WEB422 Assignment 4

Submission Deadline:

Friday, July 6th, 2017 @ 11:59 PM

Assessment Weight:

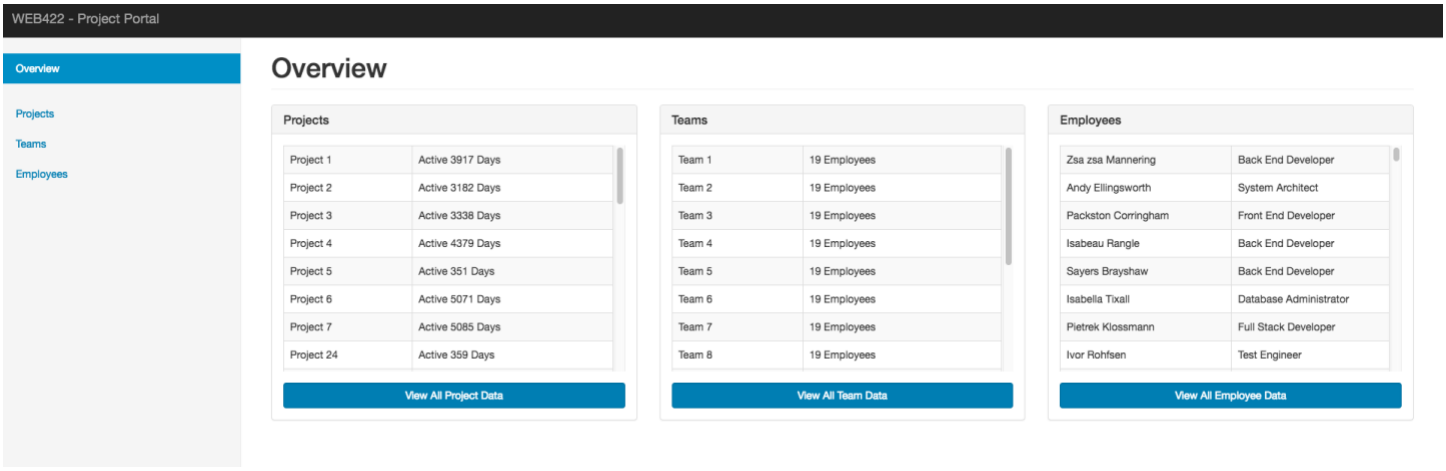
10% of your final course Grade

Objective:

To work with React and practice working with Component based architecture. We will continue to connect to our Teams API as a data source for our app. For an exemplar on how to retrieve Teams API data (using axios and the "componentDidMount" lifecycle method), refer to the Week 5 example here: https://github.com/sictweb/web422/blob/master/Code%20Examples/week5/my-app/src/Employees/Employees.js

Specification:

For this application, we will be working entirely in React. Our application will feature 4 routes: Overview, Projects, Teams & Employees. Each route will be responsible for rendering information from our Teams API (often in a <table> format) - we will *not* be updating the data at this time using forms. When complete, the main structure of the app and the "Overview" view will look like the following:

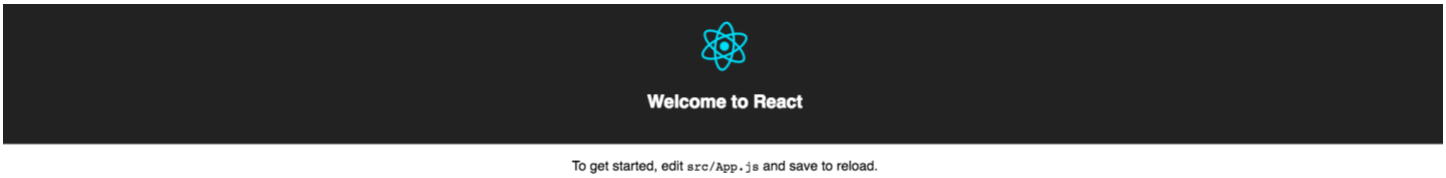


Please Note, once the app is working as expected, please feel free to **add any extra design, Images or CSS to your solution**. Be creative - this is your app.

Getting Started:

To get started, create a new React app using the command "create-react-app" - if "create-react-app" is not available on your system, you can use the command "npm install -g create-react-app".

Once you have created your new React app - open the folder in Visual Studio Code and run the command "npm start" to get the React development server running. This should open your default web browser and show you http://localhost:3000:



At this point, you may go ahead and **delete the files**:

- App.test.js
- App.css

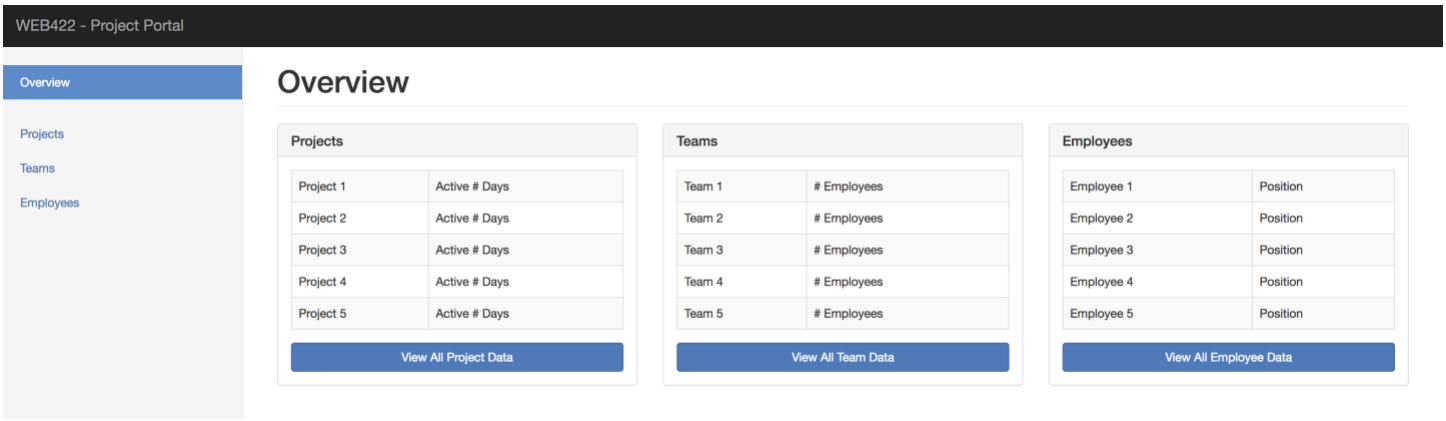
Next, use the code at the following links to **replace** your App.js & index.css files (this will give you a solid starting point and a design to work from):

- https://scs.senecac.on.ca/~patrick.crawford/shared/winter-2018/web422/A4/App.js
- https://scs.senecac.on.ca/~patrick.crawford/shared/winter-2018/web422/A4/index.css

Once this is complete, you need to update your "public/index.html" file to include the following <link> & <script> tags in their appropriate positions, ie: <link> element **before** '<link rel="manifest" href="%PUBLIC_URL%/manifest.json">') and <script> elements **before** </body>:

- <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
- <script src="https://code.jquery.com/jquery-3.2.1.min.js" integrity="sha256-hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNgt4=" crossorigin="anonymous"></script>
- <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa" crossorigin="anonymous"></script>

When you have made the above updates, your app should look like the following:



Dependencies:

This project will make use of several additional dependencies available from npm including:

- axios: https://www.npmjs.com/package/axios
- moment: https://www.npmjs.com/package/moment
- react-router-dom: https://www.npmjs.com/package/react-router-dom

At this point, it makes sense to "npm install --save …" each of the above modules, since you will need them in your application

Creating Your Components:

As we know from class, the main building block for all React applications, is the "Component". The following sections of the specification will outline each component that must be built to get your application working properly.

NOTE: All Components created for this application must exist **in their own file** (ie: ComponentName.js). The First Step is to create the following components based on the existing code within your App.js. We will shrink down the "App" component to use all of the new components

NavBar

The NavBar is the simplest component. It simply renders the <nav> element from our current "App".

SideBar

The SideBar component simply renders the "sidebar" element from our current App, ie: the " … element. Additionally, it accepts the following property:

- "highlight" - this property determines which "nav-sideber" element is "active" (it could be "Overview", "Projects", "Teams" or "Employees". For example, if the "highlight" property is "Projects", then the Projects will get the class "active"

  ie: <li class="active">Projects

  To conditionally add the class "active" to an <li> element (for the "Projects" view), you can use the code:
  <li className={(this.props.highlight === "Projects" ? 'active' : '')}>

MainContainer

The MainContainer component wraps everything that a page needs into its own component. This includes a <NavBar /> and <SideBar /> component. It must account for any "children" that may be added to the component (using this.props.children). Additionally, it accepts the following property:

- "sidebar" - this property determines the value that needs to be passed as the "highlight" property for the <SideBar /> element. This way, we can pass a "sidebar" value to the "MainContainer" component and be assured that the matching "SideBar" will be highlighted.

When completed, your "MainContainer" should include the (structural) code (see comments for hints):

```
<div>
  <NavBar />
```

```
<div className="container-fluid">
  <div className="row">
    <SideBar /> // Add the correct "highlight" property here
    <div className=" col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
      // be sure to add a reference to the "children" here
    </div>
  </div>
</div>
</div>
```

If you've updated your code correctly so far, your <App> component's render() method should return something like this

```
<MainContainer>
        <h1 className="page-header">Overview</h1>
        <div className="row">
        …
        </div>
</MainContainer>
```

ProjectsPanel

You will notice that there's a that has the "panel-title" of "Projects". This should really be its own "ProjectsPanel" component, since it's dealing with data. For this component, you will need to include the following modules

- "axios"
- "moment"

Additionally, this component must adhere to the following specifications:

- Has a "state" that includes a "projects" array
- Uses "axios" to "get" all projects from your Teams API (Heroku) and assign the results to the component state "projects" array (**Hint**: Perform this operation in the "componentDidMount()" lifecycle method)
- Renders a "Panel", where the "panel-title" is "Projects"
- The "panel-body" is a Bootstrap (responsive) table with 2 columns:
    - Project Name
    - Active # Days (HINT: You can use the moment().diff() method with the 'days' option here)
- Under the table, the panel must continue to include the "View All Project Data" link in the "panel-body"

TeamsPanel

You will notice that there's a that has the "panel-title" of "Teams". This should really be its own "TeamsPanel" component, since it's dealing with data. For this component, you will need to include the following module

- "axios"

Additionally, this component must adhere to the following specifications:

- Has a "state" that includes a "teams" array

- Uses "axios" to "get" all teams from your Teams API (Heroku) and assign the results to the component state "teams" array (**Hint**: Perform this operation in the "componentDidMount()" lifecycle method)
- Renders a "Panel", where the "panel-title" is "Teams"
- The "panel-body" is a Bootstrap (responsive) table with 2 columns:
    - Team Name
    - # Employees (this is the number of employees on the team)
- Under the table, the panel must continue to include the "View All Team Data" link in the "panel-body"

EmployeesPanel

You will notice that there's a that has the "panel-title" of "Employees". This should really be its own "EmployeesPanel" component, since it's dealing with data. For this component, you will need to include the following module

- "axios"

Additionally, this component must adhere to the following specifications:

- Has a "state" that includes a "employees" array
- Uses "axios" to "get" all employees from your Teams API (Heroku) and assign the results to the component state "employees" array (**Hint**: Perform this operation in the "componentDidMount()" lifecycle method)
- Renders a "Panel", where the "panel-title" is "Employees"
- The "panel-body" is a Bootstrap (responsive) table with 2 columns:
    - Full Employee Name (First Name + Last Name)
    - Position Name (ie: "Full Stack Developer")

Under the table, the panel must continue to include the "View All Employee Data" link in the "panel-body"

Overview

If you have been updating your <App> component to use your new components, its "render" method should return the following (this is much more optimized than what was originally provided):

```
<MainContainer>
  <h1 className="page-header">Overview</h1>
  <div className="row">
    <div className="col-md-4">
      <ProjectsPanel />
    </div>
    <div className="col-md-4">
      <TeamsPanel />
    </div>
    <div className="col-md-4">
      <EmployeesPanel />
    </div>
  </div>
</MainContainer>
```

Since this is really the "Overview" view, it should be placed in its own "Overview" component. The only change that needs to be made from the above JSX is that we must provide the text "Overview" to the "MainContainer"'s "sidebar" property, ie: <MainContainer sidebar="Overview">…</MainContainer> to ensure that the correct "sidebar" gets highlighted.

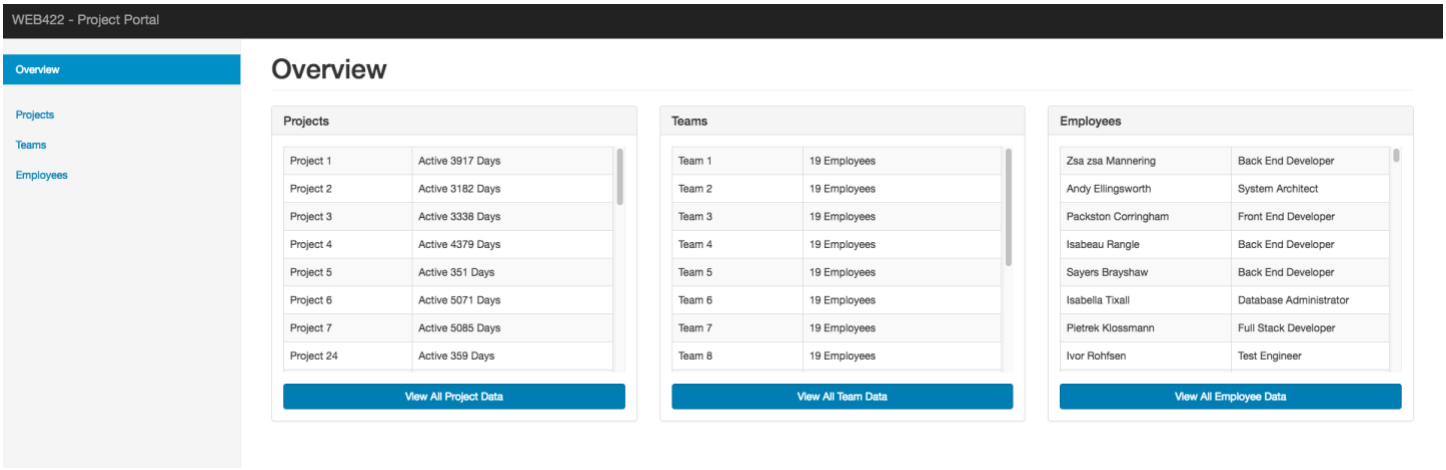
NOTE: For this component, you will need to include the following modules

- "MainContainer"
- "ProjectsPanel"
- "TeamsPanel"
- "EmployeesPanel"

Once this is complete, feel free to replace the entire return() statement for the <App> component, with your new "Overview" component for testing, ie:

```
class App extends Component {
  render() {
    return (
      <Overview />
    );
  }
}
```

Take a look in the browser - this should give you a working "Overview":



Creating Your Components pt II:

Now that your main building blocks are constructed, it's time to add the high level views (used in your routes), including:

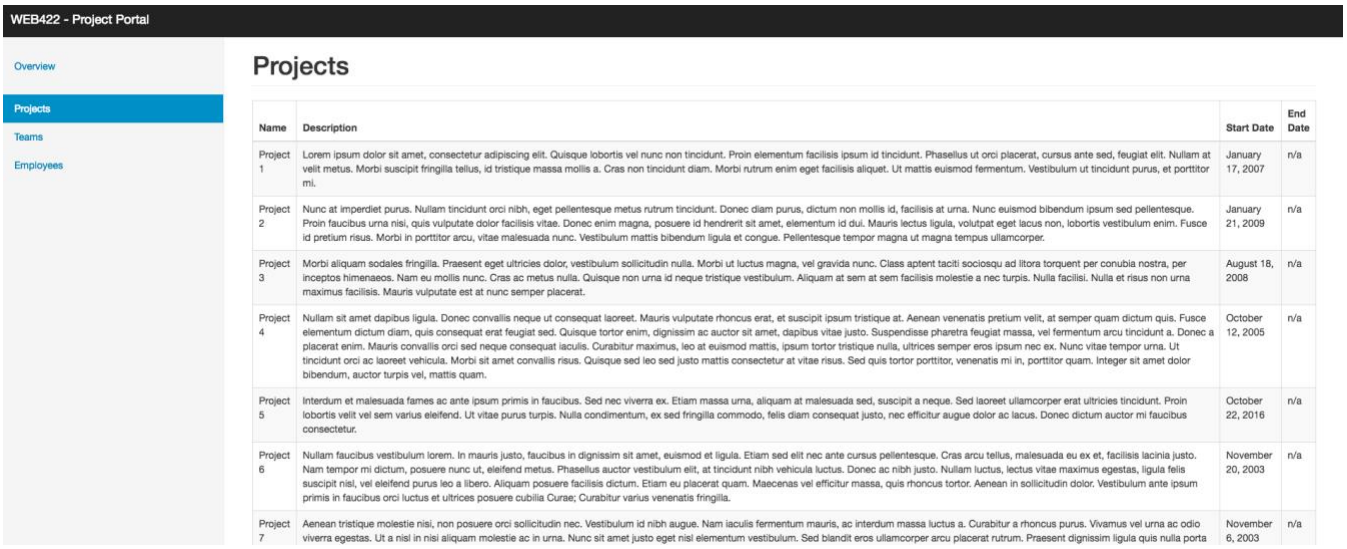
Projects

Like the "Overview" route - this route's "render()" method will be wrapped in a <MainContainer>…</MainContainer> Component. However instead of sidebar="Overview", the sidebar property will instead pass "Projects", ie <MainContainer sidebar="Projects">…</MainContainer>.

Also, the "page-header" must read "Projects".

This Component must also adhere to the below specifications:

- For this component, you will need to include the following modules
  - "axios"
  - "moment"
  - "MainContainer"
- This component has a "state" that includes a "projects" array
- Uses "axios" to "get" all projects from your Teams API (Heroku) and assign the results to the component state "projects" array (**Hint**: Perform this operation in the "componentDidMount()" lifecycle method)
- It must render a responsive bootstrap table: <table className="table table-striped table-bordered">…</table> with the following columns:
  - "Name" (the Project Name)
  - "Description" (the Project Description)
  - "Start Date" (the Project Start Date - HINT: Use "moment" to format the date using "LL")
  - "End Date" (the Project End Date - if this doesn't exist or is null, simply render "n/a", otherwise use "moment" to format the date using "LL")
- Once Complete, your component should look like:



Teams

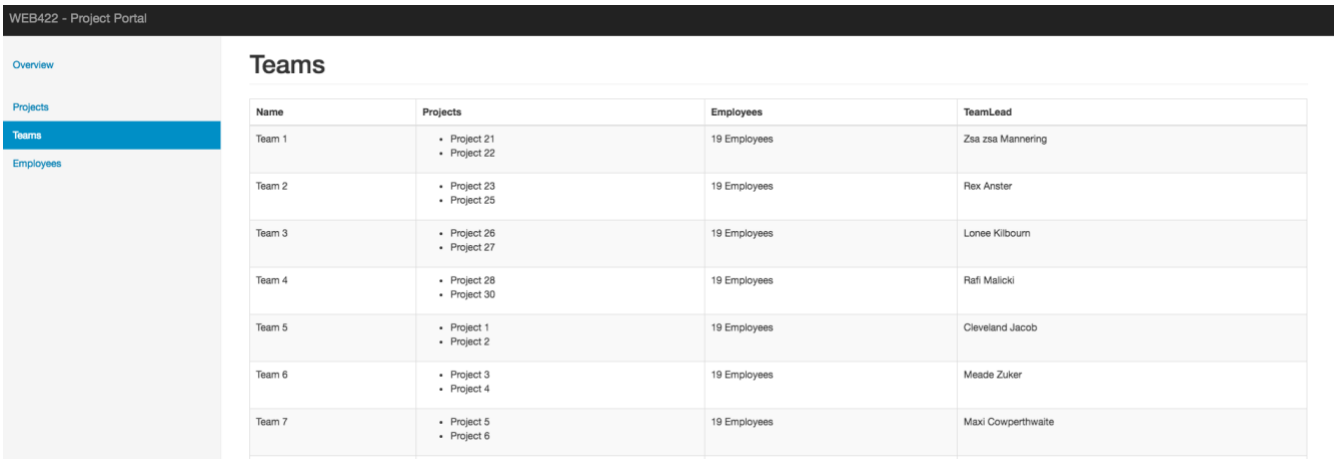
Like the "Overview" route - this route's "render()" method will be wrapped in a <MainContainer>…</MainContainer> Component. However instead of sidebar="Overview", the sidebar property will instead pass "Teams", ie <MainContainer sidebar="Teams">…</MainContainer>.

Also, the "page-header" must read "Teams".

This Component must also adhere to the below specifications:

- For this component, you will need to include the following modules
  - "axios"
  - "MainContainer"
- This component has a "state" that includes a "teams" array

- Uses "axios" to "get" all teams from your Teams API (Heroku) and assign the results to the component state "teams" array (**Hint**: Perform this operation in the "componentDidMount()" lifecycle method)
- It must render a responsive bootstrap table: <table className="table table-striped table-bordered">…</table> with the following columns:
    - "Name" (the Team Name)
    - "Projects" (an "unordered list" of all projects that the team is responsible for)
    - "Employees" (The number of employees on the team)
    - "Team Lead" (The full name (First & Last) of the Team Lead)
- Once Complete, your component should look like:



Employees

Like the "Overview" route - this route's "render()" method will be wrapped in a <MainContainer>…</MainContainer> Component. However instead of sidebar="Overview", the sidebar property will instead pass "Employees", ie <MainContainer sidebar="Employees">…</MainContainer>.

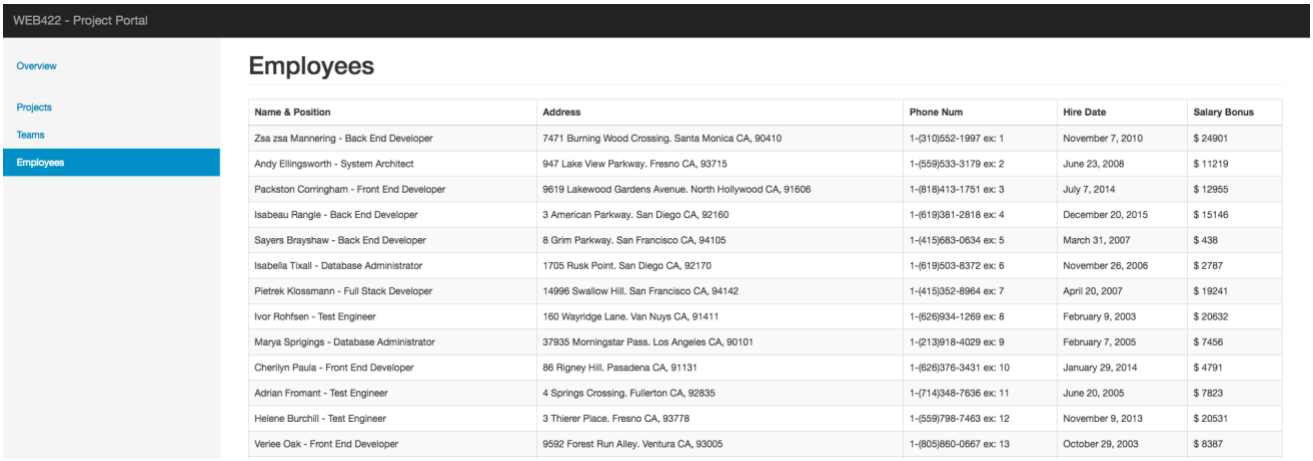
Also, the "page-header" must read "Employees".

This Component must also adhere to the below specifications:

- For this component, you will need to include the following modules
    - "axios"
    - "moment"
    - "MainContainer"
- This component has a "state" that includes an "employees" array
- Uses "axios" to "get" all employees from your Teams API (Heroku) and assign the results to the component state "employees" array (**Hint**: Perform this operation in the "componentDidMount()" lifecycle method)
- It must render a responsive bootstrap table: <table className="table table-striped table-bordered">…</table> with the following columns:
    - "Name & Position" (The full name and position of the employee)
    - "Address" (The full address of the employee)
    - "Phone Num" (The phone number + extension of the employee)
    - "Hire Date" (The hire date of the employee (HINT: Use "moment" to format the date using "LL")

- "Salary Bonus" (The employee's salary bonus, including a "$")

- Once Complete, your component should look like:

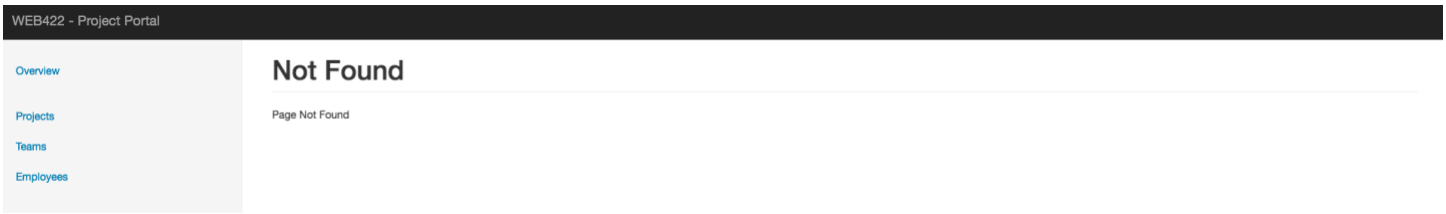


NotFound

The "NotFound" is a simple route that we will use whenever a route is not identified within the app. Like the "Overview" route - this route's "render()" method will be wrapped in a <MainContainer>…</MainContainer> Component (without a "sidebar" property).

Also, the "page-header" must read "Not Found" and the text below should simply be a element with the text: "Page Not Found".

Once Complete, your component should look like:



Creating Routes

All of our custom components are finally complete. Now we must register the routes so that our application knows how to handle navigation between routes.

- Update your "index.js" ReactDOM.render() line to wrap your <App /> component in a <BrowserRouter> element, ie:

```
<BrowserRouter>
        <App />
</BrowserRouter>
```

- (Don't forget to import {BrowserRouter} from 'react-router-dom')

Next, we must update our "App" component to register the routes according to the following specification (NOTE: Refer to the Week 5 notes on "React Routing" for further details on how to wire the below routes):

- "/" will render <Overview />
- "/projects" will render <Projects />
- "/teams" will render <Teams />
- "/employees" will render <Employees />
- Any other Route will render <NotFound />

Updating all internal Links

The last step is to ensure that all elements use the <Link> component to prevent the whole page from reloading between routes. This can be done by simply updating any anchor elements that refer to internal routes, ie:

- Everywhere you see …, replace the code with <Link to="…">…</Link>

This should eliminate any full-page reloads between routes.

HINT: Do not forget to add the line: "import {Link} from 'react-router-dom';" in any components that require the "Link" component

Assignment Submission:

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

```
/**********************************************************************************
* WEB422 – Assignment 04
* 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 all files in your Visual Studio code folder **EXCEPT the node_modules folder** (this will just make your submission unnecessarily large, and all your module dependencies should be in your package.json file anyway).
- Submit your compressed file (without the node_modules folder) to My.Seneca under **Assignments** -> **Assignment** 4

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.