Assignment 8 – Queues  
COS318 – Web Programming

All peasants seem to do is work. Never a break even when they deserve it. I guess they should have thought about that when they decided to become peasants! In assignment eight, you will be helping the peasants complete their chores. Those chores will be coming from an AWS queue. Here’s the good part: You also get to be the peasant overseer. Someone’s got to make sure that Kuzcotopia gets built! (unless of course the Emperor has a change of heart…then we should build something else)

The finished version of the assignment will contain three projects. One for creating work, one for completing that work, and a class library for shared code between the other two projects.

1. **(10 Points) Controller** 
   1. Create a controller that accepts a “work count” integer using a JSON entity in a POST endpoint. The controller should send this integer into a service which queues messages.
      1. Use a good status code for successful responses. There is a better one to use than the default 200 OK.
      2. If the work count integer is less than 1 or greater than 10 the controller should return Bad Request. Hint: remember the rules!
2. **(10 Points) Html/Javascript**
   1. Create a simple html page that allows a user to enter an integer and send it to your controller from step 1. Display errors and successes as normal.
3. **(20 Points) Queue Service (Queue the work)**
   1. Create a service that queues several JSON messages. The service will have a method that accepts an int parameter and queues that many messages when the method is called. When the controller calls this method method, it should pass the work count int as the number of messages.
   2. The JSON messages will indicate what type of work needs to be performed. The messages should have three keys, “type”, “message”, and “data”. Randomly choose a type each time you queue a message. The three types are:
      1. Carry: Some peasants need to carry the supplies to the builders. The “message” for this type should be a string of what they are carrying. (use whatever you want for the things they are carrying). The “data” key will be null.
      2. Build: These peasants have the supplies. They need to put stuff together. The “message” for this type should be what they are building. (use whatever you want for the things they are building). The “data” key indicates how many steps there are in the project. It should be a random integer between 1 and 5.
      3. Survey: This is the easiest job. These peasants just need to make sure the job is done to high standards. The “message” key should be some positive comment about how the other peasants are doing. The “data” key will be a random integer between 500 and 1000.
4. **(40 Points) Console application (Do the work)**
   1. Create a console application that reads from the queue from step 1.
   2. Depending on the “type” of message received from the queue, a different action should be taken.
      1. Carry: The peasant is carrying something! Write the “message” to the console.
      2. Build: Building often takes several steps to complete. If the “data” parameter is a positive number, send another queue message with the “type” equal to “Build”, but with a “data” parameter of one less than the current message. Then write to the console with how many steps are left to complete (the “data” parameter). If the “data” parameter is zero, the building is all done. Write to the console saying that the building is complete.
      3. Survey: It takes a little time to make sure the peasant did the job correctly. Use Task.Delay() to wait a number of milliseconds equal to the “data” of the message. Write to the console before and after the delay to indicate the surveying is happening. These two messages should be written on the same line in the console.
   3. Delete the messages out of the queue after you are finished processing them.
      1. The console application should not await the delete task, i.e. the code should continue executing without waiting for the delete task to finish. There must be no warnings in your code about not awaiting a Task return type.
   4. The console application should not exit unless the user forcibly closes it.
5. **(20 Points)** Code style, formatting, completeness, and quality.
   1. AWS accounts are free for the first one million requests. Create your own account and use your own queue.
   2. Why does she even have that lever?

Stretch Levels

If you already have a lot of experience with queues, or if you just squeaker squeak squeaken, try to complete these stretch levels for a reputation bonus. If you try for the stretch levels, make sure to type it in the comments on Moodle so I don’t miss it.

**Kuzco Level**

Add some CSS to your page to make it look nicer. Background colors, font colors, or anything that looks good.

**Bucky the Squirrel Level**

Even though we aren’t waiting for messages to be deleted from the queue, we still want to record if a queue message fails to be deleted. Use ContinueWith on your delete message task and log a message to the console. Your ContinueWith method should only execute if the delete task fails.

**Pacha Level**

If an exception is thrown in your “Do the work” application, then the application would stop processing queue messages. Modify your code that if there is any exception, it is logged to the console, but it doesn’t stop your application from processing queue messages.

The Rules

1. No inline styles or inline javascript.
2. Error messages must be “in-page” i.e. no pop-ups or alerts.
3. Any resources not created by you (images, javascript libraries, etc.) must be referenced using a CDN or URL, not directly included in your assignment submission.
4. All requests that submit a body to your server must have their entities validated with appropriate annotations, such as MinLength, Range, or Required.
5. The root path of your server must display the main page of your application.
6. Service/data/model classes must not have any http, request, or response references.
7. Controller entity classes must not be used directly to store data on the server; translate them into a model (data storage) class before saving the data. Conversely, controllers must not send any model classes to the user; translate them into controller entity classes before sending the response.
8. All service class instances must be obtained using dependency injection.
9. You may not use any synchronous methods in your C# code wherever there is an async option.