Interactivity with JavaScript and PHP

**Table of Contents**

[Introduction 2](#_Toc33547597)

[Review Week 4 Results 4](#_Toc33547598)

[Laboratory 5 Results 5](#_Toc33547599)

[Simple HTML, JavaScript and PHP Examples 6](#_Toc33547600)

[Hello World 6](#_Toc33547601)

[JavaScript Content Swapping 9](#_Toc33547602)

[JavaScript countUp and changeLook examples 11](#_Toc33547603)

[Create a Single Page Application — 3D App 16](#_Toc33547604)

[Single Page Web3D App with JavaScript 17](#_Toc33547605)

[Create an Image Gallery using JavaScript, JQuery and PHP 21](#_Toc33547606)

[Step 1: Set up your site 21](#_Toc33547607)

[Step 2: PHP Hook 24](#_Toc33547608)

[Step 3: Creating the JavaScript Controller 27](#_Toc33547609)

[Step 4: Creating the HTML File for Testing the Gallery 33](#_Toc33547610)

[Step 5: Create some CSS rules to style the gallery 35](#_Toc33547611)

[Step 6 the gallery results 35](#_Toc33547612)

[Integrate the Gallery into your simple 3D App 37](#_Toc33547613)

[Restyle the 3D App 42](#_Toc33547614)

[Lab 5 Results 44](#_Toc33547615)

[Lab 6 — Exploit AJAX with JQuery and JSON 45](#_Toc33547616)

[Appendix 46](#_Toc33547617)

[Appendix 1 — Lab 4 Results 46](#_Toc33547618)

[Appendix — CSS Selector IDs 47](#_Toc33547619)

[Appendix — Self-Directed Learning 48](#_Toc33547620)

[JavaScript tutorials 48](#_Toc33547621)

[X3DOM tutorials 49](#_Toc33547622)

# Introduction

This Lab 5 tutorial will focus on adding interactivity to the basic 3D App created in Lab 4. It is important that undergraduates complete this lab during week 5, so you will need to put some time in to catch up if you are behind. Further, we are now starting to get to grips with some JavaScript and soon we will use JavaScript to manipulate out X3DOM models (X3D in the HTML DOM) so you will need to put in extra time to more familiar with the technologies we are using, see Appendix — Self-directed Learning.

In Lab 5 we will create some simple JavaScript functionality that converts your Lab 4 basic 3D App into a single page application (SPA) in the form of a 3D App. Although we are using modern frameworks such as Bootstrap 4, and libraries such as jQuery, we will create this initial SPA the old fashioned way with just plain JavaScript using the **document.getElementById(id)** function to swap contents — some of you may already be familiar with this function from previous modules.

Along the way we will also implement a simple gallery interface using JavaScript, a little JQuery, a bit of PHP, and AJAX using XMLHttpRequest. “All modern browsers have a built-in XMLHttpRequest object to request data from a server.”[[1]](#footnote-1) You can use this simple gallery interface as a thumbnail gallery, but where should your images come from? An obvious choice is to delve into 3ds Max a bit more and render some photorealistic 3D images of your models — this extra work is a good way to get into the so called ‘going beyond what is taught’ marks!

So, in this lab, you will get an initial taste of a full technology stack minus the data store:

* HTML5,
* CSS3,
* JavaScript,
* AJAX using XMLHttpRequest,
* and PHP.

Where the XMLHttpRequest object is effectively used as an API call to transfer data (i.e. images to a gallery interface) between your web browser (3D App) and a web server (the ITS web server) — the images are stored on the web server.

You may, or may not use the resulting gallery interface in your assignment later, indeed you may swap this gallery code for a Bootstrap 4 gallery component instead. However, This gallery ‘hand crafted’ component if suitably extended and refined could serve as a component that gains extra marks.

Lab 5 will introduce you to the concept of browser (or client-side) scripting using JavaScript and give you a hint at the role of PHP for server-side scripting (the gallery interface).  Scripting allows you to manipulate the web page contents in ways that cannot be done with HTML or CSS.

To achieve a good level of dynamic control and interactivity on the web page you need to use a scripting language, in this lab we will focus on utilising JavaScript, which allows you to add functionality such as buttons and forms, etc., communicate with the server, make calculations, write interactive games, add special effects, and generally make your web pages smarter.

However, we won’t cover all this functionality (this isn’t a JavaScript module, after all), just the stuff that may prove useful for your assignments — you will need to discover more about JavaScript yourself to provide even more interesting functionality!

In laboratory 5 you should complete the following tasks:

1. Convert your Lab 4 basic 3D App from four web pages to one webpage with the same content using a JavaScript ‘content swapping’ function by exploiting the JavaScript **document.getElementById(id)** function, which has cross bowser support.
   * The **document.getElementById(id)** function is a very powerful JavaScript function that enables you to manipulate HTML elements in JavaScript.
   * It allows you to do something with an HTML element, e.g. a div tag, such as display the contents of that div tag or not, or change properties such as background color, font style, etc. In order to restyle a page interactively.
2. Create, and integrate into your 3D App, a simple image gallery interface, which (perhaps for your assignment) you could repurpose as a 3D object selection gallery utilising both PHP and JavaScript. Other uses could be to select larger images, perhaps displayed in a ‘light box’ — you should note, however, that this could also be done with the Bootstrap 4 .[img-thumbnail class](https://getbootstrap.com/docs/4.3/content/images/) along with a [Modal](https://getbootstrap.com/docs/4.3/components/modal/) to create a sort of lightbox, or use the Bootstrap 4 gallery component. In lab 6, you will see I used rendered images and a modal component for the gallery interface, see the **Live Feedback Site**. These principles could also be used to generate image-based buttons for selecting different camera views, and so on.
3. You should also take the opportunity in this lab to tidy up your CSS and create a JavaScript function to restyle dynamically your 3D App.
4. And, of course, we need to connect up your camera views, animation, and light buttons to their associated inline X3D code nodes, if you can’t figure this out I show you how in Lab 6 again. Remember, we showed you how to do this in Lab 3.
5. At the end of this laboratory we also suggest that you read around and experiment with JavaScript in general by trying out the resources indicated in Canvas to familiarise yourself some more with JavaScript. Some suggested resources to look at include, but are not limited to:
   * [W3C Schools JavaScript Tutorial](http://www.w3schools.com/js/)
   * [JavaScript Tutorials for Beginner's](http://htmldog.com/guides/javascript/)
   * [JavaScript for the Total Non-Programmer](http://webteacher.com/javascript/)

Having said that most of the examples we give you are sufficient to create a reasonably interactive Web3D App!

1. Further, you should investigate the X3DOM tutorials, which should enable you to appreciate how X3D is integrated and manipulated in the HTML DOM (Document Object Model) on the webpage —using JavaScript.

* [X3DOM Tutorials](http://doc.x3dom.org/tutorials/index.html)
  + [X3DOM Examples](http://www.x3dom.org/?page_id=5)

# Review Week 4 Results

You should have 4 key sets of results from the previous 4 laboratories. You can see in the Appendix the Lab Results expected for Lab 4. This is critical, you need these results for this Lab 5. If you have not finished these or achieved these results you need to correct or catch up. The [**Live Feedback Site**](http://users.sussex.ac.uk/~martinwh/3dapp/3dapp/labs/) located at (<http://users.sussex.ac.uk/~martinwh/3dapp/3dapp/labs/>) also gives you the correct results and codes so that you can compare and contrast.

# Laboratory 5 Results

Figure 1 illustrates something like how we expect your Lab 5 result to look like, i.e. a new version of the Lab 4 template, that has been converted into a single web page using the JavaScript ‘content swapper’ code developed in this Lab 5, including a theme restyle button and the new gallery. Note, this represents some nominal use of JavaScript, to gain marks in the assignment associated with a ‘deeper understanding’ you would be expected to go significantly beyond this level of Java Script coding — see assignment details on Canvas for further information.

|  |  |
| --- | --- |
| ../../../../../Desktop/Screen%20Shot%202018-02-04%20at%2021. | ../../../../../Desktop/Screen%20Shot%202018-02-04%20at%2021. |

Figure 1: Example single page layout, fluid grid and CSS styling.

# Simple HTML, JavaScript and PHP Examples

First things first, while this is not a module that teaches JavaScript and PHP per se, we do need to have a play around in this JavaScript and PHP pond to get our feet wet a bit. Let’s start by completing the examples discussed in your Lab 5 Notes — make sure you read these first.

## Hello World

In the Lab 5 Notes we show you a simple HTML, JavaScript and PHP example that renders Hello World to the HTML web page.

1. Create a new folder for your Lab 5 results. As you already know, I am using Visual Studio Code, Figure 2 illustrates the evolving site structure for all the 3D App labs so far. Here you can see that I have created an exampleScripts folder for these ‘simple HTML, JavaScript and PHP examples’. You should create the same sub folder, e.g. exampleScripts, in your folder structure to hold the examples scripts we will create.
2. Create a new file and save your file as helloworld.php in your exampleScripts folder. Note, all PHP files need to have the extension .php. If you use any other text editor, e.g. Notepad++, Sublime Text, you will save as a .php file.

Tip! By the time we get to lab 5 it is not unusual to have thrown file discipline out the window resulting in a well and truly mashed up site with lots of extraneous files and folders as you have progressed through the weekly laboratories, so now might be an opportunity to tidy up your site.

Your well-maintained site, however, should look something like that shown in Figure 3 at this point — I repeat, take the opportunity to tidy up if you have to! The LHS of Figure 2 shows my site structure in Visual Studio Code, and the RHS shows the same structure in my Windows 10 Laptop. If you have copied over your Lab 4 work to your Lab 5 folder, and created your exampleScripts folder and helloworld.php file, you should have something similar to that shown in Figure 3.

|  |  |
| --- | --- |
|  |  |
| Lab folder and file structure in Visual Studio Code | Same in Mac Finder |

Figure 2: Site structure for Mobile Web 3D Applications 2018.

Note the nicely laid out Lab 1, Lab 2 (this only contains the 3ds Max stuff), Lab 3, with my converted models, Lab 4, which we completed last week — this contains the 4 HTML pages for Home Coke, Sprite and Pepper, and we will be using parts of this code along with the associated CSS, scripts, assets, etc. for Lab 5 — I have copied Lab 4 results to my Lab 5 folder to start the conversion to a SPA based 3D App). And finally, the new lab 5 with the helloworld.php file just created.

1. Recalling the example in the Lab Notes 5 document, you can write the following code, Figure 3, to render hello world:

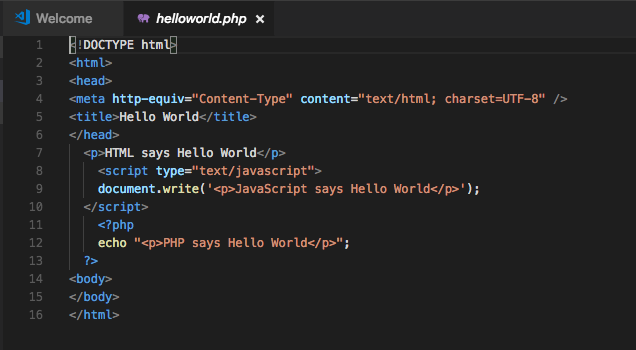


Figure 3: Hello world in HTML, JavaScript and PHP

1. When you have finished writing the code, save it and copy the file from your local directory to your public\_html directory somewhere so that it is live on the web, e.g. public\_html/3dapp/lab5/exampleScripts, or something like that.
2. Now run the helloworld.php file. How do you do this? Can you drag the file onto a web browser and does it run then? You should find it won’t run because it is a PHP file, and a PHP file needs a server environment to run, which is also PHP enabled. You might see the code rendered, or you might get asked to download the file depending on where you are trying to run the code. So, you need to check it runs by typing its URL into the web browser
3. Open up a web browser, e.g. Chrome, and enter the URL where you have placed the code, for example on the ITS Web or test it on a localhost:

* http://localhost:8888/3dapp/lab5/exampleScripts/helloworld.php
* <http://users.sussex.ac.uk/~martinwh/3dapp/3D_Apps_2019/lab5/js/exampleScripts/helloworld.php>

You can also comment out the PHP part, and save the file as an HTML file. Figure 4 shows the output.

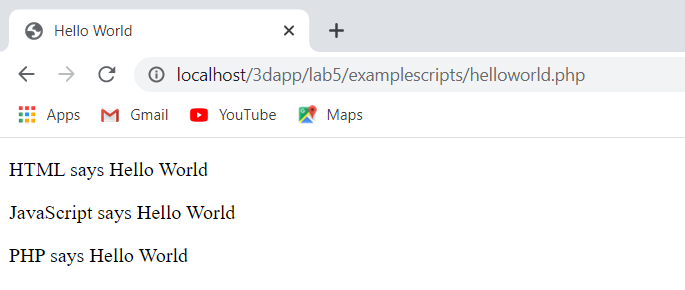


Figure 4: helloworld.php file rendering Hello World with HTML, JavaScript and PHP in Google Chrome on a Mac in a localhost

Next we will create a simple JavaScript swap function using the document.getElementById(id) function to create a single page 3D App.

## JavaScript Content Swapping

If you check out the Lab 5 Notes, we discuss the concept of using JavaScript to swap content using the document.getElementById(id) function. The getElementById() method accesses the first element with the specified id. This selector id is the id that you apply to a div tag, for example, you might create a div tag something like:

<div id=”coke”>Your coca cola web page content goes in here</div>

In which case, the id is ‘coke’. Let’s try out the actual example we illustrated in the Lab 5 Notes:

1. Create a new HTML file (note we are not using any PHP code now) and enter the code you were given in the Lab 5 Notes, however let’s make it a bit more compatible with your basic 3D App from Week 4, see Figure 6 for the code you need to enter:

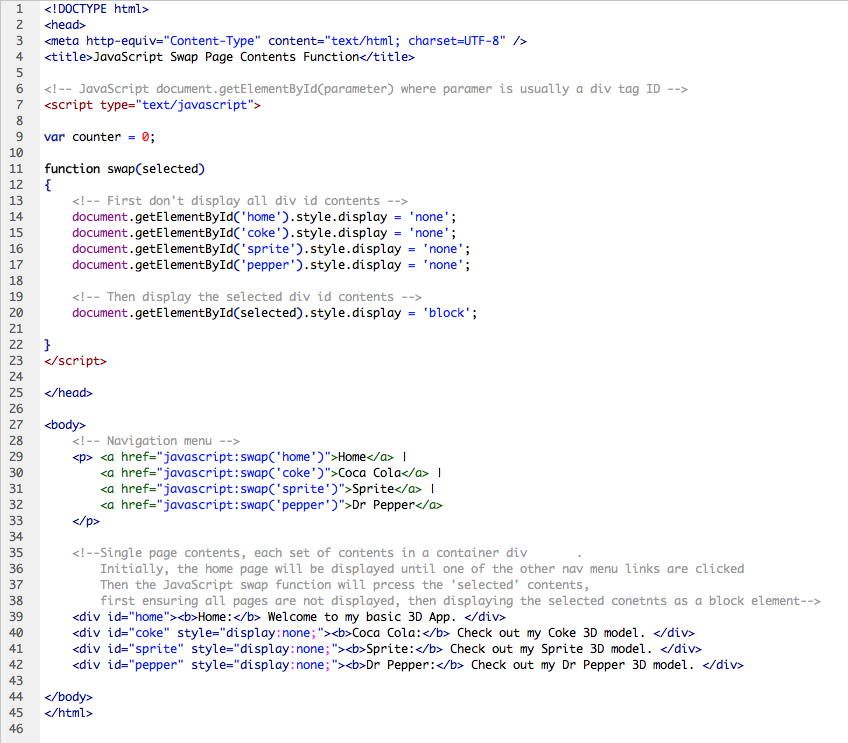


Figure 5: JavaScript content swap example

1. Once you have this code entered and saved you can view it in a web browser. For those of you who are awake, you will notice the counter variable is not used yet. Because this is an HTML file, you can simply drag it onto a web browser, but it is better to get into the habit of testing on a server. So, copy it to your exampleScripts folder (or to whatever folder structure you have created) as before and access it that way. Figure 6 illustrates the results — as you can see, I am using a localhost to test.

|  |
| --- |
|  |

Figure 6: JavaScript content swap results

1. What you have essentially created here is a menu for the Home Page, Coca Cola, Sprite and Dr Pepper page contents, but instead of linking to other web pages you are selecting new content by ‘swapping’ (i.e. enabling the selected by id) div tags. This is achieved by blanking out all the div tags by setting **style.display = ‘none’**; and then selecting the required content by passing the id associated with the required div tag to the **document.getElementById(selected)**.**style.display = ‘block’**; which then displays that div tags content as a block element. It should be obvious that this swap function will be very useful for creating a single page application (SPA), i.e. your basic 3D App.

Let’s try some other simple examples.

## JavaScript countUp and changeLook examples

1. Create the other two examples discussed in the Lab 5 Notes, but don't bother to create a new file, just build the examples into the content swap file (jsContentSwap.html). The code is illustrated in Figure 7.
2. The countUP example could be adapted to change a texture or colour on your 3D model — you will find examples that do just this on [www.x3dom.org](http://www.x3dom.org), or you could adapt it to cycle through web page styles or brands in combination with the changeLook function.
3. The changeLook function is quite interesting — you can adapt this concept to change the them.
4. Figure 8 illustrates the result.
5. Test the countUp and changeLook functions work, you should see something like that illustrated in Figures 9, 10 and 11 after clicking ‘here’ several times, and clicking the Click me! Button to restyle the page.
6. You can see that the text was re-styled to ‘pink’, and background changed to ‘grey’, etc.
7. Remember, as discussed in the Lab 5 Notes, that JavaScript loses its context (the changes) when you refresh or reload the page — it does not maintain state. Refresh the page to return to the original style, etc.
8. You can apply these simple scripts to your basic 3D App, and you should do some more research on JavaScript to see if there are any other interactive functionality that might prove interesting in your final 3D App.

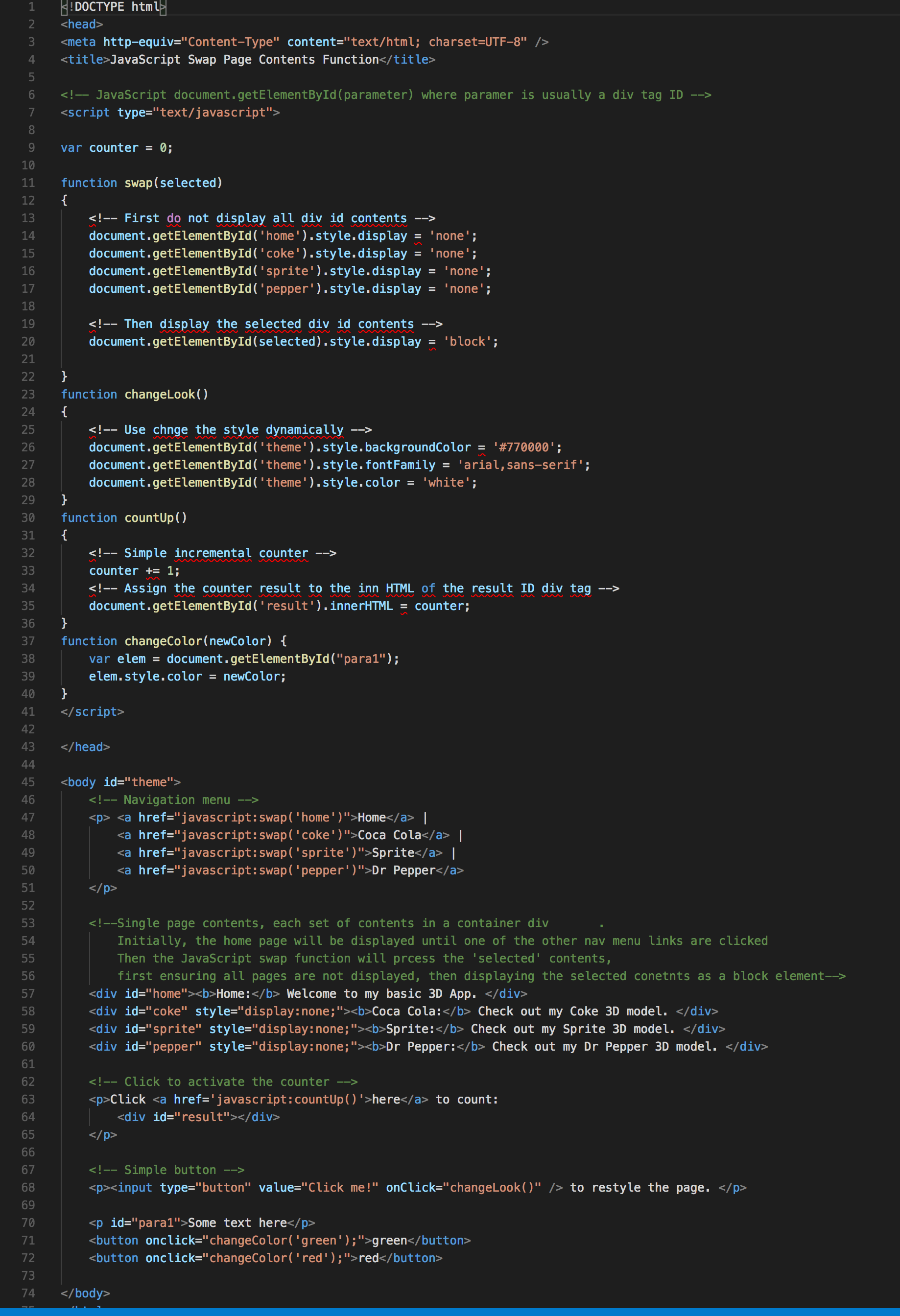


Figure 7: Adding the JavaScript countUp and changeLook function

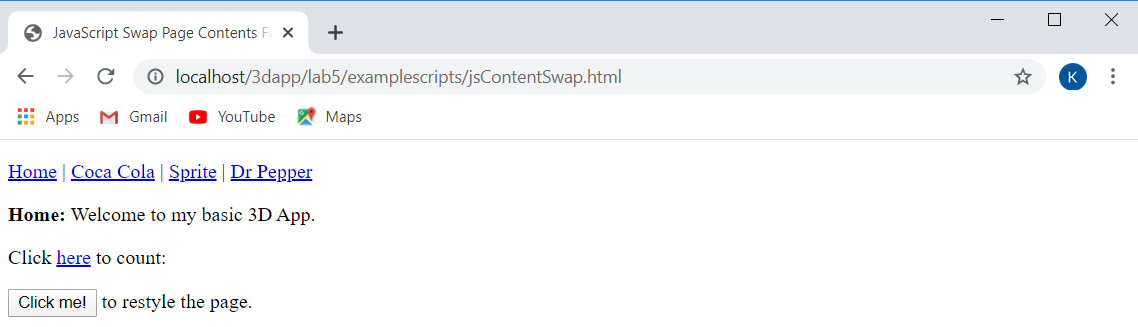


Figure 8: The results showing the Home page, click on the other pages links to see the other pages

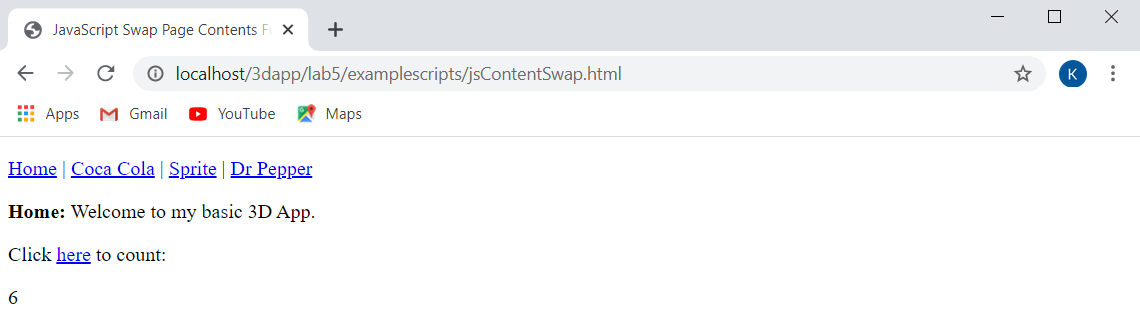


Figure 9: The count clicked 6 times

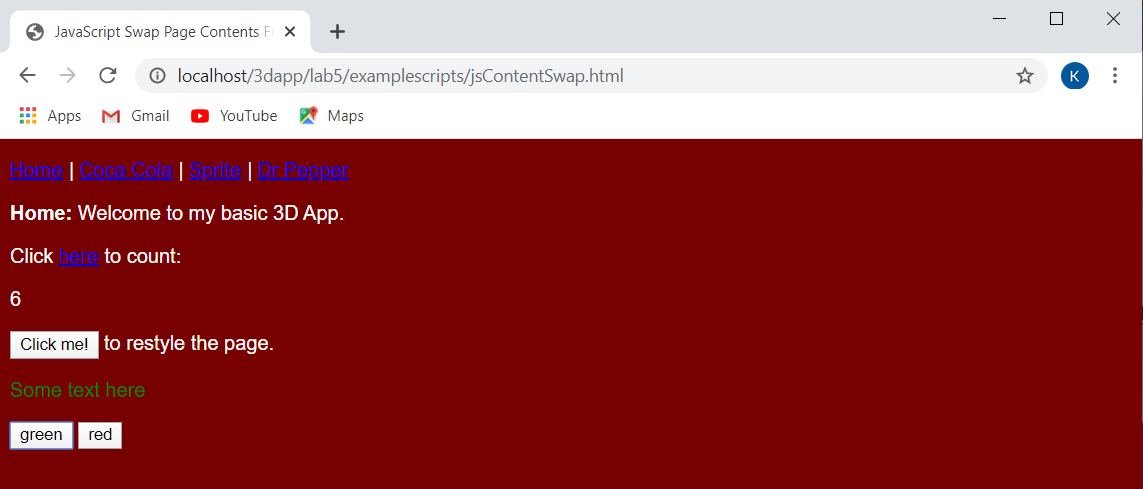


Figure 10: Clicking on the ‘Click me!’ button to change the page style

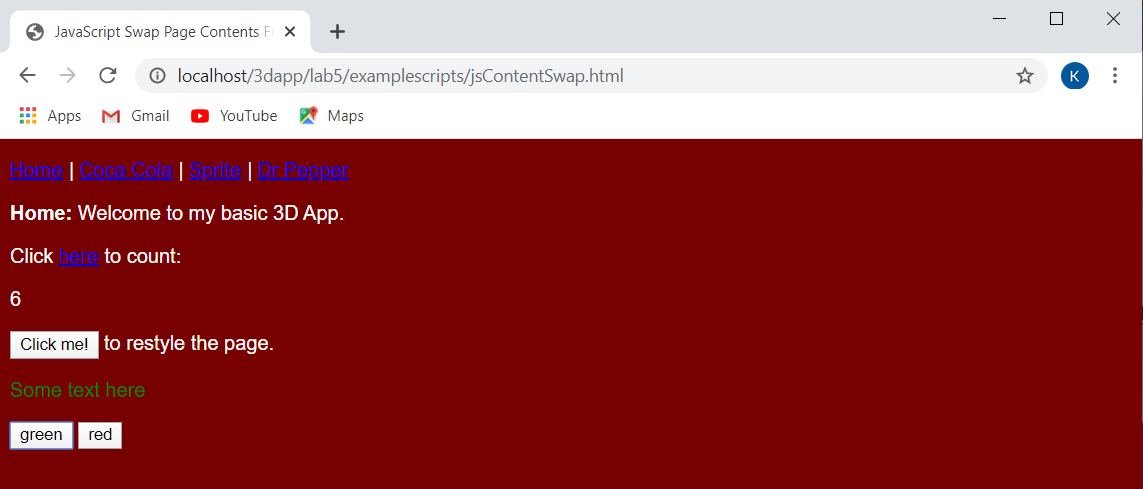
You can add other simple JavaScript functions, for example, try changing the text color, see 

Figure 10, which also shows ‘**Some text here**’ after clicking the green button. Now that you have a good understanding of these simple JavaScript functions, you should adapt them modify to your basic 3D App from Lab 4. Note, you are expected to show a deeper understanding in your assignment that show you have significantly gone beyond these nominal examples.

Now that you have finished these scripting experiment, crste a scripts folder move your exampleScripts folder into your new scripts folder, you might like to move your ‘js’ folder to ‘scripts’ to signify you don’t just have javaScript code in that folder, and as such create a php flder under the scripts folder too. You will also need to update your paths in your html files.

Check your Lab 4 work copied to your Lab 5 with the new scripts structure still works.

# Create a Single Page Application — 3D App

Now that we have a good idea of how these simple scripting functions work, let’s apply them to update our Lab 4 results. We will write JavaScript and PHP functions to implement:

* A single page application (SPA) design
* A new image gallery
* A restyle option

When you have finished, your 3D App will look something like that shown in Figure 13, based on Bootstrap 4.4. Note there is no pepper.html file, this will be 3D App based on a SPA!

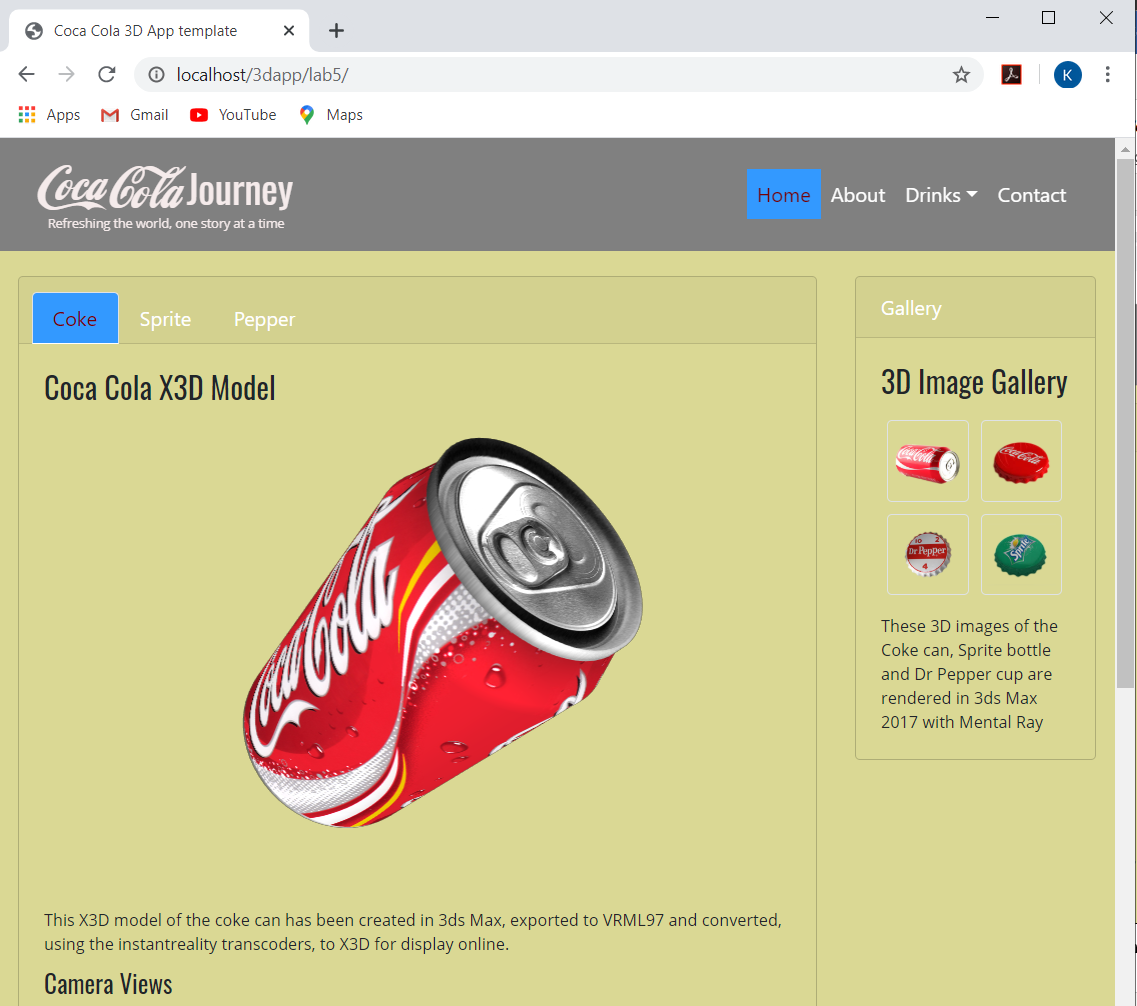


Figure 11: Note the single web page denoted by the index.html URL

## Single Page Web3D App with JavaScript

Now, I am not going to go through this step by step, because we just went through the underlying code needed in the last section. Instead, I will offer some pointers, while at the same time I will also point out that while JQuery probably has more efficient code to do the same, JQuery is really an abstract on top of JavaScript.

However, putting JQuery aside for the moment, you should now apply the simple JavaScript functions learnt above to convert your basic 3D App to a single page version and add a colour theme. To get started on re-developing your basic 3D App with a single page using the document.getElementById(id) function and methods you will:

1. We need to create 4 new fluid grid layout block elements with associated selector ids: home, coke, sprite and pepper contents — note that selector IDs have to be unique. See Appendix — CSS Selector IDs for an explanation on the use of selector IDs.

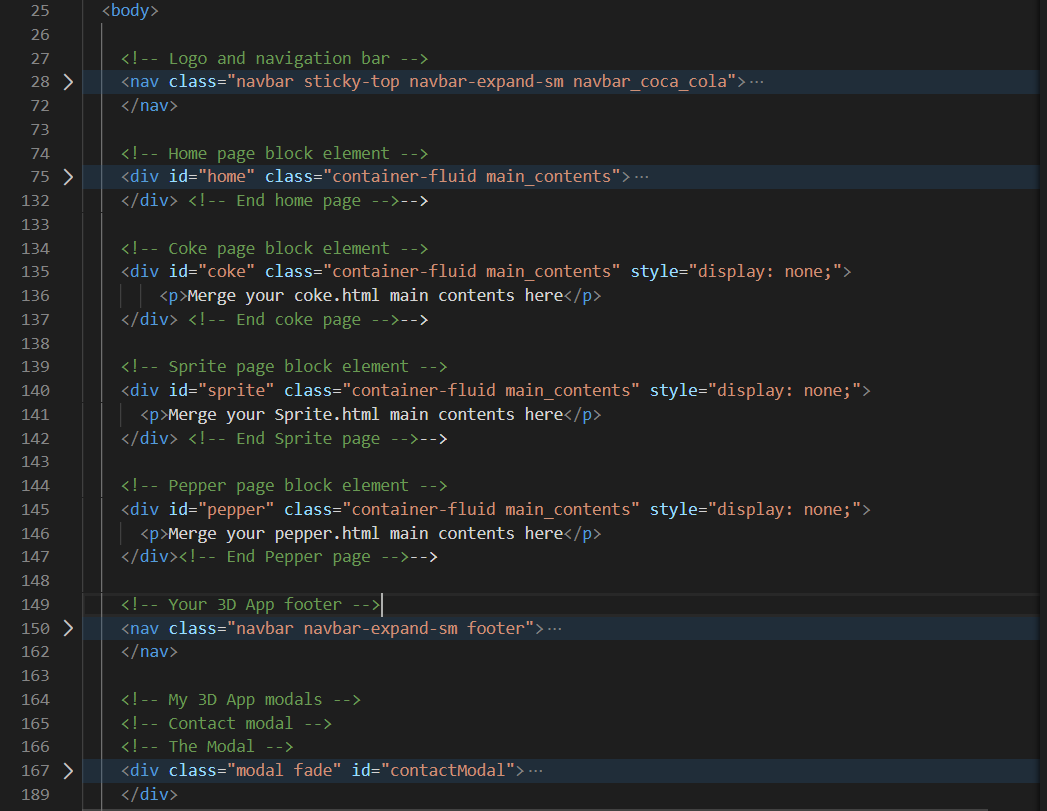


Figure 12: New HTML5 layout for the SPA to incorporate home, coke, sprite and pepper contents.

1. The home page contents block element already exists, just add the home id. Then created new block elements for the Coke, Sprite and Pepper contents. The structure should look something like that shown in Figure 12.
2. * Test one page at a time as you create the new contents
3. You can look at the **Live Feedback Site** to get an idea of what the result looks like, see Figure 13.

|  |  |
| --- | --- |
|  |  |
| Bootstrap 4.3 2019 version | Bootstrap 4.4 2020 version |

Figure 13: Home page, i.e. index.html

At this stage, if you have the structure right your Bootstrap 4 version should show an index.html as the home page — only one page now. Don’t forget to copy over the X3D libraries into the index.html page, or you won’t see your 3D!

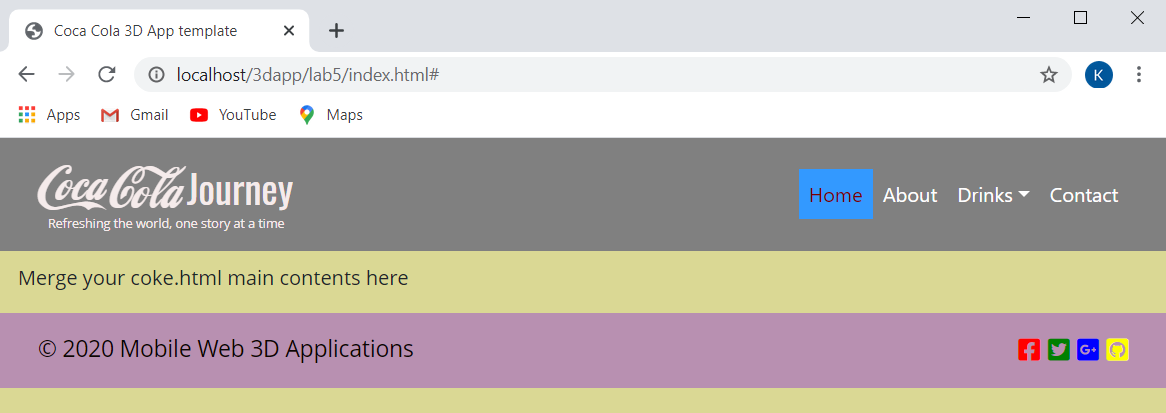


Figure 14: The sprite contents, note index.html. The coke and pepper should be the same

If you get stuck you may examine the **Live Feedback Site**, have a look at 2019 and 2020 Bootstrap versions. But, note these URLs will show the final result, not partial results

Ok, so you should now have a single page application of your 3D App using Bootstrap 4 components, make sure this works properly before you proceed to the next stage. Once it works, you can remove your coke.html, sprite.html and pepper.html files from this lab 5 folder.

You should now only have the index.html file and your Lab 5 folder should look like that shown in Figure 15. You can check the **Live Feedback Site** for more details. Note all extraneous CSS and JS files have been removed and all libraries installed rather than using CDNs. We have a single Sindex.html because we have just created a single page application (SPA) for our 3D App. Make sure you test it and it works ok.

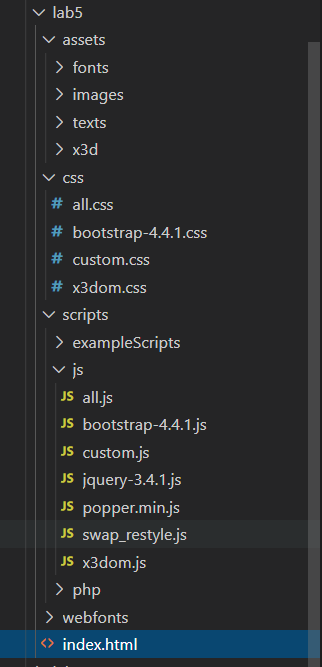


Figure 15: Your Lab 5 structure so far

Next, let’s start to add the image gallery and add some more functionality.

# Create an Image Gallery using JavaScript, JQuery and PHP

In this tutorial section, you will be creating a dynamic image gallery interface that you may find useful for your 3D App. Image galleries can use PHP (a server-side scripting language) to gather image file paths from folders, and by using some JavaScript (and/or JQuery) at the front end we can effectively generate a gallery interface dynamically. In this tutorial, the PHP server-side code is provided for you. PHP is also already enabled on the IT Web Server for students.

Your goal is to build a client-side script that reads image files from a folder dynamically, by invoking a PHP script on the server-side, to generate thumbnails on the client-side. These thumbnails can then be used as a gallery interface. You can use the gallery interface for a number of functions as previously discussed.

The gallery interface code, you will end up with, will be relatively sparse and could be improved in many ways. It serves as an introduction to PHP and provides an opportunity for you to demonstrate deeper understanding by improving and refining the code to produce a better gallery interface.

This section of the tutorial was originally developed using our own CSS rules for styling the gallery thumbnails, however since we are now using Bootstrap, we have eliminated much of the hand-built code for creating thumbnails. Thus, we now use the [Bootstrap Images](https://getbootstrap.com/docs/4.3/content/images/) and its .img-thumbnail and .img-fluid classes. The actual PHP and JavaScript code will stay largely the same though.

The code for this tutorial will be divided between four files:

* *hook.php* — a PHP script that reads image paths from a folder of your choice.
* *gallery\_generator.js* — JavaScript/JQuery code to generate the client-side gallery HTML code
* *index.html* — the gallery web page.
  + After this part of the tutorial, you will be expected to integrate the code into your Lab 5 results.
* *style.css* — the cascading style sheet of the gallery.
  + And, of course, you will need to integrate the CSS into your Lab 5 custom.css taking into account media queries.

Doing it this way makes for an independent piece of code, effectively you are creating your own Bootstrap 4, JavaScript, and PHP gallery component, which you can build upon. Alternatively, you could just as easily use a pre-built piece of gallery code, e.g. the Bootstrap 4 carousel component.

## Step 1: Set up your site

We’ll start by creating a new development sub-directory in your Lab 5 web space called gallery, e.g. …/lab5/gallery — I happen to be using …/lab5/gallery because I am managing all my previous years lab codes in Lab 5. To test the gallery, choose four images (associated with the topic of interest) from the Internet and copy them in a sub folder gallery/assets/images. I choose the images illustrated in Figure 16, later on you can change these for 3D images rendered in 3ds Max, for example.

|  |  |  |  |
| --- | --- | --- | --- |
| Macintosh HD:Users:martin:Dropbox:web3d_app_2015_labs:web3d_mobile_app:lab6:gallery:images:coke_top.jpeg | Macintosh HD:Users:martin:Dropbox:web3d_app_2015_labs:web3d_mobile_app:lab6:gallery:images:sprite_top.jpeg | Macintosh HD:Users:martin:Dropbox:web3d_app_2015_labs:web3d_mobile_app:lab6:gallery:images:not_needed:dr_pepper_top.jpg | Macintosh HD:Users:martin:Dropbox:web3d_app_2015_labs:web3d_mobile_app:lab6:gallery:images:coke_can.jpg |

Figure 16: Some random themed images borrowed from the Internet

Tip! Try and choose 4 images roughly the same size and proportions, it doesn’t matter too much the actual size — adjust them in Photoshop if you have to. The following step is optional:

**Option 1:** In the past, you would have had to create thumbnails — you can find plenty of tutorials online show you how to do this. When I created these thumbnails, I happened to use the mac Preview App, this app allowed me to set a custom size to get roughly what I needed to start with — you would have then used CSS3 rules to scale down the thumbnails for desktop, tablet and mobile sized screen for the fluid grid layouts. However, because we are now using Bootstrap, you don’t actually have to create the thumbnails, the Bootstrap .img-thumbnail class will do this for you.

**Option 2:** However, I did further prepare these images as transparent images in Photoshop, that way CSS can set the background colours effectively — Tip! In Photoshop zoom in on your image and use the quick Selection Tool and Add/Subtract Selection to select the edge of your image of interest, and then select the Edge Refine adjusting parameters to get a good selection and save to a new layer, which effectively removes the background. If your images are not exactly the same size you can scale them up and down with the Image Size (to get all your images roughly to the same scale and then use Canvas Size to set the final size of the image (I chose 240x240) in Photoshop. Then save the images as png — note jpeg does not support transparent images — you can always scale/crop to get all your images thumbnails the same size.

Now, because I already have these images prepared, I will use them anyway. But, note, mine are effectively all small images already massaged into a size for thumbnails. You can choose a different set of 4 coca cola themed images, and the [Bootsrap Images](https://getbootstrap.com/docs/4.3/content/images/) content responsive and thumbnail images classes will give you what you need to create the gallery styling without the effort outlined in the option 1 above, though you still may want to better prepare your images, option 2.

So, here’s my 4 thumbnails after preparation in Apple Preview and Photoshop, you should choose whatever images you like — each image is 240x240 pixels, transparent png, see Figure 17.

|  |  |  |  |
| --- | --- | --- | --- |
| Macintosh HD:Users:martin:Dropbox:web3d_app_2015_labs:web3d_mobile_app:lab6:gallery:images:coke_can_tn.png | Macintosh HD:Users:martin:Dropbox:web3d_app_2015_labs:web3d_mobile_app:lab6:gallery:images:coke_top_tn.png | Macintosh HD:Users:martin:Dropbox:web3d_app_2015_labs:web3d_mobile_app:lab6:gallery:images:sprite_top_tn.png | Macintosh HD:Users:martin:Dropbox:web3d_app_2015_labs:web3d_mobile_app:lab6:gallery:images:dr_pepper_top_tn.png |

Figure 17: Images prepared in Photoshop

When you have created your images place them into an assets/images folder in your gallery folder, see Figure 18.

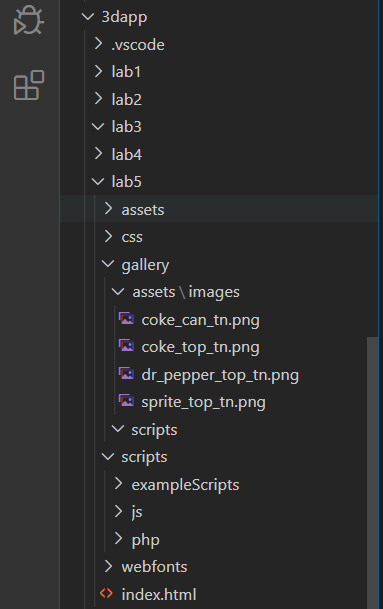


Figure 18: The gallery> assets>images> thumbnail images.

Note, that the gallery images are in the assets folder inside the gallery folder and not the main 3D App assets folder. Further, you need to also create another scripts folder inside your gallery folder because we are building the gallery as a self-contained code component that we will integrate into the 3D App later.

Note also that my images are already prepared as thumbnails from previous years, you don’t need to do this, because the Bootstrap 4 .img-thumbnail class will prepare this for you. Instead, just put your png gallery images in the indicated image folder.

## Step 2: PHP Hook

Ok, let’s get on and write the code, we are going to have a play with PHP. This PHP hook file will create a string of file paths that will be sent to the HTML code asynchronously as an XML packet using an [XMLHttpRequest object](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest).

There are more modern ways to asynchronously communicate with a server and fetch objects, e.g. the [Fetch AP](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch) and [JQuery ajax()](https://api.jquery.com/jquery.ajax/). However, in modern web-development XMLHttpRequest is still used for three reasons:

1. Historical reasons: we need to support existing scripts with XMLHttpRequest.
2. We need to support old browsers
3. We need something that fetch can’t do yet, e.g. to track upload progress.

Further, as already mentioned, this presents an opportunity for you to modernise the gallery code for those deeper understating marks in the assignment.

The hook.php file is provided for you, but you should have an understanding of how it works — see the code comments. As an alternative to using an XML packet, we could use JavaScript Object Notation (JSON: <http://www.json.org/>) packets — Lab 6 will show you how to use AJAX with JSON to read data from a file.

For those new to PHP, it is important to note that this language has no specific object casting as all variables begin with a dollar symbol ($) and a variable name — see [w3schools.com PHP 5 variables](http://www.w3schools.com/php/php_variables.asp). So, for example, you do not need to specify an array as being of type “array”— as you do with JavaScript. In the w3schools PHP 5 variables example given in the link above, if you use $txt = “Hello World”, you don’t need to declare that the variable $txt is a string, or that for $x = 5 that the variable $x is an integer, and so on. Figure 19 shows the PHP Hook code you will need create for your image gallery.

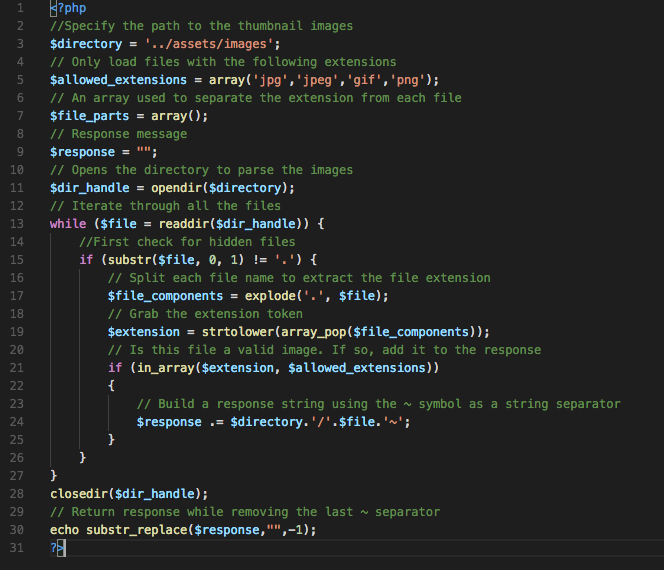


Figure 19: The hook.php file

The hook.php file begins with several variable definitions as follows, see Figure 20:

* *directory* — stores the image path/folder name in relation to your gallery web space. In this particular example, if hook.php were in the /gallery root folder, your images would be placed in /gallery/assets/images/. We can see that the ../ takes us up a directory level to the assets and scripts folder level, we then dive down into the assets/images folder to find the image files.
* *allowed\_extension* — stores an array of allowed file extensions. We want to identify only the image files in a directory so that any other file formats will be ignored by the gallery’s hook.php file. You will be able to test this feature by adding junk files (e.g. a blank .txt file).
* *file\_parts —* variable used to verify whether a file path has a valid image extension. This array splits the filename using the dot. For example, image.png will be stored in the array as {“image”,”png”}. The last element in the array will always be the extension.
* *response —* will store the server-side reply in the form of a string.
* *dir\_handle —* this isa very important PHP object that opens a directory handle to run a script to parse the image folder’s contents. It is paired with the closedir().

Copy the code into a blank file and save it as hook.php. Use Visual Studio Code again, see Figure 20. However, be careful that:

1. You place your hook.php file in the /gallery/scripts folder and **NOT** in the gallery/assets/image subfolder.
2. You save it as hook.php and **NOT** hook.php.txt.



Figure 20: Declare the PHP variables for the hook.php file.

The next part of the hook.php file is a while loop that iterates through the image files to build a server response. The server response is a string representing a list of file names. Our returned response message structure will look as follows:

../assets/images/coke\_can\_tn.png~../assets/images/coke\_top\_tn.png~../assets/images/dr\_pepper\_top\_tn.png~../assets/images/sprite\_top\_tn.png

That string will be tokenized by the JavaScript code using the tilde (~) symbol as a separator. Now, let’s look at the iteration code that builds that string. We need a while loop that executes once for every file in the /images folder. Each file name consists of a name value, a dot and an extension. We define *file\_components* as an array that splits each file name using the dot separator. All extensions are converted to lowercase. If the extension of a file name exists in the *allowed\_extensions* array, the file name path is added to the string. Finally, a tilde (~) symbol is concatenated at the end. Unlike JavaScript, PHP uses “.” instead of “+” symbols for string appending (concatenation). Figure 21 shows the PHP code needed complete with comments — to see what each individual PHP function does, you will need to explore further by looking up the PHP function, e.g. check out the [PHP opendir()](http://www.w3schools.com/php/func_directory_opendir.asp) and [PHP readdir()](http://www.w3schools.com/php/func_directory_readdir.asp) functions on w3schools. You can also play with the [explode()](http://www.w3schools.com/php/func_string_explode.asp), [strtolower()](http://www.w3schools.com/php/func_string_strtolower.asp), [array\_pop()](http://www.w3schools.com/php/func_array_pop.asp) on w3schools.



Figure 21: The while loop code used to iterate through the images in /assets/images folder

As previously mentioned, the server response is stored in the *response* variable. After the while loop has executed, we close the directory handler. We subtract the last tilde from the *response* string using the PHP substr\_replace function and echo the result, see Figure 22.

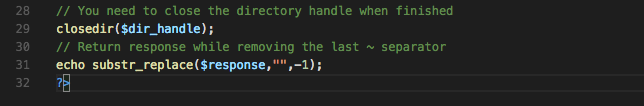


Figure 22: Close the directory and return the response string

Your combined hook.php code should look like that shown in Figure 19 above.

If you run the *hook.php* file, the result should look as follows, see Figure 23. In our example, we have the 4 images mentioned above:

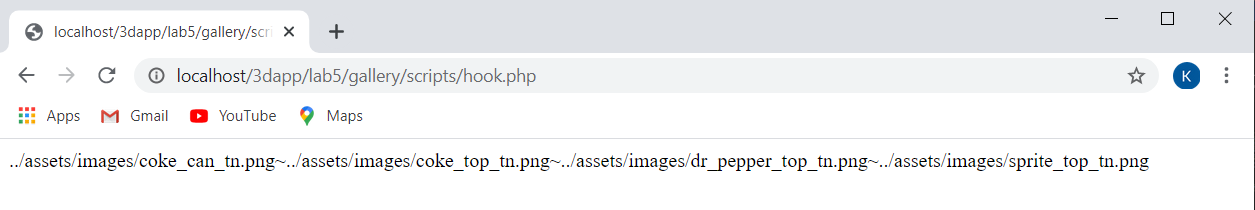


Figure 23: Result from running the hook.php file

So, now we know that the PHP hook is working, we need to develop an AJAX (Asynchronous JavaScript and XML) based call to use this PHP code.

## Step 3: Creating the JavaScript Controller

We will now write some JavaScript to generate the gallery dynamically. We will also use the [JQuery $(document).ready()](https://learn.jquery.com/using-jquery-core/document-ready/) method to pause our JavaScript until the HTML DOM is ready; we will discuss this function later.

Make a new file and save it as *gallery\_generator.js* in your *gallery*/scripts folder. The controller will create an AJAX request using XMLHttpRequest to grab the image data form the web server and parse it with the hook.php file, and generate the required HTML syntax. We will be creating a small JavaScript function that uses an [XMLHttpRequest Object](http://www.w3schools.com/xml/dom_http.asp) to communicate with the server.

We will declare a new XMLHttpRequest object as one of four JavaScript variables, see Figure 24, that we will need in our dynamic gallery generator. Note that each JavaScript definition is preceded by “var”, unlike with PHP.

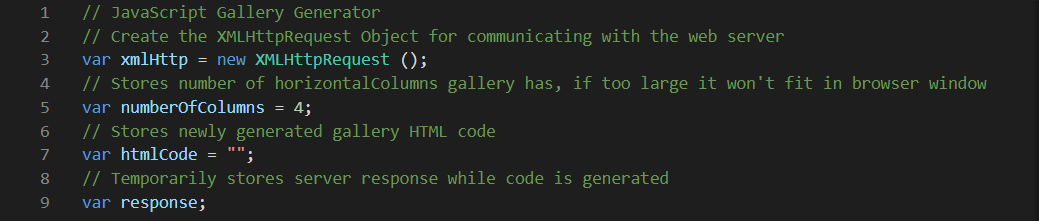


Figure 24: Declare some useful variables as described above

What are these variables used for?

* *xmlHttp* — as mentioned, will represent the object communicating to the server. We will call the function above in this variable’s definition.
* *numberOfColumns* — this is a simple integer that stores the number of horizontal columns the gallery will have. This is a parameter you can customize. However, if the number is too large, the gallery will be too wide and will not fit on your browser window.
  + For initial testing we are going to temporarily build the gallery into an HTML table
  + But, because tables do not work well in responsive web design we will redesign this gallery to function in a div tag later, when we integrate it into your Lab 5 3D App.
* *htmlCode —* variable used to store the newly generated gallery HTML code. This variable is instantiated as an empty string.
* *response —* this variable is also a string and is used temporarily to store the server response while the code is being generated.

Next, we will define a jQuery function: [$(document).ready(function()](https://learn.jquery.com/using-jquery-core/document-ready/), which is called automatically when the page loads. This function will do all the code generation. Although, code can be dumped in the file without a jQuery function, it is good practice to enclose the code in the $(document).ready(function(), see Figure 25. The reason for this is that JavaScript does not execute until all the HTML DOM has been evaluated. A web page can’t be manipulated until the document is ready.

Also, later on, in Lab 6 you will use jQuery’s $.ajax() method, more precisely the short version [.getJSON()](https://api.jquery.com/jquery.getjson/) and the [$(document).ready(function()](https://learn.jquery.com/using-jquery-core/document-ready/) to make AJAX request to PHP methods. What this means, is that, in effect we could write this gallery code in a different way, perhaps more efficient and abstract way. But, I will leave you to ponder that when you are looking back from Lab 6 — I’ll give you a clue now though, instead of XMLHttpRequest returning an XML object, you would return the image file names as a JSON object with the .getJSON() method. For now, let’s continue with the current method.

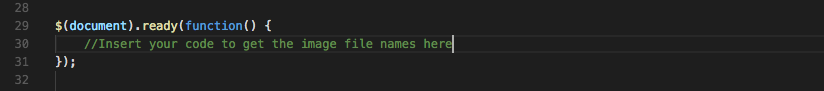


Figure 25: The JQuery $(document).ready() function that encapsulates your handler code

So, this function only executes when the HTML DOM is ready. The code inside this is often referred to as a handler. Inside the .ready function (your handler) we will define a variable called *send*. It stores the PHP file name (hook.php) so that the JavaScript code may access it. Note, if you saved your file in a different folder, you should provide the relative path.

Using the object *xmlHttp* we will interrogate the server to grab the image files. We use the .open function call and we pass the parameter “GET” and the server file destination *send*. The *.send(null)* function call is required to tell the server that the client has no outgoing message. *xmlHttp.onreadystatechange* links an embedded function asynchronously to the status of the ActiveXObject. Effectively, if the status changes (when the server replies) the function gets called. Inside the embedded function we check the status of the *xmlHttp* object. If the status equals to *4*, the server has replied. Your code should look like that shown in Figure 26.



Figure 26: Interrogate the server and grab the image file names

The HTML code generator will be added in the ‘if statement’. For testing purposes, you can add an alert box that pops up when the server replies successfully. The test code will look as follows, see Figure 27. You can see that we are only flashing up an alert box with the response. We will comment it out after testing.

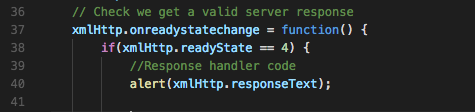


Figure 27: Check that the server response is valid

Now, as I update this tutorial every year I essentially know that it works so I am not actually going to test at this stage, but I will leave the alert box in. If you want, you may delay writing the response handler code for a minute and implement an HTML page to test the code so far, but equally you may as well crack on.

Next, we will add the response handler code previously mentioned. This is a three-step procedure.

1. First, we take the server response using *xmlHttp.responseText* and tokenize it using the tilde (~) symbol. The resulting array is stored in *response*.
2. Second, we start building the HTML code and append any created tags to the *htmlCode* variable. As mentioned above, we will build the HTML as a table, and later redesign into a div tag
3. We need put the new created HTML into the gallery table

Figure 28 shows the remaining code to parse the server response and build dynamically the HTML code around the retrieved image file names, i.e. the HTML code is added the HTML variable.



Figure 28: Building the HTML code in a table, don’t forget we need to change it later.

Previously we have defined a columns variable. In our example, every *numberOfColumns* iterations of the loop, we close the row tag and we open a new one using the code on line 54. Every value of iterator that is divisible by *numberOfColumns* with no remainder will push the next cell on a new line. At this point you might like to think a bit more about this. Eventually, if you utilise this gallery generator in your assignment, for example, you will want to achieve a nice layout. In which case, you might like to achieve a single row of gallery thumbnails, or a single column, or a rectangular block of thumbnails, but you will need to do this with div tags not table cells, so you will remove the code in Figure 29 eventually.

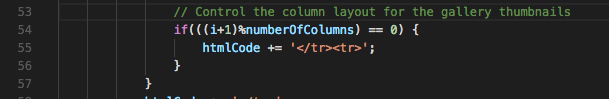


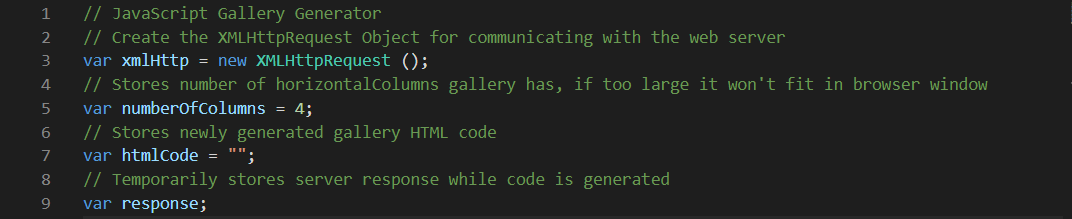
Figure 29: Check number of columns required and terminate the table HTML code.

Third, the generated code is added to a table object that has the id ‘gallery’ as follows, see Figure 30.

../../../../../Desktop/Screen%20Shot%202018-02-04%20at%2015.

Figure 30: send the HTML table back to the HTML gallery table.

The complete JavaScript code should look like that shown in Figure 31.



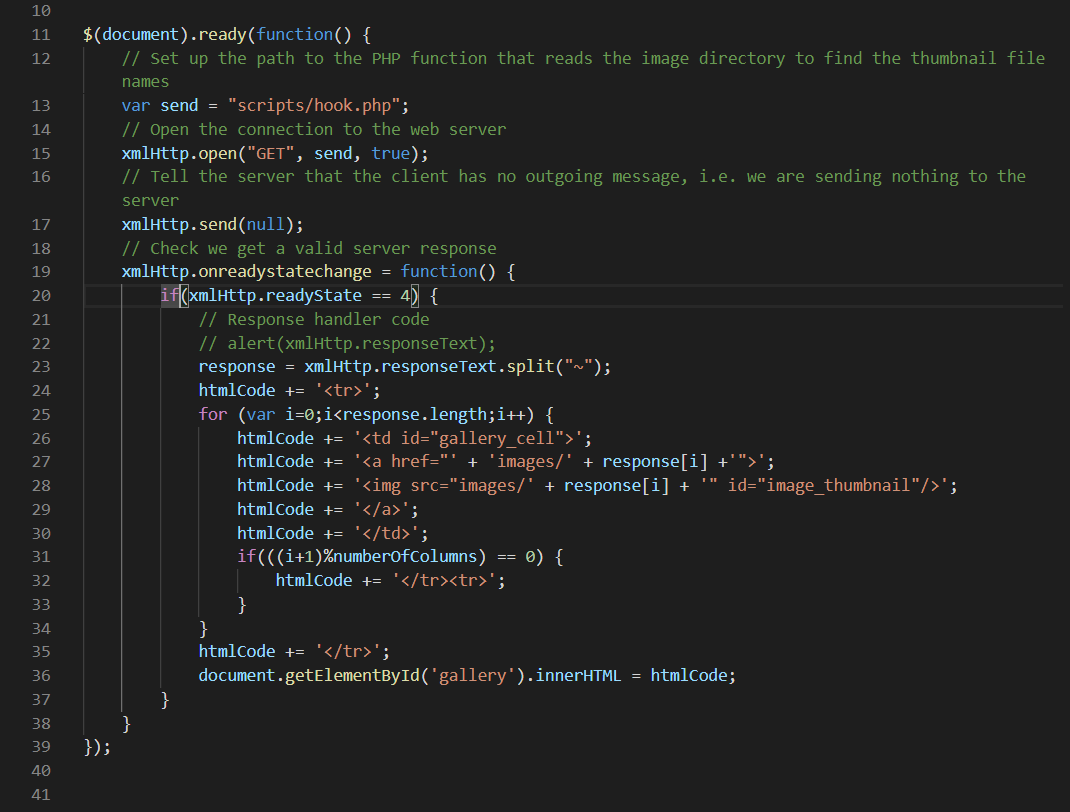


Figure 31: The complete JavaScript controller.

To integrate this into your Lab 5 3D App you will need to redesign the response handler code to create HTML for a div tag, not a table, see later. But, for now you should have a self-contained gallery generator code composed of the files and structure illustrated in Figure 32.

You still have to create test gallery.html page and some CSS rules to style it a gallery.html and CSS rules in gallery.css. Let’s create these files next. Then you will have a self-contained gallery generator that you can use and deploy in your 3D App — of course, you will wish to develop your JavaScript, HTML and CSS further to make it more usable.

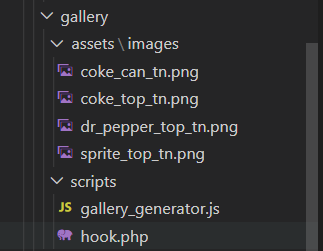


Figure 32: A JavaScript and PHP AJAX based gallery generator

## Step 4: Creating the HTML File for Testing the Gallery

Make a new file and save it as *gallery.html* in your *gallery* folder. The file is very simple as all the gallery HTML code is generated through the JavaScript handler. We will be creating (in step 4) basic CSS rules so do not forget to add a link to it in the header. We will name the CSS file simply *gallery.css* for the table version. Additionally, we need to import the jQuery library and the *gallery\_generator.js* file. It is good practice to always give the html file a header title.

The body contains a centred page title and a table that can be identified by the id “gallery”, see Figure 33. The code in step 3 will not work without that id.



Figure 33: The HTML code with the table div and gallery ID

The result should look similar to that shown in 

Figure 35. Note, the gallery is not styled and thus the images are large. At this point, if you used large images you will get 4 large images shown. Because I scaled mine down a bit in option 1 above I can see them ok in this HTML code, but they still need turning into thumbnails.

To resize them using CSS, e.g. create proper thumbnails, we will use the Bootstrap 4 .img-thumbnail class. Also, here as I have 4 images, I set ‘numberOfColumns = 4’, which allows a single row to be created, similarly I could have set to 1 and 2 to create a single column and a block of 2x2 thumbnails. As already mentioned you will need to modify the HTML code generation to eliminate the tables. When you first run the code, you will get the alert box with the server response, see Figure 34. You can comment out the alert box now.

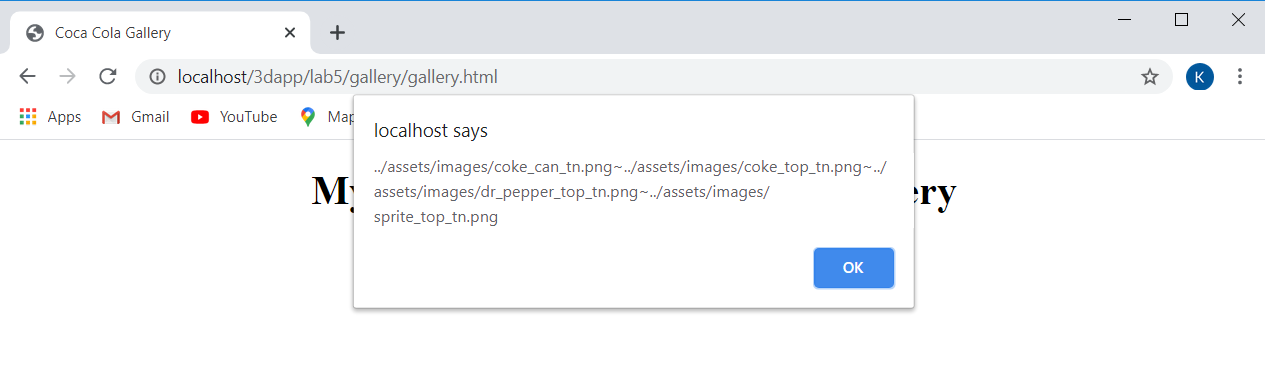


Figure 34: The Alert box showing the server response

If you don’t get this, and your gallery don’t show up, you’ll need to go back and debug.



Figure 35: The gallery generator results with CSS rules applied

## Step 5: Create some CSS rules to style the gallery

Make a new file and save it as *gallery.css* in your *gallery* folder, see Figure 36. The CSS rules contain three style ids. To begin with, we specify a width for the gallery table. Then, we style the gallery cell. No particular reason, but I have chosen an ivory background to the thumbnail, red border, radius 10px, etc. Lastly, I have resized the larger thumbnail pictures to have maximum widths 150px and height of 80px pixels. I have used those parameters instead of width/height to ensure that the thumbnails maintain the correct aspect ratio. Feel free to customize these parameters to suit your 3D App.



Figure 36: The CSS rules

## Step 6 the gallery results

With this CSS3 styling your gallery should look like that shown in Figure 37

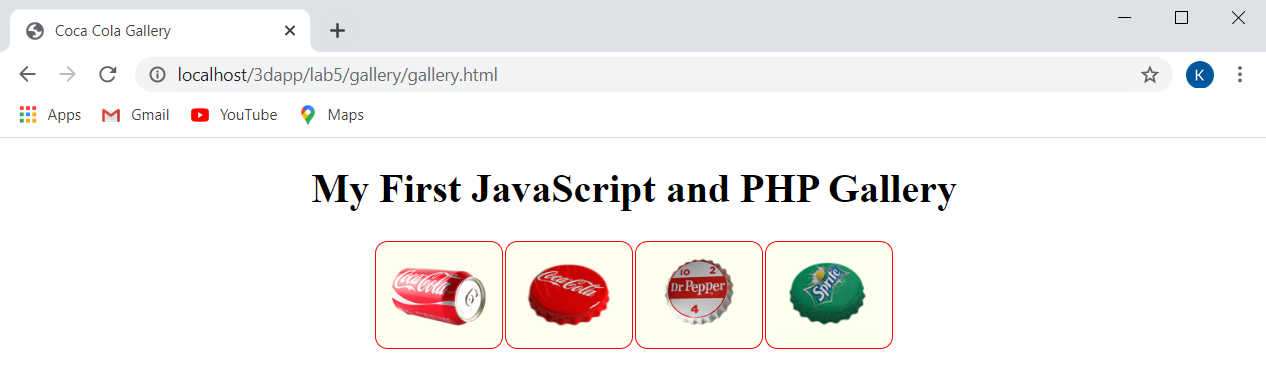


Figure 37: Your first JavaScript and PHP gallery

Clearly, you can see that these thumbnails could be used to trigger rendering of a larger image, or used as button, icons, whatever to load other media contents.

Note, up to now we have not used the Bootstrap 4 .img-thumbnail class. To do this, all you have to do is modify the gallery\_generator.js file and change the id="image\_thumbnail" code for class=”img-thumbnail”. You will also have to declare the bootstrap 4 libraries in this test gallery component’s HTML file. You might want to wait and do this when you integrate the gallery into your Lab 5 site.

When opened, the result will look as follows with the CSS rules applied, see Figure 37. All thumbnails will open the corresponding image files when clicked, but you should realise that you can change that code to load the 3D models, etc.

Here are the results with ‘numberOfColumns = 1’ and ‘numberOfColumns = 2’, see Figure 38.

|  |  |
| --- | --- |
|  |  |

Figure 38: Setting different ‘numberOfColumns’ values

# Integrate the Gallery into your simple 3D App

We now need to figure out how to integrate this gallery into the gallery Lab 4. In Lab 4 we simply use a .card class and stacked the images, now we need to integrate our gallery code so that these images are generated dynamically from a gallery images folder on the web server.

Once adapted, you should be able to update your 3D App at this stage to see the new gallery. The advantage with this gallery is that it dynamically, exploiting AJAX, loads thumbnails from the server. You could then use each thumbnail however you like. For example, you could use them to load the 3D models, or you could use this gallery code as an actual gallery by changing the current test images to 3D images and using a Bootstrap 4 modal component to show the larger image — don’t forget the Bootstrap 4 carousel component.

To integrate this gallery code into your Lab 5 results, you will need to consider the following:

* Convert the JavaScript controller, i.e. the handler, to inject div tags into the front-end view, not tables
* Modify your index.html to use the gallery:
  + That is, replace the current gallery links (3 in total) with the generated version:

<div id="gallery"></div>

* Remember the self-contained gallery generator is using identical id selectors for the 4 images. If these ids were only being used to modify styles you might get away with it (often HTML does not care depending on the browser), even though it is bad practice. So, you need to change the gallery\_cell and image\_thumbnail id selectors (Figure 31 — you will replace the img\_thumbnail with the Bootstrap 4 .img-thumbnail class anyway) to use class selectors, and change the CSS rules to class versions too, Figure 39 illustrates.
* Look at how the HTML is being built in Figure 39:
  + For example, the ‘for loop’ initially sets a counter variable i = 0, and increments the counter until it reaches the total number of images in the response variable. The size or length of the response variable is found in response.length.
  + Take the first line of the htmlcode building code: you can clearly see that this is HTML code inside ‘ ’, which is being added as a string to the htmlcode variable. The next line then simply starts to add more HTML, i.e. '<a href="' + '' + response[i] +'">'; We can see that the + symbol is concatenating the HTML together, which is define inside the ‘ ’, the image filename returned for the response is added to the end of the URL, and so on for building the rest of the HTML code. We will use this approach in Lab 6, where we will exploit AJAX with the JQuery.getJSON() function to request data from the server via a URL and bring back a JSON object that we then parse to produce HTML for the front-end view. Opportunity! you could replace the XMLHttpRequest object with .getJSON() giving it the URL to the hook.php, and modify the hook.php to return a JSON object, etc.
* Also, you need to return the HTML code and use it in 3 blocks of HTML code id’d by the coke, sprite, pepper id selectors in the JavaScript swap function. So, we need 3 galleries, one for each HTML block, but we can’t use the same gallery id 3 times, it won’t work. A quick solution is simply to create 3 unique galleries using the same innerHTML, see Figure 39 with the return of the same innerHTML 3 times. This is possible, because I happen to have decided on 3 gallery block elements.
* Even more so, if you wanted to use this gallery in your assignment with different sets of images for each 3D object modelled, you would need to adapt the code to use the hook.php more than once, each time perhaps passing hook.php a new directory to search in.

Figure 39 shows the JavaScript code modified for the Bootstrap 4 version of your Lab 5.

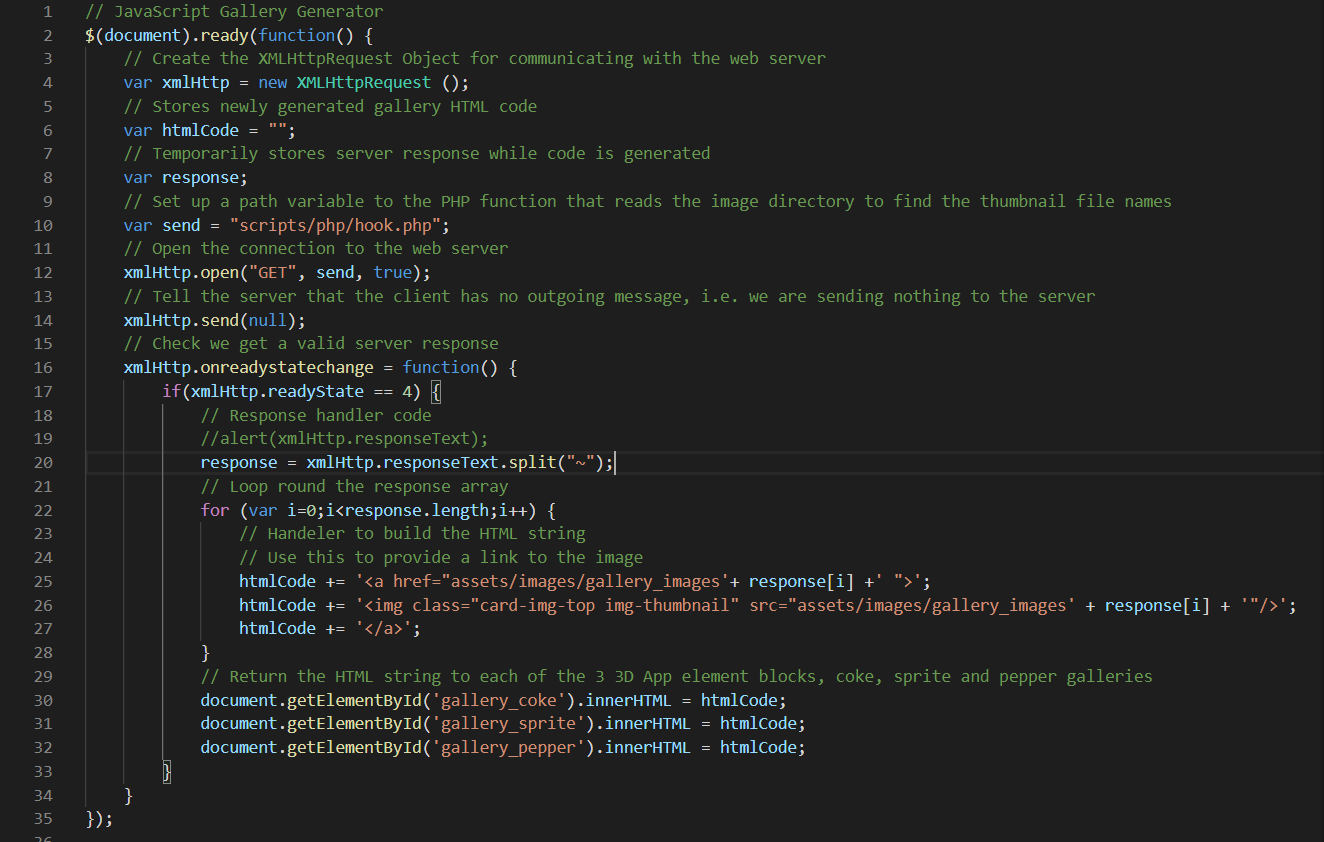


Figure 39: The JavaScript code with div tags and img-thumbnail class.

* You need to deploy the code in your Lab 5 project; at the moment, it is self-contained in the gallery folder. Compare the code for building the table in Figure 31 to that in Figure 39. We have moved the hook.php out of the /gallery/scripts folder placed it in the lab5/scripts/php folder. And we have also placed the gallery\_generator.js file in the lab5/js folder. This means changing the path to find the hook.php, and modifying the HTML code-building handler. You will note form the code shown in Figure 39 that I have insert the assets/images/gallery\_images path, which also means you don’t need hook.php to return the $directory path. This, you will need to modify the hook.php code. Note we eliminated the numberOfColumns variable, that was a nice feature, you might want to see if you can put it back to control the rows and columns of your gallery inside the .card class.
* Make sure you test your generated HTML code — check it with the Chrome inspect tool. See Figure 40 and Figure 41.

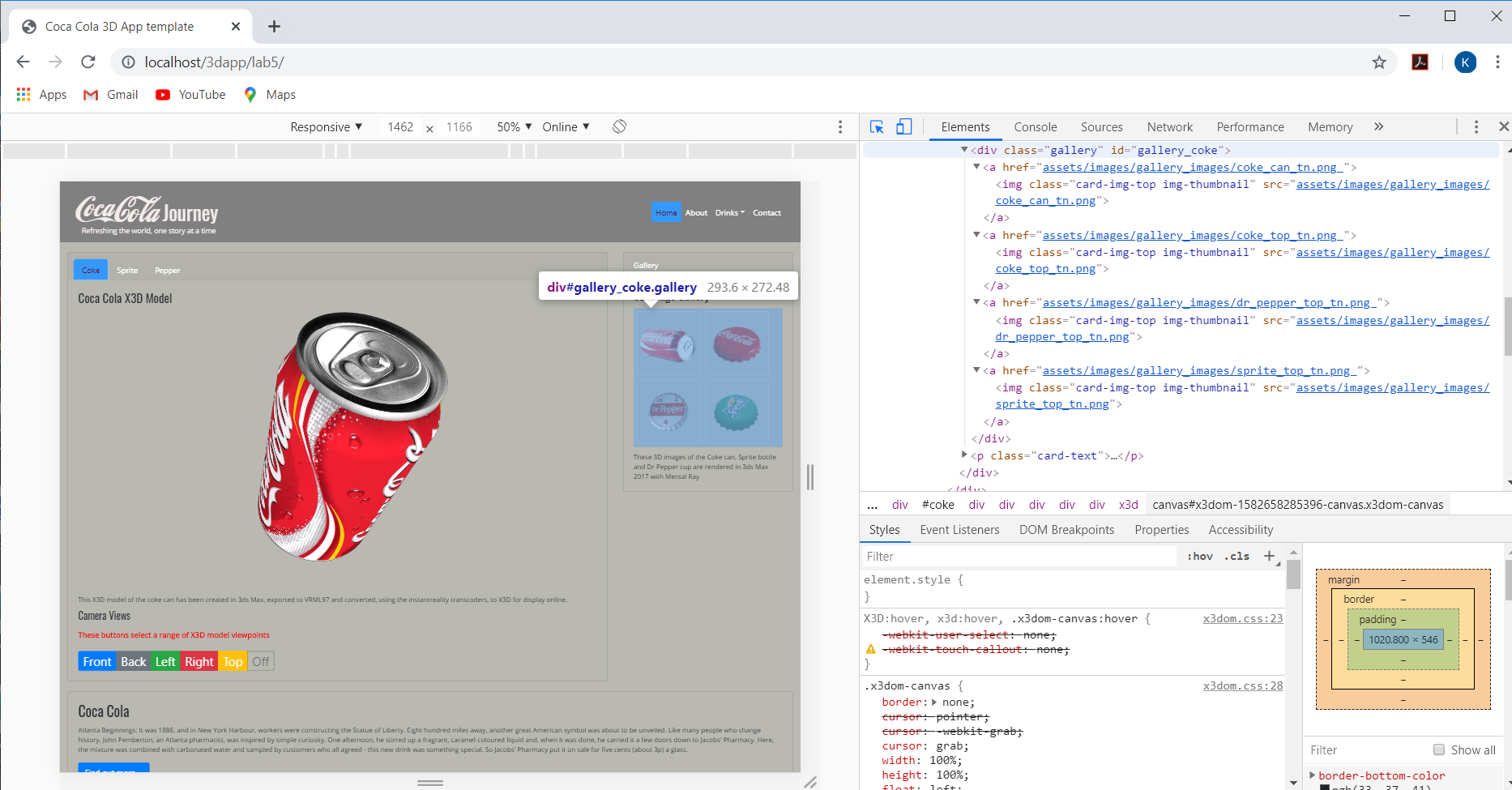


Figure 40: Chrome inspect illustrates the HTML has been correctly generated for the gallery links.

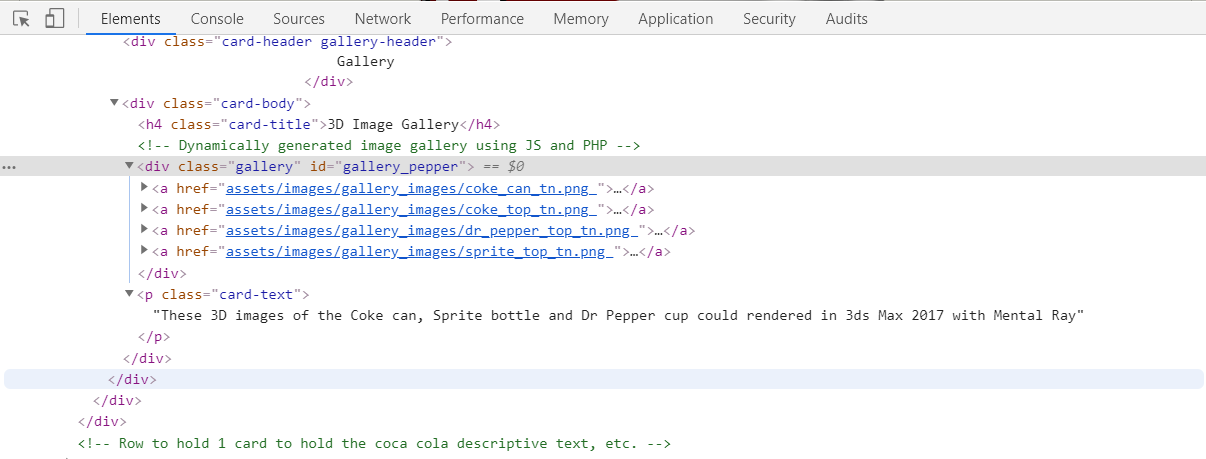


Figure 41: Inspect your new HTML gallery code

However, all we had to do was modify the Bootstrap 4 .img-thumbnail class CSS rules a little to get a reasonable layout inside the .card class based 3D Images Gallery. Note, if we do this we will also change the home page contents images, so we need to copy the current .img-thumbnails CSS rules and create a .gallery class. Figure 44 shows the relevant HTML5 code section where re replace the hardwired 3D Image Gallery from Lab 4 with the new dynamically generated 3D Image gallery. Do the same for the sprite and pepper contents.

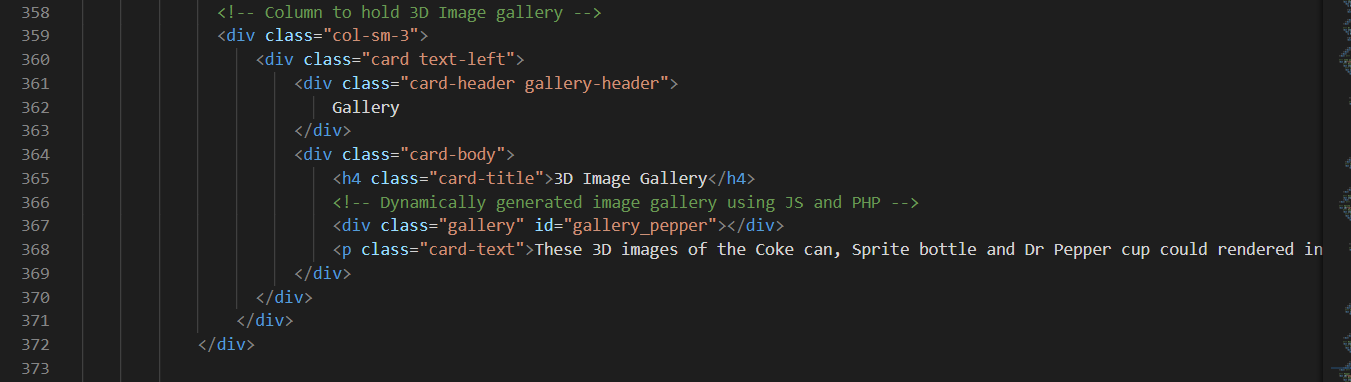


Figure 42: The Bootstrap HTML for Lab 5 for the pepper HTML block, as an example.

Figure 43 shows the new .gallery class with the .ing-thumbnail rules changed.

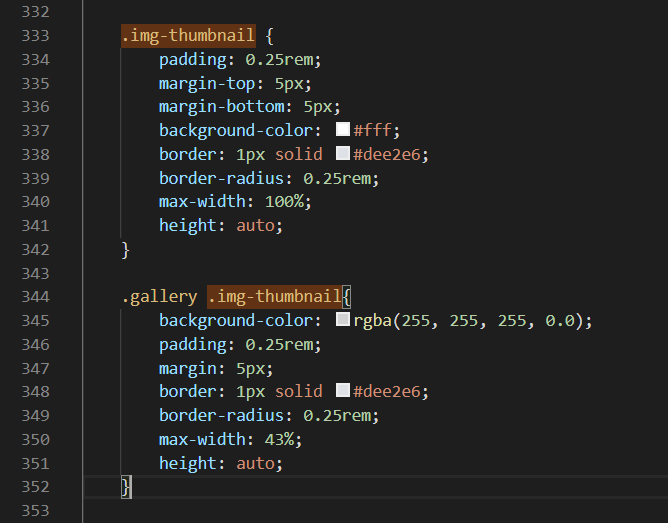


Figure 43: Corresponding Bootstrap CSS rules for mobile.

Figure 46 illustrates the new 3D Image Gallery for the coke page contents.

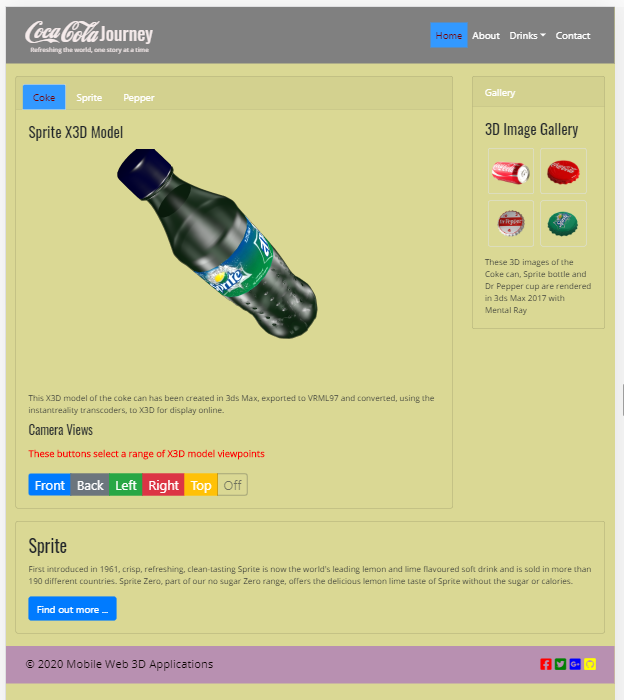


Figure 44: The new dynamic gallery in the Coke page — Bootstrap 4.

The gallery should look the same on the sprite and pepper pages. If you get stuck, you should check out my finished version at the **Live Feedback Site**.

# Restyle the 3D App

Back on page 12 we suggested you have a go at re-styling the 3D App. We have not done this so far, and if you haven’t either, now’s the time to do this task. As illustrated in the examples around page 11 you can adapt this code to create some useful JavaScript functions, see Figure 45. Later in Lab 6, we will use JQuery for swapping as it is more efficient, but for now use JavaScript.



Figure 45: JavaScript functions to swap content, change the look of the page, and reset or change back the look of the page.

Note that you already created the swap function, but you should create the other two JavaScript functions” changeLook(), and changeBack(), and integrate them into your 3D App. If you haven’ already done it, gather up your JavaScript functions and put them in a file called swap\_restyle.js (for want of a better file name). Put this JavaScript file in the scripts folder., and don’t forget to declare it in your index.html and set up the appropriate IDs for their associated block elements.

# Lab 5 Results

Figure 46 illustrates the final Lab 5 version with the JavaScript restyle and change back functions added.

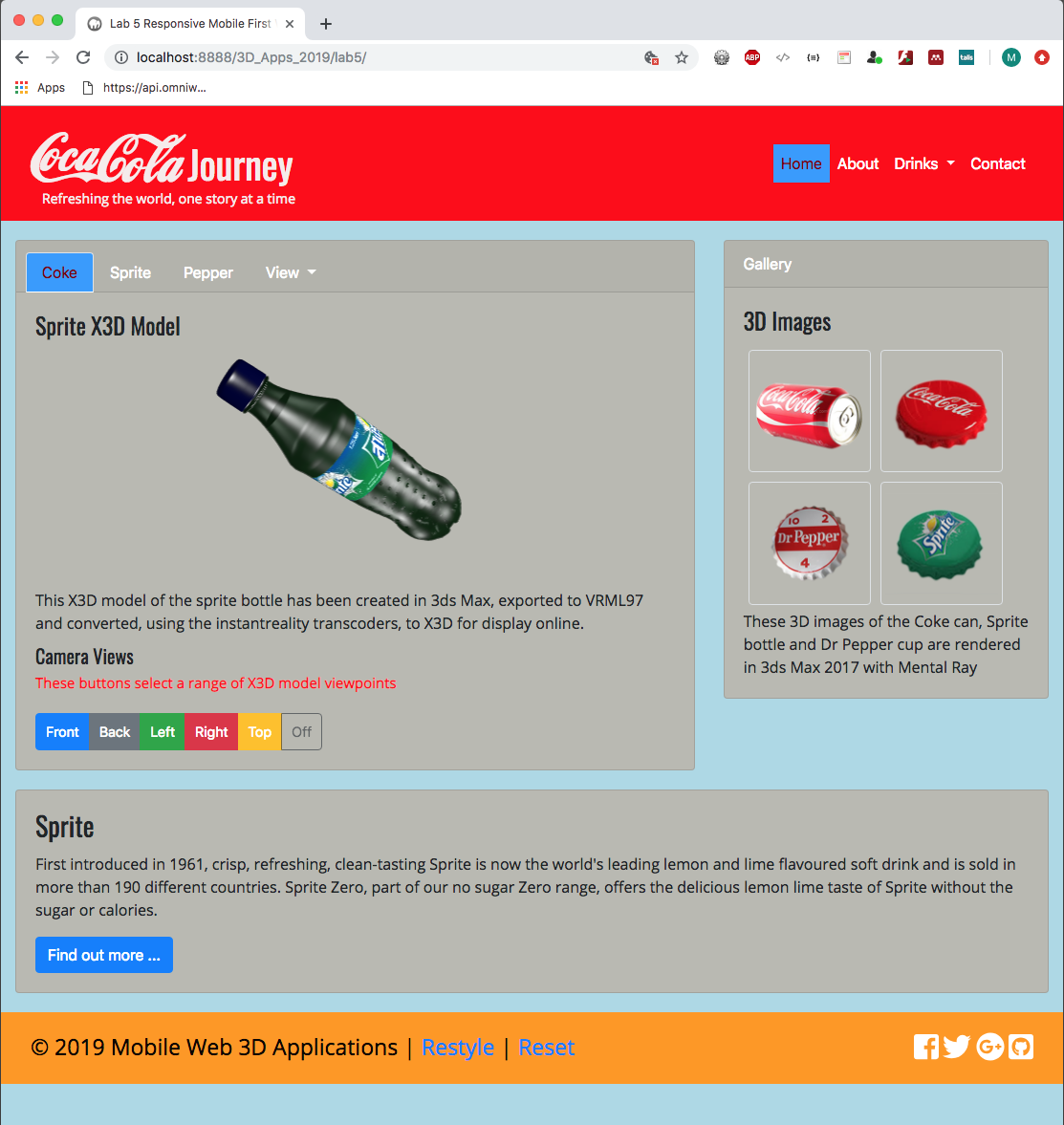
****

Figure 46: Bootstrap dynamic gallery integrated, page restyled — 2017 version.

…

You can inspect the live version the **Live Feedback Site**:

We will continue to improve your Lab 5 version in Lab 6 next week.

# Lab 6 — Exploit AJAX with JQuery and JSON

In Lab 6 the tutorial is divided into two parts:

1. Continue with the Coca Cola theme to create an AJAX version that updates from a JSON file on the backend

* As usual, we will also do some tidying up as we go along. For example, we need to restructure the layout to ensure we are using X3D resources sensibly, instantiating the interaction buttons only once in the SPA, and also instantiating the gallery only one. The only thing we need to change for each contents section is actually the 3D models.

# Appendix

## Appendix 1 — Lab 4 Results

As usual, let’s review what we have done in the previous Lab 4 so that we have a clear focus on what is expected from lab 5.

In Lab 4 you created a 4-page basic 3D App, using a lightweight text editor such as Visual Studio Code and the Bootstrap 4.4

Figure 47 shows the Bootstrap 4.4 results copied over to your Lab 5 folder structure ready to start Lab 5. I’ll just to remind you of some suggested improvements that you might consider doing:

1. Consider some design changes

* For example, replace the main\_3d\_image with a Bootstrap 4 .carousel class of 3D rendered images on the home page.
* Consider more carefully where you position the 3D model and interaction buttons, i.e. camera, animation, lighting controls, etc.
  + An important point to realise as we move forward is that we will need to hook these button controls into the associated inline X3D nodes, without knowing a great deal about the X3D language. But, that’s the beauty of VRML97 and X3D, it is a bit intuitive, so we can easily do this by way of some examples.
* How are you going to load the 3D models? Here I offered several alternatives: the .nav-bar menu dropdown, the .nav-tab menu, utilise the gallery thumbnails, build your own load buttons, etc.
* Make sure that the size of your inline X3D panel in your media screen breaks are proportional to the rest of your 3D App layout. Indeed, make sure all your 3D App components, e.g. buttons, etc. are proportional. Of course, this is all subjective!
* Consider changing the layout for the tablet and mobile, don’t just wed yourself to the layout I happened to use in the labs!
* We put some effort into developing buttons using Bootstrap 4 in Lab 4, but there is much more that can be done. For example, these are clearly Bootstrap 4 buttons, but you could design your own — have a look back through previous years examples on the Live Feedback Site. You could use icons instead, etc.

1. The next major thing to check is to adjust your CSS3 so that the 3D App renders well across all media screen breaks, the **Live Feedback Site** results for these tutorials are not necessarily all finished in this regard, so don’t use these as a benchmark of excellence!

* Make good use of the [ChromeDev Tools](https://developer.chrome.com/devtools), right click and select Inspect in Chrome, to check your 3D App works across all media screens breaks and a range of appropriate devices.

However, these are just some of my thoughts, you probably have much better ideas for improvements. You should look into this after completing the basics of Lab 5.

Please Note, if your assignment hand-in in Week 11 merely consists of the lab work you won’t score so well — while, of course, you are expected to use the codes developed and extend upon them, you are also expected to produce your own virtual museum brad and layout for your assignment.

|  |  |
| --- | --- |
|  |  |
| Bootstrap 4/3 mobile (xs) | Bootstrap 4.3 (xl) |

Figure 47: Results from Lab 4 move to the Lab 5 folder.

Note, I simply choose different CSS background colours for the Bootstrap 4 media screen breaks in Figure 47, and I haven’t done this for every HTML5 block element.

So, don’t get confused between this method of using CSS to change styling and the method above where we use JavaScript code that you will develop to change such at runtime.

## Appendix — CSS Selector IDs

A CSS id selector should be a unique id used only once to select that block HTML element. What you find, though, is that in practice you can often get away with repeated CSS id selectors that are the same. The reason for this is that the WC3 CSS specification is not precise and different browsers treat it slightly different when rendering HTML, often being quite forgiving. But, you do not know when the next browser implementation may interpret the specifications more harshly. Moreover, if you are processing an HTML block element based on a CSS id using a JavaScript function, that JavaScript function hasn’t a clue which CSS id and associated HTML block element you are trying to manipulate, so unpredictable results can arise. For example, it may pick the first instance of the id and ignore the rest.

The general solution is to change your CSS id selectors for CSS class selectors, which can be used multiple times. Take the time to study and understand CSS ID versus Class selectors, you will need to get this right when implementing your Lab 5 multiple web pages into a single web page layout and integrating your image gallery code below.

## Appendix — Self-Directed Learning

You should try and find the time to follow some external tutorials (see Canvas or try the links below) to get a better understanding and acquire more knowledge of JavaScript and read around X3DOM to better prepare yourself for the assignments. You should be looking for functionality that is useful to implement as part of your assignment. For example, maybe a ‘light box’ might be useful to adapt to display your 3D model. There are many resources/tutorials online that show you how to add a ‘light box’ to your gallery — you can use Bootstrap 4’s modal and .img-thumbnail class and the gallery generator above we coded above. Consider re-coding your gallery to use JQuery .getJSON()) as the bare bones for building quite a sophisticated gallery. Light boxes allow images to open superimposed on the page with a dimmed background and a close button — you might be able to adapt this to display your 3D model instead.

Here are some example light box libraries:

* <http://lokeshdhakar.com/projects/lightbox2/>
* <http://noelboss.github.io/featherlight/>
* <https://plugins.jquery.com/tag/lightbox/>

There are many others available. Alternatively, you might like to adapt a tutorial to create your own based on the image gallery we already started above:

* <http://webdesignerwall.com/tutorials/creating-a-modern-lightbox-with-css3-and-javascript>
* <http://webdesign.tutsplus.com/tutorials/htmlcss-tutorials/super-simple-lightbox-with-css-and-jquery/>

## JavaScript tutorials

You might like to have a look at the following JavaScript tutorials:

* [W3C Schools JavaScript Tutorial](http://www.w3schools.com/js/)
  + We recommend this, particularly for its interactive ‘try it’ feature
* [JavaScript for the Total Non-Programmer](http://webteacher.com/javascript/)
  + This is not bad, short and sweet

## X3DOM tutorials

It is a good idea to try out some of the X3DOM tutorials in your own time, because these provide the most sophisticated and cutting-edge methods for integrating 3D with the web page for your Web3D App. So, please take the time to look at X3DOM more closely:

* [X3DOM Tutorials](http://doc.x3dom.org/tutorials/index.html)
* [X3DOM Examples](http://examples.x3dom.org/)

1. <https://www.w3schools.com/xml/xml_http.asp> [↑](#footnote-ref-1)