**EAD – CA3 Report**

Anthony Kuznetsov – x00162229

Links: [Project](https://yellow-beach-0ba2fec03.2.azurestaticapps.net/) | [API](https://chandan-02.github.io/anime-facts-rest-api/) | [GitHub Repo](https://github.com/anthonykuz/ead-ca3)

The Blazor project I built is called **“The Anime Index”**. This is a simple single-page website that you can query an anime title and it returns a list of facts about that show along with a poster image and a fixed title.

**Functionality**

The HTML aspect of the page was simple enough to create. I have all the content within a global container for setting min and max width’s so that the containers don’t scale up or down past a certain threshold.

Then after that I had split each component (header, search, image, fact cards) into their own containers with unique classes for styling and positioning.

Text

Description automatically generated

Because this app was only retrieving one object at a time and not storing anything, I decided to have my code not be part of a class, to not add any unneeded complexity.

Text

Description automatically generated  
I split up all my attributes and values by inputs, query related, markup and a new type called TextInfo; I used this for formatting my string back in a readable way with capitalised first letters and removal of underscores.

The actual parsing function is simple and gets the job done, first I fix the query by putting the string to lowercase and replacing the space’s with underscores and I append that to the API URL via a template string. The rest is just awaiting a response from the client:

Text

Description automatically generated

If the API call does return something, I first clear the previous values if there are any and then update them once more. I loop through the JSON array of facts and add a string of HTML with the parsed JSON content included into a list of strings called ‘Facts’. Afterwards, I send that list into a function to insert each fact as markup.

Text

Description automatically generated

And this is also a quick and easy function that loops through all the strings and appends them to the markup string which is then converted into actual markup via **MarkupString**

Text

Description automatically generatedText

Description automatically generated

**Testing**

There is not much to test here but I did conduct a quick Selenium IDE test to see if my buttons were readable and the input was working and… It does:

Graphical user interface, text, application

Description automatically generated

**Design**

I decided to use the **“anime-facts-api”** found on the [GitHub open-source API list](https://github.com/public-apis/public-apis) because it fit all the requirements for the CA; that being it used CORS and did not require an API key.

For design - I knew that because the functionality of the app was simple, the interface should be as well. And I began where *most* students would with design… being a basic mock-up in MS Paint:

A picture containing diagram

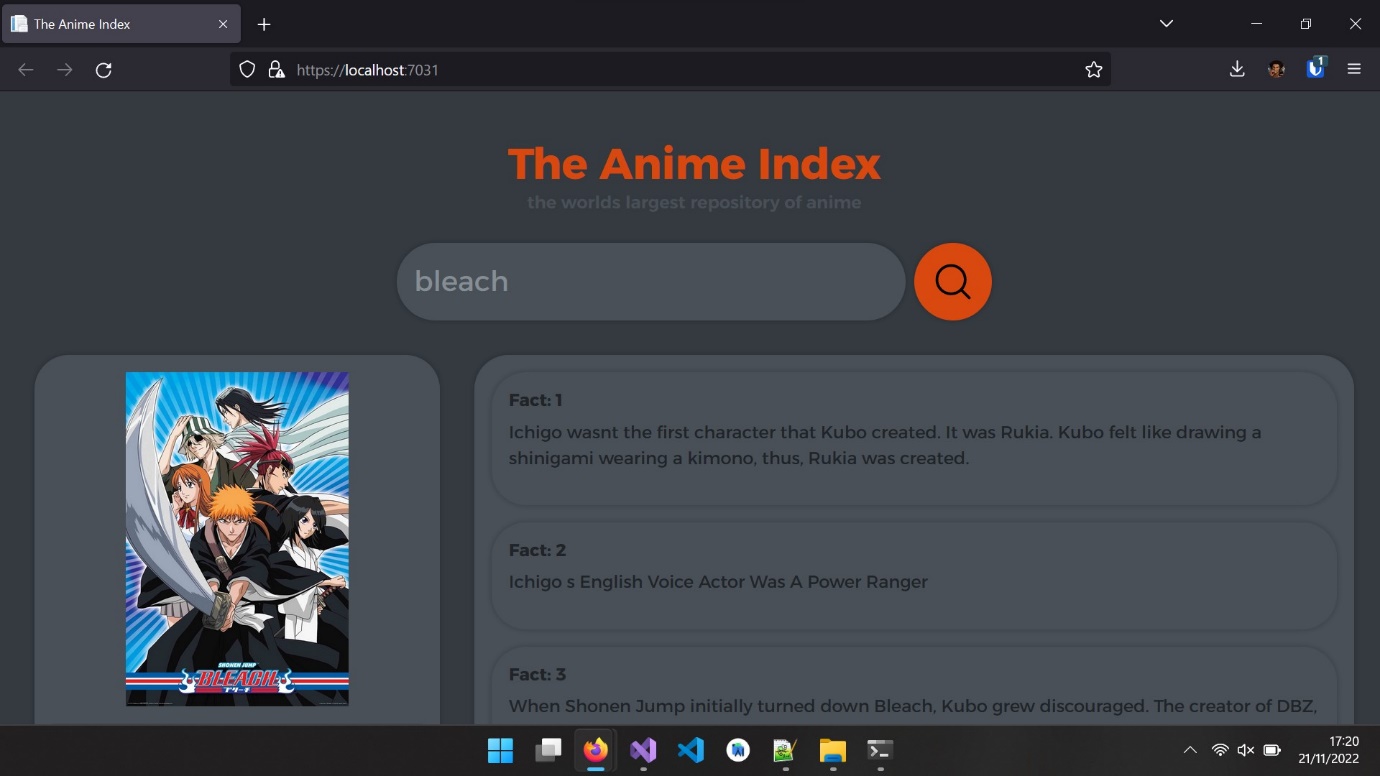
Description automatically generated

All it was comprised of was the heading and subheading at the top, with the search bar below that and then a grid of two columns:

1. (Left)  
   The anime title and cover artwork.
2. (Right)  
   The fact cards that would be gathered from the API call.

There is also a column to the left of that, which is the default Blazor bootstrap navigation menu that was removed shortly afterwards.

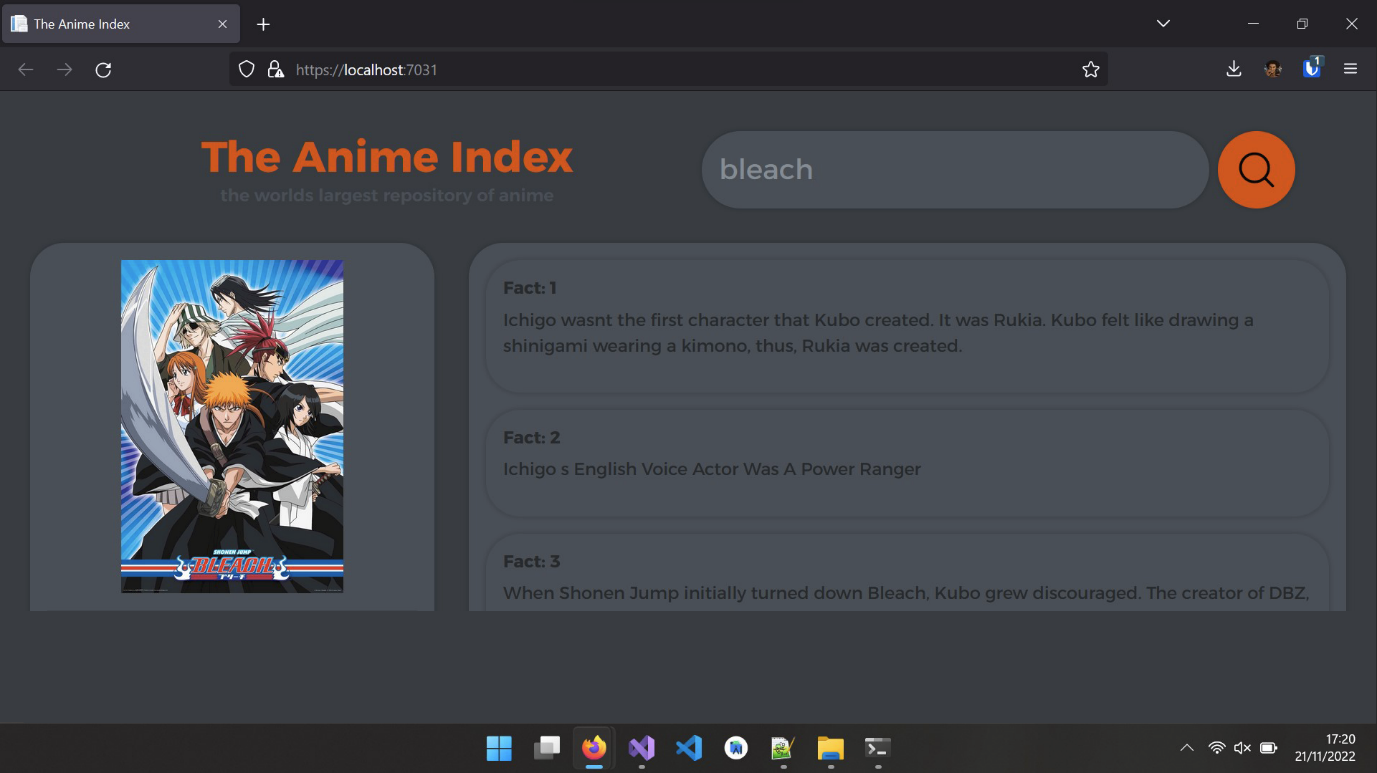
Mock-ups are an important, yet overlooked, step necessary to see the evolution of a design. From this artwork I was able to develop this:



This is the first iteration of the design which replicates all the aspects of the initial mock-up but in a much more presentable finish. I had also removed the default Blazor additions such as the “About” header and the navbar which were unnecessary components of this project.

I had asked for some additional input from none other than my own family members that suggested for me to move around the logo/header and subheading to be alongside the search bar instead of above it. And they also suggested for me to use an accordion instead of just a list to add an additional layer of interactivity and cleanliness.

And so, I came up with a quick mock-up in Photoshop that showed these changes:



As suggested, I had moved the logo to be alongside the search bar and moved up the core content. What I had done as well in practice, was have the content containers only stretch out to the max viewport so there was no unnecessary scrolling on the page, only within the fact cards container.

Along with that, I had also changed the colour scheme slightly to make text more readable, and to make it easier to see the interactable parts of the website which includes highlighting fact cards, search bar and search button on **mouse hover**.

These changes can be observed in the final design:

Graphical user interface, application

Description automatically generated

It is also worth noting that all the styling here is custom made and not utilizing any external Bootstrap libraries. I managed to achieve this look by using (maybe overusing) CSS grids for positioning the containers, flex boxes for aligning elements along rows/columns.

All of this can be found in the index.css file found in the static directory:

Text

Description automatically generated

I am also using a custom font provided by Google Fonts called [Alexandria](https://fonts.google.com/specimen/Alexandria) with font weights of 400, 600 and 700.

I managed to create the accordion (collapsed fact cards, open on click) by swapping a CSS class via a JavaScript function.