WEB422 Assignment 3

Submission Deadline:

Friday, February 18th @ 11:59pm

Assessment Weight:

9% of your final course Grade

Objective:

To continue to work with our "Restaurants" API (from Assignment 1) on the client-side to produce a rich user interface for accessing data. For this assignment, we will be leveraging our knowledge of React to create an interface for *viewing* restaurants. **Please Note**: You're free to style your app however you wish, however the following specification outlines how we can use the React-Bootstrap Components (Bootstrap 4). If you wish to add additional images, styles or functionality, please go ahead.

Sample Solution:

You can see a video of the solution running at the link below:

https://pat-crawford-sdds.netlify.app/shared/winter-2022/web422/A3/A3.mp4

Step 1: Creating a React App & Adding 3rd Party Components

The first step is to create a new directory for your solution, open it in Visual Studio Code and open the integrated terminal:

- Next, proceed to create a new react app by using "create-react-app" using the command "npx create-react-app" followed by the identifier for your app, ie: "my-app".
- Once the tool has finished creating your React App, be sure to "cd" into your new app directory and install the following modules using npm:
    - react-router-dom
    - react-bootstrap@1.6.4 bootstrap@4.6.0
    - react-router-bootstrap
    - react-leaflet leaflet
- To ensure that our newly-added Bootstrap components render properly, we must import the correct CSS file to our "src/index.js" file, ie:
    - import 'bootstrap/dist/css/bootstrap.min.css';
- Similarly, to ensure that our maps / markers are rendered correctly, we must include the leaflet CSS. This can be done by adding the following line within the <head>…</head> element of our "public/index.html" file:

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

- Next, we must clear out (delete) the CSS content from both "App.css" and "index.css", since we won't be using any of those rules within our new app. Once this is complete, place the following single line of CSS within the "index.css" file (this will ensure that any table with class "table-hover" has the "Pointer" cursor:

  .table-hover tr:hover{ cursor:pointer; }

Step 2: Route Component "Skeletons"

For the next part of the assignment, we'll add all of the components that our routes will use as well as create a navbar that persists across all routes to help the user navigate our app.

To Begin, add the following components in separate files (using the naming convention ***ComponentName.js***) within the "src" directory.

- **Restaurants** – outputs: Restaurants query: ***query*** where ***query*** is a value that we will retrieve by using the "useLocation" hook
- **Restaurant** – outputs: Restaurant id: ***id*** where ***id*** is a value that we will retrieve by using the "useParams" hook
- **About** – outputs About
- **NotFound** – outputs <h1>Not Found</h1> We can't find what you're looking for... (**Note**: please feel free to change this up to something more fun).

Step 3: App Component & NavBar

Before we add routing to our application, we must first do some work with the "App" component:

- To begin, add the following import statements
  - import { Button, Col, Container, Form, FormControl, Nav, Navbar, Row } from '**react-bootstrap**';
  - import { Navigate, Route, Routes } from '**react-router-dom**';
  - import { LinkContainer } from '**react-router-bootstrap**';
- Next, remove all of the contents from the return statement and instead simply return "null" (we will change this soon)
- Next, delete the import for "logo" – we won't be using the "logo" for our application.
  - You can also delete the src/logo.svg file
- Now we must add the following "state" value to our App component:
  - **searchString / setSearchString** – default value: ""
- Before we are done, we must add the following "handleSubmit(e)" function to the App component according the following specification:

- Prevents the default action for the ***event*** (parameter ***e***)
- Navigates to the route: /restaurants?borough=***searchString*** where ***searchString*** is the value in the state (above) – **HINT:** You can use the "useNavigate" hook – see: "'Redirecting' to a Route" https://web422.ca/notes/react-routing
- Resets the value of **searchString** to ""

This will enable us to filter our restaurants by a user-entered "borough" value.

- Finally, use the following JSX code to replace the return value of our App component. This will show a navbar built using React-Bootstrap components.

```
<>
 <Navbar bg="light" expand="lg">
  <LinkContainer to="/">
   <Navbar.Brand>New York Restaurants</Navbar.Brand>
  </LinkContainer>
  <Navbar.Toggle aria-controls="basic-navbar-nav" />
  <Navbar.Collapse id="basic-navbar-nav">
   <Nav className="mr-auto">
    <LinkContainer to="/restaurants">
     <Nav.Link>Full List</Nav.Link>
    </LinkContainer>
    <LinkContainer to="/about">
     <Nav.Link>About</Nav.Link>
    </LinkContainer>
   </Nav>
   <Form onSubmit={handleSubmit} inline>
    <FormControl type="text" placeholder="Borough" className="mr-sm-2" value={searchString}
onChange={(e) => setSearchString(e.target.value)} />
    <Button type="submit" variant="outline-success">Search</Button>
   </Form>
  </Navbar.Collapse>
 </Navbar>
 <br />
</>
```

Step 4: Adding Routes

With our Navbar in place and our App component all set up, we can now proceed to add the routes to our application:

- In your src/**index.js** file, import the "BrowserRouter" component and use it to wrap the <App /> Component (ie: the <App /> Component will be the only child of the <BrowserRouter></BrowserRouter> Component
- Next, in the return value for the <App /> Component, add the following code **beneath** the navbar:

```
<Container>
 <Row>
  <Col>
```

```
  </Col>
 </Row>
</Container>
```

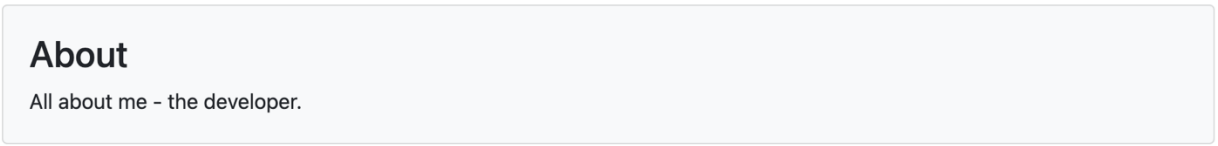
The above code simply sets up a Bootstrap grid with a single column

- Inside the <Col></Col> Component, add your <Routes></Routes> Component as well as configuration for the following routes:
    - "/" – Redirects to the "/Restaurants" Route
    - "/about" – Renders the <About /> Component
    - "/Restaurants" – Renders the <Restaurants /> Component
    - "/Restaurant/***id***" – Renders the <Restaurant /> Component
      (note: ***id*** is a route parameter, used to specify a specific restaurant)
    - (No Route) – Renders the <NotFound /> Component

Step 5: "About" Component

The first of our "view" components is the "About" Component. For this component, we will simply display information about the developer (you):

- Show a title for the page, ie:

  This can be accomplished using the **Card Component** (https://react-bootstrap-v4.netlify.app/components/cards/). However, feel free to use any other design elements, components from react-bootstrap, etc. for your header.

  NOTE: Whatever design you decide here, will be used throughout your application

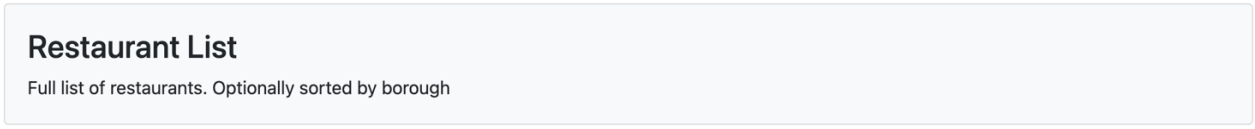
- Place information that you wish to share about interesting projects you have completed / working on, your academic career, development interests, etc.

Step 6: "Restaurants" Component

The next "view" component is the "Restaurants" Component. For this component to function properly, it needs to adhere to the following specification:

- It must import the "useLocation" hook from "react-router-dom"
- It must also have following "state" values:
    - **restaurants / setRestaurants** – default value: null
    - **page / setPage** – default value: 1
- When the component is **first mounted**, **location (from the useLocation hook)** is changed, or the **page** "state" value is changed, the following logic (ie: "effect") should be performed:
    - Execute a **fetch** request to the **api/restaurants** route of your Restaurant API on Heroku (from "Assignment 1") using the query parameters:
        - **page** – the value of **page** in the "state"
        - **perPage** – 10 (this can either be hardcoded in the request or added as a const at the top of the component.
        - **Borough** – the value of ***borough*** in the **search** property from **location** (**from the useLocation hook**). This can be obtained by "parsing" the value of **location.search**. **HINT:** You can use "URLSearchParams" – see: "Adding Query Parameters to our Routes" https://web422.ca/notes/react-routing
            - **NOTE:** If the value of "borough" is *undefined*, in the query string, then do not include it in the **fetch** request as a parameter
    - When the **fetch** operation has completed (ie: once the result is obtained and the data is parsed), set the value of **restaurants** in the "state" to the data returned
- A "previousPage()" utility function must be implemented such that:
    - It decreases the value of **page** in the "state" by one (1), only if the current value is greater than one (1)
- A "nextPage()" utility function must be implemented such that:
    - It increases the value of **page** in the "state" by one (1).
- Next, we must modify the return value to provide a way for users to view and interact with the **restaurants** array in the "state".

  First, we will need to display a header for the view (using the same design that you used for your "About" component). This will simply be a static message reminding the users that they are viewing the full list of restaurants, and that they can search by "borough", ie:



Next, you're free to use any method you wish to display the **restaurants** data. If you wish to try to duplicate the demo video, a <Table striped bordered hover>…</Table> was used, however there are other options as well, such as: <Card>…</Card>, <ListGroup>…</ListGroup>, <Accordion>…</Accordion>, etc.

Whichever method you wish to show the values in the **restaurants** array, you must ensure that for each restaurant, you show:

- name
- address building + address street
- borough
- cuisine

To navigate to a specific restaurant when an item is clicked the following code can be used:

onClick={()=>{ navigate(`/restaurant/${***restaurant***._id}`)}}

NOTE: This assumes that ***restaurant*** is the current "restaurant" object in the iteration used to display restaurants and that you have added the "useNavigate" hook - see: "'Redirecting' to a Route" https://web422.ca/notes/react-routing

To enable ***paging***, we can use the <Pagination>…</Pagination> component, specifically:

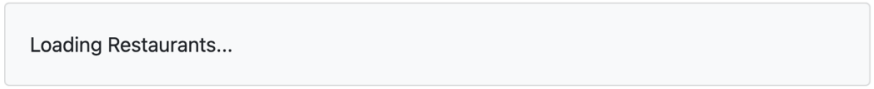
```
<Pagination>
  <Pagination.Prev />
  <Pagination.Item></Pagination.Item>
  <Pagination.Next />
</Pagination>
```

You will have to ensure that:

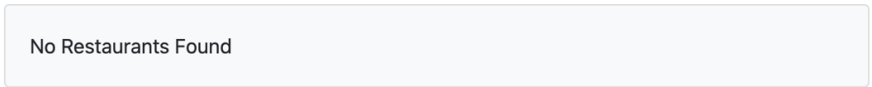
- when <Pagination.Prev /> is clicked, the **previousPage()** method is invoked
- when <Pagination.Next /> is clicked, the **nextPage()** method is invoked
- the <Pagination.Item></Pagination.Item> component contains the value for **page** in the "state"

- Finally, it is important to note that there are also some circumstances when you are unable to render the **restaurants** data, such as:
  - The request is still in progress (ie: **restaurants** is null)
  - **restaurants** is not null, but it doesn't contain any values (ie: it is an empty array)

In either case, render an appropriate message / indication to the user, ie:



or



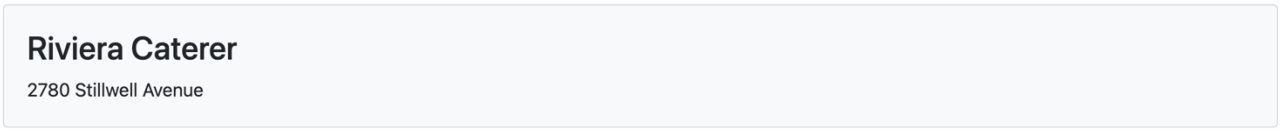
Step 7: "Restaurant" Component

The second "view" component used to render data from our API is the "Restaurant" component. When implementing this one, you will need to adhere to the following specification:

- The leaflet components must be imported, ie:
  import { MapContainer, TileLayer, Marker } from 'react-leaflet';
- It must import the "useParams" hook from "react-router-dom"
- It must also have following "state" values:
  - **restaurant / setRestaurant** – default value: null
  - **loading / setLoading** – default value: true
- When the component is **first mounted** or when **id (from the useParams hook)** is changed, the following logic (ie: "effect") should be performed:
  - Set the **loading** "state" value to true
  - Execute a **fetch** request to the **api/restaurants/id** route of your Restaurant API on Heroku (from "Assignment 1"), where **id** is the value of **id (from the useParams hook)**
  - When the **fetch** operation has completed (ie: once the result is obtained and the data is parsed), we must:
    - Set the **loading** "state" value to false
    - Make the following check to ensure that the *data* that came back was indeed a restaurant object and update the state accordingly, ie:

      if(*data*.hasOwnProperty("_id")){
        set the **restaurant** value in the "state" to *data*
      }else{
        set the **restaurant** value in the "state" to null
      }
- Next, we must modify the return value to provide a way for users to view the **restaurant** object in the "state".

  First, we will need to display a header for the view (using the same design that you used for your "About" component). This will contain the **name** of the **restaurant** as well as a combination of the **address building** and **address street** values, ie:



  Once this is complete, we will need get the map for the restaurant working. You may use the following code, where ***restaurant address coordinate*** values should be replaced with values from the address coord array:

```
<MapContainer style={{"height": "400px"}} center={[restaurant address coordinate 1, restaurant address coordinate 0]} zoom={13} scrollWheelZoom={false}>
    <TileLayer url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />
    <Marker position={[ restaurant address coordinate 1, restaurant address coordinate 0]}></Marker>
</MapContainer>
```

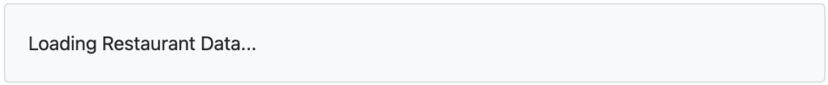
Next, we must display all of the "grades" for the restaurant, once again You're free to use any method you wish to display the **grades** data. If you wish to try to duplicate the demo video, a <CardDeck>…</CardDeck> containing multiple <Card>…</Card> components was used.

Whichever method you wish to show the values in the **grades** array (for the specific **restaurant** object in the "state"), you must ensure that for each grade, you show:

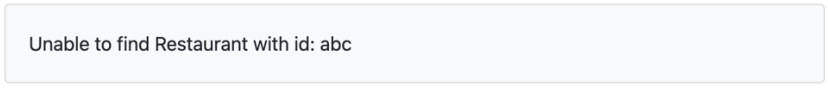
- grade
- date

- Finally, it is important to note that there are also some circumstances when you are unable to render the **restaurant** data, such as:
  - The request is still in progress (ie: **loading** is true)
  - **loading** is no longer true, but the value of **restaurant** is null

In either case, render an appropriate message / indication to the user, ie:



or



Once you have updated this final component, your assignment should be complete. Please check it against the sample video (top) and ensure that its functioning correctly before submitting.

Assignment Submission:

- For this assignment, you will be required to **build** your assignment before submitting. This will involve running the command:
  - **npm run build**

  to create a "build" folder that you will include in your submission
- Next, add the following declaration at the top of your index.js file

```
/*********************************************************************************
* WEB422 – Assignment 3
* 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 **without node_modules** (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.