WEB422 Assignment 3

Submission Deadline:

Friday, June 16th @ 11:59pm

Assessment Weight:

8% of your final course Grade

Objective:

To continue to work with our "Trips" 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 and Next.js to create an interface for *viewing* trips. **Please Note**: Once again, you're free to style your app however you wish. The following specification outlines how we can use the React-Bootstrap Components (Bootstrap 5). 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/summer-2022/web422/A3/A3.mp4

Step 1: Creating a Next 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 Next.js app by using "create-next-app" with the command "npx create-next-app" followed by the identifier for your app, ie: "my-app" and the "--use-npm" option.
- Once the tool has finished creating your Next App, be sure to "cd" into your new app directory and install the following modules using npm:
  - swr
  - bootstrap react-bootstrap
  - react-leaflet leaflet
- To ensure that our newly-added Bootstrap components render properly, we must import the correct CSS file to our "pages/_app.js" file, ie:
  - import 'bootstrap/dist/css/bootstrap.min.css';

  Add Bootstrap JavaScript library to the project by adding the following code into the App component (function), including importing the useEffect hook from react:

```
useEffect(()=>{
```

```
          import("bootstrap/dist/js/bootstrap"); // add bootstrap js library
        },[])
```

- Next, we must delete (or clear the CSS from) the "Home.module.css" file and clear the existing CSS from "globals.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 "globals.css" file (this will ensure that any table with class "table-hover" has the "Pointer" cursor, and set different color and background color for different user types – indicated as class"Subscriber" or "Customer"):

  .table-hover tr:hover{ cursor:pointer; }
  .Subscriber { background-color: …; color: …; }
  .Customer { background-color: …; color: …; }

Step 2: Component "Skeletons"

For the next part of the assignment, we'll add our "page" / custom components (with some default text) as well as create a layout for our app.

NOTE: Components that output a simple <p>…</p> element will be overwritten further in the assignment

To proceed, add the following components:

- **Home (pages/index.js)** – Remove existing JSX and all imports and modify this to output <p>Trips</p>
- **Trip (pages/trip/[id].js)** – outputs <p>Trip id: id</p> where ***id*** is a value that we will retrieve by using the "useRouter" hook
- **About (pages/about.js)** – outputs <p>About</p>
- **MainNav (components/MainNav.js)** – outputs <p>MainNav</p>
- **Layout (components/Layout.js)** – outputs:

  ```
  <MainNav />
  <br />
  <Container>
    {props.children}
  </Container>
  <br />
  ```

  NOTE: the "Container" is from "react-bootstrap", ie: import { Container } from 'react-bootstrap';
- **PageHeader (components/PageHeader.js)** – outputs <p>PageHeader</p>

Step 3: App Component & NavBar

Before we start building out our components, we must first do some work with the "App" component (pages/_app.js):

- To begin, add the following import statements:

- import Layout from '**@/components/Layout**';
- import { SWRConfig } from '**swr**';

- Next, wrap the <Component {...pageProps} /> component with the following SWRConfig component to globally define the fetcher:

  <SWRConfig value={{ fetcher: (...args) => fetch(...args).then((res) => res.json()) }}></SWRConfig>

- Finally, wrap the "SWRConfig" element (above) with our <Layout></Layout> component.

With this complete, we can now create our "MainNav" component, so that our navbar shows more than just <p>MainNav</p>:

To begin, we should (at a minimum) have the following components imported:

- **Container, Nav, Navbar** from "react-bootstrap"
- **Link** from "next/link"

Next, use the documentation from the "Navbars" section of the React-Bootstrap docs (https://react-bootstrap.netlify.app/docs/components/navbar) to obtain the JSX code for a functioning Navbar according to the following specification:

- Shows ***New York Citibike Trips*** in the **Navbar.Brand** element without an href element (ie remove href="#home" from **Navbar.Brand**).

- The **Navbar** element should have a "className" value of "fixed-top".

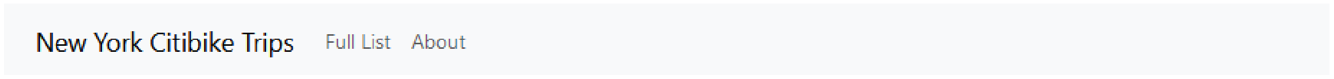
- Contains 2 <Nav.Link> elements with the text "**Full List**" and "**About**", linking to "**/**" (for **Trips**) and **"/about**" for (**About**)

  - **NOTE:** For these elements to function correctly, they must be wrapped in <Link> elements containing appropriate href properties and the "passHref" property. For example, the "Trips" Nav.Link element should be:

    <Link href="/" passHref legacyBehavior><Nav.Link>Full List</Nav.Link></Link>

- There should be **two** line breaks (<br />) after the **Navbar** element – this will ensure that the content can be seen below the *fixed* **Navbar**

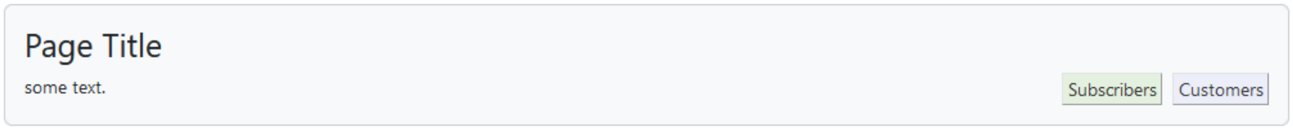
- Once complete, you should have a responsive Navbar that looks similar to the following image (with your own Name in place of **Student Name**. The "Trips" item should link to "/", while the "About" item should link to "/about":



Step 4: Page Header Component (PageHeader.js)

With the navbar and other components in place we can concentrate on building a reusable "Page Header" element for our various views according to the following specification:

- The component must accept custom properties (props): **title**, **text, showSubscriber, showCustomer** (ie: `<PageHeader title="Page Title" text="some text." showSubscriber={true} showCustomer={true} />`
  - The component must render the heading and text props in some kind of container. To achieve the same design as the example, a "Card" (see: https://react-bootstrap.github.io/docs/components/cards/) was used with the className "bg-light" and the title, text, showSubscriber, showCustomer values rendered within a <Card.Body>…</Card.Body> element (NOTE: <Card.Title>, <Card.Text> and <Card.Image> will not be used), for example:

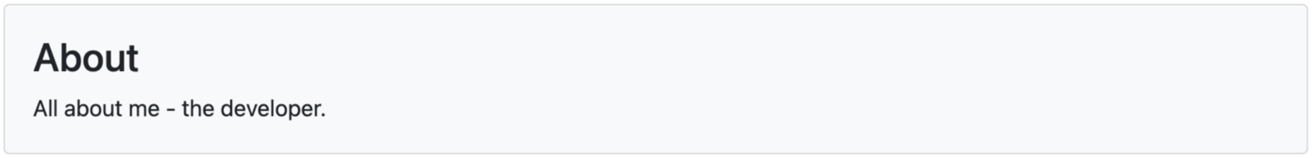


  Note:
  - "Page Title" is the value of the prop **title** passed into the component and rendered as level-3 heading.
  - "some text." is the prop **text** value passed into the component and rendered without any tags.
  - "Subscriber" is static text which is rendered using HTML <button> with the mere attribute className={"Subscriber"}. It will show up only when the prop showSubscriber is true and will be hidden otherwise.
  - "Customer" is static text which is rendered using HTML <button> with the mere attribute className={"Customer"}. It will show up only when the prop **showCustomer** is true and will be hidden otherwise.
  - Additionally, if you need to push the buttons to the right of the "Card" (as above), the "float-end" class may be used in a div element, ie: <div className="float-end"> … </div> to wrap the buttons:
  - **Card** should be imported from `"react-bootstrap"` at top of the component.
- There should be a line break (<br />) after the container.

Step 5: About Component (about.js)

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, using the PageHeader Component (PageHeader.js):


About

All about me - the developer.

The **PageHeader** should be imported from "@/components/PageHeader" and render the component with the attributes (props) and appropriate values passed to component in order to get the screenshot above.

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

Step 6: Home Component (index.js)

We now have all of the pieces in place to build our "Home" component. This is arguably one of the more complicated components as it must display data from our "Trips" API, one page at a time (similar to what was achieved in Assignment 2).

- To begin, we should (at a minimum) have the following components imported:
    - Import the 'styles/Home.module.css' file as **styles**;
    - **useSWR** from 'swr';
    - **useState, useEffect** from 'react';
    - **Pagination, Table** from 'react-bootstrap';
    - **PageHeader** from '@/components/PageHeader';
    - **useRouter** from 'next/router';

- Next, add the following **state** values to the component:
    - **page** (default value: 1)
    - **pageData** (default value: [])

- With state values in place, we can now use SWR to make a request for the data, ie:

  const { data, error } = useSWR(`**(Your Cyclic Trips API)**/api/trips?page=**page**&perPage=10`);

- We will be using the data returned from useSWR to set our "pageData" state value, rather than using it directly within our JSX for this component. To ensure that whenever the "data" has been successfully updated (as a result of a new page being requested, for example), we can leverage the "useEffect" hook as follows (assuming "setPageData()" is your setter function for your **page** state value):

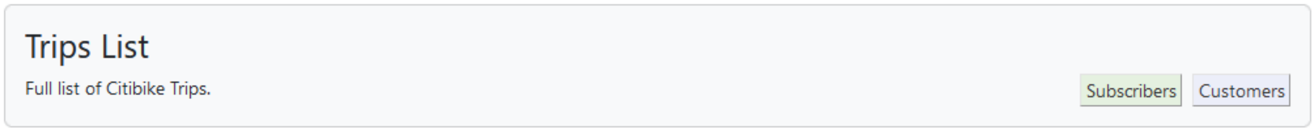
```
useEffect(() => {
 if (data) {
  setPageData(data);
 }
}, [data]);
```

- Before we can write the return statement (JSX) for this component, we must ensure that we have two functions declared:

  - **previous** – decreases the **page** state value by **1** if it is greater than **1**
  - **next** – increases the **page** state value by **1**

- Next, we must modify the return value to provide a way for users to view and interact with the **trips** array in the "state".

  First, we will need to display a header for the view using the PageHeader Component (PageHeader.js):

  Trips List

  Full list of Citibike Trips.

  The **PageHeader** should be imported from "@/components/PageHeader" and render the component with appropriate attributes (props) and values passed to component in order to get the screenshot above.

  A <PageHeader /> element with the title "Film Collection", text "Full list of Citybike Trips.", showSubscriber true and showCustomer false.

  Next, you're need to display the **trips** data. Try to duplicate the demo video, a <Table bordered hover>…</Table> was used, (ie: " import { Table } from 'react-bootstrap';". You are allowed to use other options to render the trips data as well, such as: <Card>…</Card>, <ListGroup>…</ListGroup>, etc.

  Whichever method you wish to show the values in the **trips** array, you must ensure that for each trip, you show:
    - Bike ID
    - Start Station
    - End Station
    - Duration (**NOTE**: this value is shown in minutes, using the formula: (tripduration / 60).toFixed(2)
    - (Row Highlighting) – Ensure that the correct "class" attribute is provided for each row, depending on if the trip was for a "Subscriber" or "Customer" – recall, you may use the syntax: **className={ *someClass* }.** If the CSS rules within the "globals.css" file do not work for the table cells (<td>), you may use embedded CSS WITHIN THE JSX CODE, e.g.:

```
<style jsx>{`
     .Subscriber{ background-color: ...; color: ...; }
     .Customer{ background-color: ...; color: ...; }
   `}</style>
```

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

  onClick={() => { **router**.push(`/trip/${**trip**._id}`) }}

  NOTE: This assumes that ***trip*** is the current "trip" object in the iteration used to display trips and that you have added the "useRouter" hook. The **router** above is the instance of the "useRouter" hook. - see: course site useRouter Hook about transitioning to a new route.

  Last, to enable paging, we can use the Pagination element (<Pagination>) containing the following elements:

  - <Pagination.Prev /> with a "click" event that executes the "previous" function
  - <Pagination.Item /> which shows the current **page** value
  - <Pagination.Next /> with a "click" event that executes the "next" function

Step 7: Trip Component (pages/trip/[title].js)

The Trip component is a dynamic component and will show the detailed trip information and display the trip's starting and end points info on a Leaflet map. The following components need to be imported:

  - **useRouter** from 'next/router';
  - **PageHeader** from '@/components/PageHeader';

Next, the "Trip" component will use the approach of fetching API data for pre-rendered HTML to speed up load time and provide greater SEO. The "Trip" component uses dynamic route, so you need to implement both 'getStaticPaths' and 'getStaticProps' functions:

- 'getStaticPaths' – an asynchronous function. Use Thunder Client or browser to fetch the first page of your Trips API with the URI, e.g., `(Your Cyclic Trips API)/api/trips?page=1&perPage=10`. Then use the 10 trips' **_id** values for the "params" id value of the static paths. For example, this is one of the static paths: { params: { id: "572bb8222b288919b68abf5a"} }.

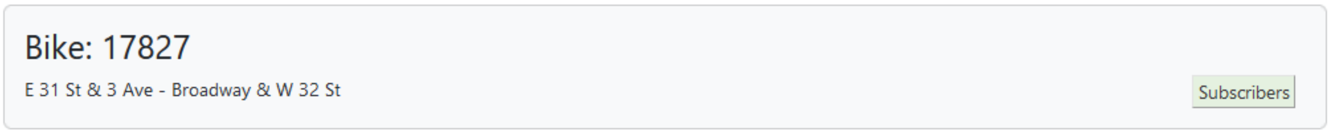
  Note: the value for **fallback** must use **'blocking',** which means if new paths not returned by getStaticPaths, the function will wait for the HTML to be generated, identical to SSR (hence why blocking), and then be cached for future requests so it only happens once per path.

- 'getStaticProps' – an asynchronous function. The function receives a parameter **context** and use Fetch API with the URI, e.g., `(Your Cyclic Trips API)/api/trips/${**context.params.id**}` to fetch the static and dynamic Trip (for trips which are not prefetched/rendered) data. The function returns props in an object with a single trip data as value.

- The trip component must receive **props** which contains the trip object data.

Before we can render anything however, we must check the value of the trip data passed in through props. If trip data is null or undefined (ie: "falsy"), return **null** (ie: don't render anything). Otherwise, render the trip data as specified below:

- First, we will need to display a title for the view using the PageHeader Component (PageHeader.js) and render the component with the attributes (props) and appropriate values passed to component in order to get the screenshot above. This will contain the **bikeid** and **usertype** of the trip as well as a combination of the **start station name** and **end station name** values which will be passed into the component with the attributes (props) title, text, showSubscriber, showCustomer. For example: <PageHeader title={`Bike: ${props.trip?.bikeid}`} … … />:

  
  Bike: 17827
  E 31 St & 3 Ave - Broadway & W 32 St

  The shows up, indicating the user type of current trip is "Subscriber".

- Once this is complete, we will need get the map for the trip working. In your Trip component, you may try to use the following code, where ***start station*** location ***coordinate (index 1, index 0) /name*** & ***end station location coordinate (index 1, index 0) / name*** values should be replaced with values from the current trip object:

  <MapContainer style={{ "height": "500px" }} center={[***start station location coordinate 1***, ***start station location coordinate 0***]} zoom={15}>

      <TileLayer url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />
          <Marker position={[***start station location coordinate 1***, ***start station location coordinate 0***]}>
              <Tooltip permanent direction='right'>Start: {***start station name***}</Tooltip>
          </Marker>
          <Marker position={[***end station location coordinate 1***, ***end station location coordinate 0***]}>
              <Tooltip permanent direction='right'>End: {***end station name***}</Tooltip>
          </Marker>
  </MapContainer>

  To make above code work, more files or components need to be imported:

  - import 'leaflet/dist/leaflet.css';
  - **MapContainer, TileLayer, Marker, Tooltip** from 'react-leaflet';

  However, the above code will give the ReferenceError: "window is not defined". That is because we are trying to use the server-side rendering or pre-rendered HTML for the Trip component, but the window object does not exist on the server side. So, we should disable server-side rendering or pre-rendered HTML on server for rendering Leaflet map (and the other parts are still pre-rendered in the server). The following is the solution for this issue.

- The solution for above issue is to create a component called Map and the Map component will be imported dynamically and disable server-side rendering/pre-rendered HTML for the Map component. The following are the detailed steps:

  - Create a folder named "Map" under the "components" folder. Then create two files named "index.js" and "Map.js" under the "Map" folder.

  - In the "Map.js" file, add code for creating the Map component and the component will return/display the Leaflet map with the code just given above (<MapContainer> component and import statements).

    Note:

    - The Map component must receive **props** passed in from its parent component.

    - To make about code work, you may need to create constant named **trip** which takes the value of props.trip. Or the Map component may receive **{trip}** rather than **props** as the passed-in data from the parent component.

    - Certainly, parts of the code above need to be updated with values from the passed-in trip object:

  - The index.js file (components/Map/index.js) is for dynamically import the Map component and disable server-side rendering or pre-rendered HTML on server. Here is the code:

```
import dynamic from "next/dynamic";

const Map = dynamic(()=> import("./Map" ), // dynamic import
        {ssr: false} // disable server side rendering
    )
export default Map;
```

- Go back to the Trip component (component/trip/[id].js), add this import statement:

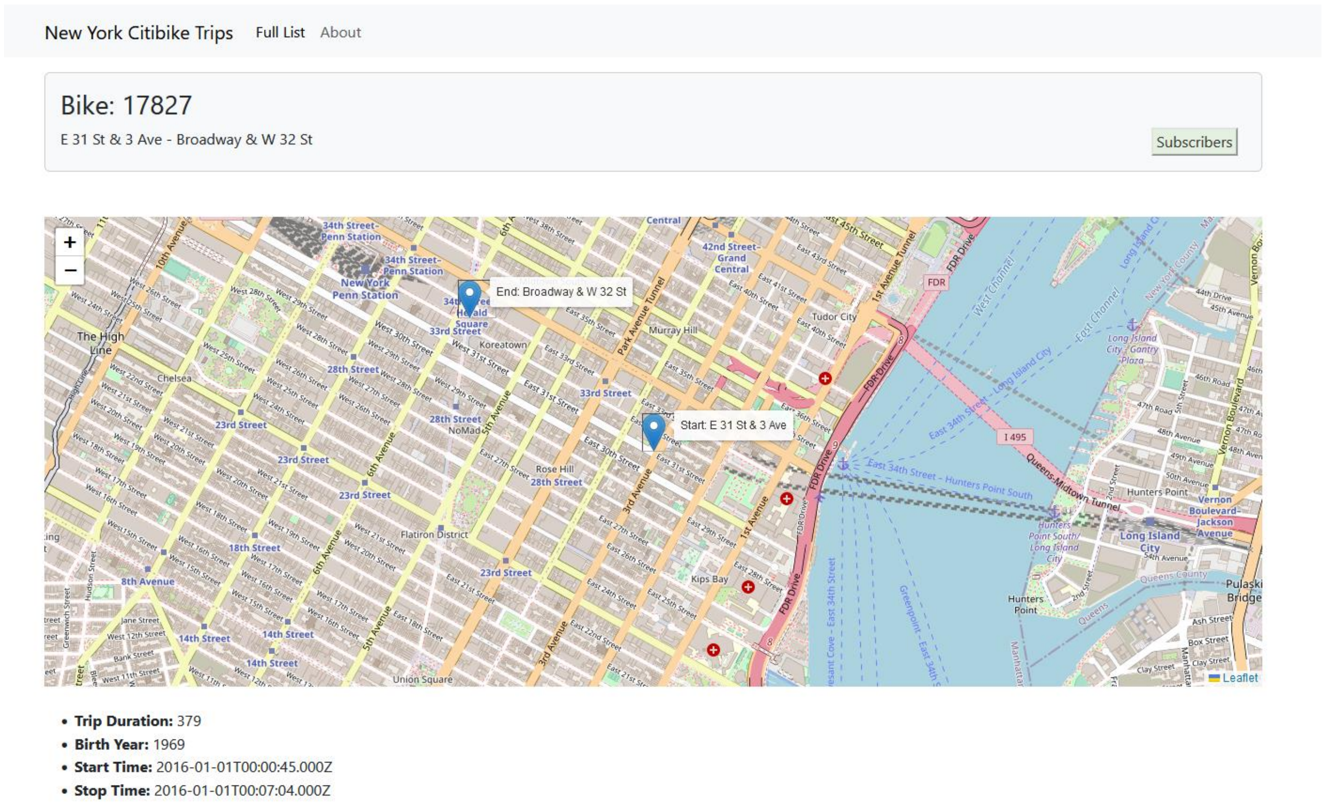
  ```
  import Map from "@/components/Map"; // will import components/Map/index.js
  ```

  Then under the Page Header, render the Map component with passing the current trip object and the value of trip attribute (prop). Now the Leaflet map with two markers for start station and end station locations should show up in your Trip component.

  Last, display more info about the trip in unordered list below the map:

  - Trip duration:
  - Birth year:
  - Start time:
  - Stop time:

  A screenshot of the Trip component:



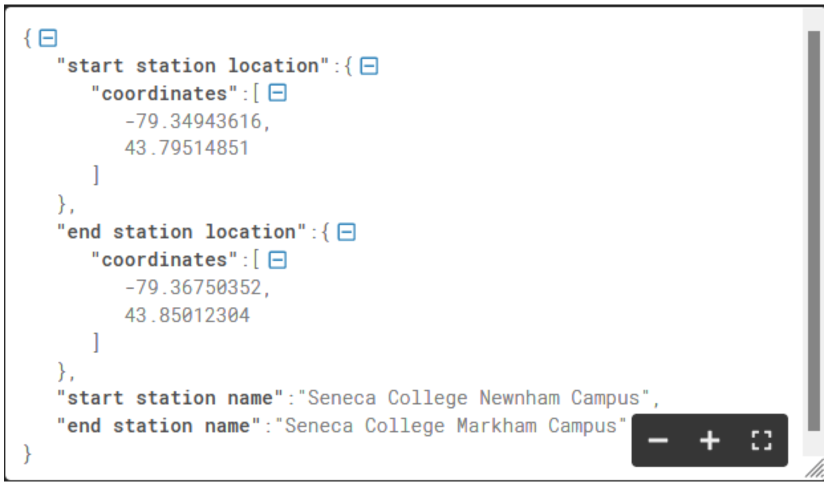
Bike: 17827

E 31 St & 3 Ave - Broadway & W 32 St

- **Trip Duration:** 379
- **Birth Year:** 1969
- **Start Time:** 2016-01-01T00:00:45.000Z
- **Stop Time:** 2016-01-01T00:07:04.000Z

Step 8: Updating About Component (pages/about.js)

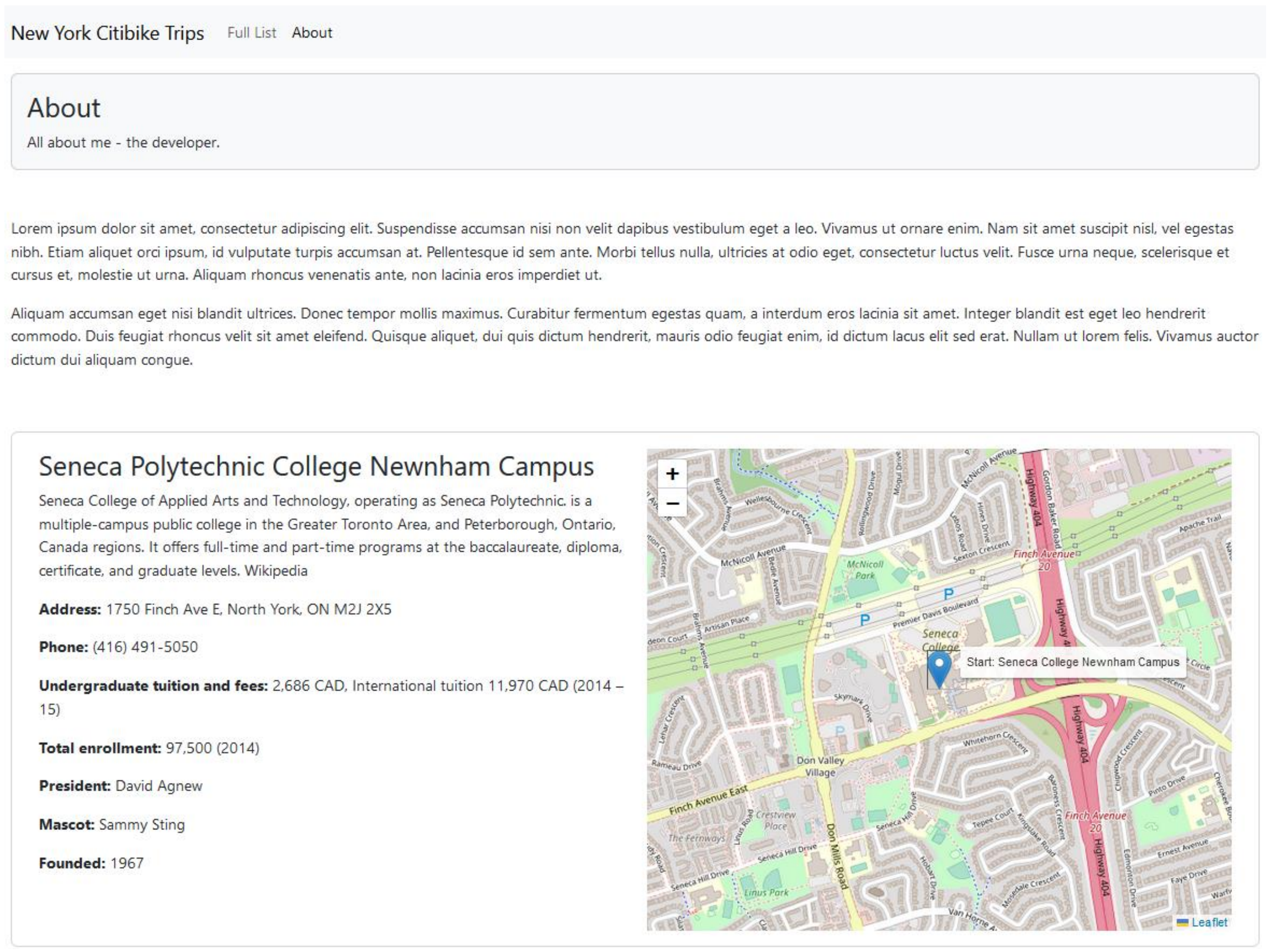
In the About component, create a constant named trip with this object as value:



```
{
   "start station location":{
      "coordinates":[
         -79.34943616,
         43.79514851
      ]
   },
   "end station location":{
      "coordinates":[
         -79.36750352,
         43.85012304
      ]
   },
   "start station name":"Seneca College Newnham Campus",
   "end station name":"Seneca College Markham Campus"
}
```

Then render Leaflet map in the About component template (UI) using the Map component by passing the trip object to the Map component.

Lastly, display information about Seneca college and use Card, Container, Row, and Col components from react-bootstrap to organize the info and map. After completed, your About component should like this:



New York Citibike Trips Full List About

About

All about me - the developer.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse accumsan nisi non velit dapibus vestibulum eget a leo. Vivamus ut ornare enim. Nam sit amet suscipit nisl, vel egestas nibh. Etiam aliquet orci ipsum, id vulputate turpis accumsan at. Pellentesque id sem ante. Morbi tellus nulla, ultricies at odio eget, consectetur luctus velit. Fusce urna neque, scelerisque et cursus et, molestie ut urna. Aliquam rhoncus venenatis ante, non lacinia eros imperdiet ut.

Aliquam accumsan eget nisi blandit ultrices. Donec tempor mollis maximus. Curabitur fermentum egestas quam, a interdum eros lacinia sit amet. Integer blandit est eget leo hendrerit commodo. Duis feugiat rhoncus velit sit amet eleifend. Quisque aliquet, dui quis dictum hendrerit, mauris odio feugiat enim, id dictum lacus elit sed erat. Nullam ut lorem felis. Vivamus auctor dictum dui aliquam congue.

Seneca Polytechnic College Newnham Campus

Seneca College of Applied Arts and Technology, operating as Seneca Polytechnic. is a multiple-campus public college in the Greater Toronto Area, and Peterborough, Ontario, Canada regions. It offers full-time and part-time programs at the baccalaureate, diploma, certificate, and graduate levels. Wikipedia

Address: 1750 Finch Ave E, North York, ON M2J 2X5

Phone: (416) 491-5050

Undergraduate tuition and fees: 2,686 CAD, International tuition 11,970 CAD (2014 – 15)

Total enrollment: 97,500 (2014)

President: David Agnew

Mascot: Sammy Sting

Founded: 1967

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 production build in the ".next" folder to be included in your submission. **Please Note:** you will be required to fix any errors that are identified during the build process.

- 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.