WEB422 Assignment 1

Submission Deadline:

Friday, May 17th @ 11:59pm

Assessment Weight:

5% of your final course Grade

Objective:

This first assignment will get the "Teams" Web API ready for use with the WEB422 course as well as reinforce the use of the jQuery and Bootstrap frameworks for rendering API data.

Software Requirements:

Before you begin, make sure you have the following installed:

1. VS Code
2. node.js (10.x LTS version) and npm
3. git
4. heroku CLI

Specification:

For this assignment, we will be publishing our own personal version of the "Teams" API on Heroku. We will then write jQuery code to request data and render it on the page using Bootstrap (3.3.7).

Step 1: Obtaining the Code Examples / teams-api from GitHub

The first step will be downloading ("cloning") The web422 repository, located here:

https://github.com/sictweb/web422

This can be done by opening a Command Prompt / terminal window and entering the command:

git clone https://github.com/sictweb/web422.git

Once this operation completes successfully, open the “web422" folder and then "Code Examples.”. Here you will find a "**teams-api**" folder - this is the main folder with all of the API / Server logic in it.

Copy this **“teams-api”** directory out of "Code Examples" and into another, separate location (e.g., put it in your course WEB422 folder). We will be updating this code during this and *future* assignments.

Step 2: Following the Guide

Now that you have obtained the teams-api source code and have the folder open in VS Code, the next step involves following along with the instructions located at:

https://github.com/sictweb/web422/tree/master/Code%20Examples/teams-api#teams-api

This will guide you through the MongoDB Atlas Database set up, as well as configure the code correctly and push the solution to Heroku.

When you are done, you should be able to request data from your teams-api node service on Heroku, and have it return data from your MongoDB Atlas database.

Step 3: Boilerplate Client Side Code

Once you have completed the guide (Step 2), and have the Teams API running on Heroku (this can be tested by accessing the newly-created Heroku URL in a web browser and using one of the API routes such as "/teams") we need to create some code on the client-side to work with the data.

To get started, use the provided **assignment-1** folder from the assignment-1.zip file.

Start by running the `**npm install**` command to install necessary dependencies. Next, open the code in VS Code.

- You should see the following folder / file structure
  - src/
    - index.html

        - index.js
        - styles.css
    - README.md (NOTE: all other files are discussed in the README.md)
- Next, we will use some boiler-plate code to start our index.html. You can copy and paste the index.html file from the WEB422 "static-server" example, which can be found here:
-

https://github.com/sictweb/web422/blob/master/Code%20Examples/static-server/public/index.html

- Be sure to add a CSS reference to “styles.css" and a JavaScript reference to “index.js"
- Lastly, ensure that the <title> element is changed from "Welcome" to "*student* - WEB422" where *student* is your first and last name.

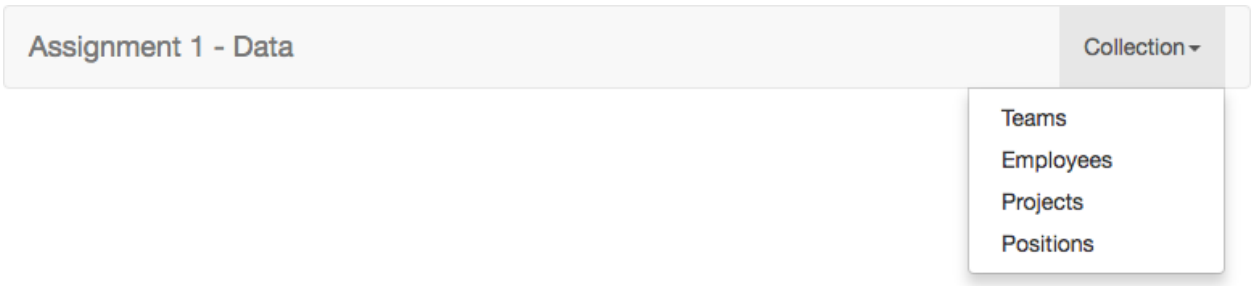
Step 4: Static Content

Before we start accessing the API and populating our page with data, we should create some static page elements that support our dynamic data. Make use of the Official Bootstrap 3 Documentation to create the following static page elements on your index.html page:

- A Navbar element (using the main example) consisting of the following:
    - The "navbar-brand" text should read "Assignment 1 - Data"
    - **Do not** include the element: <ul class="nav navbar-nav">…
    - **Do not** include the element: <form class="navbar-form navbar-left">…</form>
    - The element: <ul class="nav navbar-nav navbar-right">… must consist of a **dropdown** menu, with a **dropdown-toggle** that reads "**Collection**" containing the following items:

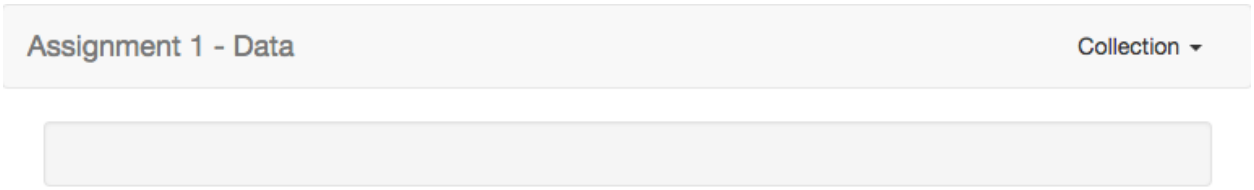
      ```
      <li><a href="#" id="teams-menu">Teams</a></li>
      <li><a href="#" id="employees-menu">Employees</a></li>
      <li><a href="#" id="projects-menu">Projects</a></li>
      <li><a href="#" id="positions-menu">Positions</a></li>
      ```

    - When complete, the navbar should look like the image below:



- Next, you must include A responsive grid with a single column (ie, "col-md-12" - see the notes on "Responsive Grid System" from Week 11 in WEB322) that contains a single "well" with id="data". Essentially, we're creating a horizontally centered container for our data.
- To ensure that the "well" doesn't grow too large when populated with data, add some CSS to your main.css file to ensure that it **does not** exceed **300px** high (HINT: use the **max-height** and **overflow-y** properties)

  When complete, the page should look like this:



Step 5: Accessing the API and updating the DOM

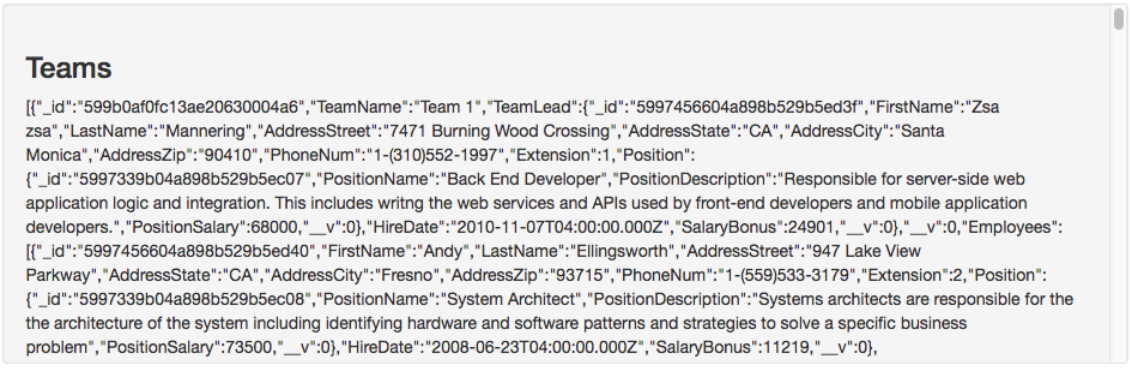
Now that we have a reliable WEB API that we can use to access data, as well as an HTML page with the correct controls and container, we can start writing code to fetch the data and update the DOM.

The first step is to create a DOM ready handler inside your index.js file to execute all of your jQuery code. Inside the ready callback, output some text to the console (ie: "jQuery working") so that you know that jQuery is working properly before you proceed.

Next, we need to wire up all 4 menu items in the "Collection" menu to fetch some data and update the DOM when clicked (HINT: use the "on" method for each element to bind the "click" event) according to the following specification:

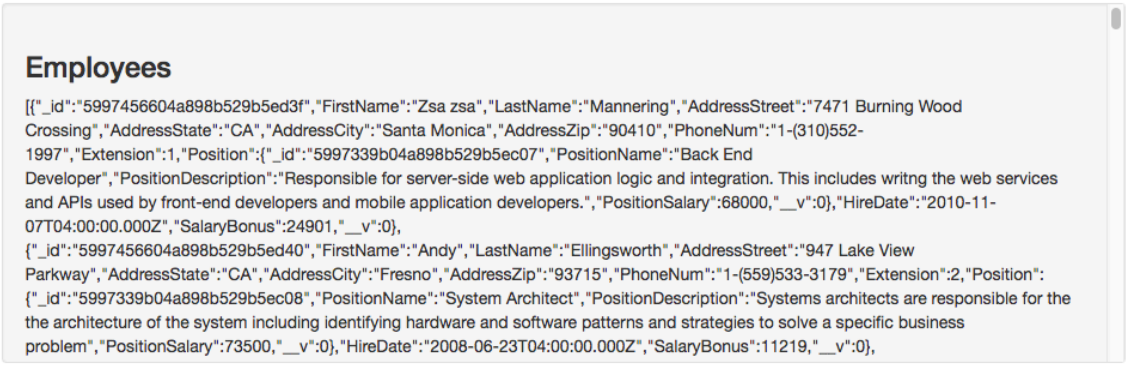
Teams is Clicked

- Prevent the "default action" using event.preventDefault(). This will stop the element from behaving like a regular link.
- Make an AJAX GET request to your Teams API hosted on Heroku to GET all teams (/teams). When this is "done":
    - Clear the contents of the "well" (id="data") element using the .empty() method
    - Add the element **<h3>Teams</h3>** to the "well"
    - Append the results of the AJAX query to the "well". HINT: Be sure to call JSON.Stringify(data) on the returned data, so that it can render as plain text in the browser
- When complete, it should look like this:



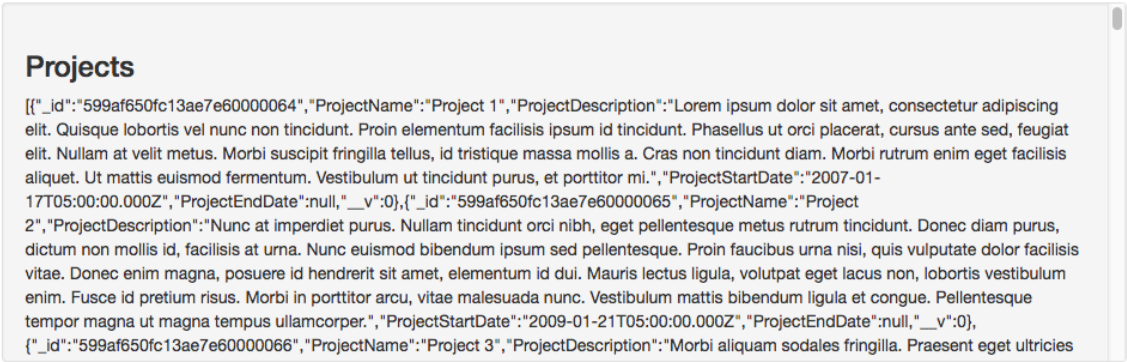
Employees is Clicked

- Prevent the "default action" using event.preventDefault(). This will stop the element from behaving like a regular link.
- Make an AJAX GET request to your Teams API hosted on Heroku to GET all employees (/employees). When this is "done":
    - Clear the contents of the "well" (id="data") element using the .empty() method
    - Add the element **<h3>Employees</h3>** to the "well"
    - Append the results of the AJAX query to the "well". HINT: Be sure to call JSON.Stringify(data) on the returned data, so that it can render as plain text in the browser
- When complete, it should look like this:



Projects is Clicked

- Prevent the "default action" using event.preventDefault(). This will stop the element from behaving like a regular link.
- Make an AJAX GET request to your Teams API hosted on Heroku to GET all projects (/projects). When this is "done":
    - Clear the contents of the "well" (id="data") element using the .empty() method
    - Add the element **<h3>Projects</h3>** to the "well"
    - Append the results of the AJAX query to the "well". HINT: Be sure to call JSON.Stringify(data) on the returned data, so that it can render as plain text in the browser
- When complete, it should look like this:



Positions is Clicked

- Prevent the "default action" using event.preventDefault(). This will stop the element from behaving like a regular link.

- Make an AJAX GET request to your Teams API hosted on Heroku to GET all positions (/positions). When this is "done":

    - Clear the contents of the "well" (id="data") element using the .empty() method

    - Add the element **<h3>Positions</h3>** to the "well"

    - Append the results of the AJAX query to the "well". HINT: Be sure to call JSON.Stringify(data) on the returned data, so that it can render as plain text in the browser

- When complete, it should look like this:

Step 6: Pretty Print your Output JSON

Now that we have the data showing in our web UI, let's improve the look by using a JSON pretty-printing JavaScript library:

https://github.com/center-key/pretty-print-json

Use the **pretty-print-json** library to properly indent and colour-code your JSON output in the page for each of the sections in your web page. Use the instructions in the **Browser** section to get jsdelivr.com CDN links for the CSS and JS files, then use the instructions under the **Usage** section to alter your previous code.

When finished, all JSON output should be properly indented, include whitespace, and be colour coded by data type. It should look something like this:

Teams


```
[
  {
    Projects: [
      {
        _id: "599af650fc13ae7e60000078",
        ProjectName: "Project 21",
        ProjectDescription: "Sed sed dui turpis. Praesent porta in massa ac tincidunt. Quisque gravida neque a
        ProjectStartDate: "2012-03-22T04:00:00.000Z",
        ProjectEndDate: null,
        __v: 0
      },
      {
```


Assignment Submission:

1. Add the following declaration at the top of your index.js file

/**
* WEB422 – Assignment 1
* 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: _____
*
*
**/

2. Run the `**npm run prepare-submission**` script to create the **assignment1-submission.zip** file, and upload this to Blackboard. See the **README.md** file for further instructions.

Important Note:

- **NO LATE SUBMISSIONS** for assignments. Late assignment submissions will not be accepted and will receive a **grade of zero (0)**.
- 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.
- After the end (11:59PM) of the due date, the assignment submission link on My.Seneca will no longer be available.