Tutorial/Laboratory 03

Creating Dynamic, Interactive Websites with JavaScript

1. LEARNING OUTCOMES

Upon completion of these laboratory exercises, you should be able to:

- Implement collapsible, mobile-friendly navigation menus with Bootstrap JS.
- Utilize client-side scripting with JavaScript to build dynamic, interactive websites.
- Debug JavaScript applications using the Chrome or Firefox debuggers.

2. REQUIRED SOFTWARE

- Apache NetBeans 11.2 (or later): https://netbeans.apache.org/download/index.html
- FireFox (https://www.mozilla.org/en-US/firefox/new/) or Chrome (https://www.google.com/chrome/) web browser.
- Bootstrap 4: https://getbootstrap.com/
- jQuery: https://jquery.com/

3. INITIAL SETUP

3.1 In this practical assignment, we'll continue with the *World of Pets* website that we built in previous labs. Therefore, you should first make a copy of the most recent project and rename it to Lab03 (or similar). Remember, do NOT use spaces or special characters when naming your projects, as this becomes part of the URL when you run or publish to the web server.

Also, don't forget to edit the **Run** configuration for the newly copied project, just like we did previously. Choose all the same options as before, and be sure to change **Web Root** setting to **/Lab03.**

3.2 Test the new project by clicking the green **Run** button.

In the following exercises, we will improve our website by making it more dynamic and interactive. Namely:

1. Improve the menu by implementing Bootstrap's JavaScript and the collapsible mobile menu.
2. Add event handlers so that when the user clicks on the pet images, we'll display the larger image in a popup window, rather than opening the image in the browser.

4. EXERCISE 1: INSTALLING BOOTSTRAP JAVASCRIPT AND JQUERY

4.1 In the previous Lab, we only installed Bootstrap CSS. But Bootstrap also provides JavaScript for enabling some features, such as the collapsible mobile menu. Bootstrap JS also depends on two other JavaScript libraries, Popper.js and jQuery, but the bundled version of Bootstrap includes Popper.js, so we just need to add that and jQuery. We'll use the CDN method again for both.

4.2 Copy the following code into the <head> element in index.html, just below the existing <link> element:

```
<!--jQuery-->
<script defer
    src="https://code.jquery.com/jquery-3.4.1.min.js"
    integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSFlBw8HfCJo="
    crossorigin="anonymous">
</script>

<!--Bootstrap JS-->
<script defer
    src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.bundle.min.js"
    integrity="sha384-6khuMg9gaYr5AxOqhkVIODVIvm9ynTT5J4V1cfthmT+emCG6yVmEZsRHdxlotUnm"
    crossorigin="anonymous">
</script>
```

Note the use of the **defer** attribute in our <script> element. This tells the browser to wait until the complete web page has been loaded and parsed before loading the JavaScript. Before this attribute was added in HTML5, developers typically placed the <script> element at the end of the document, just before the closing </body> tag.

Discussion Questions:

a. Does it matter when JavaScript files are loaded?

b. What are some of the ramifications of loading JavaScript before the HTML document is fully parsed and displayed?

c. What are the advantages of putting all <script> elements in the <head>?

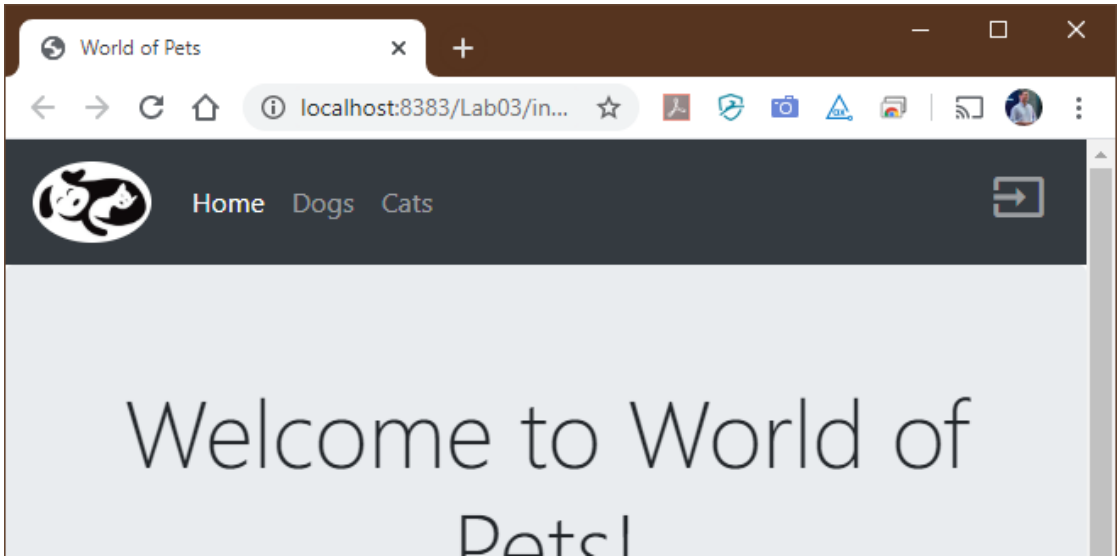
d. What does the **async** attribute do and how is it different from **defer**?

5. EXERCISE 2: IMPLEMENTING THE COLLAPSIBLE MOBILE MENU

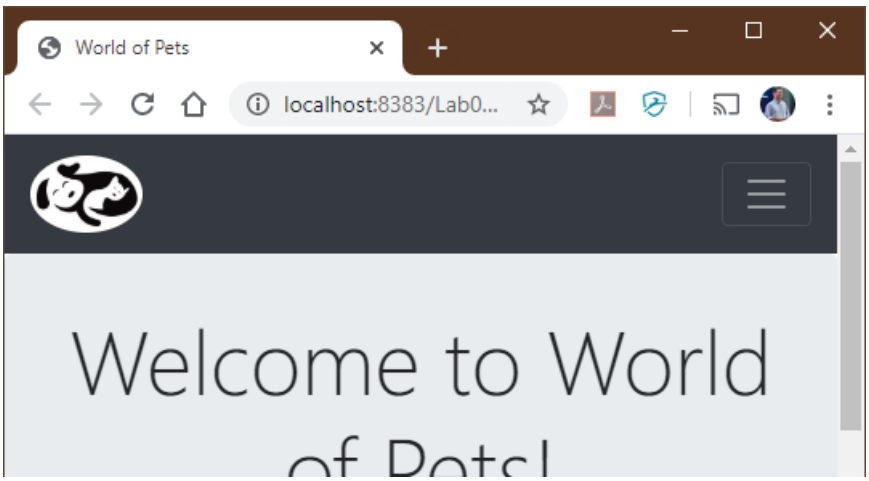
5.1 To make our navigation menu more responsive and usable on mobile devices, we'll take advantage of Bootstrap's collapsible menu feature.

Refer to the examples given at https://getbootstrap.com/docs/4.4/components/navbar/#responsive-behaviors, then Modify your existing <nav> element so that when the browser is resized past the small break point, the menu collapses to the mobile version. Example:

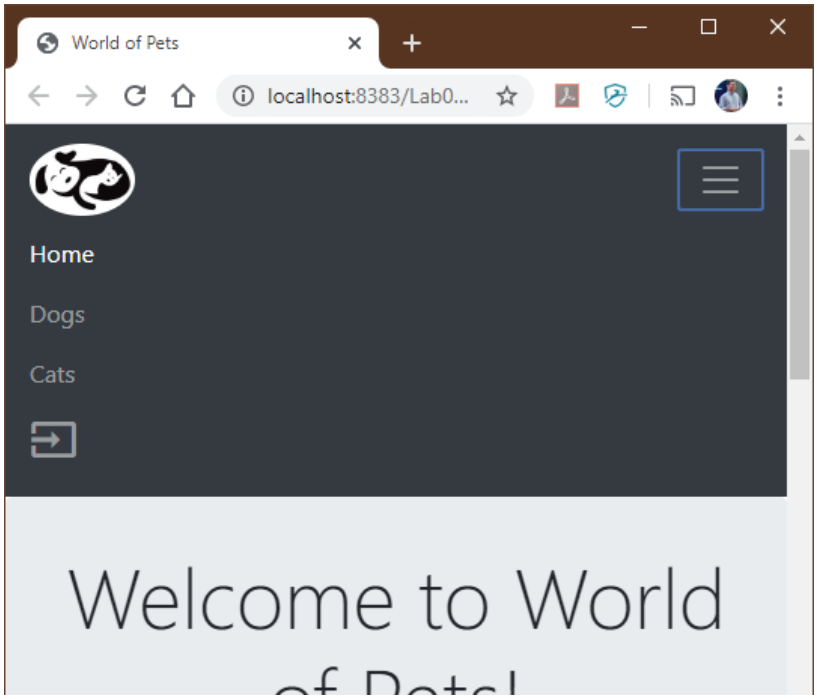
Normal menu:



Collapsed:



Mobile menu:



6. EXERCISE 3: JAVASCRIPT EVENT HANDLERS

6.1 One of the primary uses of client-side scripting is to respond dynamically to events that occur while the user is interacting with the web page. In this exercise, we'll demonstrate this by adding JavaScript *event listeners* that respond to the user clicking on the pet article images. Instead of opening the larger image in the browser, we'll display it in a popup window. Then when the user clicks the image again, the popup will close.

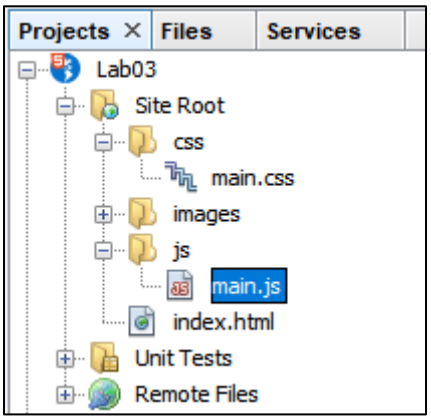
Here are the suggested steps you may want to follow:

a. Remove the existing <a> elements that wrap each <img> element.

b. Add a class attribute (e.g. class="img-thumbnail" or similar) to each <img> tag so that your JavaScript function can select only the images that should have a popup when clicked. You could have other images in the page for which you do *not* desire the popup effect, so the class attribute is a handy way to designate the ones that should.

c. Edit your external CSS file (main.css) and add a ruleset for styling the popup images. You can do this by adding a new class, say ".img-popup", which will be dynamically applied to the popup image in your JavaScript code. Add property-value pairs for styling items such as the image size, border, background, padding, position, etc.

d. Add a custom external JavaScript file (e.g. "main.js") to your project. You should put this in a sub-folder called **js** that's just below the Site Root (public_html), similar to how you added your custom CSS file previously:



Don't forget to include your JavaScript file in index.html by adding another <script> element below the existing ones:

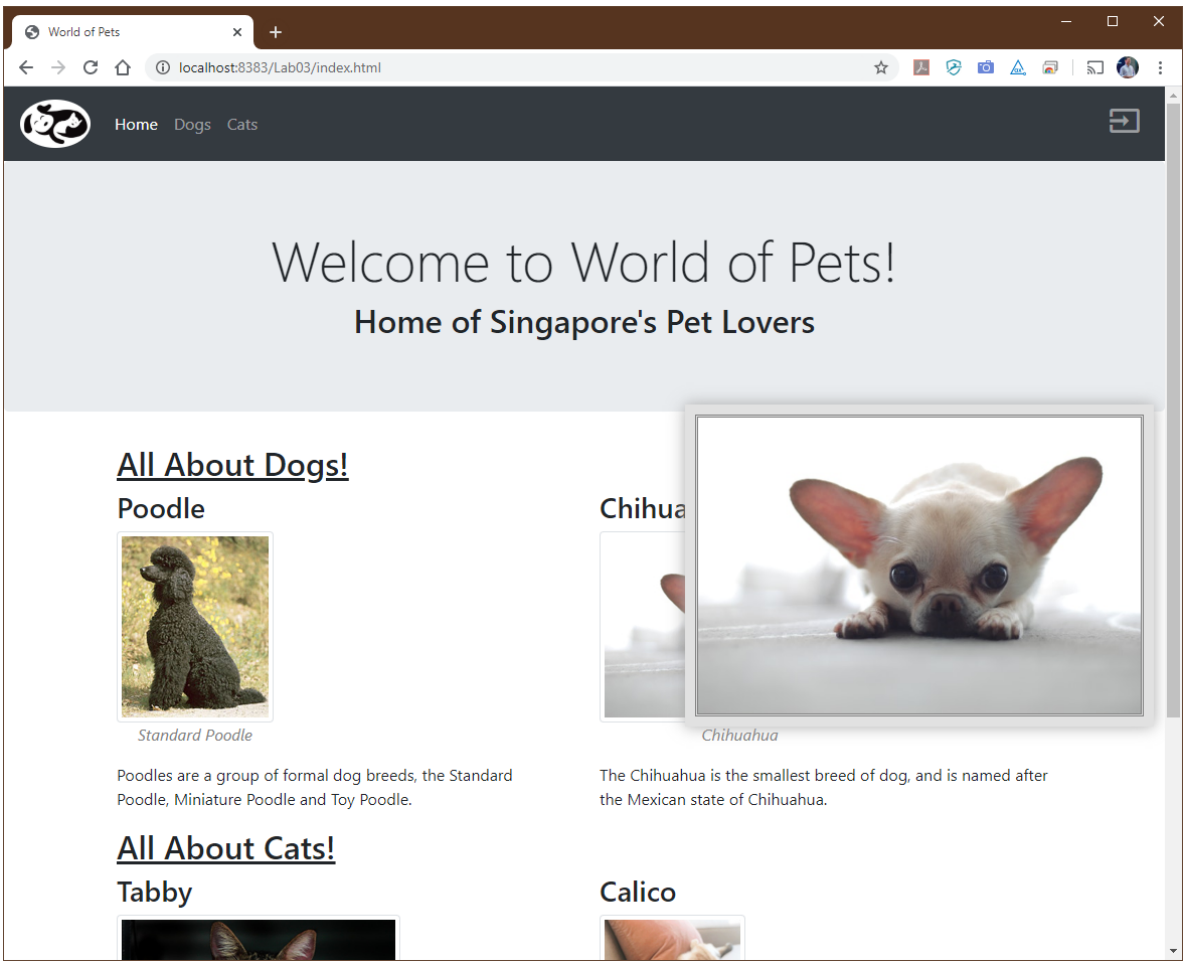
```
<!-- Custom JS -->
<script defer src="js/main.js"></script>
```

e. Edit main.js and add a "click" event listener to all elements of the img-thumbnail class. **Tips**:

- You'll want the code that registers your event listeners to be executed automatically when the page first loads. To do this, you can take advantage of jQuery's $(document).ready() method.
- You can make use of the standard getElementsByClassName() JavaScript function to get an array of all <img> elements in that class, then loop through the array to add the event listener to each one. Alternatively, you can use jQuery's event handling functions and achieve the above with much less code.
- To dynamically create the popup image, you can use the createElement() JavaScript function (a **span** element should work fine). Then use setAttribute() to set the "img-popup" class to the new element, and the innerHTML property to add the new <img> element within the .

You may refer to https://www.w3schools.com/js/js_htmldom_eventlistener.asp for more details on event listeners.

Example showing popup after clicking on an image:



There are other ways to achieve this and you are welcome to experiment.

Discussion Questions:

a. Why do we prefer to use the event listener method as opposed to the older inline JavaScript event handlers (e.g. “onclick=” attribute in the HTML element)?

a)
1. easier to use multiple functions/and all
2. security: users can put malicious code to run code in the onclick

b. What are some of the advantages of putting your JavaScript code in an external file?

neater, easier to collaborate

c. What happens if the user has disabled JavaScript or it’s not supported by the device/browser? As a web developer, how should you prepare for that?

1. turn them away; inform that they need javascript
2. limited features, but can run on the server side.
to detect if no javascript: <noscript></noscript>

7. EXERCISE 4: DEBUGGING JAVASCRIPT

7.1 For anyone wishing to make their career in software development, debugging skills are invaluable. JavaScript applications are especially prone to errors and learning how to properly debug them will save you countless hours and headaches.

7.2 For this exercise, go to https://developers.google.com/web/tools/chrome-devtools/javascript and run though the tutorial on debugging JavaScript in Chrome. Although this is specific to Chrome, the debugger in FireFox and other browsers work very similarly, and the concepts are the same.

Note: there is no submission required for the debugging exercise, however you are responsible for knowing the concepts, including:

- Setting breakpoints
- Stepping through JavaScript code
- Inspecting variable values
- Using the console

8. SUBMISSION OF LAB ASSIGNMENT

8.1 In order to receive credit for this Lab assignment, you must submit your completed work to xSiTe LMS before the end of the Lab session. To submit your work:

a. Save all files and close NetBeans.

b. In File Explorer, navigate to the location where you saved your project and right-click on the folder name, then select 'Send to -> Compressed (zipped) folder' and ZIP up your entire NetBeans project. **Note: only .zip format is acceptable, do not use .rar, .7z, or any other format.**

c. In the ICT1004 module on xSiTe, go to **Assessments->DropBox** and locate the Dropbox folder corresponding to this Lab. Click the link to open the Dropbox then hit the **Add a File** button to submit your .zip file. You may also add comments if desired. Be sure to hit **Submit** to complete your submission.

d. Remember to save a copy of your work as we will be building upon this website in subsequent Lab assignments.

9. ADDITIONAL PRACTICE

9.1 Once you've completed this Lab assignment, you are encouraged to try out the following online tutorials:

a. JavaScript: https://www.w3schools.com/js/default.asp

b. jQuery: https://www.w3schools.com/jquery/default.asp

c. AJAX: https://www.w3schools.com/js/js_ajax_intro.asp

d. JSON: https://www.w3schools.com/js/js_json_intro.asp