WEB422 Assignment 3

Submission Deadline:

Monday June 17th @ 11:59pm

Assessment Weight:

9% of your final course Grade

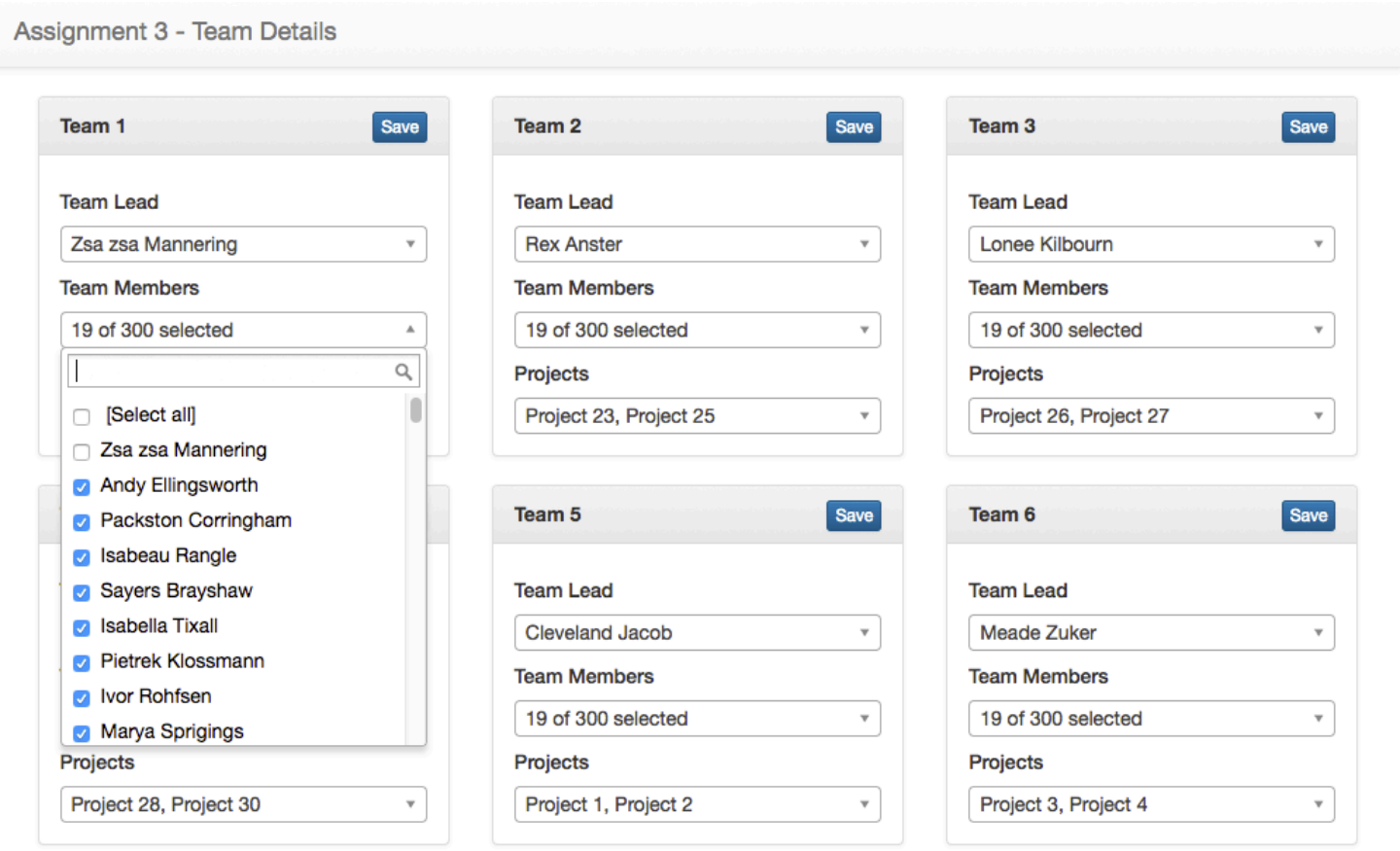
Objective:

To practice writing client-side JavaScript code using the MVVM pattern, build and use Component Architectures, and work with one of Vue.js or React. 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 card/panel-style interface for each Team's information.

Here's a general guideline on how things should look. NOTE: this is only a guide, not a required layout/UI; your app should be different from other students', depending on the components you choose, and your CSS:



Getting Started (Dependencies):

First, decide between **Vue.js** and **React**. You can use either, but need to **pick only one**. Depending on your choice, use either the **Vue CLI** or **create-react-app** to build your application.

Second, decide on a **layout system**. You can continue using Bootstrap if you wish, or try something else, like Google's Material design. You can either hand-code everything in your component's HTML/CSS yourself, or use pre-made layout components:

- Vue
  - https://bootstrap-vue.js.org/
  - https://vuematerial.io/
  - https://vuetifyjs.com/en/
- React
  - https://react-bootstrap.github.io/
  - https://material-ui.com/

If you find another UI toolkit for Vue/React that you want to use, that's fine too.

Third, you'll need some way to show **multiple items in a select component**. There are lots of existing components you can use, for example:

- Vue
  - https://vue-multiselect.js.org/
  - https://mdbootstrap.com/docs/vue/forms/multiselect/
- React
  - https://react-select.com/home
  - https://github.com/Khan/react-multi-select

Requirements:

Rather than specify every aspect of how you should write your code, and what the final project should look like, you are encouraged to experiment, and think through how to build this app. Below are a list of requirements for things that your app must have or do:

1. Include a header/nav across the top of the page that says **"Assignment 3 - Team Details"**, similar to your earlier assignments. Make it a different colour, font size, etc. from the rest of your page.
2. Use a "grid" style layout for all Team "cards". Make it responsive so that it will show as many as can fit within a given screen width. Also consider how it looks on mobile.
3. Create a Team Component that manages/displays each of the individual team info cards. You are free to use/create other Components within this component (i.e., it doesn't need to be a single, giant component):
    1. A Team Component should use a "card" or "panel" style, with a header and body. Include the **Team Name** in the header. You should also include a Button to "Save" or "Update" the Team

when the user changes any of the information. Only enable or show the button if it is necessary (i.e., if the data has changed).

2. A Team Component's body should have a set of form elements to edit the Team information, including: **Team Lead, Team Members, Projects**. Use appropriate form controls/Components, including a suitable **Multi-Select** component for the Team Members and Projects lists.

3. **Bonus**: consider giving your Team Component two states: a View state, and an Edit state. In the View state, provide a button or icon to enter the Edit state (e.g., a Pencil icon that when clicked, changes all text to form elements). In the Edit state, allow the user to click Save or Cancel to exit the Edit state.

4. Use your Teams API to get all the data necessary for your app. When your app starts, download this data using **fetch()** or **axios** (https://github.com/axios/axios) or some other appropriate "XHR" library for your chosen framework. Your app should manage the teams, employees, and projects state data. Use the following routes:

    1. Teams: **/teams-raw** (NOTE: /teams will return full Objects for things like Employees, Projects, use /teams-raw to get back data with _id values)

    2. Employees: **/employees**

    3. Projects: **/projects**

5. Include appropriate **error messages** if network requests fail. You could use a Toast style popup, or show a banner at the top of the app, or use a modal dialog. If any of your network requests ever fail, make sure the user is given enough information to decide what to do next. Using console.log() is not sufficient.

6. Once all your data is available, connect your downloaded teams, employees, and projects state to display your Team Components in your UI grid. Use data binding to allow users to update any values in the UI.

7. Allow the user to **Save/Update** the data for a Team by editing the form fields, and then clicking a button. Doing so should send the updated data to the correct route in the Teams API using a **PUT** HTTP request.

    1. Use the **/team/:teamId** route, passing the _id for the current Team. When the update is successful, indicate to the user that things worked (e.g., use a green colour in the header or show a checkmark or use some other visible action). If it fails, indicate that as well.

    2. The data sent in the body of the PUT request should match the format expected by the server (see teams-api server.js /team/:teamId, data-service.js updateTeamById(), and models/team.js). Specifically:

        - **TeamName**: String
        - **TeamLead**: _id value for the Employee leading the project
        - **Projects**: Array of Project _id values
        - **Employees**: Array of Employee _id values

8. Make sure there are no errors or warnings in the browser console when you run your app.

9. Include a **README.md** file in your project with details about how to run and build your app, and information about all the various components, libraries, and other third-party tools you used. Please also list any extra features you added, and bugs or limitations you are aware of in your app.

10. Bonus: consider adding and using eslint and prettier in your package.json file to keep your code tidy and free of bugs.

Assignment Submission:

- Add the following declaration at the top of your app's main JavaScript 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: _____
*
*********************************************************************************/
```

- **Build your project** into a final web site in dist/ using `**npm run build**`
- **Delete the node_modules/** folder (it will be massive, and increase the size of your .zip by too much)
- Create an assignment3.zip file containing your project's root folder (i.e., unzipping assignment3.zip should create a folder containing your project tree).
- 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 (11:59PM) 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.