then parsed to separate the pieces into the command type, exchange, and queue. If the given command type is a produce request, the message body is then extracted from the input. This continues by alerting the user that the command was added to the MongoDB local database, then published to the RabbitMQ queue, as well as incrementing the appropriate key value pair by one.If the command type is a consume request, the program knows there is no message body and uses the given information to extract the messages from the desired queue. This will invoke the callback function, where a check is conducted to see if there are any messages in the given queue. If none, the program will continue and provide a message indicating there are no messages in the queue. If there are messages, it will pop all the messages contained in the queue and print them out on the console screen while decrementing the appropriate key value pair.

This process continues until the user exits the program or uses a keyboard interrupt to terminate the program.

Outcome:

The outcome of this project was that the code would continuously request a command in the appropriate format from the user. When the appropriate command is given, it parses the given request and takes the appropriate action to complete the request. This project gave us a better understanding of the Message brokers using RabbitMQ with direct exchanges, the AMQ Protocol and Data persistence through MongoDB.