Assignment 2

For this assignment we had to implement a message queue using RabbitMQ to send the message to the queue as JSON tuples and then to the consumer. Using a WebSocket, in the case that a measurement exceeded the maximum capacity of a device, the user will be alerted in real time, signaling the problem. The queue itself was already built-in, I have used CloudAMQP. The producer reads the data from a CSV file (simulating values sent from a sensor). Normally, it should send messages once an hour, but for simplicity and to prove the solution works, the data is sent once every 2 seconds, in order to be easier to present. We use a model in the producer that is then serialized in JSON format and sent to the queue. On the server side of the application there is the hosted consumer service(it is always running pinging the queue and looking for new messages). If new messages are present on the queue and were not consumed, then it they will be sent asynchronously as JSON tuples. There we will deserialize them and map as a Measurement model, thus creating an object. Next step is making the required checks to see if the maximum value was exceeded. There are two situations, if the value is not exceeded, then the new measurement will be added to the database and the user will be able to see it in the table. On the other side, if there are problems, then using a WebSocket the user will be alerted via a notification that there were problems and to contact the company immediately. The WebSocket is implemented using SignalR, a .NET library that allows server code to send asynchronous notifications to the client application. It sends a notification to the hub with the user where the problem was and a message, and from there to the clients. Then, in the client application a connection to the WebSocket is initialized, and if the current user is logged in, then a pop up will appear. Note that the client is always listening for the server notifications, thus responding in real time. At the start of the application I have set a 2 seconds timeout to wait the connection to establish.

I think that the problems encountered were making the consumer work and not send too many requests to the database thus being the risk of blocking the other functionalities of the application. Another issue that was tackled was the problem of synchronization, for the consumer to wait the socket send the notification and then resume the seeding.