WEB422 Assignment 3

Submission Deadline:

Monday, October 14th @ 11:00pm

Assessment Weight:

9% of your final course Grade

Objective:

To practice writing client-side JavaScript code using the MVVM pattern using Knockout.js, jQuery and Bootstrap. We will work with our Teams-API to enable additional options for accessing and updating our data.

Specification:

For this assignment, we will be creating a friendly user interface to allow users to edit **existing** team data.   This includes changing the **Team Lead**, the current **Projects** as well as the **Members** (Employees) of each of the 15 teams in the system. We will use a "panel" driven interface provided by Bootstrap 3.3.7 to organize the form controls (<select> elements) for each team.   A jQuery Plugin will be used to make the controls much easier to use, and Knockout.js will provide two-way binding for quick updates. When it's complete, the interface should look like the following screenshot:

However, once the app is working as expected, please feel free to **add any extra design, Images or CSS to your solution**. Be creative - this is your app.

Getting Started (Dependencies):

This project will make use of a number of client-side dependencies, including: **Bootstrap (3)**, **jQuery**, **Knockout (and the "mapping" plugin)**, and a **jQuery Plugin - "Multiple Select"**.

The attached basic index.html boilerplate file can be used to start your project.

(**Note**: do not forget to update the **<title>** element)

This file assumes that you have done the following:

- Downloaded knockout-3.4.2.js and placed it in a "lib" folder within your solution folder

- Downloaded knockout.mapping-latest.js and placed it in a "lib" folder within your solution folder

- Downloaded multiple-select-master, extracted it and placed the complete "multiple-select-master" folder in a "lib" folder within your solution folder

- Included a main.js file in a "js" folder and a main.css file in a "css folder in your solution folder

- **Note**: The code in this assignment uses ES6 Promises (not currently supported in IE11 - https://kangax.github.io/compat-table/es6/). If this is a concern, the following "polyfill" can be used: https://github.com/taylorhakes/promise-polyfill

Updating index.html (part 1):

If you chose to use the above index.html boilerplate file, you will notice that the **<body>** element does not have any content (except for the **<script>** elements at the end). To get our solution looking somewhat like the sample, we need to add a few things:

<nav> element

For this assignment, you can use the exact same **<nav>** element as Assignment 2. However, the **"navbar-brand"** link should read: **Assignment 3 - Team Details**

Assignment 3 - Team Details

Generic Bootstrap "Modal" element

This assignment will once again rely on a generic Bootstrap modal window to show messages to the user. For this, you may use the same code from Assignment 2 (See: '**"Generic" Modal Window Container**' in the **Assignment 2** spec, if this was not completed).

Place your "modal" .html code at the bottom of your index.html file, before the <script> tags (as we did in Assignment 2)

Bootstrap Grid

The next thing that you should add is the framework for a responsive grid with a single column of width "**col-md-4**".
Recall: this will involve using **<div>** elements with the classes "**container**", "**row**", and "**col-md-4**" (See the WEB322 notes on "Responsive Grid System" if you require a refresher)

Bootstrap "Panel"

The main component that we will use to organize the User Interface will be the Bootstrap 3 Panel.

- Place the "**Panel heading without title**" example panel inside your "**col-md-4**" column

- In the "panel-heading" <div> element, place a **<strong>** element with the text "**[Team Name]**"

- In the "panel-heading" <div> element, place a **<button>** element with a class attribute of "**btn btn-primary btn-xs pull-right**" and the text "**Save**"

- In the "panel-body" <div> element, place the text "**[Team Info]**"

If you followed the instructions as above, your page should appear as:
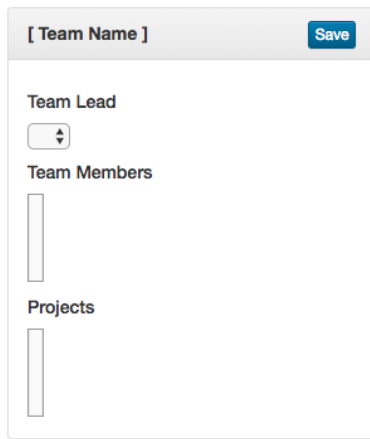
Form Controls

The primary form element that we will be using to interact with the Team data, is the <select> element. In the "panel-body" <div> element replace the "[Team Info]" text with the following elements:

- <h5> element with the text "Team Lead"
- <select> element with the class "single"
- <h5> element with the text "Team Members"
- <select multiple> element with the class "multiple"
- <h5> element with the text "Projects"
- <select multiple> element with the class "multiple"

If you added the elements in the above order, your panel should now look something like the below:

Nothing special, but remember we haven't wired up the elements to talk to our data yet. We will also be using our "multiple-select" jQuery plugin to dress up the <select> elements and make them more useable.

Updating main.js (part 1):

This is the primary JavaScript file that will do all the heavy lifting for our application. The following specifications will apply to this file:

Defining a viewModel

At the top of the file, add a new object called "viewModel" with the following **knockout "observable"** properties and values:

- property: "teams" - value: **empty array**, ie: []
- property: "employees" - value: **empty array**, ie: []
- property: "projects" - value: **empty array**, ie: []

Function showGenericModal(title,message)

This is the exact same function that you defined in your Assignment 2. (See: '**Function showGenericModal(title,message)**' in the **Assignment 2** spec, if this was not completed)

Function: initializeTeams()

The initializeTeams function is responsible for populating your observable "teams" property within your "viewModel" with data. It must adhere the following specifications:

- Return a Promise

- Make a **GET Request** using AJAX to your Teams API on Heroku using the "/teams-raw" route

  - If the request **completes successfully**, set the value of the "teams" property to the data returned from the AJAX call (using the **fromJS** method of the **ko.mapping** object, ie: **ko.mapping.fromJS(data)**) and **resolve** the promise

  - If the request **does not complete successfully**, **reject** the promise with the string: "Error loading the team data."

Function: initializeEmployees()

The initializeEmployees function is responsible for populating your observable "employees" property within your "viewModel" with data. It must adhere the following specifications:

- Return a Promise

- Make a **GET Request** using AJAX to your Teams API on Heroku using the "/employees" route

    - If the request **completes successfully**, set the value of the "employees" property to the data returned from the AJAX call (using the **fromJS** method of the **ko.mapping** object, ie: **ko.mapping.fromJS(data)**) and **resolve** the promise

    - If the request **does not complete successfully**, **reject** the promise with the string: "Error loading the employee data."

Function: initializeProjects()

The initializeProjects function is responsible for populating your observable "projects" property within your "viewModel" with data. It must adhere the following specifications:

- Return a Promise

- Make a **GET Request** using AJAX to your Teams API on Heroku using the "/projects" route

    - If the request **completes successfully**, set the value of the "projects" property to the data returned from the AJAX call (using the **fromJS** method of the **ko.mapping** object, ie: **ko.mapping.fromJS(data)**) and **resolve** the promise

    - If the request **does not complete successfully**, **reject** the promise with the string: "Error loading the 'project' data."

jQuery DOM "ready" function: $(function() { … });

The code inside our "ready" function will be responsible for invoking all our "initialize" methods as defined above and applying the knockout bindings. It will also update our existing <select> elements using the multiple-select plugin:

- Create a promise chain by invoking the "**initializeTeams()**" method, "**then**" providing the "**initializeEmployees**" method, "**then**" providing the "**initializeProjects**" method, "**then**" providing an **anonymous function** that performs the following functions:

    - Use knockout to apply the bindings (applybindings) to the document using the "viewModel" (defined at the top of our file

    - Use jQuery to select all "select" elements with class "multiple" and invoke the following method: **.multipleSelect({ filter: true });**

    - Use jQuery to select all "select" elements with class "single" and invoke the following method: .**multipleSelect({ single: true, filter: true });**

- At the end of the promise chain, provide a "**catch**" function that takes the message from the rejected promise and provides it to a generic error modal as the **message** value. The **title** of the modal should read "Error"

- **NOTE:** If you require a refresher on creating promises / chaining promises see, **Promises** and **Chaining Promises** from the WEB322 Week 3 Notes

Updating main.css:

If you check on your index.html file in the browser, you shouldn't see any errors, but you shouldn't see anything exciting either. Before we continue to update our index.html and add data-bind attributes to our elements, we should first clean up the <select> elements using CSS.
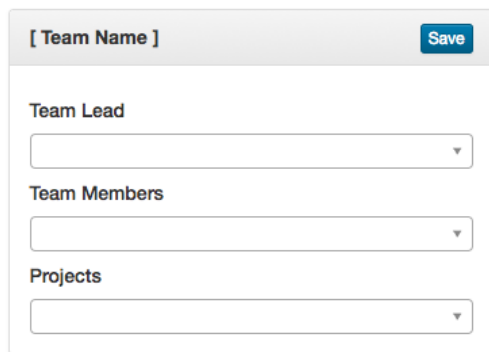
In your main.css file, add the following .css:

.ms-parent{
 width: 100% !important;
}

.ms-drop input[type="checkbox"]{
 margin-right: 8px;
}

.ms-drop input[type="radio"]{
 margin-right: 8px;
}

This should fix the <select> elements and (once your AJAX requests have successfully completed), your "Panel" should look like this:

Updating index.html (part 2):

In order to get our view actually rendering data from our viewModel, we must add "data-bind" attributes to certain elements and add other elements:

- Create a new <div> element to surround your "col-md-4" column with the following properties:
    - property: "style", value: "display: none"
    - property: "data-bind", values:
        - use the "visible" binding with the value of "true" (this is a trick in Knockout.js to prevent that flash of html content that occurs before our bindings are applied)
        - use the "foreach" binding with our viewModel "teams" array
- *Test the page* (you should see 15 empty Panels)
- Delete the text "[Team Name]" from the <strong> element in the "panel-heading" and add the following property to the <strong> element:
    - Property: "data-bind", value:
        - use the "text" binding to show the "TeamName" property of the current team
- *Test the page* (you should see 15 Panels with the correct team names)

- In the "single" <select> element, below "Team Lead", add the following property:

  - Property: "data-bind", values:

    - use the "value" binding to set the selected item to the "TeamLead" property of the current team

    - use the "options" binding to bind the <option> elements to the contents of the viewModel "employees" array (**HINT:** you can use **$parent.employees** here)

    - use the "optionsText" binding to use the "FirstName" and "LastName" properties of the "$parent.employees" objects as text for the generated <option> elements. **HINT:** you can use the following inline function here:
      function(item) { return ko.unwrap(item.FirstName) + ' ' + ko.unwrap(item.LastName) }

    - use the "optionsValue" binding to use the "_id" properties of the "$parent.employees" objects as the value for the generated <option> elements. **HINT:** you can use the following inline function here: **function(item) { return item._id }**

- *Test the page* (you should see 15 Panels with the correct Team Lead!)

- In the "multiple" <select> element, below "Team Members", add the following property:

  - Property: "data-bind", values:

    - use the "selectedOptions" binding to set the selected items to the "Employees" property of the current team

    - use the "options" binding to bind the <option> elements to the contents of the viewModel "employees" array (**HINT:** you can use **$parent.employees** here)

    - use the "optionsText" binding to use the "FirstName" and "LastName" properties of the "$parent.employees" objects as text for the generated <option> elements. **HINT:** you can use the following inline function here:
      function(item) { return ko.unwrap(item.FirstName) + ' ' + ko.unwrap(item.LastName) }

    - use the "optionsValue" binding to use the "_id" properties of the "$parent.employees" objects as the value for the generated <option> elements **HINT:** you can use the following inline function here: **function(item) { return item._id }**

- *Test the page* (you should see 15 Panels with the correct Team Members!)

- In the "multiple" <select> element, below "Projects", add the following property:

  - Property: "data-bind", values:

    - use the "selectedOptions" binding to set the selected items to the "Projects" property of the current team

    - use the "options" binding to bind the <option> elements to the contents of the viewModel "projects" array (**HINT:** you can use **$parent.projects** here)

    - use the "optionsText" binding to use the "ProjectName" property of the "$parent.projects" objects as text for the generated <option> elements. **HINT:** you can use the following inline function here: **function(item) { return item.ProjectName }**

    - use the "optionsValue" binding to use the "_id" properties of the "$parent.projects" objects as the value for the generated <option> elements **HINT:** you can use the following inline function here: **function(item) { return item._id }**

- *Test the page* (you should see 15 Panels with the correct Projects!)

- In the "save" <button> element, add the following property:

  o Property: "data-bind", values:

    ▪ use the "click" binding to invoke the method "saveTeam" (defined below)

Updating main.js (part 2):

Function: saveTeam()

The saveTeam function is responsible for sending the updated team data to the correct route in the API. To ensure that this works correctly, it must adhere to the following specifications below.

Note: "this" in the context of this function will be a single observable "team" object from our **viewModel.teams** array - this is because this function is invoked from a "click" binding from the view (index.html).

- Set the value of **this** to a local variable, ie: "currentTeam"
- Make a **PUT Request** using AJAX to your Teams API on Heroku using the "/team/:teamId" route:

  o Use the _id property of the "currentTeam" as the ":teamId" (**Recall**: to access the value of an "observable" property, we need to invoke it as a function with no parameters, ie: **_id()**)

  o For the "data", create an object literal with the following properties:

    ▪ Property: "Projects", value: the "Projects" value of the "currentTeam"

    ▪ Property: "Employees", value: the "Employees" value of the "currentTeam"

    ▪ Property: "TeamLead", value: the "TeamLead" value of the "currentTeam"

    NOTE: Do not forget to "stringify" your new object literal using **JSON.stringify().**

  o If the request **completes successfully**, show a **generic modal** with the **title:** "Success" and **message** "[TeamName] Updated Successfully", where [TeamName] is the "TeamName" of "currentTeam"

- If the request **does not complete successfully**, show a **generic modal** with the **title:** "Error" and **message** " Error updating the team information."

Assignment Submission:

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

```
/*********************************************************************************
* WEB422 – Assignment 03
* 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** (ie, .html, .css/.scss, .js, etc. files).
- Submit your compressed file to My.Seneca under **Assignments** -> **Assignment** 3

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 (11:00PM) of the due date, the assignment submission link on My.Seneca will no longer be available and submissions will not be accepted.

- 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.

- **Allow enough time to upload or deal with unexpected issues. Do not wait for the last moment** since there are transmission time/queuing delay/processing time etc. from your machine to the Blackboard server.

- Late penalty is 100%.

  Good luck!