WEB422 Assignment 2

Submission Deadline:

Friday, February 5th @ 11:59pm

Assessment Weight:

9% of your final course Grade

Objective:

To work with our "Restaurants" API (from Assignment 1) on the client-side to produce a rich user interface for accessing data. We will practice using well-known CSS/JS code and libraries including Lodash, Moment.js, Leaflet, jQuery & Bootstrap

Sample Solution:

You can see a video of the solution running at the following location:
https://pat-crawford-sdds.netlify.app/shared/winter-2021/web422/A2/A2.mp4

Specification:

For this assignment, we will create a single table that shows a subset of the restaurant data (ie: columns: **Name, Cuisine, Address and Average Score**). When the user clicks on a specific restaurant (row) in the table, they will be shown a modal window that shows an interactive map indicating where the restaurant is located. We will be making use of the Bootstrap framework to help design our UI, jQuery to work with the DOM, Lodash to specify our template(s) and Leaflet to render the map.

The Solution Directory

The first step is to create a new directory for your solution, open it in Visual Studio Code and add following folders / files:

We will not be including any of the JavaScript / CSS libraries locally. Instead, we will be leveraging their CDN locations (See the Week 2 notes for the <script> and <link> elements necessary to include jQuery, Bootstrap, Lodash and Leaflet). **Note:** Remember that the order is important, ie: jQuery should be included *before* the Bootstrap JavaScript and your main.js file should be included last.

Creating the Static HTML:

Next, we must create some Static HTML as a framework for the dynamic content.

Open your index.html file and add the minimum code required for an HTML5 page (HINT: type **!** and then immediately type the **tab** key to get an HTML 5 skeleton). Once this is complete, include links for:

- The Bootstrap 3.4.1 Minified CSS File (Using the CDN)
- The CSS for **Leaflet**, ie:

  ```
  <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" integrity="sha512-xodZBNTC5n17Xt2atTPuE1HxjVMSvLVW9ocqUKLsCC5CXdbqCmblAshOMAS6/keqq/sMZMZ19scR4PsZChSR7A==" crossorigin=""/>
  ```

- Your **main.css** file (**NOTE**: This file will only consist of two selectors (for now) to ensure that your "restaurant-table" (or whatever you wish to call it) causes the cursor to change to a "pointer" whenever a user moves their mouse over a row and that your map is a specific height, ie:

  ```
  #restaurant-table tr:hover{ cursor:pointer; }
  #leaflet { height: 400px; }
  ```

- The jQuery 3.4.1 Slim, Minified JS File (Using the CDN)
- The Bootstrap 3.4.1 Minified JS File (Using the CDN)
- The Lodash 4.17.5 Minified JS File (Using the CDN)
- The JS File for **Leaflet**, ie:

  ```
  <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js" integrity="sha512-XQoYMqMTK8LvdxXYG3nZ448hOEQiglfqkJs1NOQV44cWnUrBc8PkAOcXy20w0vlaXaVUearIOBhiXZ5V3ynxwA==" crossorigin=""></script>
  ```

- Your **main.js** file

With all of our libraries and files in place, we can concentrate on placing the static HTML content on the page. This includes the following:

Navbar

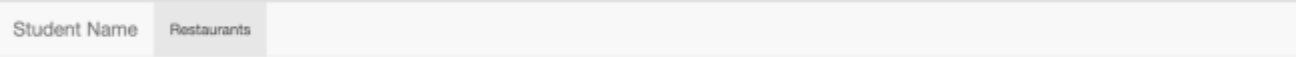
Assignment 2 will use an extremely simplified Bootstrap 3 navbar. Begin by copying the full **Default Navbar** example HTML code from the official documentation: https://getbootstrap.com/docs/3.4/components/#navbar-default and pasting it as the first element within the <body> of your file.

- Next, proceed to remove **all child elements** from the "bs-example-navbar-collapse-1" element
- In the (now empty) "bs-example-navbar-collapse-1" element, put back a single navigation item and label it "Restaurants", ie:

```
<ul class="nav navbar-nav">
  <li class="active"><a href="#">Restaurants<span class="sr-only">(current)</span></a></li>
</ul>
```

- Finally, change the "navbar-brand" to be your name

When completed, your navbar should look like the following



Bootstrap Grid System (1 Column)

Since we are leveraging Bootstrap for this assignment, we should make use of their excellent responsive grid system. Beneath the navbar, add the following HTML

- Include a element with the class "container" (so that our content is centered)
- Within the "container", create a with class "row"
- Within the "row", create a with class "col-md-12" (we will only have one column to show our data)

Main Table Skeleton

The main interface that users will interact with to view data in our application is a HTML table consisting of **4 columns: Name, Cuisine, Address** and **Average Score**. Create this table within your "col-md-12" container according to the following specification:

- The <table> element should have the class "table" and a unique id, ie: "restaurant-table", since we will be accessing it programmatically from JavaScript
- The <thead> element should contain one row
- The single header row should have 4 table heading elements with the text:
  - Name
  - Cuisine
  - Address
  - Average Score
- The <tbody> element should be empty

Once your table is in place, your app should look like the following:

Paging Control

Since our "restaurants" collection contains approximately 25000 documents, we will leverage our Web API's pagination feature when pulling restaurants from the database (ie: **/api/restaurants?page=1&perPage=10**, etc). To give the user some control over which page they wish to see, we must include a primitive pagination control (for this assignment, we will not let them "jump" to a specific page, but instead we will let them go back and forth between the pages in sequence). To accomplish this, we must place the pagination buttons on our page before wiring up their functionality using jQuery:

- Begin by copying the full **Pagination** HTML code from the official documentation: https://getbootstrap.com/docs/3.4/components/#pagination and pasting it directly underneath your newly created "restaurant-table".
- Next, delete the list items that contain the numbers **2**, **3**, **4** and **5** (leaving just **1**)
- Give each of the 3 remaining <a> elements (nested within the <li> elements) unique id values such as "previous-page", "current-page" and "next-page" (we will use these id values to add functionality to the links and display the current page)
- Finally, remove the text **1** from the middle link (it will be added dynamically later).

Once your pagination control is in place, your app skeleton should look like the following:

"Generic" Modal Window Container

We will be showing our map for a specific restaurant in a Bootstrap modal window. Since every time we show the modal window, it will have different content (Specific to the Restaurant that was clicked), we must add an empty, **generic** modal window to the bottom of our page.

To get the correct HTML to use for your Bootstrap modal window, use the following example from the documentation as a starting point.

Once you have copied and pasted the "modal" HTML into the bottom of your <body> element (ie, below the other content) , make the following changes:

- Give your with the class "**modal fade**" a unique **id**, ie: "**restaurant-modal**". We will need to reference this element every time we wish to show / work with the **modal window.**
- Remove the "Modal Title" text from the <h4> element with class "**modal-title**". We will be using jQuery to populate this with the selected Restaurant Name.
- Remove the <p> element with the text "One fine body…" from the element with class "**modal-body**" and replace it with the following two elements:

```
<div id="leaflet"></div>
<div id="restaurant-address"></div>
```

These are placeholders for both the map, and restaurant address respectfully.

- Finally, remove the button element with the text "Save Changes". This modal is used to display information only, so a "save" button is not required

JavaScript File (main.js):

Now that we have all of our static HTML / CSS in place, we can start dynamically adding content and responding both user and bootstrap events using JavaScript. In your **main.js** file add the following variables & functions at the top of the file:

- **restaurantData** (array)

  This should be an empty array (we will populate it later with a "fetch" call to our back end API)

- **currentRestaurant** (object)

  This should be an empty object (ie: {} – we will populate it later once the user clicks on a specific restaurant within our UI)

- **page** (number)

  This will keep track of the current page that the user is viewing. Set it to **1** as the default value

- **perPage** (number)

  This will be a constant value that we will use to reference how many restaurant items we wish to view on each page of our application. For this assignment, we will set it to **10**.

- **map** (leaflet "map" object)

  This will be a reference to our current "map", once it has been initialized. For now, simply assign it a value of **null**

- **avg(grades)** (function)

  This function can be used to help you calculate the average score, given an array of "grades" objects for a

specific restaurant (ie: [{date, grade, score}, …] as its input parameter. This function will loop through the grades array to determine average score and return it (formatted using .toFixed(2)).

- **tableRows** (Lodash template)

  This will be a constant value that consists solely of a Lodash template (defined using the _.template() function). The idea for this template is that it will provide the functionality to return all the rows for our main "restaurant-table", given an array of "restaurant" data. For example, if the **tableRows** template was invoked with the first two results back from our **Web API** (***left***), it should output the following **HTML** (***right***):

```
{
  "address": {
    "coord": [ … ], // 2 items
    "building": "2300",
    "street": "Southern Boulevard",
    "zipcode": "10460"
  },
  "_id": "5eb3d668b31de5d588f42933",
  "borough": "Bronx",
  "cuisine": "American",
  "grades": [ … ], // 3 items
  "name": "Wild Asia",
  "restaurant_id": "40357217"
},
{
  "address": {
    "coord": [ … ], // 2 items
    "building": "7715",
    "street": "18 Avenue",
    "zipcode": "11214"
  },
  "_id": "5eb3d668b31de5d588f42934",
  "borough": "Brooklyn",
  "cuisine": "American",
  "grades": [ … ], // 4 items
  "name": "C & C Catering Service",
  "restaurant_id": "40357437"
},
```

```
<tr data-id="5eb3d668b31de5d588f42933">
  <td>Wild Asia</td>
  <td>American</td>
  <td>2300 Southern Boulevard</td>
  <td>6.00</td>
</tr>

<tr data-id="5eb3d668b31de5d588f42934">
  <td>C &amp;amp; C Catering Service</td>
  <td>American</td>
  <td>7715 18 Avenue</td>
  <td>3.50</td>
</tr>
```

You will notice a few things about the formatting of each row in the returned result, specifically:

- The template loops through each object in the array to produce a <tr> element (**HINT**: .forEach() can be used here)
- Each <tr> has a "data-id" attribute that matches the **_id** property of the object in the current iteration
- The first <td> contains the **name** property of the object in the current iteration
- The second <td> contains the **cuisine** property of the object in the current iteration

- The third <td> contains a combination of the **address.building** and **address.street** properties of the object in the current iteration

- The final <td> content contains the average grade score for the object in the current iteration. This can **be** obtained by invoking the **avg(grades)** function and providing the "grades" array

HINT: Place all the HTML / Code for your template within a string defined using `` (this will allow you to write our template string across multiple lines.

- **loadRestaurantData()** (function)

  Now that our templates and global variables are in place, we can write a utility function to actually **populate** the **restaurantData** array with data from our API created in Assignment 1 (now sitting on Heroku). To achieve this, the loadRestaurantData function must:

  - make a "fetch" request to our Web API hosted on Heroku using the route:
    /api/restaurants?page=***page***&perPage=***perPage***
    Here, the values of **page** and **perPage** must be the values of the variables: **page** and **perPage** that you declared at the top of the file at the beginning of this assignment - **perPage** is a constant value and **page** is the current working page.
  - When the fetch request has returned and the json data has been parsed:
    - set the global **restaurantData** array to be the data returned from the request
    - invoke the **tableRows** template with the data returned from the request (ie: **tableRows({restaurants: data})**) and store the return value in a variable.
    - Select the <tbody> element of your main "restaurant-table" and set its html (.html()) to be the returned value from when you invoked the **tableRows** template function, above.
    - Select the **current-page** element (from your pagination control) and set its html (.html()) to be the value of the current **page**

Now that our loadRestaurantData() utility function as well as our templates and global variables are in place, let's add the following code to be executed when the document is ***ready*** (ie: within the **$(function(){** … **});** callback):

The first thing that needs to be done, is to invoke the **loadRestaurantData()** function to populate our table with data and set the current working page

Next, we must wire up the following **5 events** using jQuery**:**

1) <u>Click</u> <u>event for all **tr** elements within the **tbody** of the **restaurant-table**</u>

Once this event is triggered, we must perform the following actions:

- Locate the restaurant object in the **restaurantData** array whose "_id" property matches the "data-id" property of the row that was clicked, and store it as the **currentRestaurant** object (declared at the top of the file). This will allow us to work with the clicked restaurant object (**currentRestaurant**) in our other events (below).

  HINT: the "data-id" value can be obtained by using the jQuery code: $(**this**).attr("data-id");)

- Set the content of the "modal title" (ie: <h4 class="modal-title"></h4>) for the "restaurant-model" to the **name** of the **currentRestaurant**

- Set the contents of the "restaurant address" (ie:) to a combination of the **address.building** and **address.street** properties of the **currentRestaurant** in order to give a full address

- Open the "Restaurant" Modal window (ie: <div id= "restaurant-modal" … > … ").

2) **Click** event for the "previous page" pagination button

When this event is triggered we simply need to check if the current value of **page** (declared at the top of the file) is greater than **1**. If it is, then we decrease the value of **page** by 1 and invoke the **loadRestaurantData** function to refresh the rows in the table with the new page value.

3) **Click** event for the "next page" pagination button

This event behaves almost exactly like the click event for the "previous page", except that instead of *decreasing* the value of page, we **increase** the value of **page** by 1 and invoke the **loadRestaurantData** function to refresh the rows in the table with the new page value.

4) **shown.bs.modal** event for the "Restaurant" modal window

This event is actually a Bootstrap event that triggers once a modal window is fully shown (after the CSS transitions have completed). To wire up this event, we can use the following jQuery code:

```
$('#restaurant-modal').on('shown.bs.modal', function () {});
```

Once the modal has been "shown", we must include the following code in order to correctly render a map using our **Leaflet** library:

```
map = new L.Map('leaflet', {
  center: [Restaurant Address Coordinate 1, Restaurant Address Coordinate 0],
  zoom: 18,
  layers: [
    new L.TileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png')
  ]
});

L.marker([Restaurant Address Coordinate 1, Restaurant Address Coordinate 0]).addTo(map);
```

You will notice that there are placeholders for ***Restaurant Address Coordinate* 1** and **0**. These should be replaced with the values from the **coord** array in the **address** property of the **currentRestaurant**.

Additionally, you will notice that the **new** L.Map('leaflet', { … }); code creates an object that we assign to our **map** variable (originally defined as **null** at the top of our file). This is so that we are able to correctly *remove* the map once the modal window has closed.

5) **hidden.bs.modal** event for the "Restaurant" modal window

This event is wired up in the exact same way as the "shown.bs.modal" event, defined above. However, this time instead of creating a new map, we must remove the existing map using the code:

```
map.remove();
```

Assignment Submission:

- Add the following declaration at the top of your main.js file

```
/*********************************************************************************
* WEB422 – Assignment 2
* I declare that this assignment is my own work in accordance with Seneca Academic Policy.
* No part of this assignment has been copied manually or electronically from any other source
* (including web sites) or distributed to other students.
*
* Name: _____ Student ID: _____ Date: _____
*
*
*********************************************************************************/
```

- Compress (.zip) the files in your Visual Studio working directory (this is the folder that you opened in Visual Studio to create your client side code).

Important Note:

- **NO LATE SUBMISSIONS** for assignments. Late assignment submissions will not be accepted and will receive a **grade of zero (0)**.
- After the end (11:59PM) of the due date, the assignment submission link on My.Seneca will no longer be available.
- Submitted assignments must run locally, ie: start up errors causing the assignment/app to fail on startup will result in a **grade of zero (0)** for the assignment.