Arman Abgaryan

Professor Marin

Comp 484

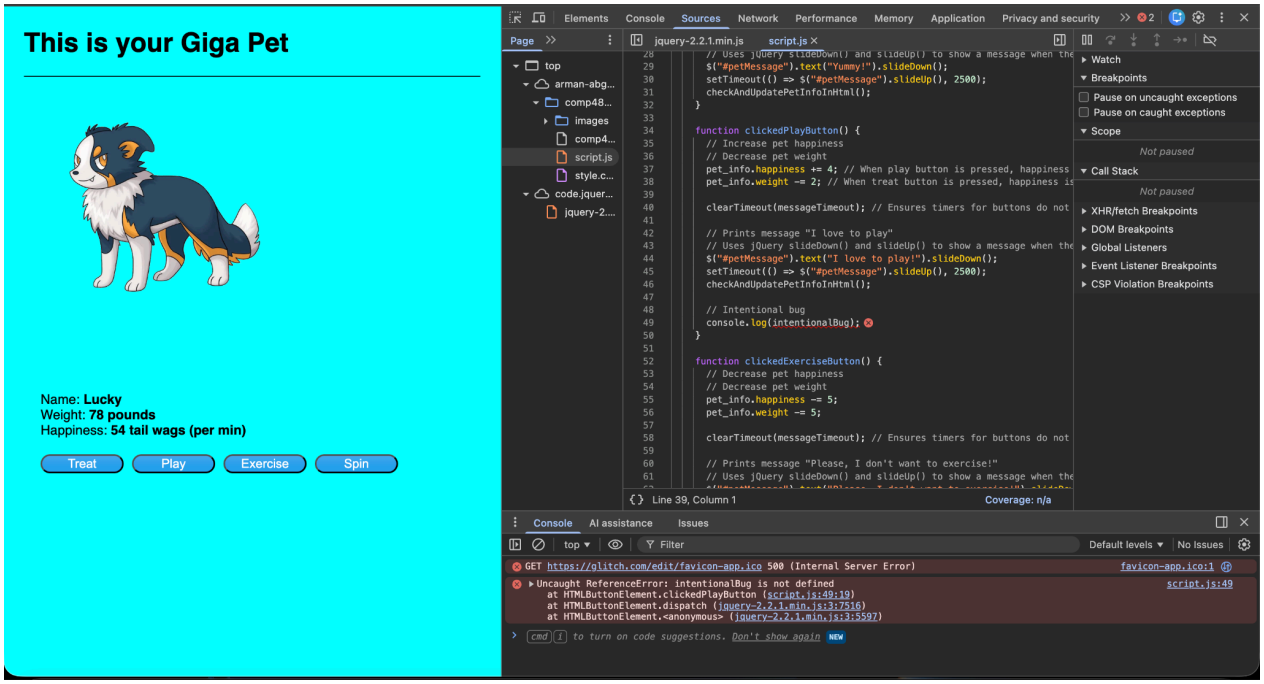
01 December 2025

HW10

Examples from https://developer.chrome.com/docs/devtools/javascript:

Reproduce a bug, Get Familiar with the Sources UI:

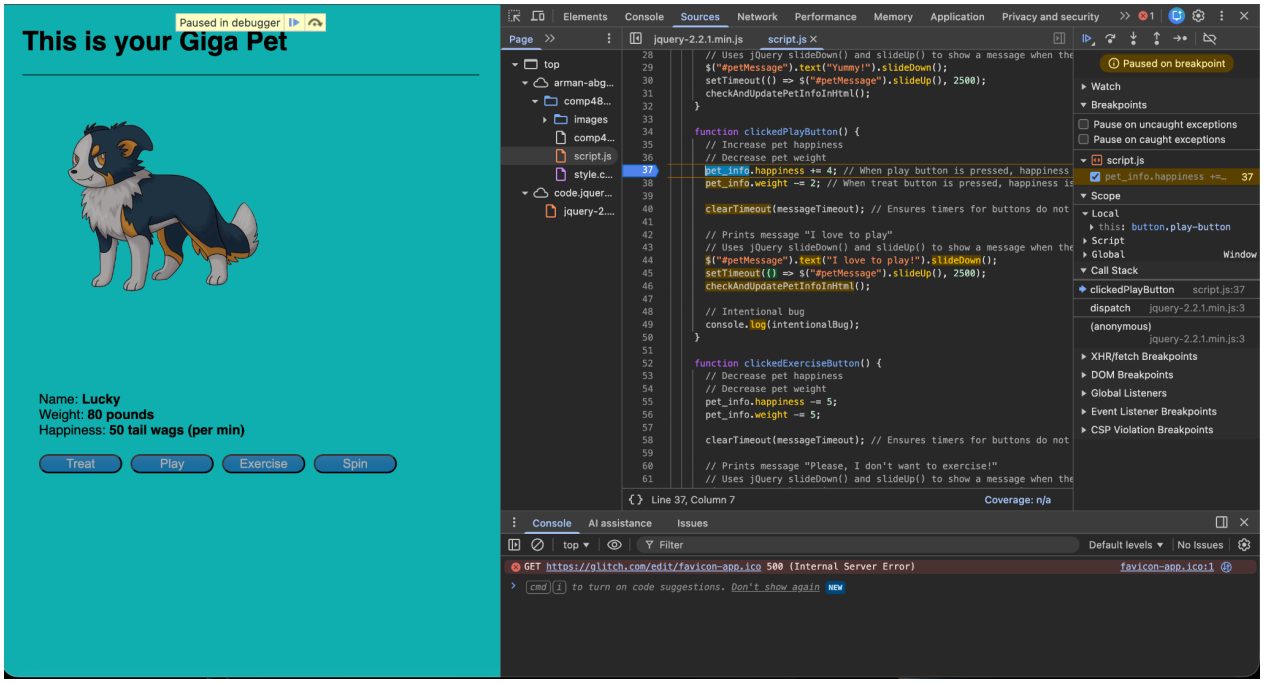
In order to show how Chrome DevTools can be used to identify and debug errors in JavaScript code, I intentionally introduced a bug into the code of my Project 2. I did so by adding “console.log(intentionalBug);” at the end of my script.js file. I then pushed these changes to my repository on GitHub and then launched it using GitHub Pages. Since “intentionalBug” is not defined anywhere in the file, I get a reference error. Using Chrome DevTools, I inspected the page, navigated to the sources tab, selected script.js, and I was able to see the reference error.



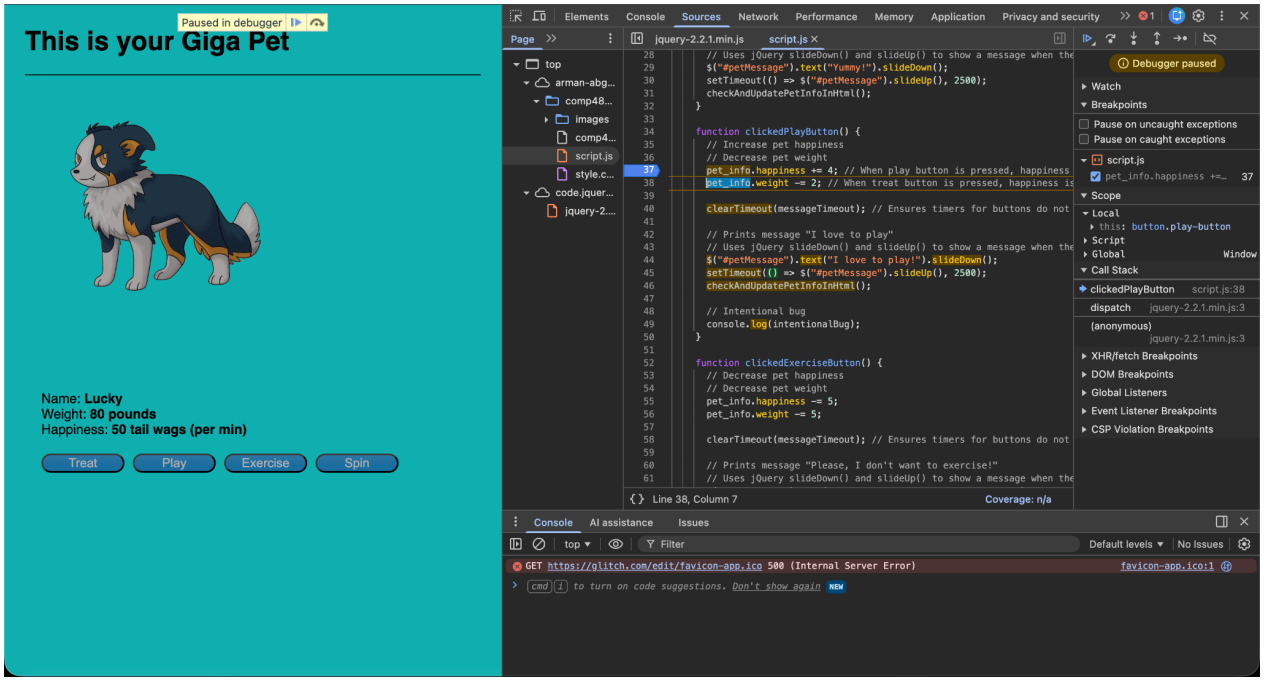
Pause the code, Step through code, Set a line-of-code breakpoint:

You can use Chrome DevTools in order to set breakpoints and step through code. To do so, I once again opened the pages link for my project 2, pressed inspect, navigated to script.js in the sources tab, and then selected the clicked on the line number of a random line of code. For this example, I chose to click line 37 which is within the clickedPlayButton() function. This line is responsible for increasing the pets happiness when the play button is pressed. By doing this, I was able to stop the code from executing right before the happiness value was adjusted, allowing me to then be able to step through each line of code in the function. Below I have screenshots of when I set the initial breakpoint along with stepping through different lines of code, ultimately reaching the end of the function.

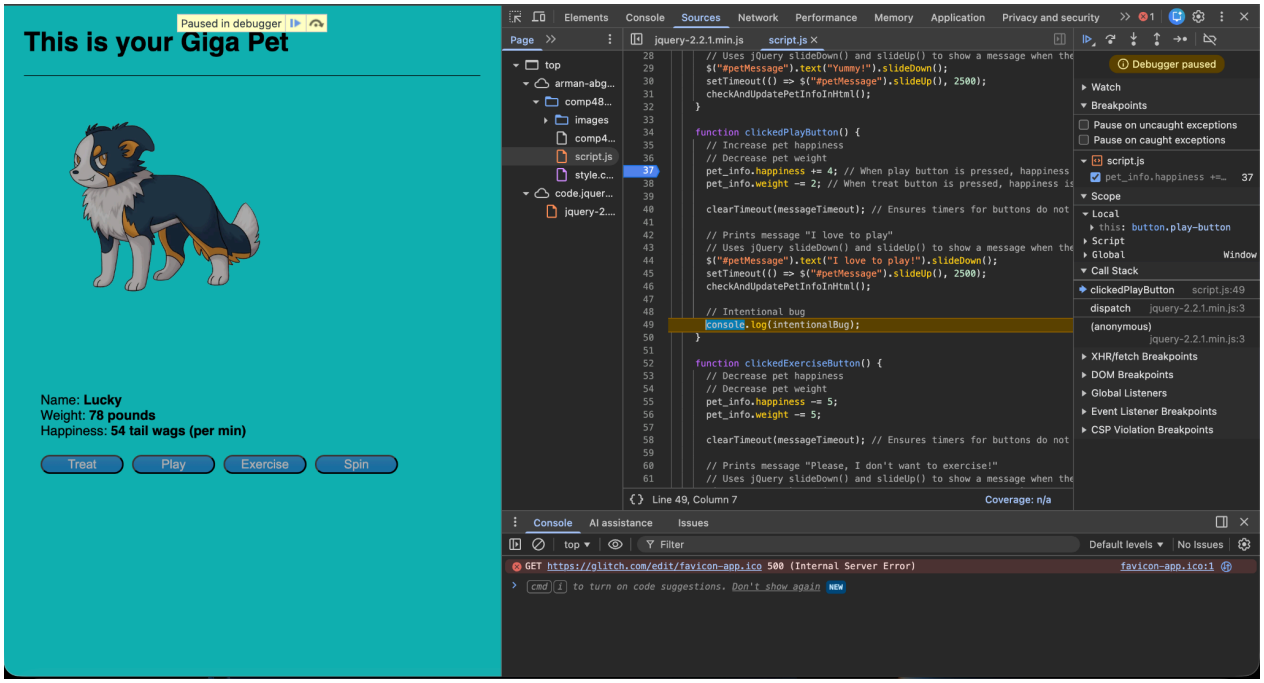
Initial breakpoint:



First step through:



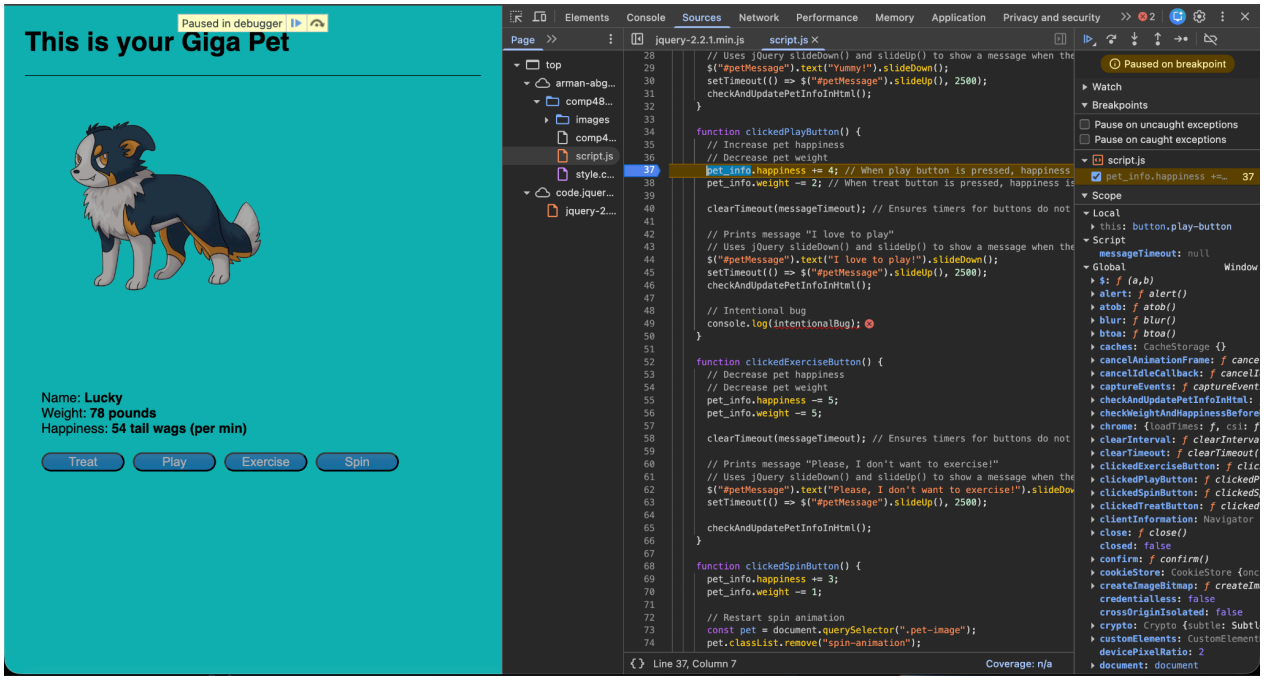
Multiple step-throughs to the intentional bug/end of function:



Check variable values (Method 1, 2, 3):

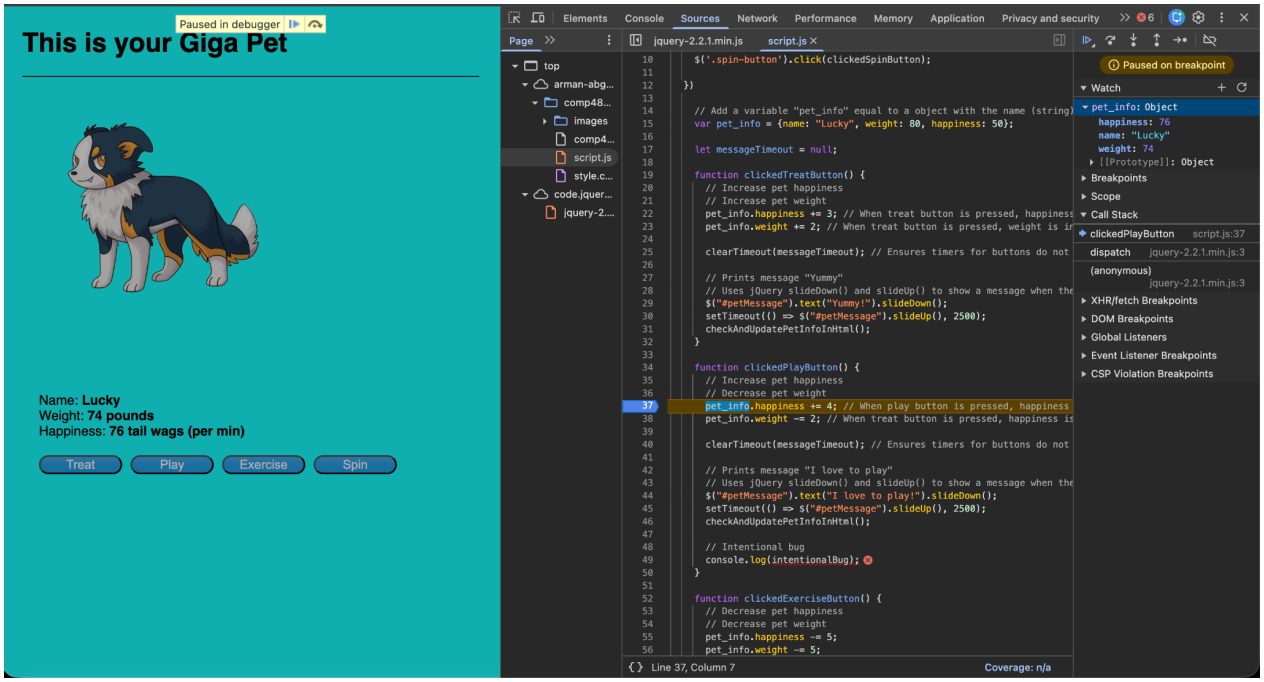
Method 1: Inspect the Scope

As noted on the website, when you are paused on a line of code while stepping through, you can navigate to the Scope tab to see the local and global variables that have been defined up to that point in execution. Back in my project 2, when traversing through clickedPlayButton(), I navigated to the Scope tab on the bottom right of the page. In the Scope, I saw drop-downs for the local variables, global variables, and the script. Upon expanding each of them, I was able to see all types of variables that were declared and used up to this point of execution.

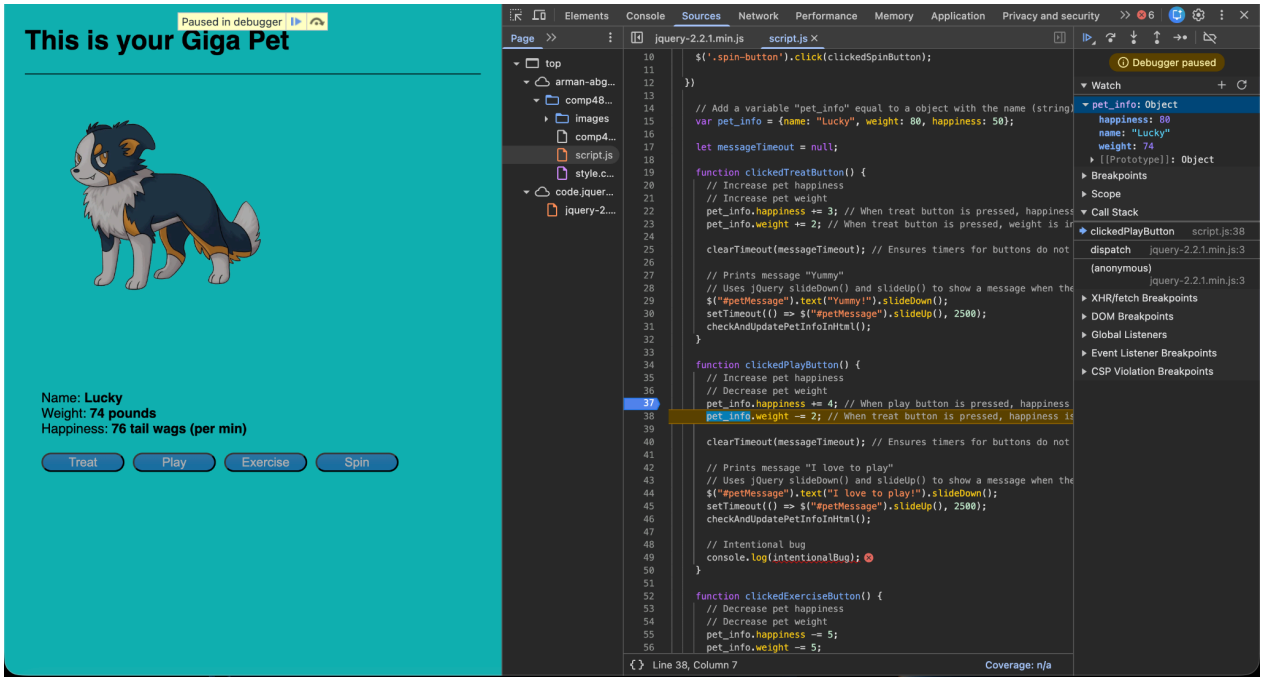


Method 2: Watch expressions

You can also use the watch tab right above the scope to monitor how the values of different variables change over time as you step through different lines of code. To demonstrate how the tab works, I once again placed a breakpoint in the clickedPlayButton() function. During the initial breakpoint and step through, I expanded the watch tab, clicked on the “+” button, and then added the variable pet_info. I was able to see the current values of my giga pet’s happiness, name, and weight.

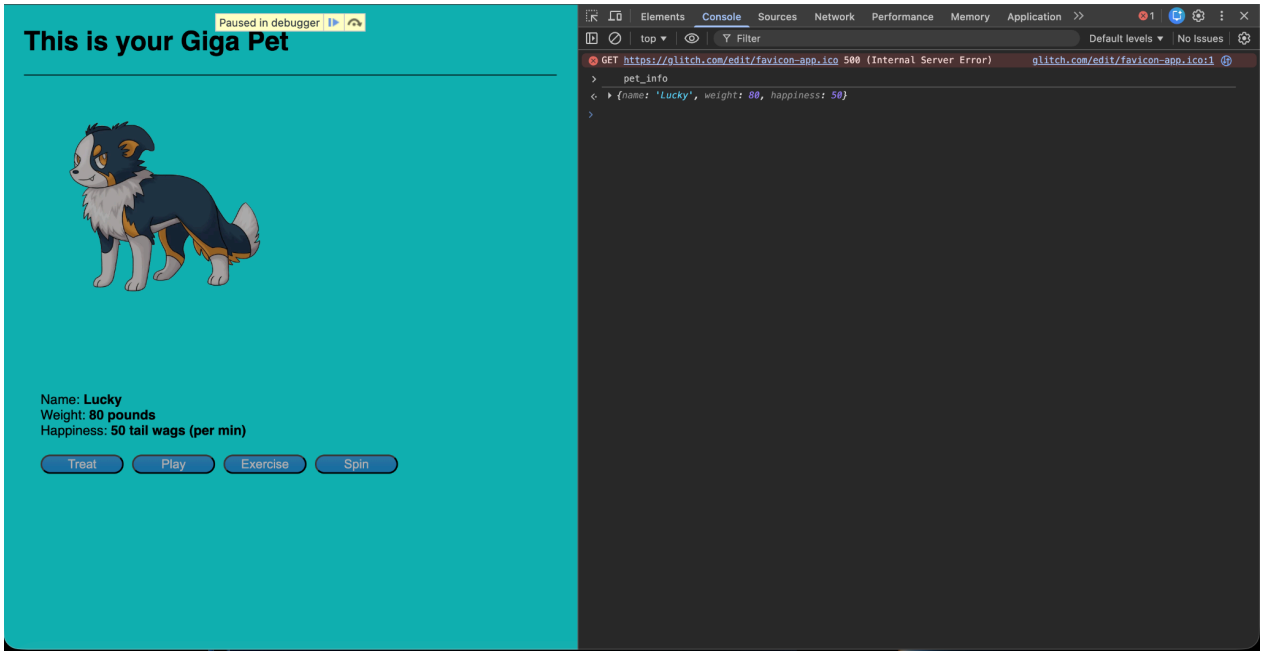


I then stepped through to the next line of code, and I was able to see the value of happiness change in the watch tab, updating from 76 to 80. The watch panel demonstrated in real time how the values of variables change when a line of code executes.

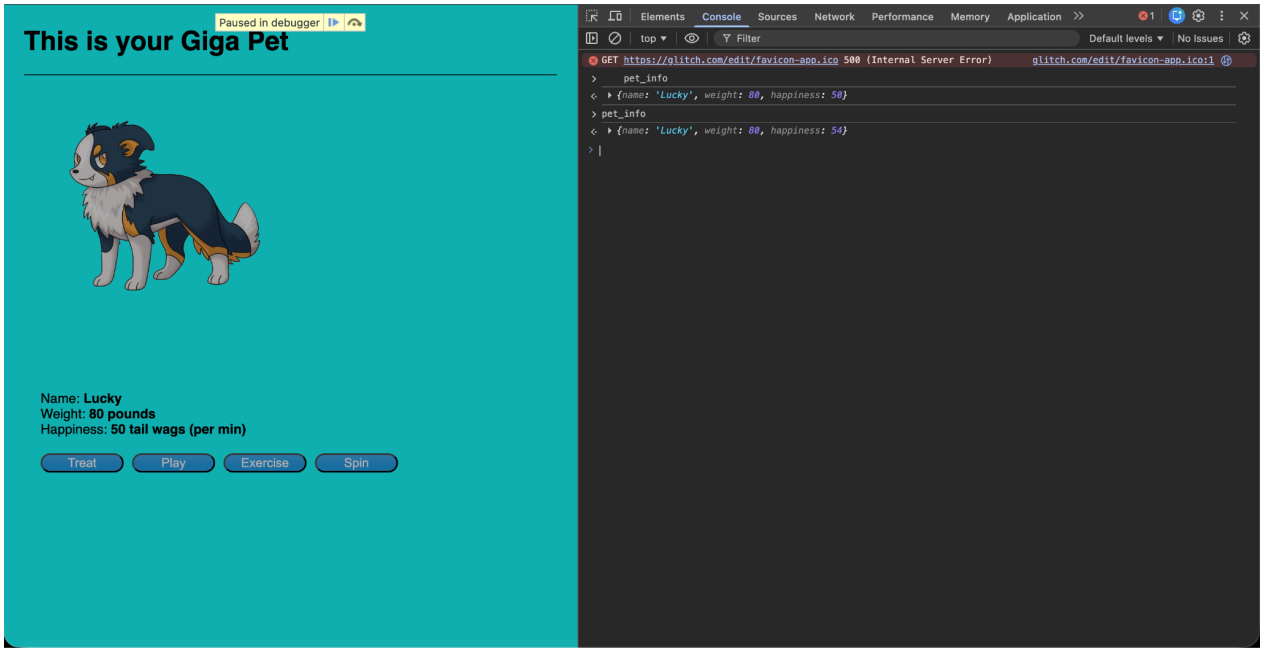


Method 3: The Console

Similar to watch, we can use the console in Chrome DevTools to check the values of variables. I once again stepped through the clickedPlayButton() function and used the console to monitor the values of the variable pet_info simply by typing “pet_info” in the console. The console then printed the current values of the name, weight, and happiness of the pet.

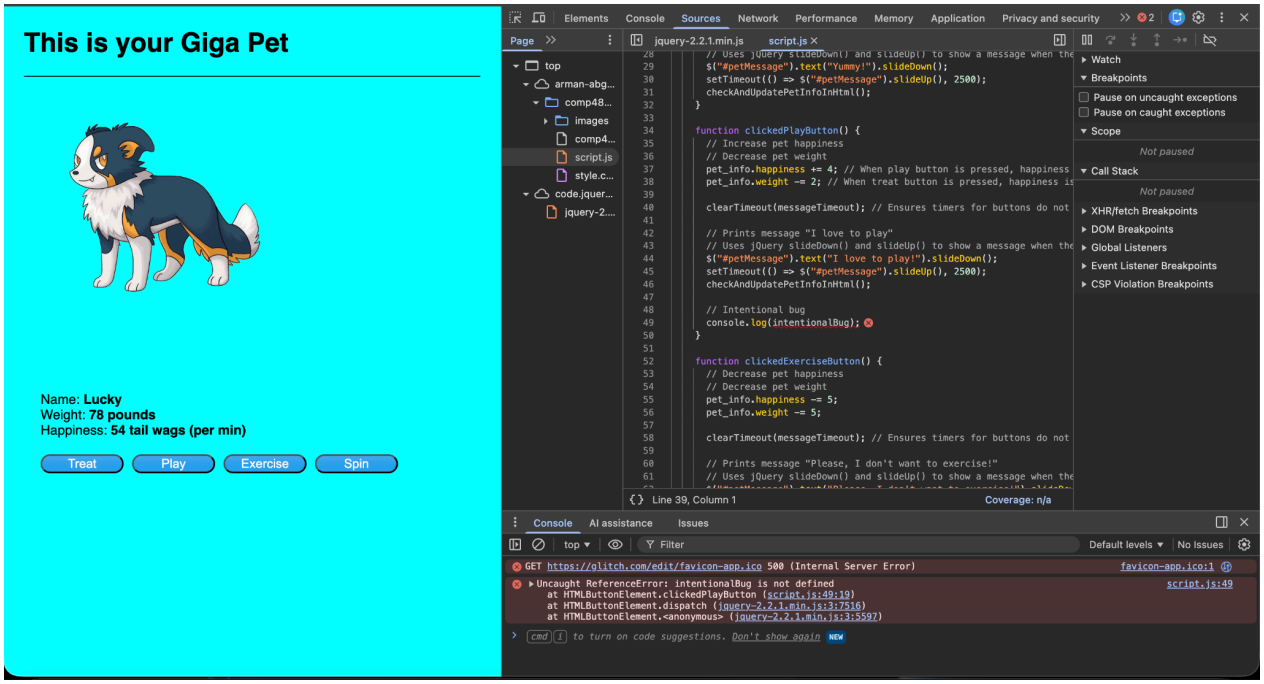


I then stepped through to the next line of code, which is responsible for increasing the pet's happiness by 4. I once again typed “pet_info” in the console and was prompted with the new and updated values of pet_info.

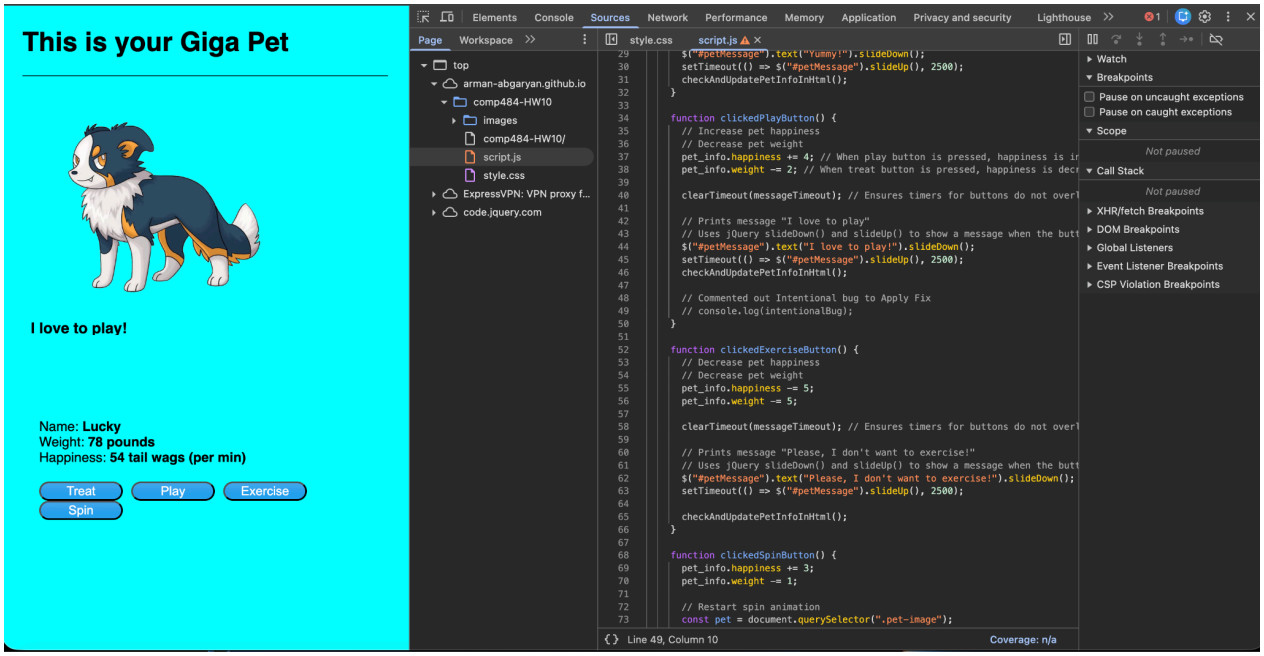


Apply a fix:

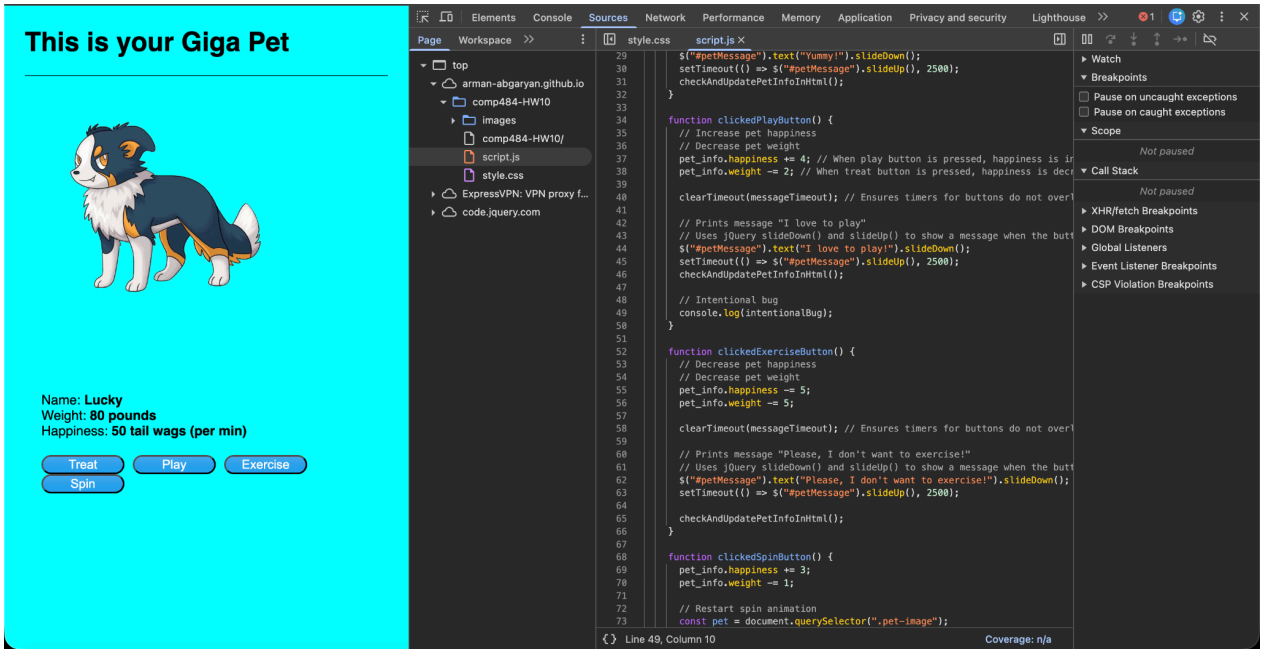
As demonstrated in the examples, we can directly apply a fix to the bug in our code using Chrome DevTools. The specific bug I introduced once again was “console.log(intentionalBug)” which would result in a reference error as intentionalBug was not defined anywhere in the code. As a result, when I would click the play button, I would be prompted with a reference error.



What I did to fix the bug was simply comment out “console.log(intentionalBug).” I then saved the changes I made in DevTools using command + C (control + C for windows) and then clicked the play button again. As a result of the fix I applied, the code ran properly with no reference errors.



Using DevTools, I was able to apply a fix directly to my script.js file. This change, however, is temporary, as once the page is refreshed, the original version of script.js is reloaded, and the fix I implemented is no longer there.

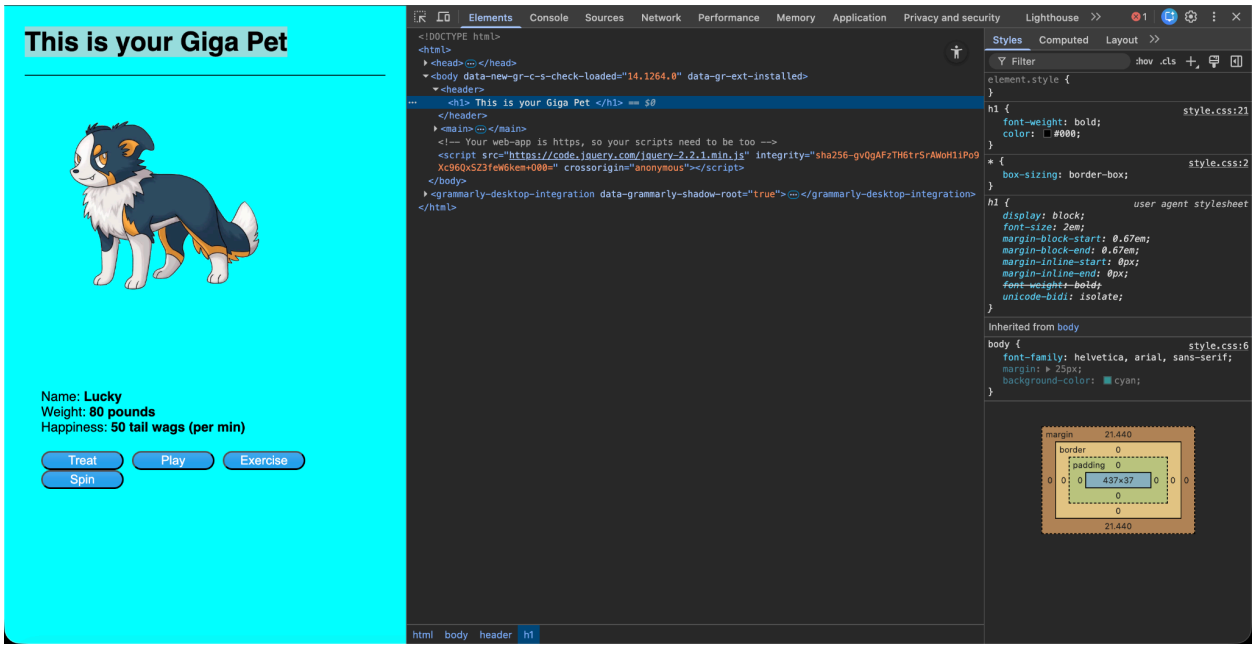


Examples from https://developer.chrome.com/docs/devtools/dom:

View DOM Nodes (All sub parts):

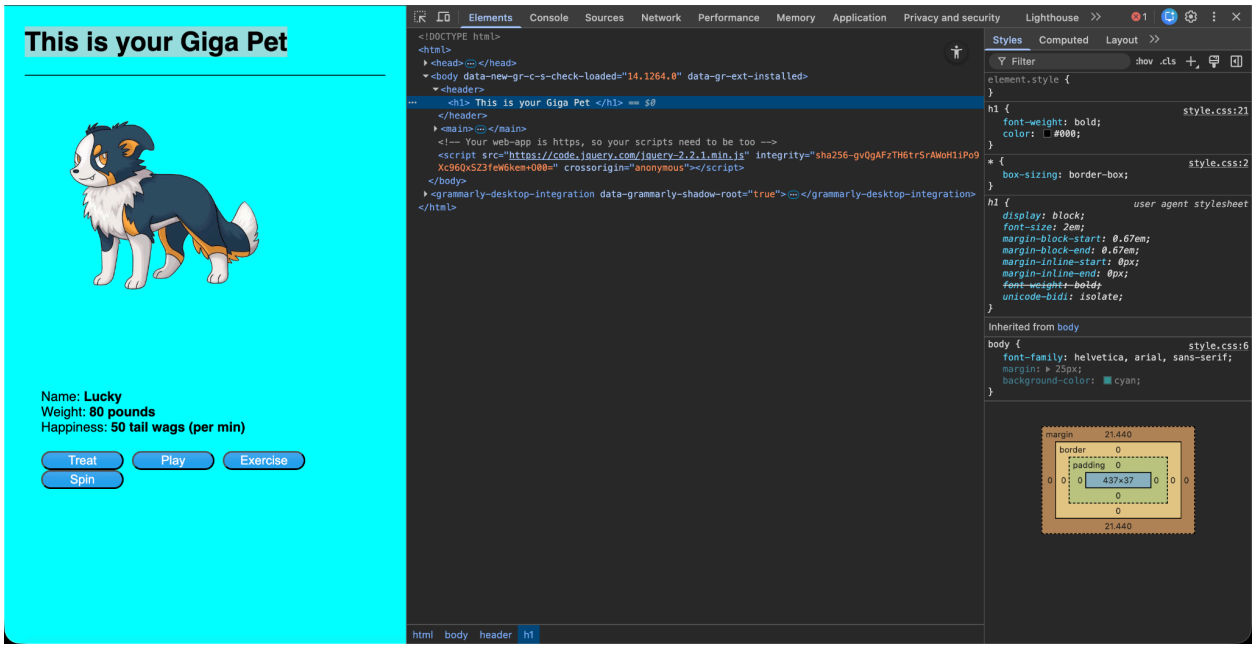
Inspect a node:

For this part of viewing the DOM Nodes, I chose to inspect the header of the page. After opening my project 2, I highlighted the text at the top of the page (“This is your Giga Pet”) and right-clicked on the element. I then selected inspect and this took me to the elements page in the DevTools, showing me the corresponding DOM node.

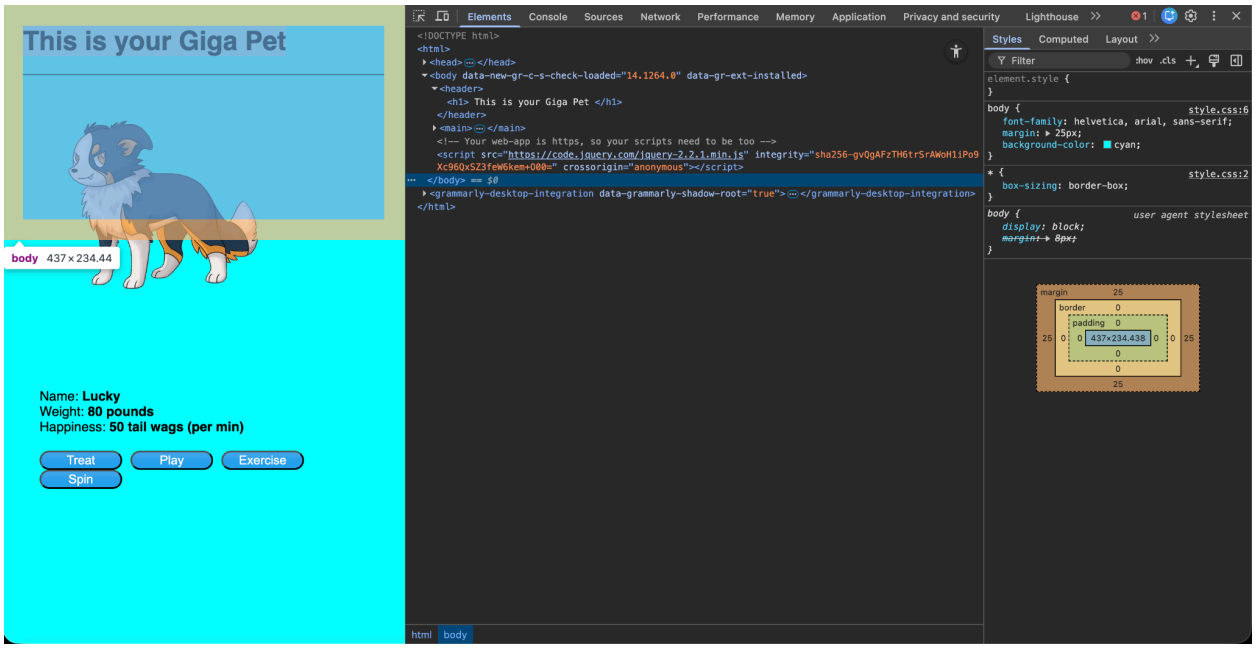


Navigate the DOM Tree with a keyboard:

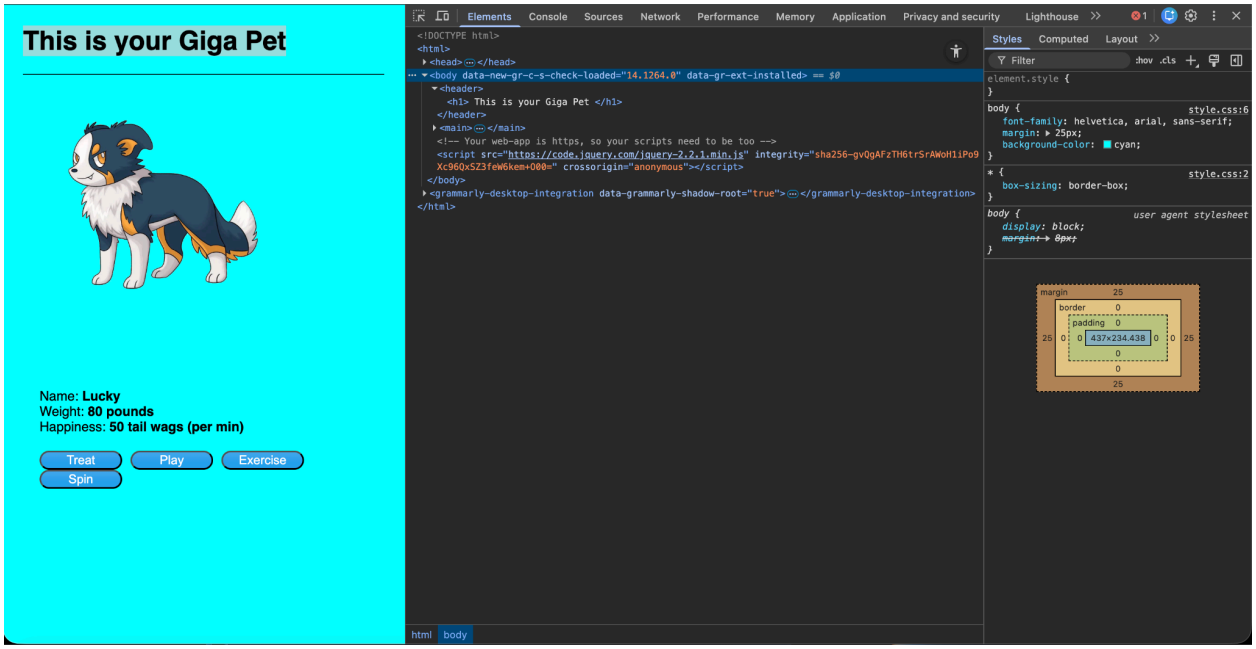
After selecting the DOM node in the Elements tab, I was then able to navigate the DOM tree using my keyboard. I used the downwards arrow on my keyboard to traverse down the tree.



I continued to traverse down the DOM tree until I reach the closing body tag which was below the script.

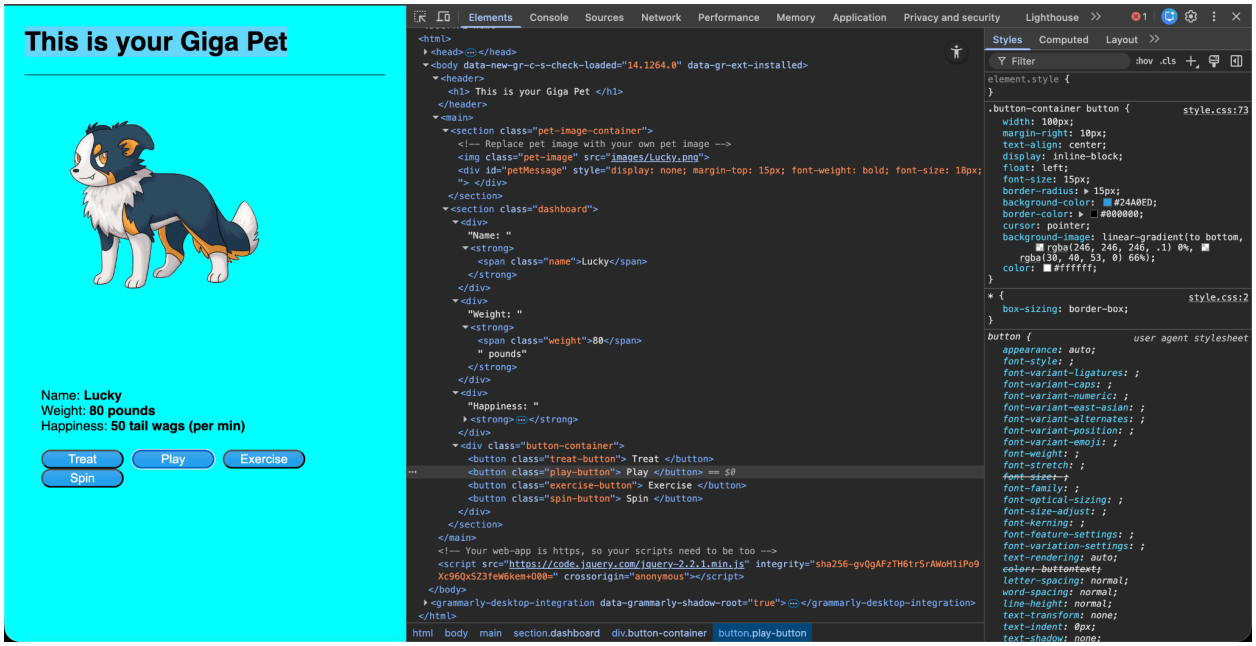


I also used the left and right arrows on my keyboard to both expand and close different nodes in the DOM such as head, header, body, etc.



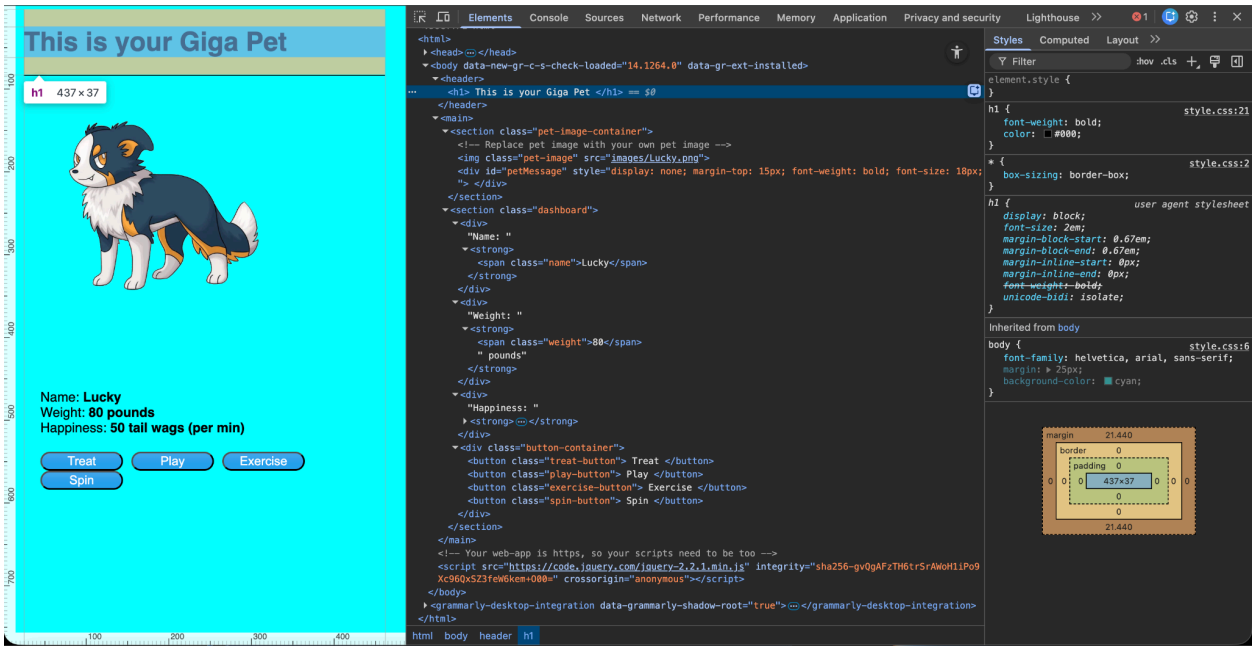
Scroll into view:

For this section, I first selected the node in the DOM tree for the “play” button. I first clicked on the node itself to highlight it and right clicked on the node and selected “Scroll Into View” from the list options. As a result, my viewport scrolled up so I can see button node.



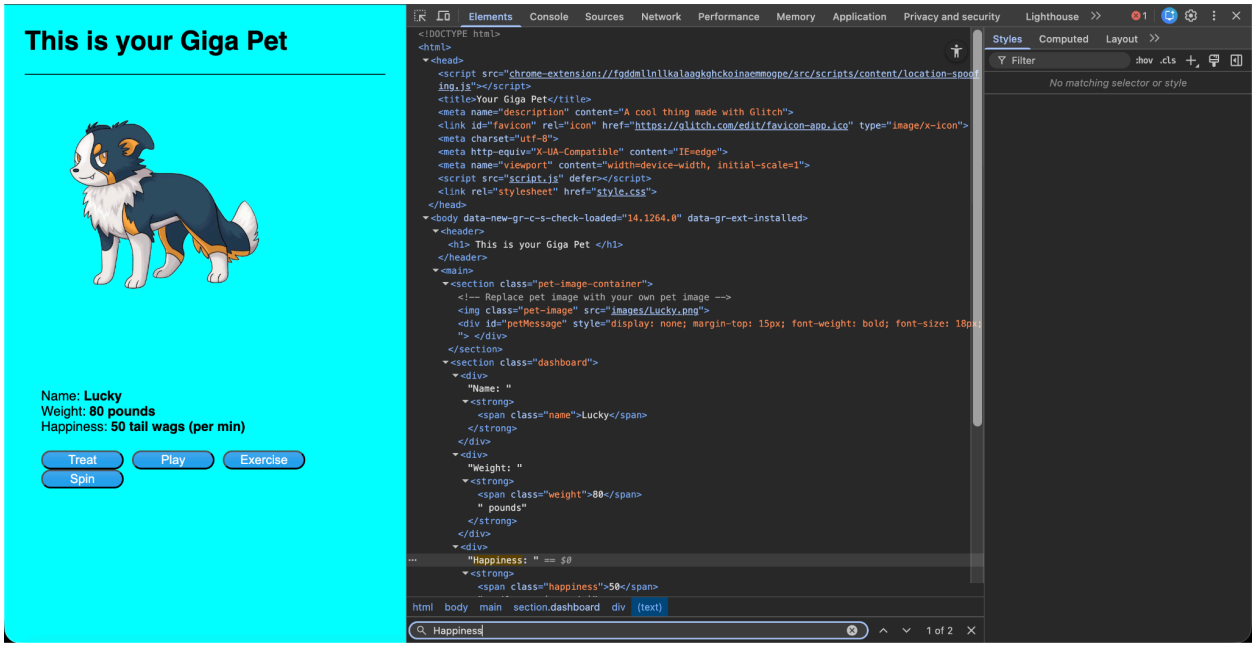
Show rulers:

In order to show the rulers while hovering over an node in the DOM tree, I first had to go into the settings and enable “Show rulers on hover.” After enabling this setting, I returned to the elements tab and hovered over the h1 node in the tree. In doing so, “This is your Giga Pet” was highlighted and the rulers demonstrated the width and height of the header.



Search for nodes:

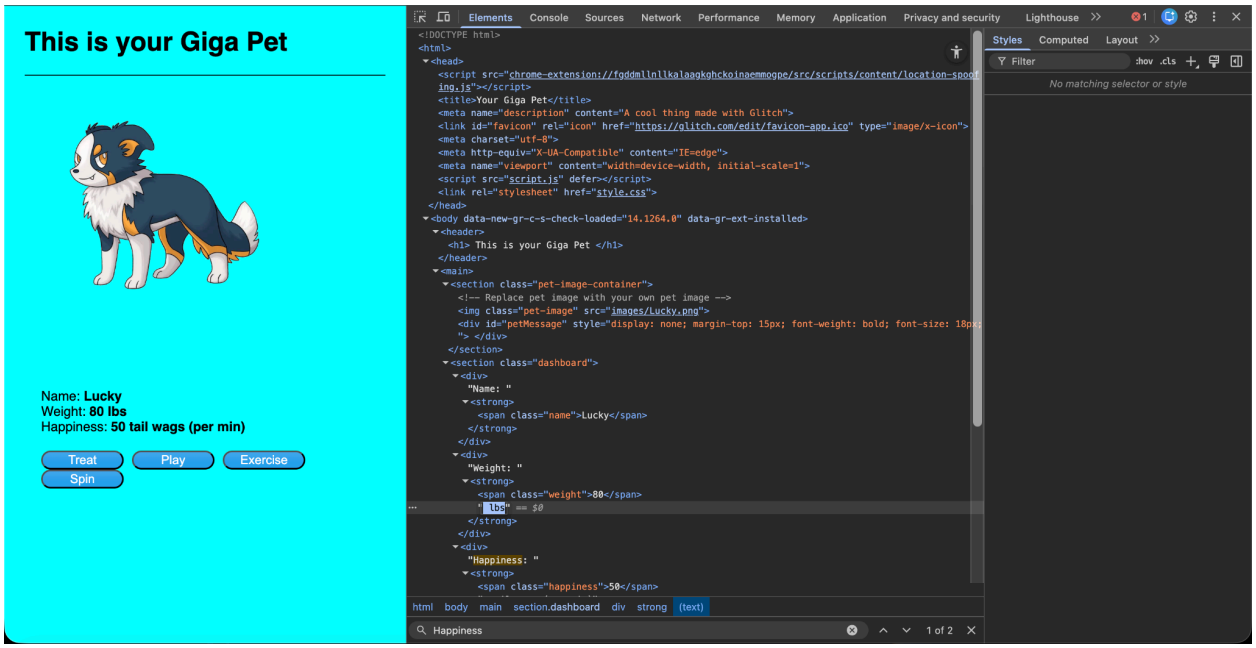
To search for a node in the DOM tree, use the shortcut command + F (control + F on windows) which will open a search bar at the bottom of the page that you can use to search for a specific node. The node I searched for was “Happiness.”



Edit the DOM (Edit the content, Edit attributes, Edit node type, Edit HTML):

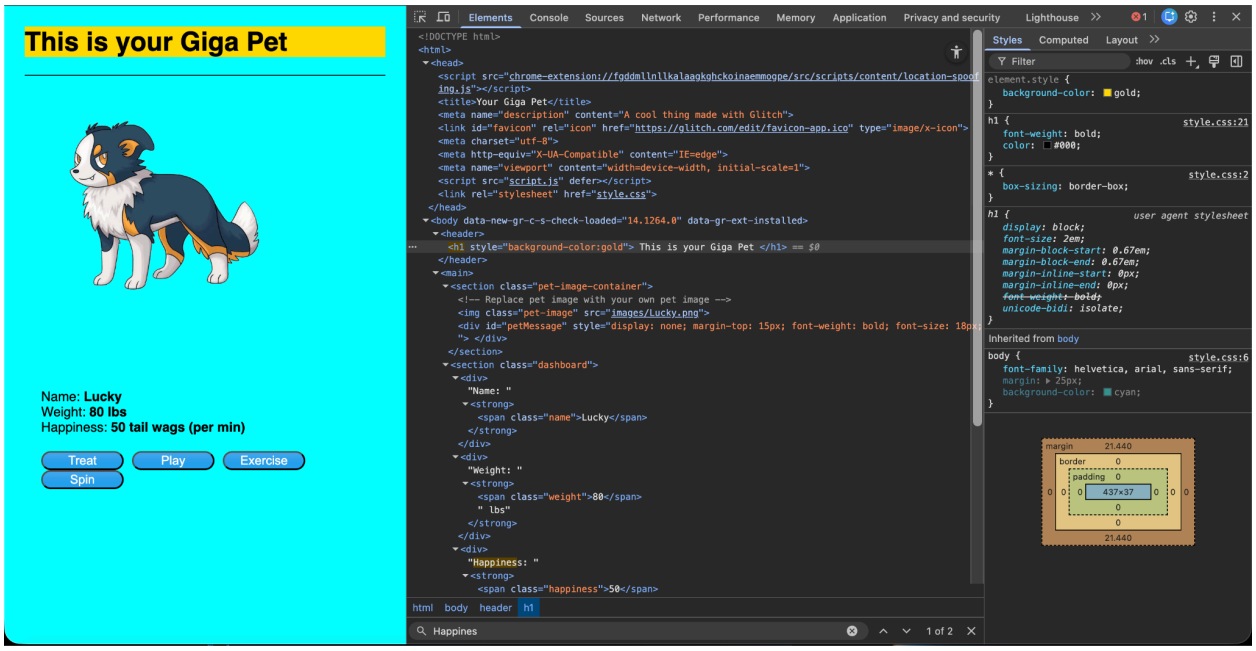
Edit the content:

To edit the content in the DOM, select a node within the tree and double click the element. This will allow you to edit the element and change it to whatever you'd like. For example, I double clicked "pounds" and edited it to "lbs."



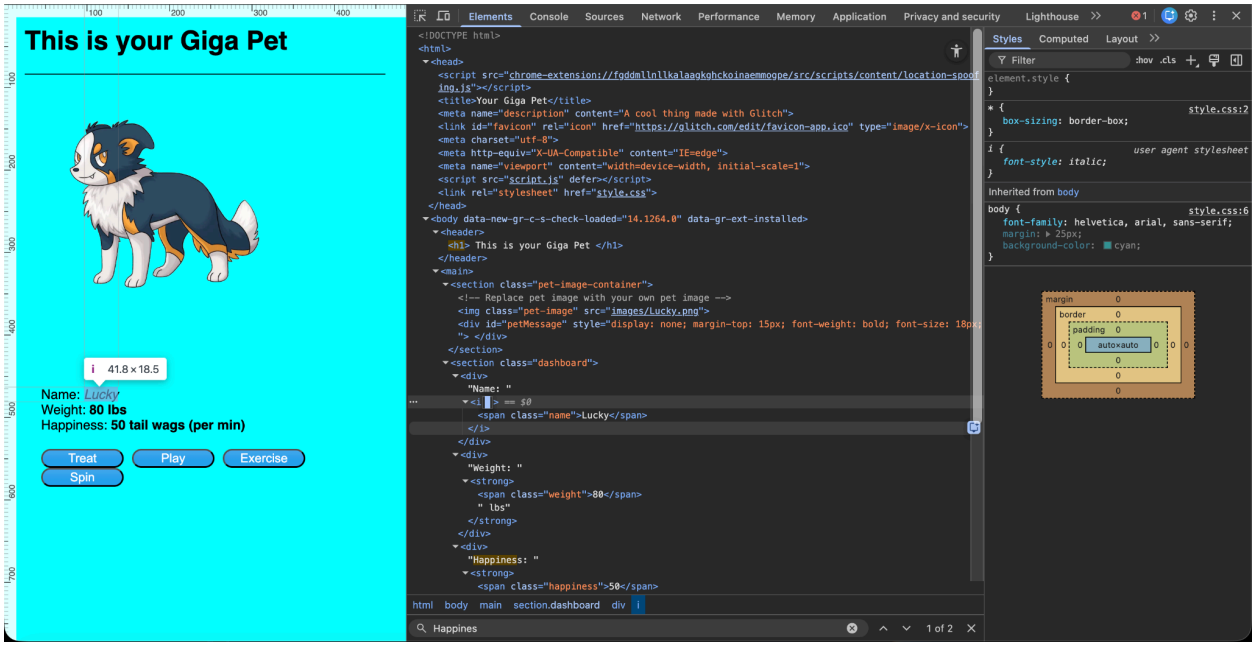
Edit attributes:

To edit the attributes of an element, double click on a specific element, press the right arrow and then space on the keyboard, and then add any sort of change to the element. For example, I decided to edit the background color of <h1> This is your Giga Pet </h1> and change it to gold, similar to the example in the website.



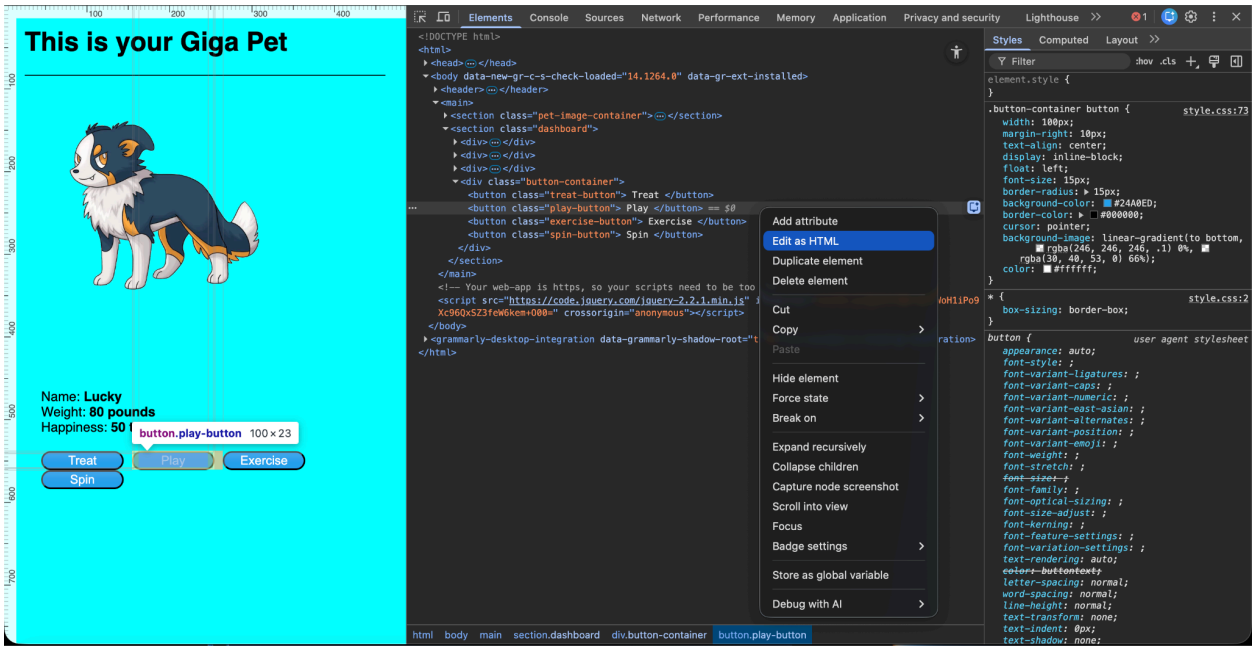
Edit node type:

Similar to editing the attribute, to directly edit the node type, select a specific node to be changed, double click on the node's type (i.e. <h1>, <p>, etc.), and then edit it to be another node type. For example, I changed the appearance/style of "Lucky" from   to .

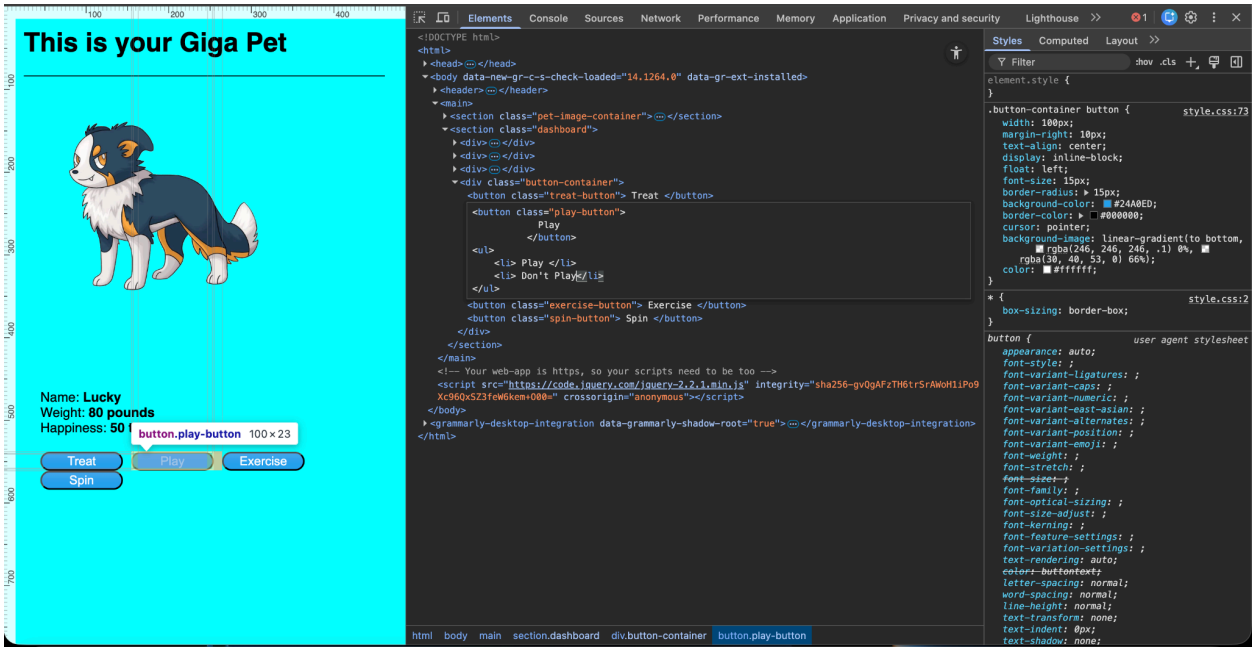


Edit HTML:

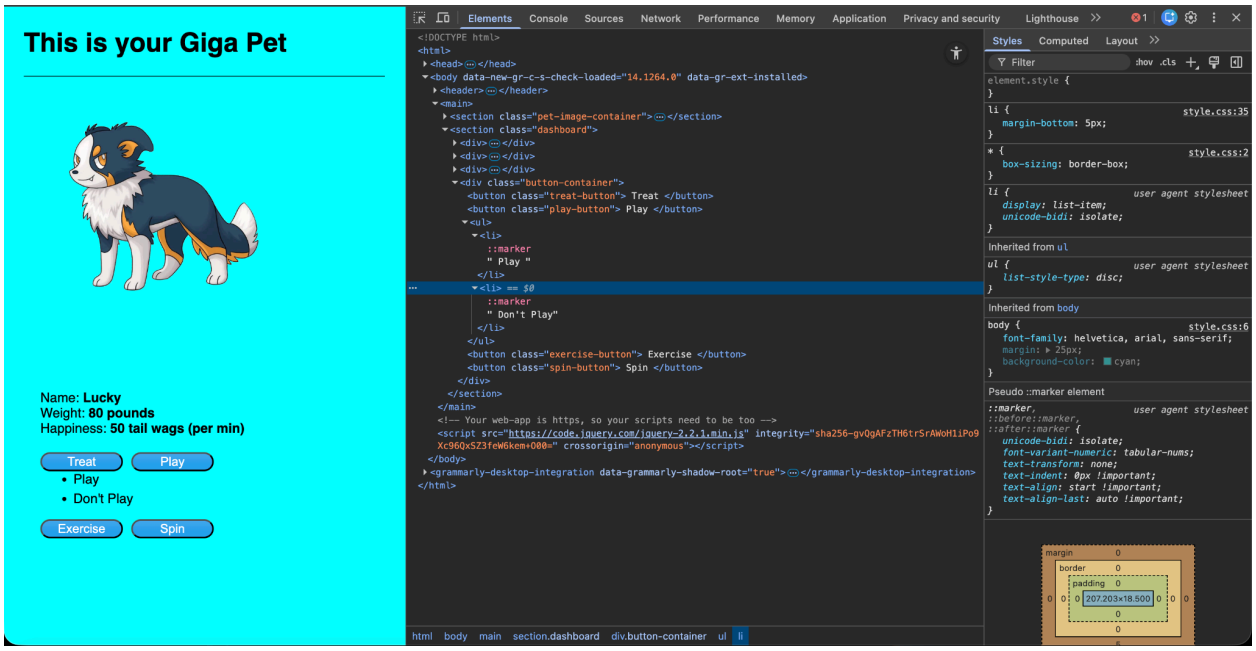
To edit the HTML of a doc, once again select a specific node to be edited, right click on the node and then select “Edit as HTML” from the list of options.



After selecting “Edit as HTML,” press enter to begin typing in a new line and then input a new line of HTML code.



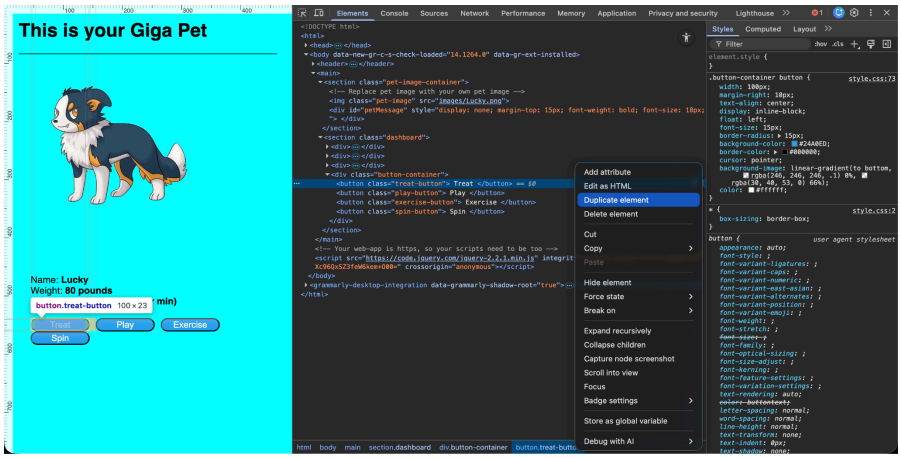
Once the new line of code has been typed, press command + enter (control + enter on windows) to apply the change. The page will automatically load and display the changes.



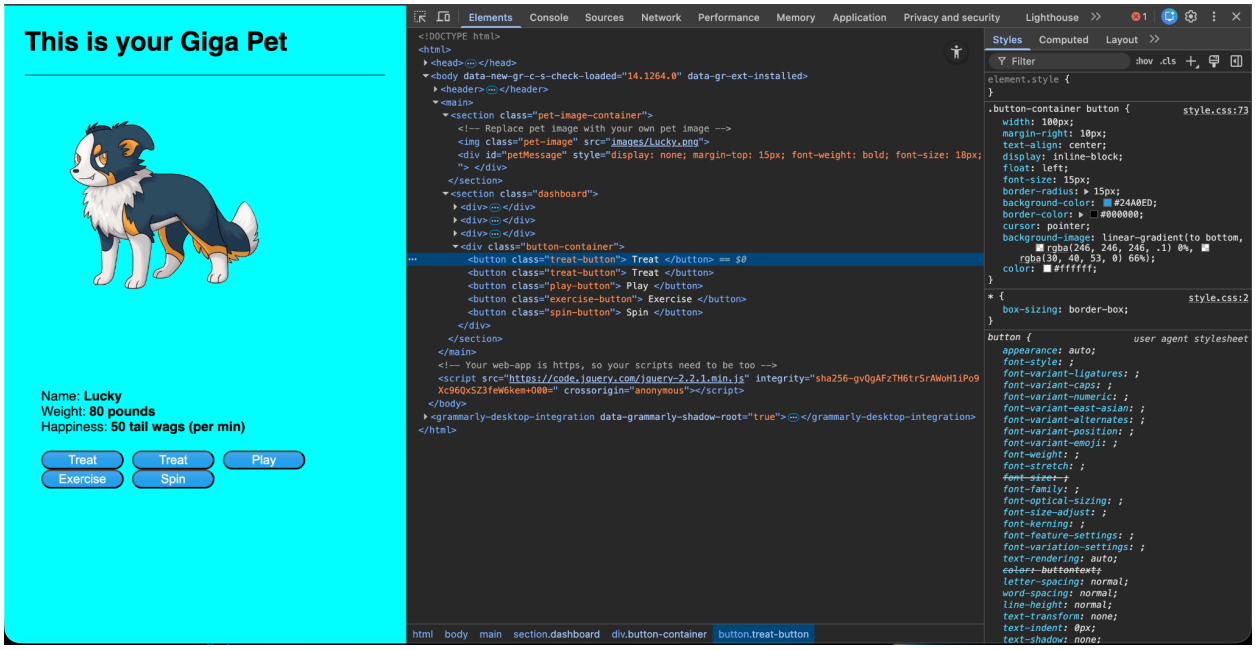
Edit the DOM (Duplicate a node, Capture a screenshot, Reorder DOM nodes, Force state, Hide a node, Delete a node):

Duplicate a node:

You are able to duplicate a node within the DOM. To do so, select a specific node to be duplicated, right click on the node and select “Duplicate element” from the list of options that appears.

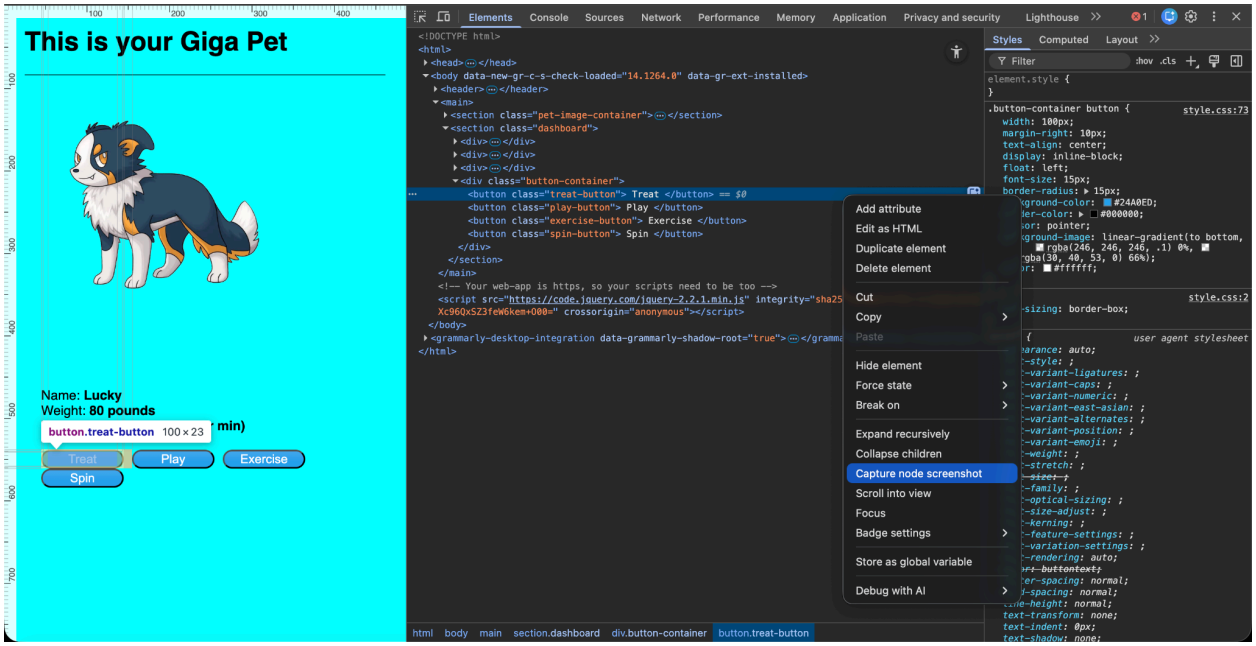


After pressing “Duplicate element” the respective element will be duplicated. For example, I selected the node for the “treat” button. Upon selecting “Duplicate element,” the button was duplicated and displayed on the page.



Capture a screenshot:

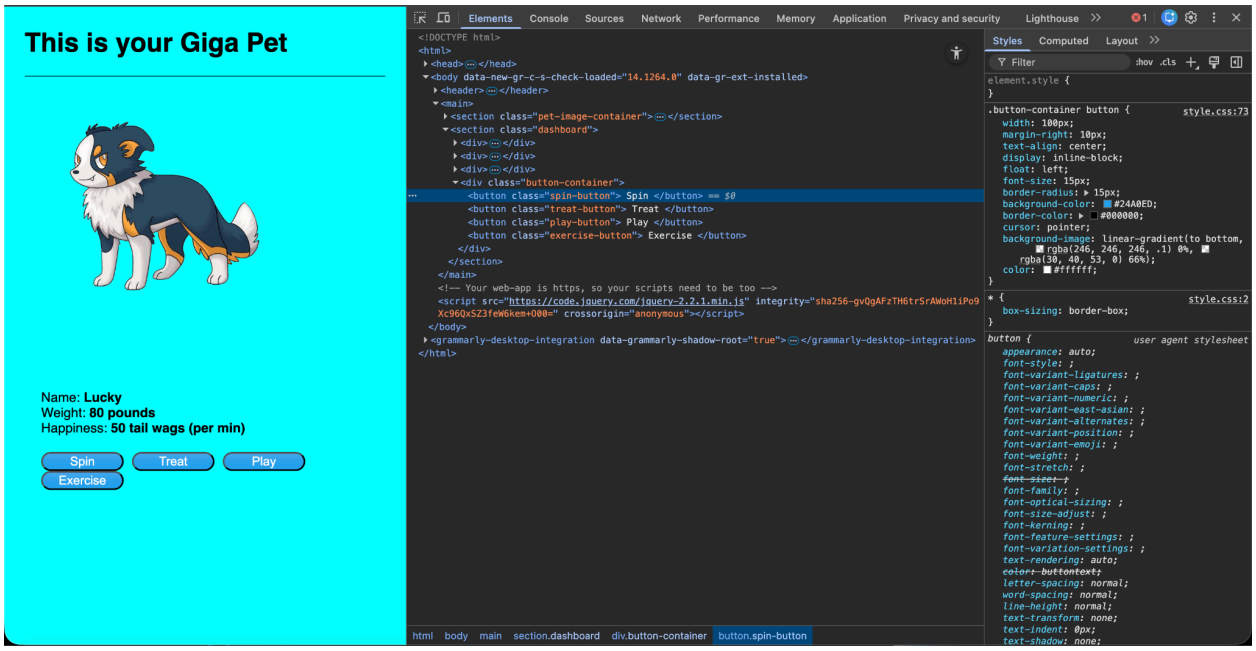
To capture a screenshot of a specific node within the DOM tree, once again select a node to be screenshotted, right click on the node, and then select “Capture node screenshot” from the list of options.



The node will then be screenshotted and automatically stored in your downloads.

Reorder DOM nodes:

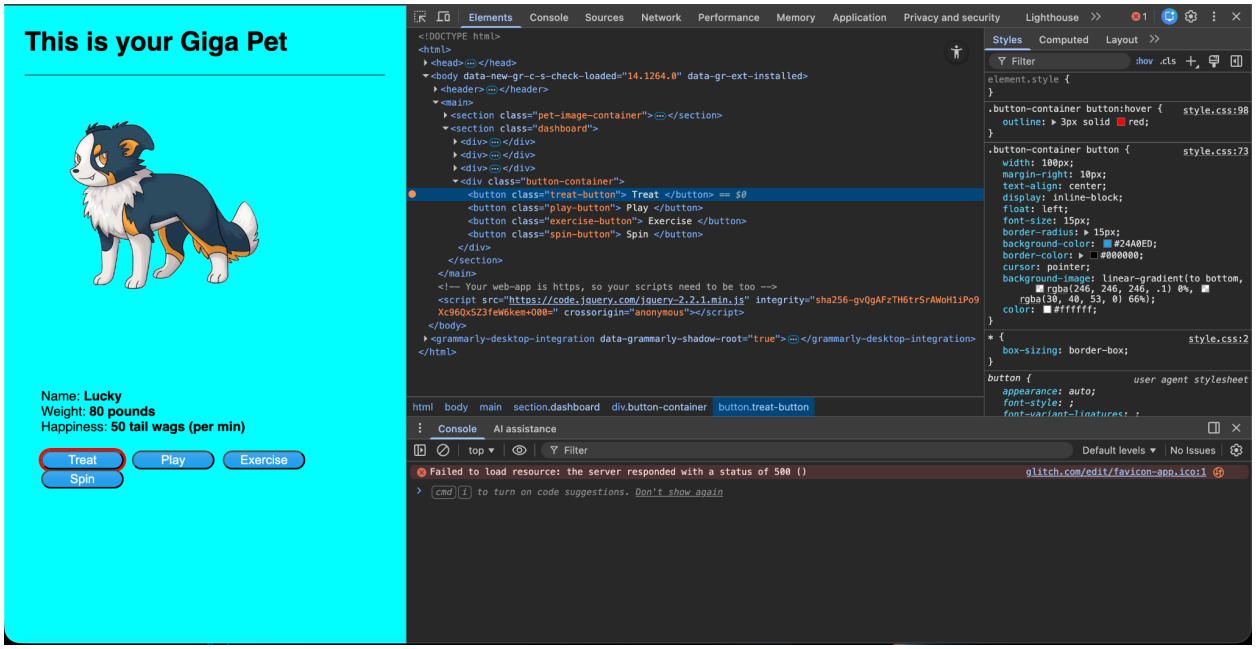
To reorder DOM nodes, it is as simple as selecting the node you'd like to reorder and dragging it and dropping it to a different location within the tree. For example, spin was originally the last button on the page. With the ability to reorder nodes, spin was moved to be the first button on the page.



Force state:

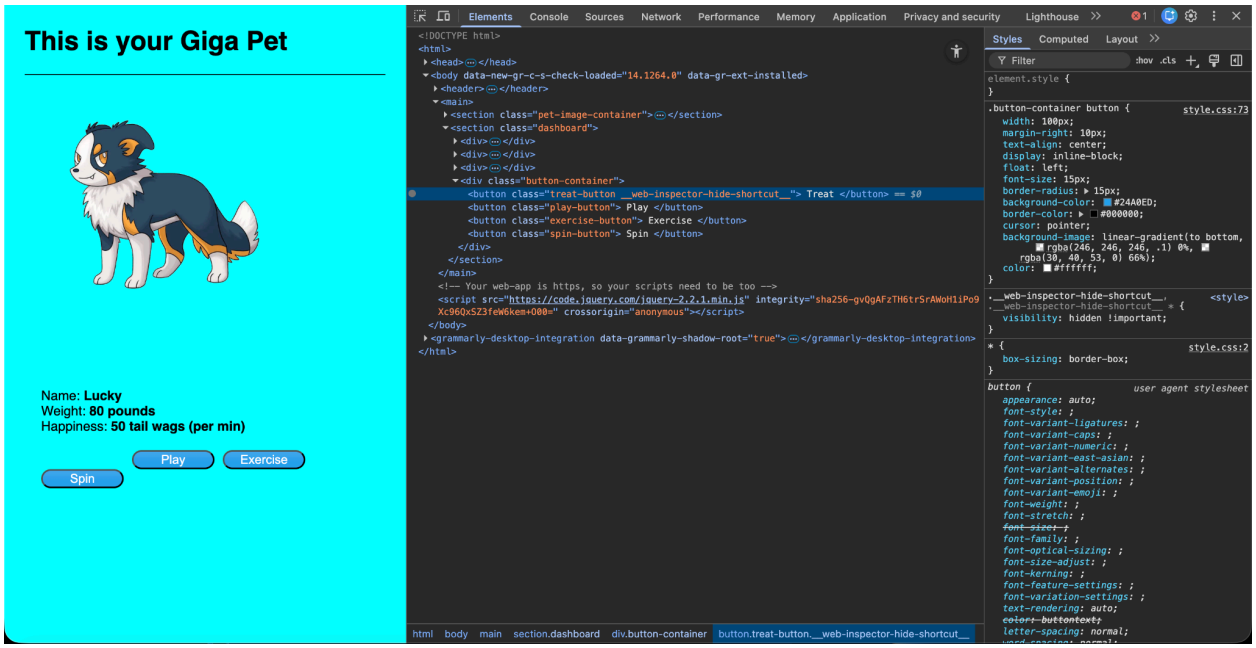
Force allows a user to force a node to remain in a particular state such as :active, :hover, and etc. To do so, select the specific node to be modified, right click on the element and select “Force State” from the list of options, and then select the state you’d like to force for the node. For example, for the treat button, the :hover state was forced, resulting in the button having a constant red border despite not actually hovering over the node itself. (In order for this to work for my project 2, I had to edit the styles.css file and add the :hover for the button as it was not

letting me force a specific state as :hover was not being used at all in my project before).



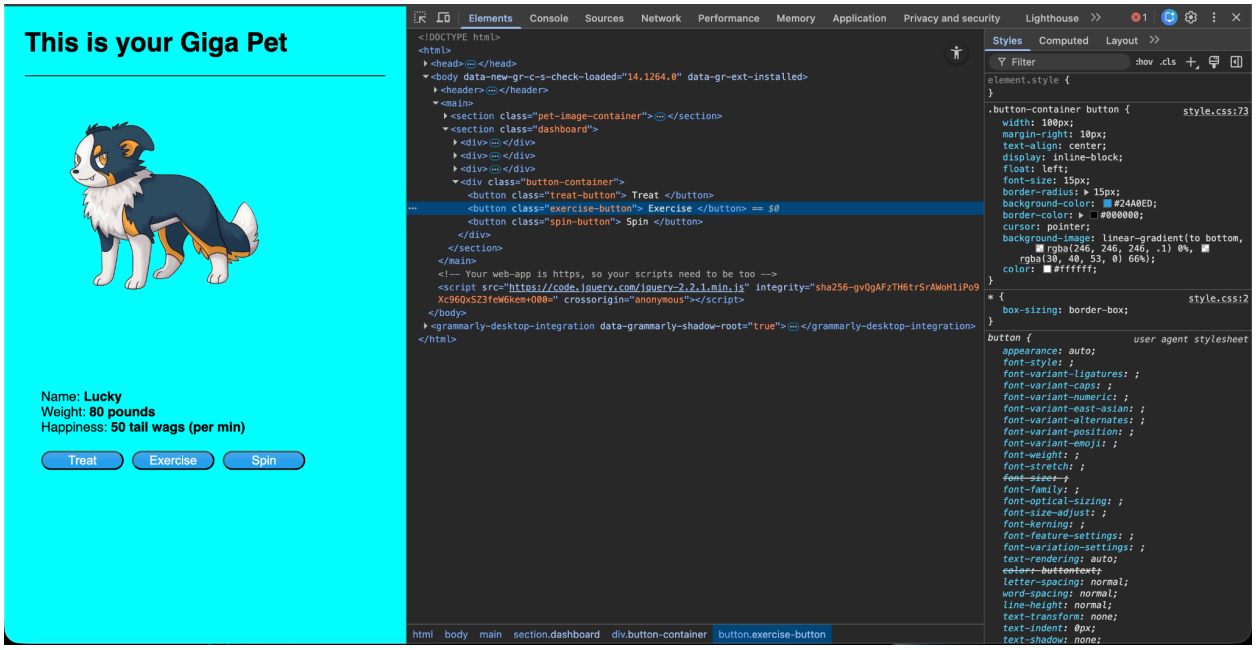
Hide a node:

To hide a node, simply select the node you'd like to hide and press "H" on your keyboard. In doing so, the node will be hidden and also be removed from the page. For example, I hid the treat button and it was no longer visible on the page.



Delete a node:

Similar to hiding a node, to delete a node fully from the tree, select the specific node to delete and then simply press “delete” on the keyboard. This will fully delete the node and remove it from the page.



However, notice how when I deleted the play button, the order of the buttons and how they were displayed changed but this was not the case for hiding the node. This is because hiding the node does not “delete” it even though it may seem like it does, all it does is simply just hide it from being visible.

Access nodes in the Console:

When inspecting a node within the DOM tree, the “== $0” indicates that node can be accessed and searched for in the console. In order to do so, simply select and element to be searched for, press esc or select the console tab to navigate to the console, and then simply type “$0.” This will bring evaluate to the highlighted node.