LAB: JSON/ AJAX

Lesson Time: 60 Minutes

In this lab, we'll show you step-by step how to pull data from a JSON file and populate HTML with the data. This is a challenging lab, so we want to walk you through it. Study this code and then try to re-create it on your own. It's important to understand exactly how it works. If you understand what this code does, you have reached a major milestone in understanding web development!

We've created a JSON file that has a list of beverage products and made it publically available on github. To view the file on github, visit this link.

https://github.com/Steve-Rossiter/SD101/blob/master/mydata.json

The Raw JSON version is found here;

https://raw.githubusercontent.com/Steve-Rossiter/SD101/master/mydata.json

1. Begin by downloading the jsTemplate.html file as a starting point for this lab.
    1a. Delete the element from the HTML but keep the button.
2. Create a table in the HTML file.
    - The table will have 5 columns and 5 rows
        - 1 header row, 4 rows for our json data
        - 5 columns, for each field in the JSON
    - Apply a nice CSS formatting of your choice to the table.
    - The first row will be column headings and have the values product name, serving, package, price, qty
    - The remaining rows will display the data values that come from the JSON file.

To do this, re-create the following HTML code:

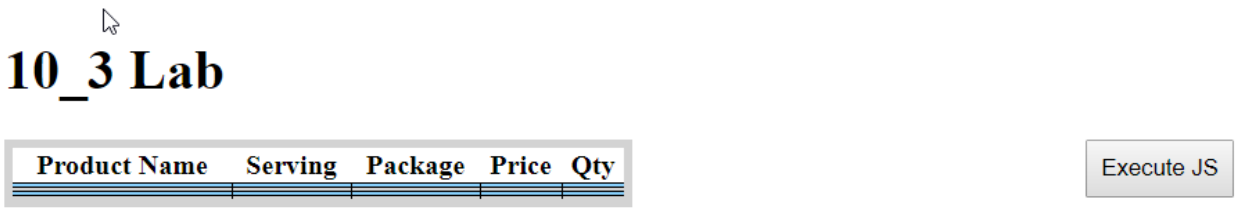
```
<table id="table">
    <tr>
        <th>Product Name</th>
        <th>Serving</th>
        <th>Package </th>
        <th>Price</th>
        <th>Qty</th>
    </tr>
    <tr>
        <td class='product'></td>
        <td class='serving'></td>
        <td class='package'></td>
        <td class='price'></td>
        <td class='qty'></td>
    </tr>
    <tr>
        <td class='product'></td>
        <td class='serving'></td>
        <td class='package'></td>
        <td class='price'></td>
        <td class='qty'></td>
    </tr>
```

```
<tr>
    <td class='product'></td>
    <td class='serving'></td>
    <td class='package'></td>
    <td class='price'></td>
    <td class='qty'></td>
</tr>
<tr>
    <td class='product'></td>
    <td class='serving'></td>
    <td class='package'></td>
    <td class='price'></td>
    <td class='qty'></td>
</tr>
```

Notice how for each cell in the table, we've assigned the cell one of five classes.

- Every cell in the first column has a class of "product"
- Ever cell in the second column has a class of "serving"
- And so on..

Right now, our HTML looks like shown below. We've already applied some CSS to the table, but you can add any CSS you want.



Our next step is to update the table cells with the data from JSON file on git hub.

3. Update the JsFunction to do the following;

- Send an XMLHttp request to GitHub to get the JSON data

- Store the Json Data in a variable
- Update each element of the table with the JSON data

To do this, we'll write the following code into our jsFunction()

```
var requestURL = 'https://raw.githubusercontent.com/Steve-Rossiter/SD101/master/mydata.json';
var request = new XMLHttpRequest;
request.open('GET', requestURL);                                    1

request.onreadystatechange = function(){
    console.log(this.readyState + " " + this.status + " " + this.statusText);
    if (this.readyState == 4 && this.status == 200){
        var jsonData = this.responseText;                           2
        console.log(jsonData);
        var objectData = JSON.parse(jsonData);
```

1. First we will create a variable called requestURL. It will hold the URL we want to pull our JSON data from.
2. Then, create a variable called request, which will be a new AJAX XMLHttpRequest
3. Finally we call the open method to GET the requestURL. Our web connection to gitHub is open.

4. In the next block of code, we code what we want to have happen when the HTTP GET Request is completed. Keep in mind, we have not send our request yet!

`Request.onreadystatechange` tracks the status of our HTTP request. We always write `Request.onreadystatechange = function(){}` . This lets us assign a function to tell JS what we want to do with the data once it's ready.

Because we only care about data that is ready, we write an if statement
`if(this.readyState == 4 && this.status == 200)`

This tells JS only execute if the HTTP request is ready and ok. This means if there was a network error and the request did not transmit, the code will not execute.

Next, we need to get our json data. We'll create a variable jsonData and set it to this.responseText. ResponseText holds the JSON returned by the web request. It's helpful to log this to the console to see we did get our JSON data.

Next, we need to change our json text into an JSON object with parse(jsonData)

```
let productColumns = document.getElementsByClassName("product");
for (let p = 0; p < productColumns.length; p++){
    html = productColumns[p] ;
    html.innerHTML = objectData.products[p].productName;
}
```

Next, we create a variable called productColumns. This variable uses getelementsByClassName to get all of our HTML elements with the class “product”. This is why we assigned all of the cells in column 1 the class “product.

Next, we create a for loop. For each html element that belongs to class “products”, we’ll set the innerHTML value to matching products.productname from our JS Object.

This works because we know the number of records in our products array, and we’ve created an equal number of HTML rows in our table.

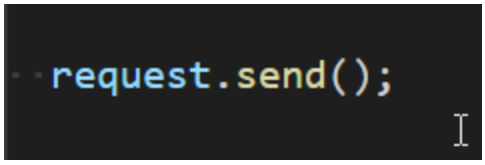
Finally there is nothing left to do but repeat this process for each of the other columns. We do the same thing again for each of our remaining HTML classes and product objects key.

```
let servingColumns = document.getElementsByClassName("serving");
for (let p = 0; p < servingColumns.length; p++){
    html = servingColumns[p] ;
    html.innerHTML = objectData.products[p].serving;
}

let packageColumns = document.getElementsByClassName("package");
for (let p = 0; p < packageColumns.length; p++){
    html = packageColumns[p] ;
    html.innerHTML = objectData.products[p].package;
}
```

We could improve our code here by creating a larger loop that goes through all of our classes, but we left it like this so you could get a better understanding of what the code is doing.

Lastly at the end of the script, we must remember to run request.send(). Without this, the request will not execute and the code in the onreadystate sub-function will not fire



.

Congratulations! If you understand what we've done in this example, you are really on your way to understanding how JS and the Web Server work together to create dynamic HTML!

To summarize what we've accomplished in the lab .

1. We had data in a JSON format on a web server (github)
2. We used JS to call the web server and make a request to get the data using AJAX, without reloading the page.
3. When the data was returned from the web server, we processed the data and displayed it nicely on our page in a table format.

Key Terms	
Lesson Files	10_3_solution.html
Additional Resources	
Further Learning	