**Enterprise Applications Development**

**CA3 – ASP.NET Core Blazor Project – Documentation**

***James Lynam – X00073019***

**GitHub Repo:** [**https://github.com/JamesLynamX00073019/EAD\_CA3**](https://github.com/JamesLynamX00073019/EAD_CA3) **Deployed App URL:** [**https://ead-ca3-x00073019.azurewebsites.net/**](https://ead-ca3-x00073019.azurewebsites.net/)

For my CA3 in EAD1, I chose to use a public API known as [CatAPI](https://thecatapi.com/). This API allows us to extract many types of information about cats, but has a focus on displaying cat pictures.

For my project, I wanted to focus on a few things:

* Displaying random cat pictures on the click of a button.
* Displaying a number of facts or information about different cats.
* Changing the cat image to specific types of images based on different buttons.
* Sorting, searching, and filtering cat breeds.

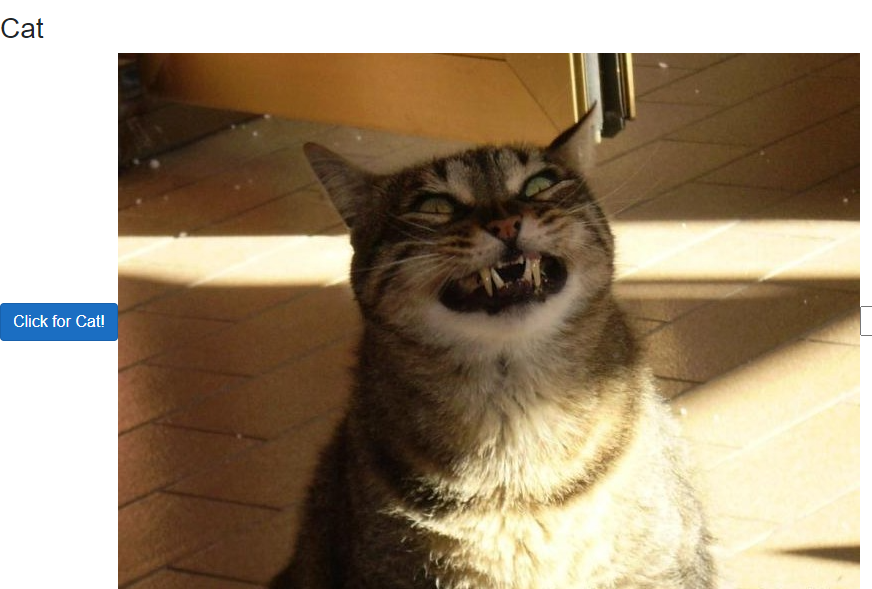
The first thing I did was use the basic template for the weather that is created when you first start a Blazor app. I identified how to implement my API and went on to create each razor page.

I decided to separate the C# code into separate files and reference them in the page after, in order to make the code easier to read.

The first issue that I encountered was when I was receiving a null reference exception for pulling in data to display the random cat image on the main page. I examined the console in the browser and saw that I was receiving the exception because I had not set up CORS(Cross-Origin Resource Sharing) correctly.

I went back and looked at the videos and attempted to apply the logic I found through watching them to my project, to no avail. I spoke to some classmates about this, and one suggested something I had seen while researching online but had not tried yet.

I included my API key in the request and response for the headers, and tried to run the page again. This time it worked, and was randomly displaying cat images.



As I went forward with the project, I decided that I need to show more than just cat pictures. I needed another level of complexity. I started to work on pulling out some information about different cat breeds, and thankfully the API has information such as name, lifespan, origin etc that I could work with.

I created a list in the HTML on the razor page and pulled out the rows from the API link I read in on the C# file. I first started with the name, and progressively added what I considered to be the most important attributes. I then decided to add a filter bar that doubled as a dynamic search, where the user could simply enter letters of a cats name, and the results would filter.

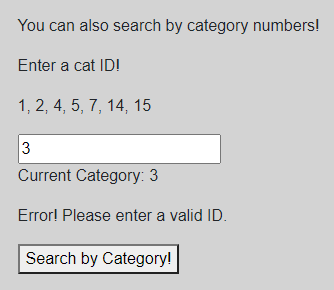
For example, the user can type “Sphy” and the only result will be a Sphynx cat. Or they could type “Be” and get “Bengal”, “Nebelung”, and “Siberian”, as each have a “be” in the name.

I also created a Category page, to see if I could extract the different categories of cat images that was available to me. There was a fair amount, though some of them seemed “experimental” in the sense that they either displayed from a list of 3-4 images, or none at all.

I pulled out the most common categories and displayed them on a page for me to reference later. This page was simply for this purpose.

I then decided to add more functionality to the original cat image page by using these categories to allow the user to change how the image is generated. Now whenever the user clicked a category button, only images in that category would appear. For example, only cats in hats would appear if that button was pressed. I also added a search bar with some validation that allows the user to enter a number between 1 and 15 to display categories of that type, although 8 of these 15 are the experimental categories I spoke of previously. Some of these will throw exceptions as they are simply empty arrays.

To avoid the user seeing this, I made it so the input field will only accept the IDs of the categories that are stable, and printed a list of these IDs.



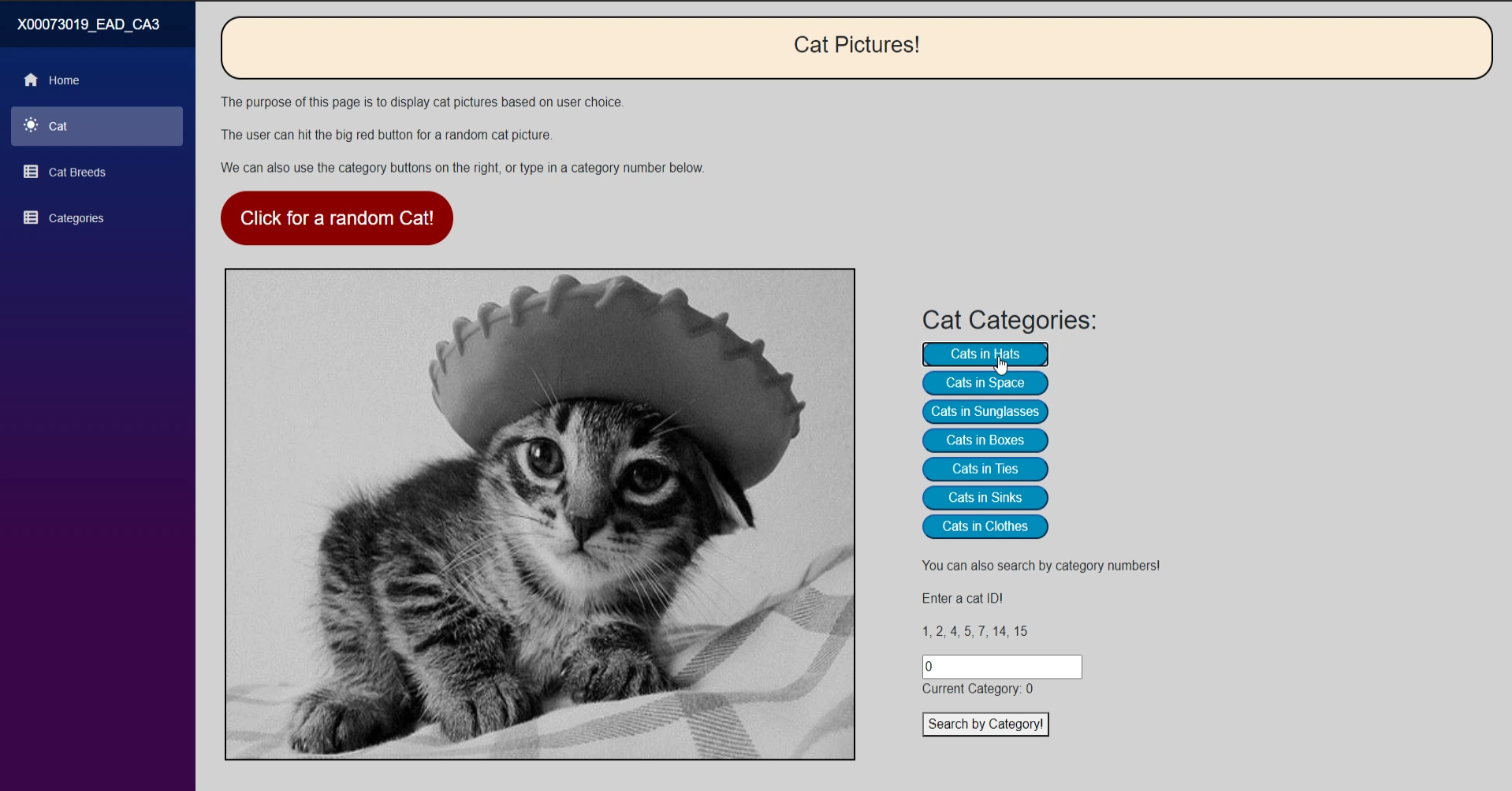
I also a developed 2 unit tests in MSTest to test object creation and the Filter method.

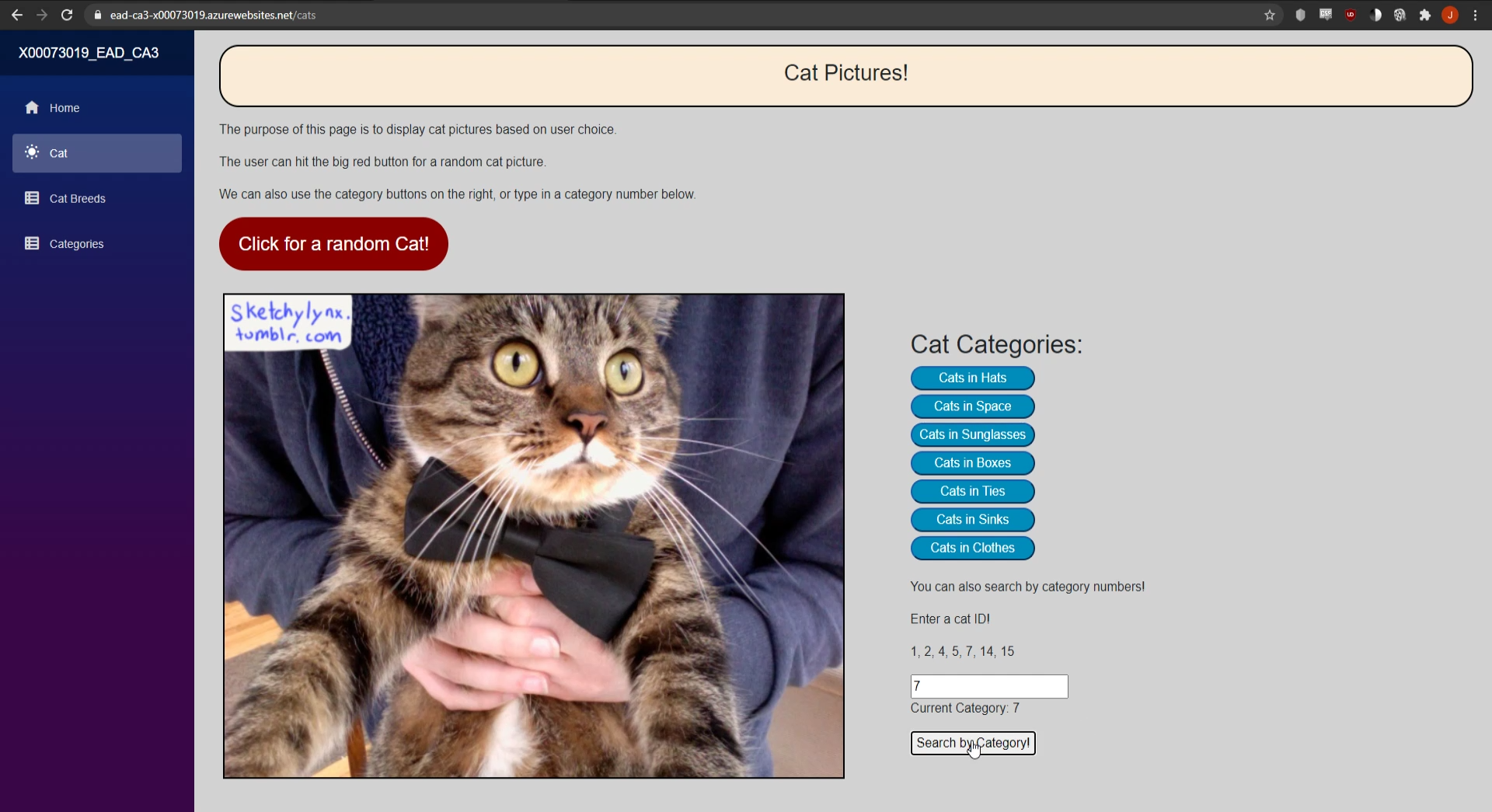
The filter test tested whether or not a created object was passed in successfully, and the object creation test simply tested if objects were created successfully. I would have liked to test the Asynchronous methods, but I could not get this to work.

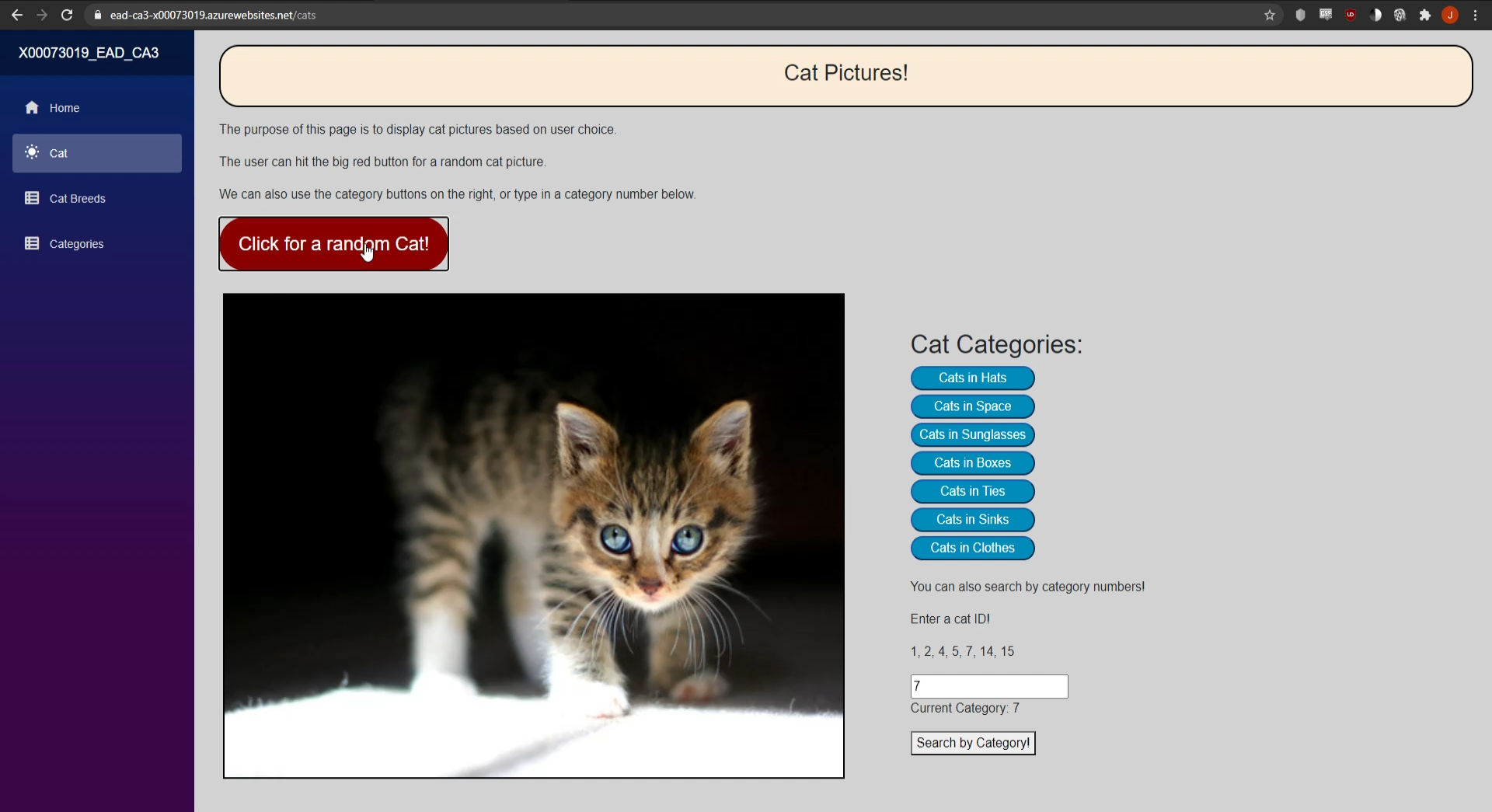
All code has been checked with SonarLint and cleaned to the best of my ability, in an attempt to increase readability.

**Screenshots of Functionality**

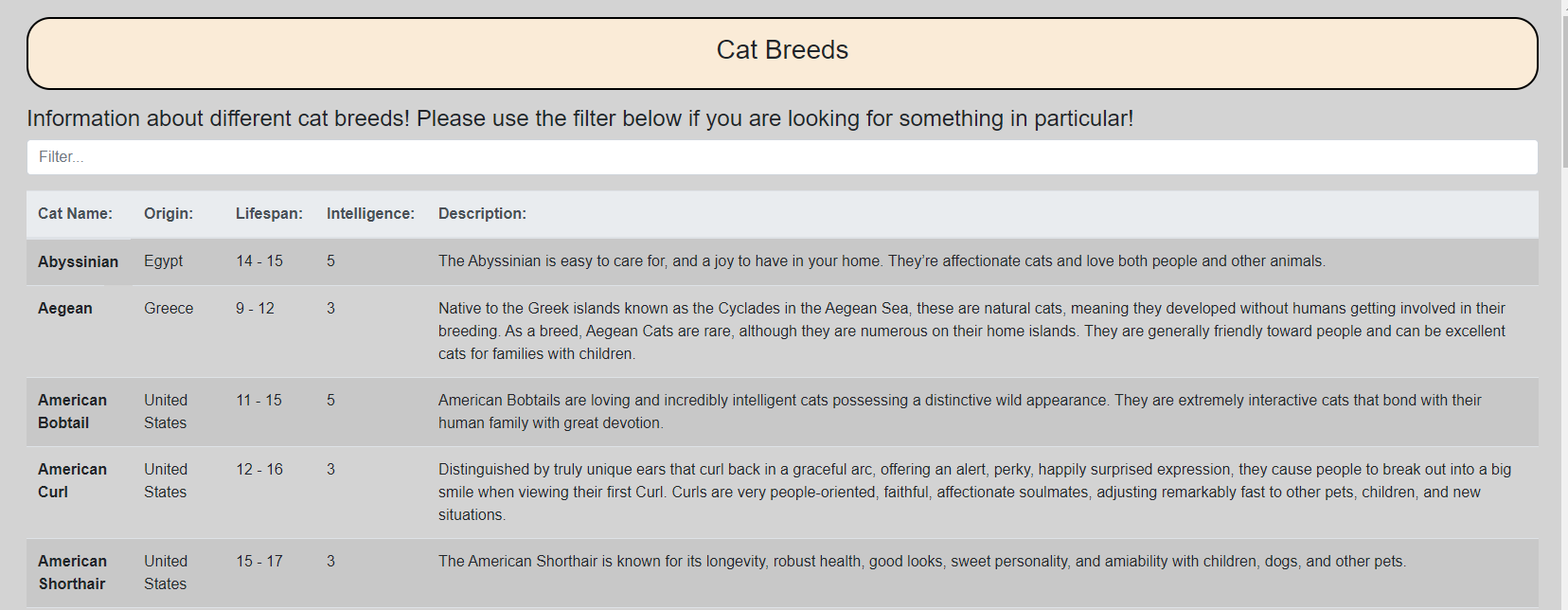
**Random Image and Button Filters:**

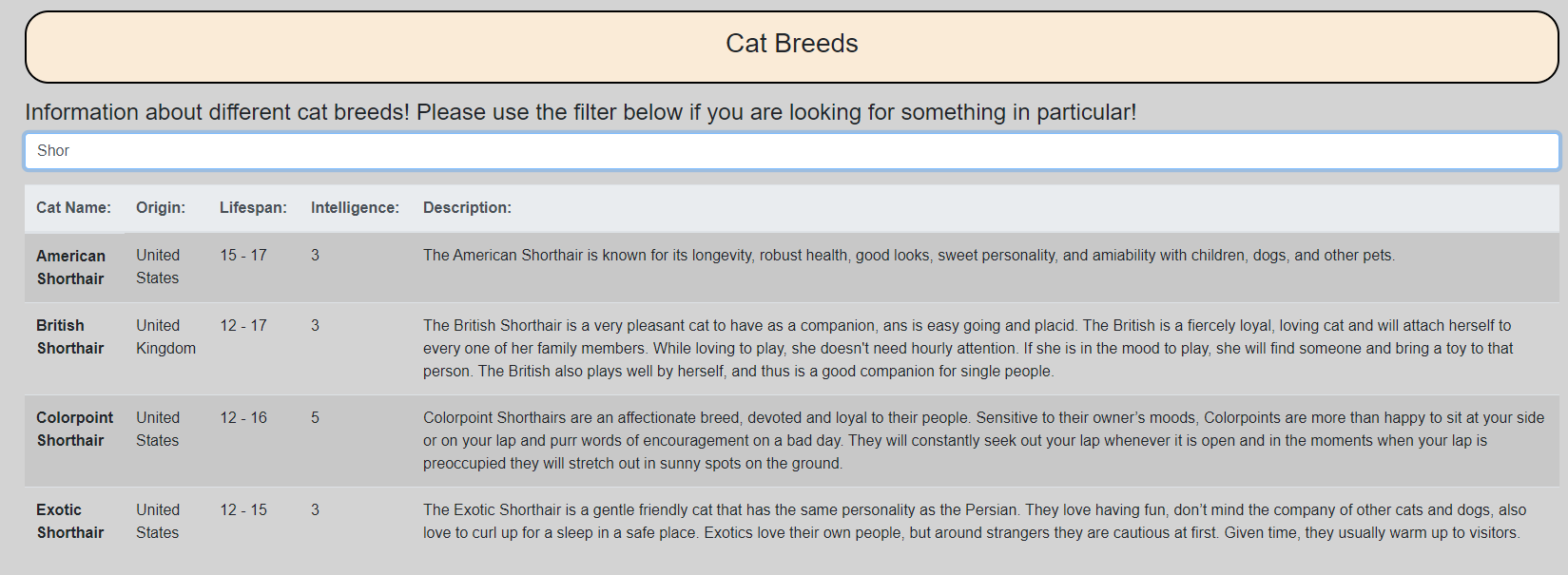
****

****

****

**Cat Breeds with Filter:**

****

****