Assignment – TFH, async, await  
COS318 – Web Programming

Grab a star and run like crazy. This assignment is full of koopas, goombas, and pirana plants. In this assignment you’ll be creating a controller that is dependent on another service. There is a problem though. This service doesn’t allow more than one request per second. That is certainly going to be annoying.

1. **(40 Points) Html and Javascript**
   1. Create an html page with javascript that looks like a level from a Mario game. Some ideas of things you can add are a background with sky and ground, a flag pole, clouds, blocks, and anything else that looks great. The main thing to make sure you have is an image of Mario on the left side of the screen.
   2. Add a button to the HTML page that begins the level.
   3. When the level begins, start by making a request to your MarioLevelController with a random value selected from the four actions. (walk, jump, wait, or run) If the server returns a successful response, then move Mario towards the right side based on the action that was selected.
      1. The amount to move Mario for each action type is Walk: 5%, Jump: 5%, Wait: 0%, Run: 10%, where these are percentages of the screen. The jump does not need to have any animations so it will look the same on the screen as “Walk.”
         1. Hint: If you put the Mario image inside of a div tag, then you can use javascript to change the “left” CSS attribute of that div tag to move it. Div tags can only be moved with the “left” attribute if you set the “position” attribute also.
      2. After the request is successful, then make another request with another random action (walk, jump, wait, or run). Keep on making these requests until Mario reaches the right hand side of the screen or the server returns that Mario has died.
      3. Each time you make a request to the server, also display the message that comes back in the response.
      4. The button should not be active while the level is in progress.
      5. Hint: The URL that you will use to make requests to **your server** from **your javascript** will look like /api/mariolevel/{move}, where {move} is the random action that your javascript selects. (This is not the same as the url for the external service.)
      6. Hint: Your javascript code will not be making any requests to the external server. It will only be communicating with the server you create, so the URL it will be using will start with “https://localhost”.
2. **(20 Points)** **MarioLevelController**
   1. Create a controller with a single method which will accept GET requests and one string URL parameter. Hint: You can use HttpGet[“{move}”] and a string parameter “move” in your method to get the value that your javascript is sending.
   2. Validate that the URL parameter sent to MarioLevelController is one of the four valid actions, “walk”, “jump”, “wait”, or “run.”
   3. MarioLevelController needs to return a MoveEntity (as a JsonResult) which contains only a single string field “message.” This message will come from a method you create in a class you create called MarioService. If the MarioService throws an exception (either from running out of its RetryPolicy or the external server returned a 500, then MarioLevelController should catch the exception and then return a message indicating that Mario died. This would still be a 200-OK response, since you are hiding the external failure.
3. **(20 Points) MarioService and IMarioService**
   1. Create a class called MarioService and create an interface IMarioService that it implements.
   2. MarioService (and IMarioService) needs a single method that will make a request to an external server. This method takes one string parameter, called “move”, which is the value that was sent to your controller as a URL parameter.
      1. The URL to use for the request is https://webprogrammingmario.azurewebsites.net/api/mario/{move}.
      2. The request must be wrapped in a Retry Policy with a maximum of 10 retries. Your policy should wait progressively longer with each retry.
      3. The external request has three possible responses:
         1. **200 – OK:** A JSON document with Message and NextStep keys. Message is a friendly string of what happened to Mario. The method can return this string so that it can be sent as a JSON response. NextStep is only used if you are doing the Luigi’s mansion stretch level.
         2. **503 – Service Unavailable:** The server can’t process the request because another request was processed too recently. This is a transient failure and should be retried.
         3. **500 – Internal Server Error:** Mario died. Any request to the external server has a small chance for this response. This is NOT a transient failure. If this is encountered, your MarioService method should let the exception go to the controller immediately without retrying.
   3. The IMarioService/MarioService should not expose any fields or methods that would imply that it is making requests to a server to perform its work.
   4. Use dependency injection to give an instance of this service to your controller. The controller should use the IMarioService interface, not the MarioService class directly.
4. **(20 Points)** Code style, formatting, completeness, and quality.
   1. The javascript code should never be exposed to any errors from the external service, nor should it have any knowledge that the external service is being used by MarioLevelController

Stretch Levels

If you already have a lot of experience transient fault handling or just really want to rescue the princess, 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.

**Toad House Level**

Retry polices work best when they add some “fuzziness” to the time between requests so that multiple retry policies don’t make all their requests in sync. Change the function that returns the TimeSpan delay between requests to add some randomness. It should still follow an exponential increase in time, but instead of the time being 200, 400, 800, etc., it would look like (as an example, there are plenty of good ranges or random numbers to choose) 100 – 300, 300 – 500, 700 – 900, etc.

**Luigi’s Mansion Level**

Make Mario smoothly transition whenever he moves instead of instantly moving to the new location. Use CSS to do this, not javascript. Then use CSS (and the javascript toggleClass method) to make Mario jump upwards on the screen whenever that action is selected. Hint: Use setTimeout to toggle the class back off again to end the jump after a short duration. Mario should move smoothly exactly like the Toad House Level.

**Bowser’s Castle Level**

The external service returns a NextStep in addition to a message. Modify your MarioService and MarioLevelController to pass NextStep as part of your JSON response (this will be in addition to the Message field). Then instead of randomly choosing the next action Mario will take, use the value from NextStep.

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. **New Rule:** You may not use any synchronous methods in your C# code wherever there is an async option.