Blazor Portfolio Site Progress Diary

# Initial

Blazor is weird… it doesn’t use C# as much as I expected, it feels like standard web development except with C# instead of javascript for writing functionality. Blazor seems more useful for web applications that require desktop-application-like speeds, especially web applications that interface heavily with servers. For a portfolio site that is mostly static, there doesn’t seem to be a benefit to using Blazor (oops).

# (12/26/22) Giving up on trying to build my website from scratch

I’ve been having some headaches trying to make everything look as I want it to using HTML and .css, so I thought to myself: what would a web developer actually do? No professional web developer is expected to make their website look good without using third party UI component libraries! There exists at least one Blazor UI component library—Blazorise—which luckily for me seems very comprehensive! I began integrating it today and already it’s making the UI so much simpler and nicer-looking. It’s many built-in components handle element spacing, consistent styling, differing aspect ratios, and likely even more! This saves a lot of time and headache, and I would be happy never attempting to do web UI from scratch again…

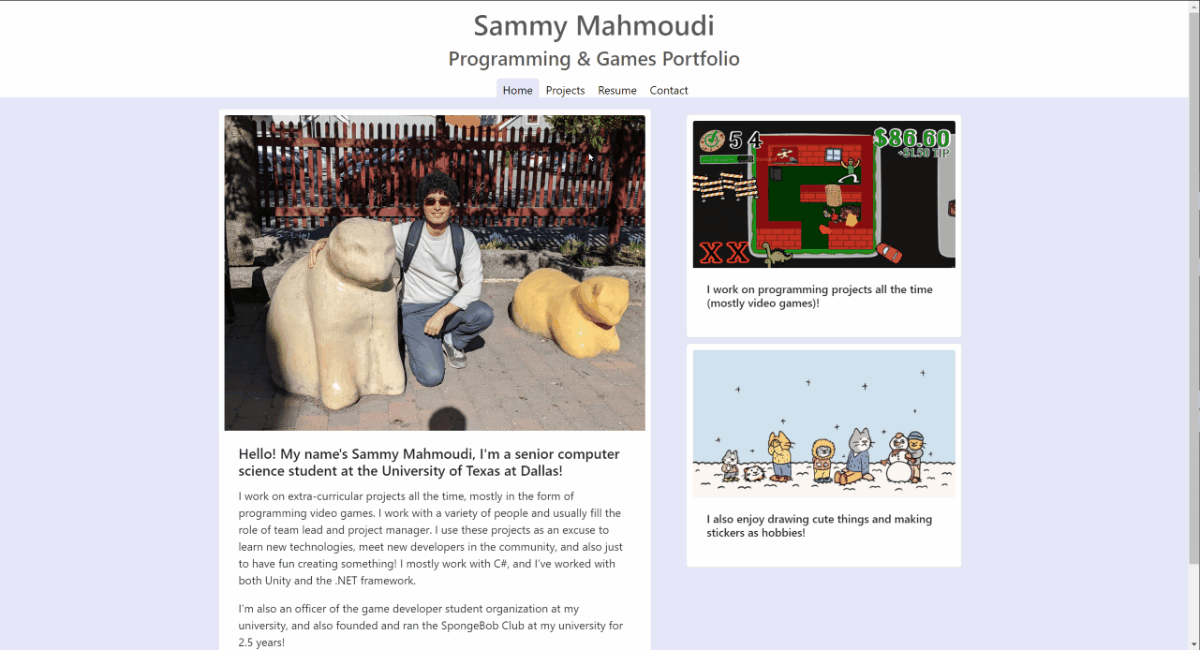
Graphical user interface

Description automatically generated Why isn’t the white box shrinking to fit the screen? (Start 30 minute googling adventure…)

# (12/27/22) Blazorise Animations

Blazorise animations are finicky AND THERE ARE NO GODDAMN FORUM POSTS ANYWHERE AND THE DOCUMENTATION IS VAGUE!!

Apparently I can't activate the same animation on a page more than once without unrendering the component and re-rendering it? For my cool slide animation effect I'm having to put an animator component on each page instead of just a single one on the main page that re-plays when the page is changed. So ugly...



# (12/28/22) WHERE ARE MY SHORTCUTS??

Just realized that many of the Visual Studio C# shortcuts don’t work in razor files. I can’t do CTRL+. to fix issues, imports don’t add automatically as I type, CTRL+. to refactor code into a separate method doesn’t work, AI autocomplete doesn’t work.

# (12/28/22) Compile errors don’t show in visual studio error window until I open the problematic file

It’s supposed to show all errors in all files in the solution… WHAT?

# (12/29/22) Ohhh, there is a razor formatting function in Visual Studio

It doesn’t seem to work on save, but manually applying formatting to the document via CTRL + K D formats! (not super well, but good enough! Trying to be more positive in here because I am actually enjoying working in Blazor)

# (12/29/22) Didn’t host in Azure, hosted in Netlify

Was originally planning on hosting in Azure, because that’s the one that’s integrated with Visual Studio. However, Azure is dumb! The site is insanely complicated, the pricing tiers are vague (pricing is in approximations, and what are CPU minutes?). Seems like Azure is more designed for hosting services, and not really intended for hosting static websites. The pricing is indicative of this, the service would be free for me but I have to move up to the ~$9.50/month tier to use a custom domain. So… instead I tried hosting in Bill Dang’s recommended service Netlify! It’s super easy to use and is free to use for my needs. It even lets me use a custom domain for free! The only bad thing is that the build automation tools it has doesn’t support building Blazor applications, so I wouldn’t be able to build the site automatically from my Git repo. I could build it on github via github actions, but I couldn’t figure out how to do that… for now I’ll just be publishing manually, which isn’t that much work.

# (1/3/22) Added dark mode switch

Added custom themes for light mode and dark mode! There’s a system in Blazorise for switching themes on runtime, so utilized that. Overwrote the css of some of the built-in blazor components (which don’t have easily-visible class names) by checking the class names in Chrome’s inspect element. Needed to set my custom css file to load after the Blazor one so my changes overwrote any Blazor-defined properties. I have a global variable that maintains the dark mode state, which is used by some components that need specific styling, like the tabs which need to match the page background only when its associated page matches the active page.

# (1/2/22) Dark mode to change social icons?

There’s a slight issue with the dark mode: it doesn’t change the color of the social icons. This is because the social icons use icons that aren’t part of the blazorize icon library, therefore aren’t read as fonts (and change color along with the fonts). They remain black.

My first solution was to create white and black versions of each of these social icons, then use an if statement to selectively render one or the other.  
Application

Description automatically generated with low confidence

This solution turned out to have some complications. It turns out that when dark mode was toggled, the component wouldn’t re-render. The icon would only be the correct color if I reload the page. My solution to this was to create an event on dark mode toggle, and call the built-in StateHasChanged whenever on dark mode toggle. Another complication: StateHasChanged doesn’t re-render the subpages. So I couldn’t just have this logic in my one parent page, but I’d have to have it on *every page*. This would work… but at this point I was realizing that this was a silly solution, too complex for such a simple issue.  
Text

Description automatically generated with low confidence

So, I created a base class to hold this logic, and had all the pages inherit this base class.

Also to simplify the previous solution of switching between black and white icons, I instead opted to remove the black icons and change the icon between black and white using styling.

# (1/5/22) Dark mode auto-sets on load using browser dark mode setting

I get the darkmode setting from the browser using a javascript script, which is called from a C# script as soon as the main page is loaded.

This was a bit hard to figure out since there aren’t many forum posts for Blazor and this is somewhat a niche issue. There didn’t seem to be any way to detect a browser’s darkmode state from a C# or razor file, which was problematic because my code for toggling dark mode state was all in C#. It’s unclear if blazor just doesn’t have this capability or if the documentation is incomplete. Luckily, javascript code is still executable in a blazor app, so I looked into how to detect browser dark mode via javascript.

I found a simple solution! It can detect a browser’s darkmode state with just one line of JS code. Next, I found some .NET documentation that explained how to call .NET C# methods from a JS script and calling a JS method from C# code. My solution was as follows: As soon as my MainLayout component is initialized (ready to render), call a javascript method that returns a Boolean true/false if the browser is in dark mode or not, and then use this returned value to update the dark mode state of the app and re-render (if different).

Text

Description automatically generated with low confidence

Text

Description automatically generated

Discord post writeup: This was a bit annoying because I couldn't find a way in .NET C# to detect the browser's dark mode state (if there is a way, I couldn't find anything in forums or the documentation...). But luckily, Blazor also allows executing javascript! I wrote/stole a tiny javascript method to return the browser's dark mode state, and I have it called when the site first loads to set my app's dark mode state 🙂

# Auto-Deploy with GHA

Created a workflow script to run automatically each time I push to main.

My workflow script has a few steps:

* Set up .NET 7.0
* Run dotnet publish to build the site
* Run a series of commands to fix Blazor-specific quirks with running website using GitHub pages.
* Deploy built site to a branch gh-pages

After all of this: GitHub detects changes to the gh-pages branch and deploys the website.

Changed workflow to no longer use a separate branch. GHA now has steps:

* Setup .NET 7.0
* Run dotnet publish to build the site
* Upload built project as a pages artifact
* Deploy artifact to GitHub pages