**Blazor Tutorial 8 - Rendering Modes**

**(20 pts)**

**Part 1** Watch the [Rendering Modes](https://www.youtube.com/watch?v=HZAdXx7-PnM&list=PLdo4fOcmZ0oXNZX1Q8rB-5xgTSKR8qA5k&index=9) video. Then answer the following:

1. (1 pts) Which rendering mode is the default? Static server side rendering (Static SSR)
2. (1 pts) True/False: A page using SSR cannot contain C# code. True
3. (2 pts) In the example of enhanced rendering of SSR, give an example of two specific things that were NOT reloaded when he navigated from one page to another. Javascript files and css stylesheets
4. (1 pts) In general, what sort of page benefits from using Streaming rendering? Long running asynchronous tasks
5. (1 pts) In the Weather page, which line of code initiates the rendering of the initial html (before the simulated weather data is available)? Copy the line of code below. @attribute [StreamRendering]
6. (1 pts) What is a major advantage of using ServerInteractivity (versus Webassembly)? A realtime connection with the server, while blazor efficiently changes the components
7. (1 pts) What is a major advantage of using Webassembly (versus ServerInteractivity )? Runs directly from the browser, offloads works from server, caches info
8. (1 pts) When using the WebAssembly project, how do you determine whether a razor file should be placed in the Client project versus the Server project (hint: an unenlightened answer is “it depends on whether you want it to run on the client versus the server). Files placed in the Client project will be rendered in the browser, while server files will be static
9. (1 pts) What is the potential performance issue of using WebAssembly render mode? It doesn’t have direct access so it can cause slow times when fetching information
10. (1 pts) How does “publishing” your web app make it smaller? It removes unnecessary files and dependencies which increases the efficiency

**Part 2**

Recall the breakfast code we used to experiment with async code (Blazor Tutorial 6). Since we were using a console project, it was hard to appreciate the significance of async. For this part, you will need to reference your “breakfast” project code; you will use some of that code again.

1. Add a “Breakfast” page to your site (and add a reference to it in the navigation). The page will need to use Streaming rendering and InteractiveServer rendermode.
2. Add a **Pour Coffee** button to the page. This button should display the message “Coffee ready “ followed by the current timestamp (the timestamp is so we can clearly see when the label gets refreshed). Click your button several times to ensure the label gets updated correctly.
3. Add a **Fry Eggs** button to the page, as well as an **eggStatus** label. Use the following code for the button event handler (this is to simulate a task that takes about 6 seconds to complete, with various updates along the way):

private void FryEggs()

{

eggStatus = "";

eggStatus = "cooking the eggs ...";

Task.Delay(3000).Wait();

eggStatus = "cooking the eggs ...";

Task.Delay(3000).Wait();

eggStatus = "Put eggs on plate";

}

1. Test as follows:
   1. Click the Pour Coffee button (label should update immediately)
   2. Click the Fry Eggs button, then immediately click the Pour Coffee button several times.
      1. Explain the reason for the behavior of the Pour Coffee button. The Task.Delay().Wait means no other actions can be executed until the eggs have finished cooking
      2. Explain the reason for the behavior of the Fry Eggs button/status label. It does not change the value again
      3. Why do we only see the “Eggs are Ready” label (why not the intermediate labels?). It only updates the value of eggStatus when the method is finished
2. Add another button **Fry Eggs Async**. Add an Async version of the Fry Eggs event handler (note: *refer to your code from Blazor Tutorial 6*). Read the following before you write the code:
   1. Note: The async methods on the previous assignment were static methods, you’ll need to remove the static so that your code can access instance variables for setting the status messages in the html.
   2. The async version uses Task.Delay(3000) (NOT Task.Delay(3000).Wait())
   3. Because we are only simulating tasks taking a long time using Task.Delay, sometimes blazor won’t detect that it needs to refresh after updating a variable referenced by the html. If this happens, you can use **this.StateHasChanged()** to tell blazor something has changed, and therefore it needs to update the UI.
3. Test as follows:
   1. Click the Fry Eggs Async button (eggStatus should update immediately to show “Warming the pan”.
   2. Click the Pour Coffee button several times in succession (pour coffee label should update immediately each time).
   3. Explain the following: Which specific statement in your async event handler keeps the UI functional while the eggs are frying? In terms of threads, explain how this works. await Task.Delay() allows it to continue using other methods during this time along with the Task<Egg> in the method name. It can utilize a second thread to complete both tasks simultaneously

**Submit:**

Make a video that shows your async method working (demonstrate coffee button presses during eggs frying), and showing egg status messages from beginning to end. Paste a clickable link to the video below.

<https://www.loom.com/share/227c1900f2ef4c5dacff961e2846cf84>