Developing with React.js, TypeScript and Webpack

Lab Time: 60 minutes

Lab Folder: C:\Student\Modules\03_React\Lab

Lab Overview: In this lab you will develop a single page application (SPA) using React.js, TypeScript and webpack. You will begin with a starter project that already contains the npm packages and configuration files for TypeScript and webpack. You work will involve adding the npm packages for React.js and creating a hierarchy of React components. You will also integrate the react router to provide your React application with a basic route map and navigation. By the end of this lab, you will get experience using the React Fetch API to call and retrieve data from an OData web service across the Internet.

Exercise 1: Create a New Project with React.js, TypeScript and Webpack

In this exercise you will learn how to create a web application using the React.js library. You will begin with a starter project that already contains the npm packages and the configuration files for webpack and the Typescript compiler. First, you make a copy of the starter project. Next, you will get this starter project up and running in the Visual Studio Code development environment. After that, you will go through the process of adding the required packages for **React.js** and creating a top-level React component named **App**.

1. Inspect the project starter files in the folder named **react-lab**.

    a) Using Windows Explorer, open the folder at **C:\Student\Modules\03_React\Lab\StarterProjects\react-lab**.

    b) Make a copy of the **react-lab** folder outside the **StarterProjects** project **c:\Student\Modules\03_React\Lab\react-lab**.

    c) If you look inside the **react-lab** folder you just copied, you will see it already contains files that should be familiar to you including **package.json**. **tsconfig.json** and **webpack.config.js**.
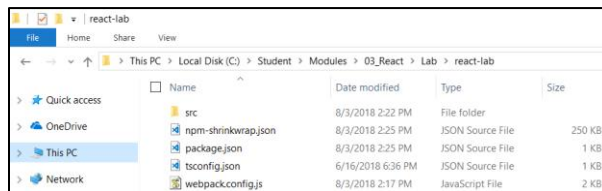
2. Open the **react-lab** folder with Visual Studio Code.

    a) Launch Visual Studio Code.

    b) Use the **File >> Open Folder** command to open the folder at **c:\Student\Modules\03_React\Lab\react-lab**.

    c) Take a moment to review the names of the files that already exist within this project.
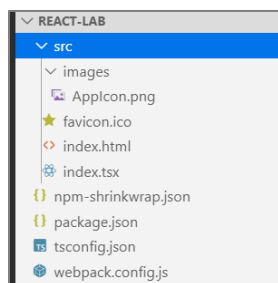
3. Examine the contents of the project's primary configuration files.

    a) Open and inspect the contents of **package.json** to see what packages have already been installed.

Note that this project already has the npm package for **bootstrap** installed. You will be using **bootstrap** to style your application.

    b) Open and inspect the contents of **tsconfig.json** to see how the TypeScript compilation process has been configured.

    c) Open and inspect the contents of **webpack.config.js** see how the webpack build process has been configured.

Note the **module.exports**.**resolve** property in **webpack.config.js** has been configured to process **tsx** files in addition to **ts** files.

    d) Close **package.json**, **tsconfig.json** and **webpack.config.js** without saving any changes.

4.   Inspect the contents **index.html**.

   a)   Locate the **index.html** file in the **src** folder and double-click on it to open it in an editor window.

   b)   Examine the HTML content inside and notice it's minimal layout.

```
<!DOCTYPE html>
<html>

<head>
  <title>React Lab App</title>
  <meta charset="utf-8" />
</head>

<body>
  <div id="react-target" />
</body>

</html>
```

   c)   Close **index.html** without saving any change.

5.   Inspect the contents **index.tsx**.

   a)   Locate the **index.tsx** file in the **src** folder and double-click on it to open it in an editor window.

   b)   Examine the two lines of TSX content inside.

```
var target = document.getElementById('react-target');

target!.innerHTML = "Just getting started. Time to get to work and build a react.js application!";
```

6.   Restore the project's packages using the **npm install** command.

   a)   Use the **View > Terminal** menu command to display the Integrated Terminal.

   b)   Locate the console of the **Integrated Terminal** where you can type in and execute **npm** commands.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

PS C:\Student\Modules\03_React\Lab\react-lab> |
```

   c)   Type and execute the **npm install** command to restore all the project packages
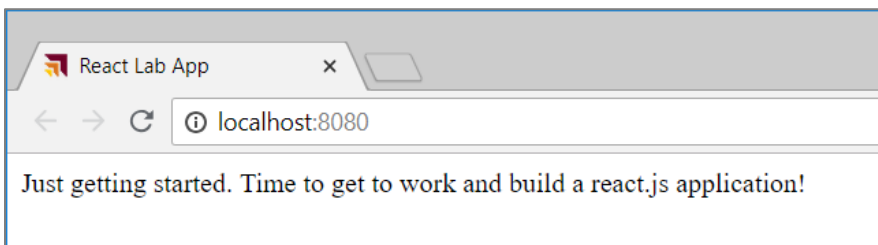
```
npm install
```

   d)   Wait until the **npm install** command completes.

7.   Run and test the React application using the webpack dev server.

   a)   In the Integrated Terminal, execute the **npm run start** command to start up the weback dev server and test he application

```
npm run start
```

   b)   Wait while webpack builds your application.

   c)   When the application launches in the browser, it should display a very minimal interface as shown in the following screenshot

```
React Lab App           ✕

←  →  C    ⓘ localhost:8080

Just getting started. Time to get to work and build a react.js application!
```

   d)   Close the browser and stop the webpack dev server by typing **CTRL + C** in the Integrated Terminal console.

At this point you have gotten the project up and running. However, the project doesn't currently provide any support for React.js. Over the next few steps you will add the npm packages required for React.js development.

8. Add the npm packages for **react** and **react-dom**.

   a) Move back to the console of the Integrated Terminal.

   b) Type and execute the following command to install the packages for **react** and its Typed Definition files.

   ```
   npm install react @types/react --save-dev
   ```

   c) Type and execute the following command to install the packages for **react-dom** and its Typed Definition files.

   ```
   npm install react-dom @types/react-dom --save-dev
   ```

9. Add a new top-level React component named **App**.

   a) Inside the **src** folder, create a new folder named **components**.

   b) Inside the new **components** folder, create a new TSX source file named **App.tsx**.
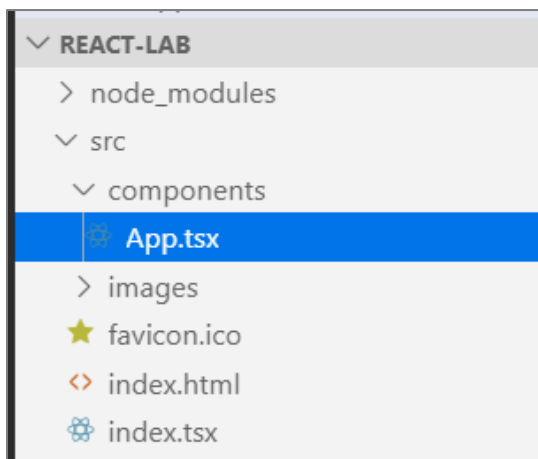
   c) Copy and paste the following code into **App.tsx** to provide the starter code for a React component.

   ```
   import * as React from 'react';

   export default class App extends React.Component<any, any> {

     render() {
       return <div>Hello React</div>;
     }

   }
   ```

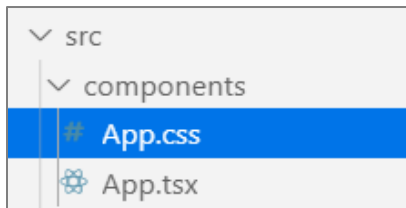   d) Replace the **render** method with the following code to create an HTML page with a navbar and a content area.

   ```
   render() {

     return (
       <div id="page-container" className="container">
         <div className="row navbar navbar-expand-sm navbar-dark bg-dark" role="navigation" >
           <h1 style={{ 'color': 'white' }} >React Lab</h1>
         </div>
         <div className="jumbotron">
           <div>TODO: Add Content</div>
         </div>
       </div>
     );

   }
   ```

10. Add a CSS file to the application named **App.css**.

    a) Inside the **components** folder, create a new CSS file named **App.css**.

    b) Add the following code to **App.css**.

```
body {
  margin: 0px;
  padding: 0px;
  background-color: orange;
  font-family: 'Gill Sans', 'Gill Sans MT', Calibri, 'Trebuchet MS', sans-serif
}

#page-container{
  background-color: white;
  min-height: 600px;
  border-bottom-left-radius: 8px;
  border-bottom-right-radius: 8px;
  border: 1px solid black;
}

.content-body {
  padding: 24px;
}

.jumbotron {
  padding-top: 18px;
  padding-bottom: 16px;
}
```

    c) Save and close **App.css**.

11. Import CSS styles into **App.tsx**.

    a) Open **App.tsx** in an editor window.

    b) Place your cursor below the **import** statement for react

    c) Add the following two **import** statements to add the bootstrap CSS style library to your application.

```
import * as React from 'react';

import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap';
```

The first **import** statement adds all the CSS styles from the bootstrap library. The second **import 'bootstrap'** statement is used to load the JavaScript library for the bootstrap library which is **bootstrap.js**.

    d) Add another **import** statement to import the CSS styles from **App.css**.

```
import * as React from 'react';

import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap';

import './App.css';
```

Note that the **import** statement for **App.css** has been added after the **import** statement for **boostrap.min.css**. That means any CSS rules you add to **App.css** can override any styles from the bootstrap library.

e) At this point, the code you have added to **App.tsx** should match the following code.

```
import * as React from 'react';

import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap';

import './App.css';

export default class App extends React.Component<any, any> {

  render() {

    return (
      <div id="page-container" className="container">
        <div className="row navbar navbar-expand-sm navbar-dark bg-dark" role="navigation" >
          <h1 style={{ 'color': 'white' }} >React Lab</h1>
        </div>
        <div className="jumbotron">
          <div>TODO: Add Content</div>
        </div>
      </div>
    );

  }

}
```

f) Save your changes to **App.tsx**.

12. Update **Index.tsc** to create an instance of the **App** component.

a) Open **Index.tsx** in an editor window.

b) Delete the existing contents of **Index.tsx** and replace it with the following code.

```
import * as React from 'react';
import { render } from 'react-dom';
import App from './components/App';

var topLevelAppComponent = <App />;
var target = document.getElementById('react-target');

render(topLevelAppComponent, target);
```
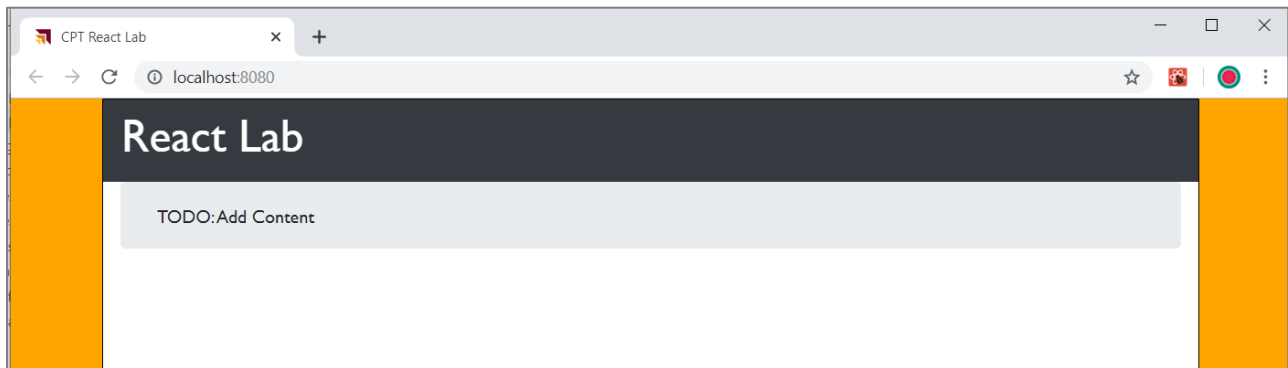
c) Save and close **Index.tsx**.

13. Run the application in the weback dev server to test your new React component.

a) Execute the **npm run start** command from the Integrated Terminal console.

b) When the application launches in the browser, it should appear as the application shown in the screenshot below.


c) After you have tested the application, close the browser and stop the webpack dev server from running.

Exercise 2: Create a React Component Hierarchy

At this point, you have already created your first React component named **App** that will serve as the top-level component in the design of your application. In this exercise you will add a few more React components to create a React component hierarchy.

1. Create the **Banner** component.

   a) Inside the new **components** folder, create two new source file named **Banner.tsx**. and **Banner.css**.
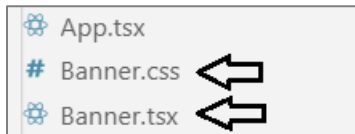
   b) Add the following code to Banner.css.

```css
#banner {
   padding: 0px;
   color: white;
   border-bottom: 1px solid black;
}

#app-icon {
   float: left;
   height: 47px;
   width: 47px;
   display: flex;
   align-items: center;
   justify-content: center;
   background-image: url('./../images/AppIcon.png')
}

#app-title {
   float: left;
   display: flex;
   align-items: center;
   height: 48px;
   padding-left: 12px;
   font-size: 24px;
}
```

   c) Save and close **Banner.css**.

   d) Add the following code to **Banner.tsx**.

```tsx
import * as React from 'react';

import './Banner.css';

interface BannerProperties {
   appTitle: string;
}

export default class Banner extends React.Component<BannerProperties, any> {
   render() {
      return (
         <div id="banner" className="row navbar navbar-expand-sm navbar-dark bg-dark" role="navigation" >
            <div id="app-icon"   ></div>
            <div id="app-title">{this.props.appTitle}</div>
            {this.props.children}
         </div>
      );
   }
}
```
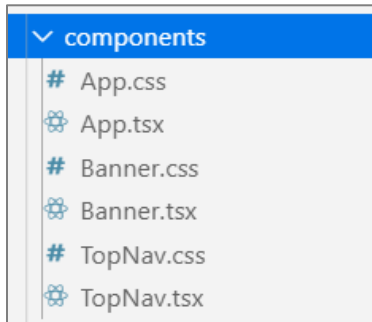
   e) Save your changes to **Banner.tsx**.

Note that the **render** method uses **{this.props.children}** to render all its child elements inside a top-level div.

2. Create the **TopNav** component.

   a) Inside the new **components** folder, create a new source file named **TopNav.tsx**.

   b) Inside the new **components** folder, create a new source file named **TopNav.css**.

   c) Add the following CSS styles to **TopNav.css**.

   ```css
   #top-nav ul {
     line-height: 1em;
     padding-left: 24px;
   }

   #top-nav ul li {
     display: inline-block;
     margin-right: 16px;
     color: #FEBF0F;
   }

   #top-nav li a {
     color: #FEBF0F;
     font-size: 1.0em;
     text-decoration: none;
   }

   #top-nav li a.active-nav-link {
     color: yellow;
   }
   ```

   d) Save your changes and close **TopNav.css**.

   e) Add the following code to **TopNav.tsx**.

   ```tsx
   import * as React from 'react';

   import "./TopNav.css";

   export default class TopNav extends React.Component<any, any> {

     render() {
       return (
         <div id="top-nav" className="navbar-collapse collapse" >
           <nav>
             <ul className="nav navbar-nav" >
               <li className="nav-item" ><a href="#">Home</a></li>
               <li className="nav-item" ><a href="#">Customers</a></li>
               <li className="nav-item" ><a href="#">About</a></li>
             </ul>
           </nav>
         </div>
       );
     }

   }
   ```

   f) Save your changes and close **TopNav.tsx**.

3. Modify the **render** method in the **App** component in **App.tsx**.

   a) Open **App.tsx** in an editor window.

   b) At the top of **App.tsx**, add two **import** statements to import the **Banner** component and the **TopNav** component.

```
import * as React from 'react';

import Banner from "./Banner";
import TopNav from "./TopNav";
```

   c) Inside the **App** class, replace the implementation of the **render** method of the with the following code.

```
render() {
  return (
    <div id="page-container" className="container">
      <Banner appTitle="React Lab App" >
        <TopNav />
      </Banner>
      <div className="jumbotron">
        <div>TODO: Add Content</div>
      </div>
    </div>
  );
}
```

4. Run and test the application.

   a) Make sure you have saved your changes to all source files in the project.

   b) Execute the **npm run start** command from the Integrated Terminal console.

   c) When the application runs, it should appear with a banner and with TopNav links as shown in the screenshot below.
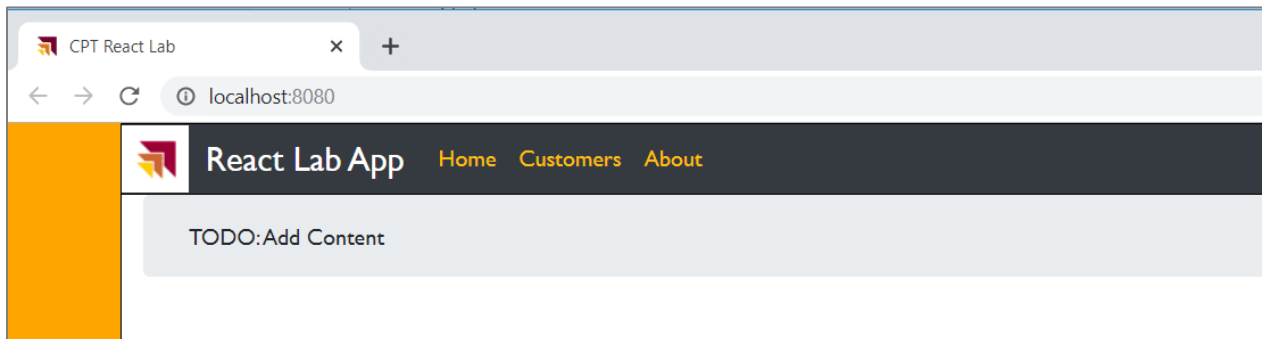
The three links in the TopNav bar should not do anything yet. You will fix that by the end of the next exercise.

   d) After you have tested the application, close the browser and stop the webpack dev server from running.

Exercise 3: Extend Your React Project using the React Router

In this exercise, you will modify your project to add support for the React Router component. You will also create three new view components to provide your application with different display for each of the three links in the TopNav menu.

1. Add the npm packages for **react-router** and **react-router-dom**.

   a) Navigate to the console of the Integrated Terminal.

   b) Run the following command to install the packages for **react-router** and its Typed Definition files.

```
npm install react-router @types/react-router --save-dev
```

   c) Run the following command to install the packages for **react-router-dom** and its Typed Definition files.

```
npm install react-router-dom @types/react-router-dom --save-dev
```

2. Create the **ViewHome** component

   a) Inside the **components** folder, create a child folder named **ViewHome**.

   b) Inside the **ViewHome** folder, create two new source files named **ViewHome.css** and **ViewHome.tsx**.

```
∨ src
  ∨ components
    ∨ ViewHome
      # ViewHome.css
      ⚛ ViewHome.tsx
```
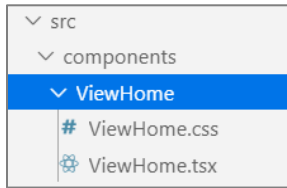
   c) Add the following CSS styles to **ViewHome.css**.

```css
#view-home h4 {
   color: darkblue;
   border-bottom: 1px solid darkblue;
}
```
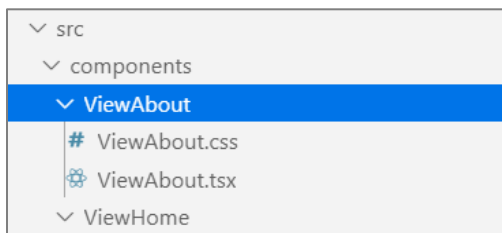
   d) Save your changes and close **ViewHome.css**.

   e) Add the following code to **ViewHome.tsx**.

```tsx
import * as React from 'react';
import './ViewHome.css';

export default class ViewHome extends React.Component<any, any> {
  render() {
    return (
      <div id="view-home" className="content-body" >
        <div className="row">
          <div className="jumbotron col">
            <h3>My Home Page</h3>
            <p>This is my React.js lab app</p>
          </div>
        </div>
        <div className="row">
          <div className="col">
            <h4>React.js is awesome</h4>
            <div>You're going to love it.</div>
          </div>
          <div className="col">
            <h4>React.js is wholesome</h4>
            <div>You can build apps that are huge.</div>
          </div>
        </div>
      </div>
    );
  }
}
```

   f) Save your changes and close **ViewHome.tsx**.

3. Create the **ViewAbout** component.

   a) Inside the **components** folder, create a child folder named **ViewAbout**.

   b) Inside the **ViewAbout** folder, create two new source files named **ViewAbout.css** and **ViewAbout.tsx**.

```
∨ src
  ∨ components
    ∨ ViewAbout
      # ViewAbout.css
      ⚛ ViewAbout.tsx
    ∨ ViewHome
```

c) Add the following CSS styles to **ViewAbout.css**.

```
#view-about P {
  font-size: 1.25em;
  color: darkblue;
}
```

d) Save your changes and close **ViewAbout.css**.

e) Add the following code to **ViewAbout.tsx**.

```
import * as React from 'react';
import './ViewAbout.css';

export default class ViewAbout extends React.Component<any, any> {

  render() {
    return (
      <div id="view-about" className="content-body" >
        <div className="row">
          <div className="jumbotron col">
            <h3>About this app</h3>
            <p className="about-text">This React.js app was created by me.</p>
          </div>
        </div>
      </div>
    );
  }
}
```

f) Save your changes and close **ViewAbout.tsx**.

4. Create the **ViewCustomers** component.

   a) Inside the **components** folder, create a child folder named **ViewCustomers**.

   b) Inside the **ViewCustomers** folder, create two new source files named **ViewCustomers.css** and **ViewCustomers.tsx**.

   c) Add the following code to **ViewCustomers.css**.

```
#view-customers p {
    color: darkblue;
}
```

   d) Save your changes and close **ViewCustomers.css**.

   e) Add the following code to **ViewCustomers.tsx**.

```
import * as React from 'react';
import './ViewCustomers.css';

export default class ViewCustomers extends React.Component<any, any> {

  render() {
    return (
      <div id="view-customers" className="content-body" >
        <div className="row">
          <div className="jumbotron col">
            <h3>Customers view</h3>
            <p>TODO: Implement this view component.</p>
          </div>
        </div>
      </div>
    );
  }

}
```

   f) Save your changes and close **ViewCustomers.tsx**.

5. Update **index.tsx** to initialize the application with a route map using the **HashRouter** component.

    a) Open **index.tsx** in an editor window.

    b) Replace the contents of **index.tsx** with the following code.

```
import * as React from 'react';
import { render } from 'react-dom';
import App from './components/App';
import { HashRouter } from 'react-router-dom';

var topLevelAppComponent =
    <HashRouter>
        <App />
    </HashRouter>;

var target = document.getElementById('react-target');

render(topLevelAppComponent, target);
```

The **HashRouter** component uses the **#** character establish routes defined by the right-hand side of the URLs used in a single page application (SPA). The React-Dom-Router package provides another router component named **BrowserRouter** which makes it possible to remove **#** characters from the URLs shown in the address bar in the browser. However, the **BrowserRouter** is more complicated to use because it requires additional server-side support from your Node.js project.

    c) Save your changes and close **index.tsx**.

6. Modify **App.tsx** to create a route map which includes the three view components.

    a) Open **App.tsx** in an editor window.

    b) Add an **import** statement to import three components from the **react-router-dom** package named **Link, Route** and **Switch**.

```
import * as React from 'react';
import { Route, Switch } from 'react-router-dom';
```

    c) Add three new **import** statements to import the three new components named **ViewHome**, **ViewCustomers** and **ViewAbout**.

```
import * as React from 'react';
import { Route, Switch } from 'react-router-dom';

import Banner from "./Banner";
import TopNav from "./Topnav";

import ViewHome from './ViewHome/ViewHome';
import ViewCustomers from './ViewCustomers/ViewCustomers';
import ViewAbout from './ViewAbout/ViewAbout';
```

In the next step you will update the **render** method by adding a **Switch** component. The **Switch** component will acts as a view port which can switch between the three different view components depending on the current route.

    d) Update the implementation of **render** method of the **App** components with the following code.

```
render() {

    return (
      <div id="page-container" className="container">
        <Banner appTitle="React Lab App" >
          <TopNav />
        </Banner>
        <Switch>
          <Route path="/" exact component={ViewHome} />
          <Route path="/customers" component={ViewCustomers} />
          <Route path="/about" component={ViewAbout} />
        </Switch>
      </div>
    );

}
```

> In this step, you have added a **Switch** component with three inner **Route** components to create a route map which includes three routes to map to the three view components named **ViewHome**, **ViewCustomers** and **ViewAbout**.

    e)   At this point, the contents of **App.tsx** should match the following code listing.

```tsx
import * as React from 'react';
import { Route, Switch } from 'react-router-dom';

import Banner from "./Banner";
import TopNav from "./Topnav";

import ViewHome from './ViewHome/ViewHome';
import ViewCustomers from './ViewCustomers/ViewCustomers';
import ViewAbout from './ViewAbout/ViewAbout';

import 'bootstrap/dist/css/bootstrap.min.css';
import 'bootstrap';

import './App.css';

export default class App extends React.Component<any, any> {

  render() {

    return (
      <div id="page-container" className="container">
        <Banner appTitle="React Lab App" >
          <TopNav />
        </Banner>
        <Switch>
          <Route path="/" exact component={ViewHome} />
          <Route path="/customers" component={ViewCustomers} />
          <Route path="/about" component={ViewAbout} />
        </Switch>
      </div>
    );

  }

}
```

    f)   Save your changes and close **App.tsx**.

> There is just one more thing you need to do before testing your application. You must configure the three links you added to the **TopNav** component so that clicking one of these links redirects the user to the target view component. You will accomplish this by adding **NavLink** components to **TopNav.tsx**.

7.   Update each of the links in **TopNav.tsx** to navigate to a specific route.

    a)   Open **TopNav.tsx** in an editor window.

    b)   Just underneath the **import** statement for **react**, add another an **import** statement to import the **Link** component and the **NavLink** component from the **react-router-dom** package.

```tsx
import * as React from 'react';
import { NavLink } from 'react-router-dom';
```

    c)   Move down to the implementation of **render** inside **TopNav.tsx** and locate the anchor (**<a>)** tag with the inner text of "Home".

```tsx
<a href="#">Home</a>
```

    d)   Replace the **Home** anchor tag with the following **NavLink** element which will redirect the user to the **ViewHome** component.

```tsx
<NavLink exact to="/" className="navbar-link" activeClassName="active-nav-link" >
   Home
</NavLink>
```

e) Locate the anchor tag for **Customers** and replace it with the following **NavLink** element.

```
<NavLink to="/customers" className="navbar-link" activeClassName="active-nav-link" >
  Customers
</NavLink>
```

f) Locate the anchor tag for **About** and replace it with the following **NavLink** element.

```
<NavLink to="/about" className="navbar-link" activeClassName="active-nav-link">
  About
</NavLink>
```

g) At this point, the code you have added to **TopNav.tsx** should match the following code listing.

```
import * as React from 'react';
import { NavLink } from 'react-router-dom';

import "./TopNav.css";

export default class TopNav extends React.Component<any, any> {

  render() {
    return (
      <div id="top-nav" className="navbar-collapse collapse" >
        <nav>
          <ul className="nav navbar-nav" >
            <li className="nav-item" >
              <NavLink exact to="/" className="navbar-link" activeClassName="active-nav-link" >
                Home
              </NavLink>
            </li>
            <li className="nav-item" >
              <NavLink to="/customers" className="navbar-link" activeClassName="active-nav-link" >
                Customers
              </NavLink>
            </li>
            <li className="nav-item" >
              <NavLink to="/about" className="navbar-link" activeClassName="active-nav-link">
                About
              </NavLink>
            </li>
          </ul>
        </nav>
      </div>
    );
  }

}
```

h) Save your changes and close **TopNav.tsx**.

8. Run and test the application.

   a) Make sure you have saved your changes to all source files in the project.

   b) Execute the **npm run start** command from the Integrated Terminal console.

   c) When the application runs, it should appear with a banner and with **TopNav** links as shown in the screenshot below.
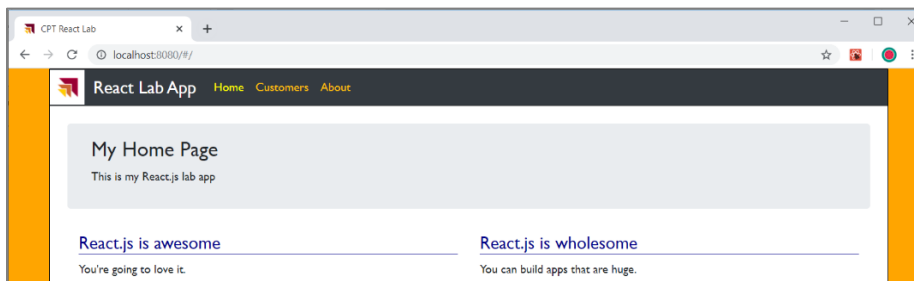
d) Click the **About** link in the TopNav bar and verify you are able to navigate to the **ViewAbout** component.
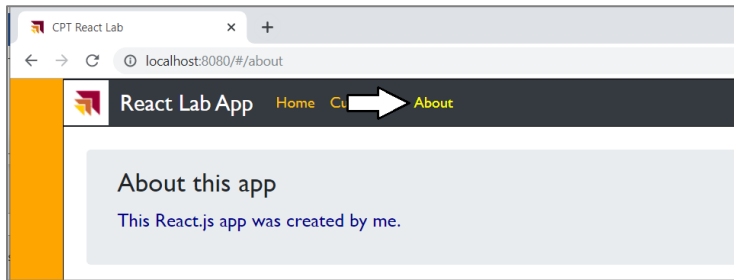
e) Click the **Customers** link in the TopNav bar and verify you are able to navigate to the **ViewCustomers** component.
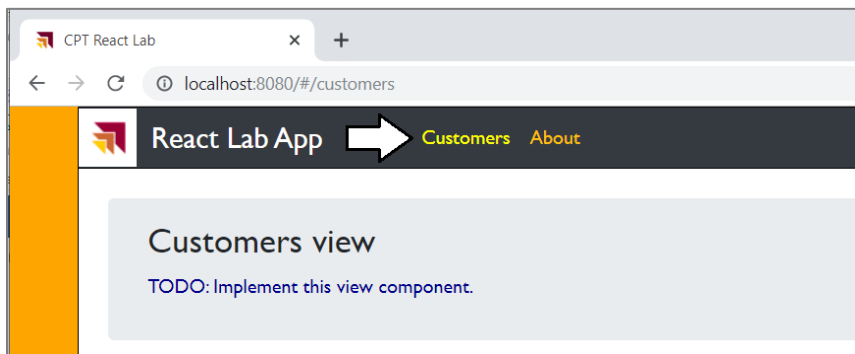
f) After you have tested the application, close the browser and stop the webpack dev server from running.

After all that work you finally have a solid starting point for an SPA created with React.js. Now it's time to move ahead in the next exercise and begin build out a user experience which allows a user to search and view customer data.

Exercise 4: Build a User Interface to Search and View Customer Data

In this lab you will extend the **ViewCustomers** component to build out a user interface experience which allows the user to search and view customer data. You will begin by creating three new interface definitions to establish programming contracts between different components in the application. Then you will add a class named **MockCustomersService** that can be used to retrieve customer data. After that, you will work through the process of building a user interface experience by creating several new child components that will be used by the **ViewCustomers** component. Note that in this exercise you will use customer data that is hard-code into the **MockCustomersService** class. In the following exercise, you will replace the **MockCustomersService** service class with a second service class named **CustomersService** that actually calls across the network to retrieve its customer data.

1. Create the three interfaces required for working with customer data.
   a) Inside the **src** folder, create a new folder named **models**.
   b) Inside the **models** folder, create three new source files named **ICustomer.ts**, **ICustomerDetail.ts**, **ICustomersService.ts**.
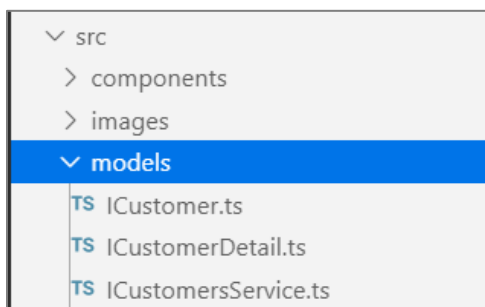
c) Add the following code to **ICustomer.ts** to define a new interface named **ICustomer**.

```
export default interface ICustomer {
  CustomerId: string;
  FirstName: string;
  LastName: string;
  Company: string;
  EmailAddress: string;
  WorkPhone: string;
  HomePhone: string;
}
```

d) Save and close **ICustomer.ts**.

e) Add the following code to **ICustomerDetail.ts** to define a new interface named **ICustomerDetail**.

```
import ICustomer from './ICustomer';

export default interface ICustomerDetail extends ICustomer {
    Address: string;
    City: string;
    State: string;
    Zipcode: string;
    Gender: string;
    BirthDate: string;
}
```

f) Save and close **ICustomerDetail.ts**.

g) Add the following code to **ICustomersService.ts** to define a new interface named **ICustomersService**.

```
import ICustomer from "./ICustomer"
import ICustomerDetail from "./ICustomerDetail";

export default interface ICustomersService {
  getCustomers(): Promise<ICustomer[]>;
  getCustomersByLastName(lastNameSearch: string): Promise<ICustomer[]>;
  getCustomer(customerId: string): Promise<ICustomerDetail>;
}
```

h) Save and close **ICustomersSevice.ts**.

2. Add a new service class named **MockCustomersService** to provide sample customer data that is hard-coded into the class.

a) Inside the **src** folder, create a new folder named **services**.

b) Inside the **services** folder, create a source file named **MockCustomersService.ts**.
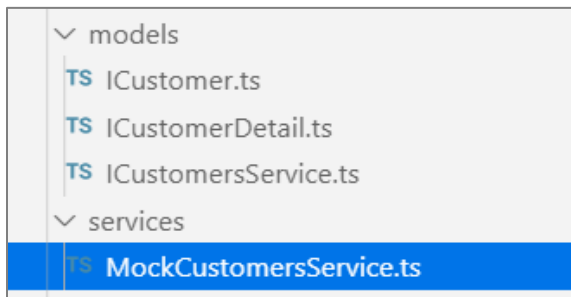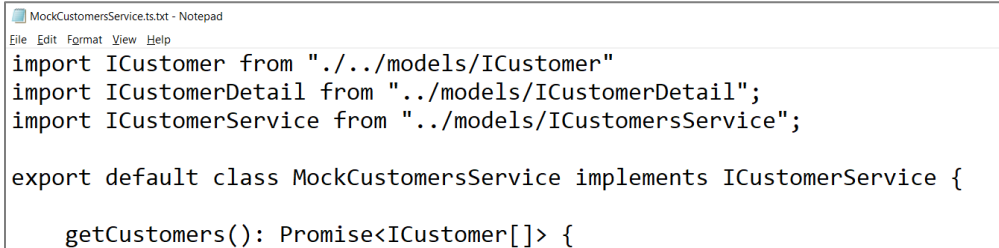
In the next step you will copy-and-paste the code for the **MockCustomersService** class. This class definition is large because it contains a large amount of hard-coded customer data in a JSON format. Rather than have you copy-and-paste the code for the **MockCustomersService** class from this document, you will open a separate text file in the **StarterFiles** folder named **MockCustomersService.ts.txt**. This will make it easier to copy and paste the code you need for the **MockCustomersService** class.

c) Using Windows Explorer, locate the file at the following path.

```
C:\Student\Modules\03_React\Lab\StarterFiles\MockCustomersService.ts.txt
```

d) Double-click the file named **MockCustomersService.ts.txt** to open it in Notepad.exe.

```
MockCustomersService.ts.txt - Notepad
File  Edit  Format  View  Help
import ICustomer from "./../models/ICustomer"
import ICustomerDetail from "../models/ICustomerDetail";
import ICustomerService from "../models/ICustomersService";

export default class MockCustomersService implements ICustomerService {

    getCustomers(): Promise<ICustomer[]> {
```

e) Select all the code in **MockCustomersService.ts.txt** and copy it to the Windows clipboard.

f) Return to our project in Visual Studio Code and paste the code into **MockCustomersService.ts**.

> Take a moment to examine the code inside **MockCustomersService.ts**. You can see that this class implements the **ICustomersService** interface using hard-coded customer data.

g) Save and close **MockCustomersService.ts**.

3. Add a new React component named **CustomersTable** to display customer data.

a) Inside the **src/ViewCustomers** folder, add two new source files named **CustomersTable.css** and **CustomersTable.tsx**.

```
∨ src
  ∨ components
    > ViewAbout
    ∨ ViewCustomers
      #  CustomersTable.css
      ⊕  CustomersTable.tsx
```
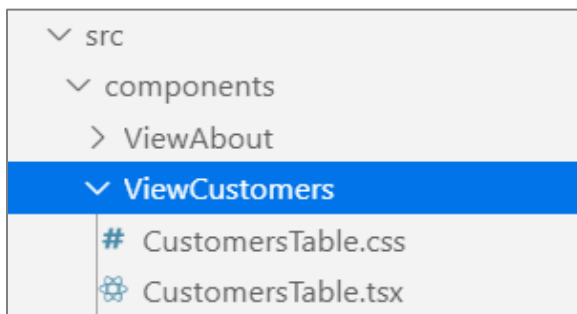
b) Add the following CSS rule to **CustomersTable.css**.

```css
#customer-table {
   font-size: 0.75em;
}
```

c) Save and close to **CustomersTable.css**.

d) Add the following React component starter code to **CustomersTable.tsx**.

```tsx
import * as React from 'react';

import ICustomer from "../../models/ICustomer";

import './CustomersTable.css';

interface CustomersTableProperties {
   customers: ICustomer[];
}

export default class CustomersTable extends React.Component<CustomersTableProperties, any> {


}
```

> Note that the **CustomersTableProperties** interface is defined with a **ICustomer[]** property named **customers**. That means the code in the **render** method of this component will have access to that array of customer data.

e) Inside the **CustomersTable** class, add the following implementation of the **render** method.

```
render() {
  return (
    <div className="row" >
      {this.props.customers.length > 0 ? (
        <table id="customer-table"
               className="col customers-table table table-striped table-bordered table-hover table-sm">
          <thead className="thead-dark">
            <tr>
              <th>ID</th>
              <th>First Name</th>
              <th>Last Name</th>
              <th>Company</th>
              <th>Email</th>
              <th>Work Phone</th>
              <th>Home Phone</th>
            </tr>
          </thead>
          <tbody>
            {this.props.customers.map((customer: ICustomer) =>
              <tr key={customer.CustomerId} >
                <td>{customer.CustomerId}</td>
                <td>{customer.FirstName}</td>
                <td>{customer.LastName}</td>
                <td>{customer.Company}</td>
                <td>{customer.EmailAddress}</td>
                <td>{customer.WorkPhone}</td>
                <td>{customer.HomePhone}</td>
              </tr>
            )}
          </tbody>
        </table>
      ) : (
        <div className="col content-body">
          <div className="alert alert-info" role="alert">
            <strong>No customers returned.</strong> Please refine your search query.
          </div>
        </div>
      )
    }
    </div>
  );
}
```

You can see that the **render** method only generates an HTML table when the **customers** array contains one or more elements.

4. Modify the **render** method of the **ViewCustomers** component to create an instance of **CustomersTable** as a child component.

a) Open the source file named **ViewCustomers.tsx** if it is not already open.

b) The following code listing shows what the **ViewCustomers.tsx** source file should look like at this point.

```
import * as React from 'react';
import './ViewCustomers.css';

export default class ViewCustomers extends React.Component<any, any> {
  render() {
    return (
      <div id="view-customers" className="content-body" >
        <div className="row">
          <div className="jumbotron col">
            <h3>Customers view</h3>
            <p>TODO: Implement this view component.</p>
          </div>
        </div>
      </div>
    );
  }
}
```

c) At the top of in **ViewCustomers.tsx**, add **import** statements for **ICustomer**, **ICustomersService**, **MockCustomersService** and **CustomersTable**.

```
import * as React from 'react';

import ICustomer from '../../models/ICustomer';
import ICustomersService from '../../models/ICustomersService';
import MockCustomersService from "./../../services/MockCustomersService";
import CustomersTable from './CustomersTable';

import './ViewCustomers.css';
```

d) Underneath all the **import** statements, add the following code to define a **type** named **CustomerViewType** and an interface named **ViewCustomersState**.

```
type CustomerViewType = 'table' | 'cards';

interface ViewCustomersState {
  viewType: CustomerViewType;
  customerService: ICustomersService;
  customers: ICustomer[];
  loading: boolean;
}
```

e) Move down an examine the line of code that begins the **ViewCustomers** class definition.

```
export default class ViewCustomers extends React.Component<any, any> {
```

f) Modify the class definition by passing the **ViewCustomersState** interface as the state type parameter for the **ViewCustomers** class as shown in the following code listing.

```
export default class ViewCustomers extends React.Component<any, ViewCustomersState> {
```

g) Place your cursor inside the **ViewCustomers** class and above the **render** method and add the following code to initialize the component's state.

```
state: ViewCustomersState = {
  viewType: 'table',
  customerService: new MockCustomersService(),
  customers: [],
  loading: false
}
```

You can see that this initialization code creates a new instance of the **MockCustomersService** to retrieve customer data.

h) Ensure the state initializer you added is inside the **ViewCustomers** class and above the **render** method as shown below.

```
export default class ViewCustomers extends React.Component<any, ViewCustomersState> {

  state: ViewCustomersState = {
    viewType: 'table',
    customerService: new MockCustomersService(),
    customers: [],
    loading: false
  }

  render() {
```

i) Replace the implementation of the **render** method inside **ViewCustomers.tsx** with the following code.

```
render() {
  return (
    <div>
      <div className="view-customers" >
        <CustomersTable customers={this.state.customers} />
      </div>
    </div>
  );
}
```

> Note how you're passing the **customers** property from the **ViewCustomer** state to the child component **CustomersTable**. Next, you must add code to actually populate the **customers** state property with data. The proper place to accomplish this is typically in the component lifecycle method named **componentDidMount**.
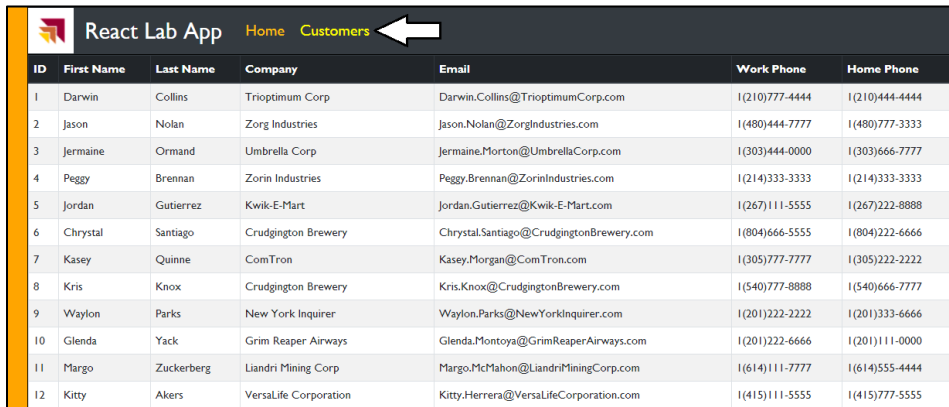
    j) Just below the **render** method, add the following implementation of the React lifecycle method named **componentDidMount**.

```
componentDidMount() {
  this.setState({ loading: true });
  this.state.customerService.getCustomers().then((customers: ICustomer[]) => {
    this.setState({ customers: customers, loading: false });
  })
}
```

    k) Save your changes to **ViewCustomers.tsx**.

5. Run and test the application.

    a) Make sure you have saved your changes to all source files in the project.

    b) Execute the **npm run start** command from the Integrated Terminal console.

    c) When the application starts up, click the **Customer** link to test the **ViewCustomers** component.

    d) You should see the application generates an HTML table as shown in the following screenshot.

> While the application is now displaying customer data in an HTML table, there is an issue. There are too many customer records to display them all at once. You will now work to create a user interface experience which allows for more specific customer searches.

6. Limit the customer data return when the **ViewCustomers** component first loads.

    a) Return to **ViewCustomer.tsx** in an editor window.

    b) Reduce the number of customer returned by replacing the call to **getCustomers** method with a call to **getCustomersByLastName**. When calling **getCustomersByLastName**, pass a capital "A".

```
componentDidMount() {
  this.setState({ loading: true });
  this.state.customerService.getCustomersByLastName("A").then((customers: ICustomer[]) => {
    this.setState({ customers: customers, loading: false });
  })
}
```

    c) Save our changes to **ViewCustomers.tsx**.

    d) Run and test the application again. You should see now there are only two customer records returned.

7. Add a new React component to create a toolbar for searching and filtering customer data.

   a) Return to your project in Visual Studio Code.

   b) Inside the **ViewCustomers** folder, add two new source files named **CustomersToolbar.css** and **CustomersToolbar.tsx**.
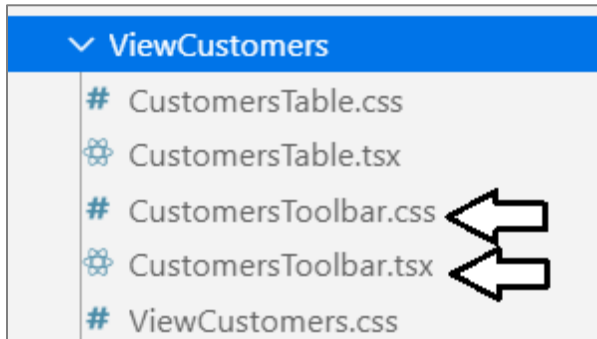
   c) Add the following CSS rules into **CustomersToolbar.css**.

```css
.customers-toolbar {
  padding: 4px;
  background-image: linear-gradient(to bottom, #555 0%, #444 50%, #555 100%);
}

.customers-toolbar .view-menu {
  margin-right: 16px;
}

.customers-toolbar .filter-menu {
  padding: 0px;
}

.customers-toolbar .filter-menu button {
  padding-left: 8px;
  padding-right: 8px;
}

.customers-toolbar .search-menu {
  margin-left: 16px;
}
```

   d) Save and close **CustomersToolbar.css**.

   e) Add the following React component starter code to **CustomersToolbar.tsx**.

```tsx
import * as React from 'react';

import ViewCustomers from './ViewCustomers'
import ICustomer from '../../models/ICustomer';
import ICustomersService from '../../models/ICustomersService';

import './CustomersToolbar.css';

interface CustomersToolbarProperties {
  ViewCustomers: ViewCustomers
}


export default class CustomersToolbar extends React.Component<CustomersToolbarProperties, any> {

}
```

In the next step you will implement the **render** method to display a search box which allows users to search for customers.

f) Inside the **CustomersToolbar** class, add the following implementation for the **render** method.

```
render() {
  return (
    <div className="row btn-toolbar customers-toolbar" role="toolbar" >
      <nav className="container-fluid navbar navbar-expand-xl">
        <div className="search-menu input-group input-group-sm ml-auto">
          <div className="input-group-prepend">
            <span className="input-group-text" id="basic-addon1">Search</span>
          </div>
          <input id="searchbox" type="text" className="form-control form-control-sm" placeholder=""
            onChange={(event: React.ChangeEvent<HTMLInputElement>) => {
              let customerService: ICustomersService = this.props.ViewCustomers.state.customerService;
              let searchString: string = event.target.value;
              if (searchString != "") {
                customerService.getCustomersByLastName(searchString).then((customers: ICustomer[]) => {
                  this.props.ViewCustomers.setState({ customers: customers, loading: false });
                });
              }
              else {
                customerService.getCustomers().then((customers: ICustomer[]) => {
                  this.props.ViewCustomers.setState({ customers: customers, loading: false });
                });
                this.props.ViewCustomers.setState({ customers: [], loading: false });
              }
            }} />
        </div>
      </nav>
    </div>
  );
}
```

8. Modify **ViewCustomers.tsx** to integrate the **CustomersToolbar** component.

   a) Return to **ViewCustomers.tsx** in a code editor window.

   b) Add an **import** statement for the **CustomersToolbar** component.

```
import CustomersToolbar from './CustomersToolbar'
```

   c) Currently, the **render** method of the **ViewCustomers** class should match the following code listing.

```
render() {
  return (
    <div>
      <div className="view-customers" >
        <CustomersTable customers={this.state.customers} />
      </div>
    </div>
  );
}
```

   d) Modify the **render** method by adding the **CustomersToolbar** component as shown in the following listing.

```
render() {
  return (
    <div>
      <CustomersToolbar ViewCustomers={this} />
      <div className="view-customers" >
        <CustomersTable customers={this.state.customers} />
      </div>
    </div>
  );
}
```

9. Run and test the application.

    a) Make sure you have saved your changes to all source files in the project.

    b) Execute the **npm run start** command from the Integrated Terminal console.

    c) When the application starts up, click the **Customer** link to test the **ViewCustomers** component.

    d) You should now see the **CustomersToolbar** component with a search box aligned to the right-side.
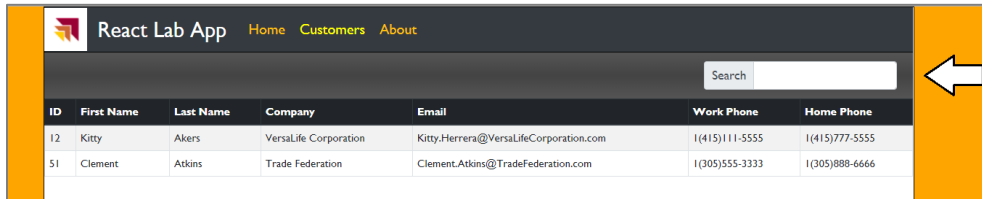
    e) Try a search by typing a capital "C" in the search box. You should now see customers with last name starting with "C".
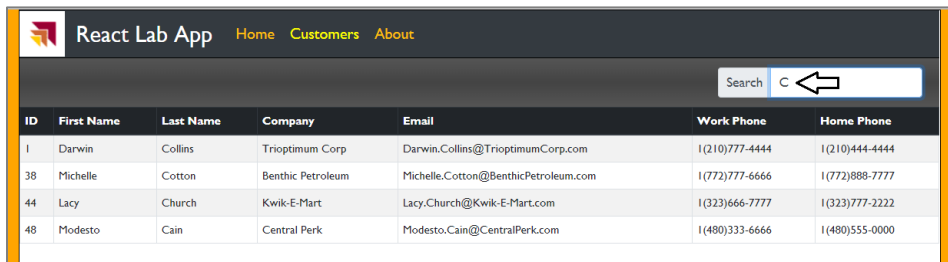
    f) Now type "Co" into the search box to further refine your search.
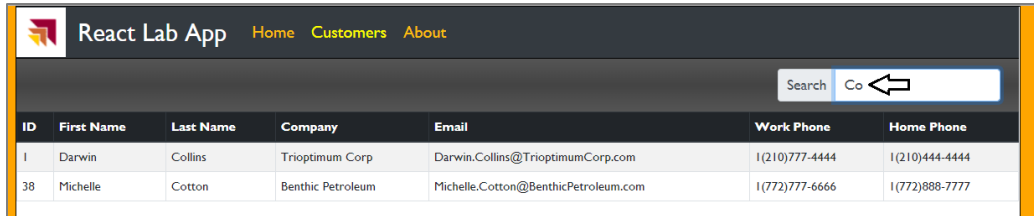
    g) Type "Cod" into the search box the search box and observe the message indicating no customer were returned.
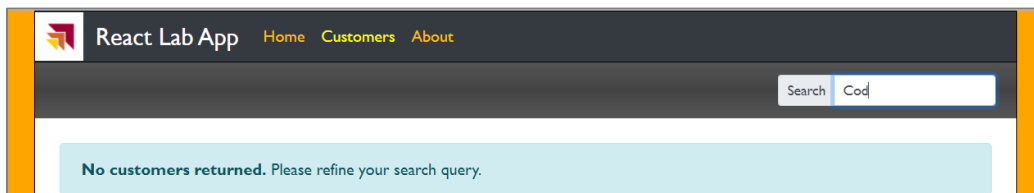
    h) After you have tested the application, close the browser and stop the webpack dev server from running.

10. Add an alphabetic filtering menu to the **CustomersToolbar** component.

    a) Open **CustomersToolbar.tsx** in an editor window.

    b) Inside the **CustomersToolbar** class, add a private field named **letters** containing a capital letter for each letter in the alphabet.

```
export default class CustomersToolbar extends React.Component<CustomersToolbarProperties, any> {

  private letters = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M",
                     "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"];

  render() {
```

c) Place your cursor inside the **nav** section above the **div** with the **search-menu** class and add a few lines to extra create space.

```
export default class CustomersToolbar extends React.Component<CustomersToolbarProperties, any> {

    private letters = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M",
                       "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"];

    render() {
        return (
            <div className="row btn-toolbar customers-toolbar" role="toolbar" >
                <nav className="container-fluid navbar navbar-expand-xl">



                    <div className="search-menu input-group input-group-sm ml-auto">
                        <div className="input-group-prepend">
```

d) Copy and paste the following code just under the opening **nav** element tag where you made extra space.

```
<div className="filter-menu btn-group btn-group-sm " role="group" >
  {this.letters.map((letter: string) =>
    <button type="button" key={letter} className="btn btn-sm btn-secondary"
      onClick={() => {
        (document.getElementById('searchbox') as HTMLInputElement).value = letter;
        let customerService: ICustomersService = this.props.ViewCustomers.state.customerService;
        customerService.getCustomersByLastName(letter).then((customers: ICustomer[]) => {
          this.props.ViewCustomers.setState({ customers: customers, loading: false });
        });
      }} >
      {letter}
    </button>
  )}
</div>
```

11. Run and test the application.

   a) Make sure you have saved your changes to all source files in the project.

   b) Execute the **npm run start** command from the Integrated Terminal console.

   c) When the application starts up, click the **Customer** link to test the **ViewCustomers** component.

   d) You should now see the **CustomersToolbar** component now displays a new filtering menu.

   e) Click on the button with the letter "C" to test the filtering behavior.
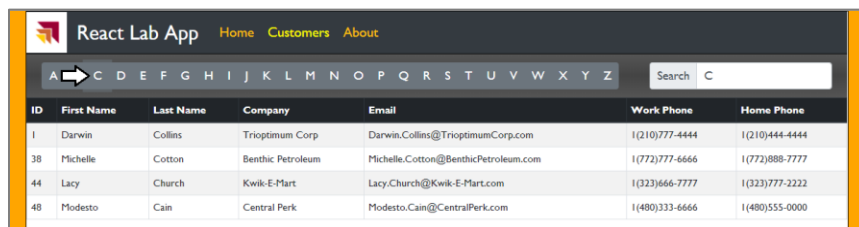
   f) Note that clicking a button also add that character to the search box to show the user the current search filtering.


   g) Click on the letter "X". Your user interface experience should display a view indicating no customers were returned.
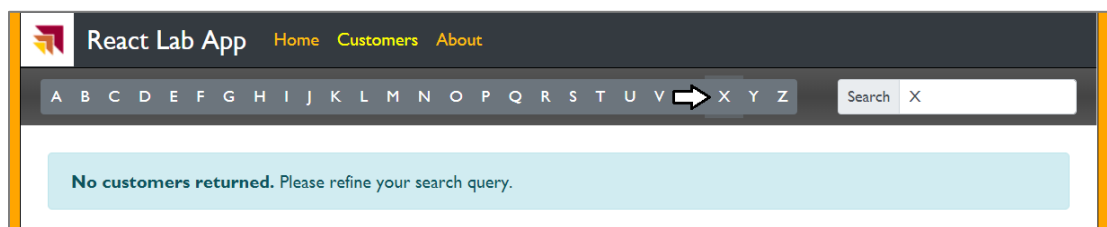
Over the next few steps you will create a new React component named **CustomerCard** to display customer data in an alternate fashion. After creating the **CustomerCard** component, you will then modify the **ViewCustomers** component so the user can switch between displaying customer data with the **CustomersTable** component and the **CustomerCard** component.

12. Add new React component named **CustomerCard**.

    a) Inside the **src/ViewCustomers** folder, add two new source files named **CustomersCard.css** and **CustomersCard.tsx**.
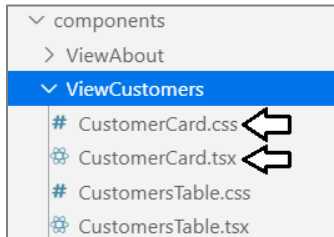
    b) Add the following CSS rules to **CustomerCard.css**.

```css
.view-customers .customer-card {
  margin: 4px;
  padding: 0px;
  border: 1px #888 solid;
  display: inline-block;
  width: 210px;
  height: 90px;
  background-color: #EEE;
}

.customer-card:hover {
  border: 2px solid darkred;
  background-color: lightyellow;
}

.customer-card .card-header {
  font-size: 1.0em;
  padding: 2px;
  padding-left: 4px;
  background-color: black;
  color: white;
}

.customer-card .card-body {
  padding: 8px;
  font-size: 0.85em;
  max-height: 40px;
}
```

    c) Save and close **CustomerCard.css**.

A page break has been introduced to prevent breaking the next code listing across pages.

d) Add the following code to **CustomerCard.tsx**.

```
import * as React from 'react';

import ICustomer from "./../../models/ICustomer";

import './CustomerCard.css';

interface CustomerCardProperties {
  customer: ICustomer;
}

export default class CustomerCard extends React.Component<CustomerCardProperties, any> {

  render() {
    return (
      <div className="card customer-card" >
        <div className="card-header">
          {this.props.customer.FirstName + " " + this.props.customer.LastName}
        </div>
        <div className="card-body">
          <div className="card-text" >
            Work Phone: <strong>{this.props.customer.WorkPhone}</strong>
          </div>
          <div className="card-text" >
            Home Phone: <strong>{this.props.customer.HomePhone}</strong>
          </div>
        </div>
      </div>
    );
  }
}
```

e) Save and close **CustomerCard.tsx**.

Note that the **CustomerCard** component is different than the **CustomersTable** because it displays data for a single customer instead of a set of customers. When you integrate the **CustomerCard** component by modifying the **render** method of the **ViewCustomers** component, you will create a separate instance of the **CustomerCard** component for each customer. However, before you update the **ViewCustomers** component, you must first update the **CustomersToolbar** component to add a menu to toggle between views.

13. Update the **CustomersToolbar.tsx** with a new menu to toggle between views.

a) open **CustomersToolbar.tsx** in editor window.

b) Place your cursor inside the **render** method above the **return** statement and add the following code.

```
let inTableView: boolean = this.props.ViewCustomers.state.viewType == "table";
```

c) The new line you added should appear at the top of the **render** method as shown in the following code.

```
export default class CustomersToolbar extends React.Component<CustomersToolbarProperties, any> {

    private letters = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M",
      "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"];

    render() {
      let inTableView: boolean = this.props.ViewCustomers.state.viewType == "table";

      return (
        <div className="row btn-toolbar customers-toolbar" role="toolbar" >
```

d) Place your cursor inside the **nav** section above the **div** with the **filter-menu** class and add a few lines to create some space.

```
return (
    <div className="row btn-toolbar customers-toolbar" role="toolbar" >
        <nav className="container-fluid navbar navbar-expand-xl">


            <div className="filter-menu btn-group btn-group-sm " role="group" >
                {this.letters.map((letter: string) =>
```

e) Copy and paste the following code just under the opening **nav** element tag where you made extra space.

```
<div className="view-menu btn-group btn-group-sm" role="group" >
  <button type="button"
    className={"btn btn-sm btn-secondary" + (inTableView ? " active" : "")}
    onClick={() => { this.props.ViewCustomers.setState({ 'viewType': 'table' }) }} >Table View</button>
  <button type="button"
    className={"btn btn-sm btn-secondary" + (!inTableView ? " active" : "")}
    onClick={() => { this.props.ViewCustomers.setState({ 'viewType': 'cards' }) }} >Cards View</button>
</div>
```

f) Save and close **CustomersToolbar.tsx**.

> Note that the **CustomersToolbar** component gets passed a direct reference to the **ViewCustomers** component. This design makes it possible for the **CustomersToolbar** component to view and set the state of the **ViewCustomers** component. Now it's time to modify the **ViewCustomers** component to *react* to the user switching between views.

14. Update the **ViewCustomers** component to integrate a secondary view based on the **CustomerCard** component.

a) Open **ViewCustomers.tsx** in an editor window if it's not already open.

b) At the top of **ViewCustomers.tsx** add a new **import** statement for the **CustomerCard** component.

```
import CustomerCard from './CustomerCard';
```

c) Move down inside **ViewCustomers.tsx** and examine the code inside the **render** method.

d) Locate the **div** element with the **view-customer** class which currently contains a single **CustomersTable** element.

```
render() {
  return (
    <div>
      <CustomersToolbar ViewCustomers={this} />
      <div className="view-customers" >
        <CustomersTable customers={this.state.customers} />
      </div>
    </div>
  );
}
```

e) Replace the **div** element with **view-customer** class with the following HTML

```
<div className="view-customers" >
  {this.state.viewType === "table" ?
    <CustomersTable customers={this.state.customers} /> :
    (this.state.customers.map((customer: ICustomer) => <CustomerCard customer={customer} />))
  }
</div>
```

f) Save our changes to **ViewCustomers.tsx**.

15. Run and test the application.

a) Make sure you have saved your changes to all source files in the project.

b) Execute the **npm run start** command from the Integrated Terminal console.

c) When the application starts up, click the **Customer** link to test the **ViewCustomers** component.

d) You should now see the **CustomersToolbar** component now displays a new menu for switching views.
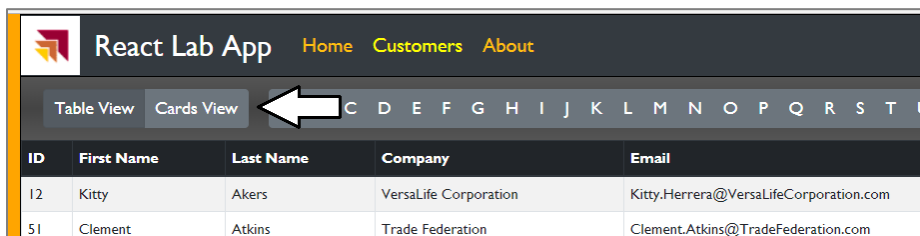
e) Click on the **Cards View** button and verify you can switch between views.
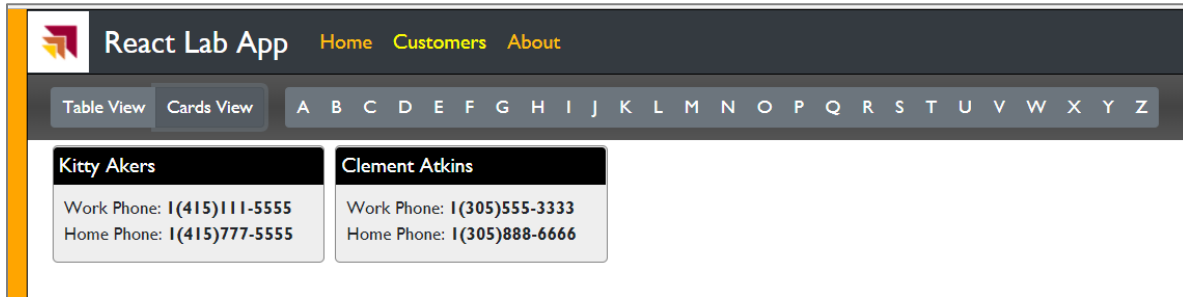
f) While in Card View, experiment using the search box and by pressing different filter buttons.
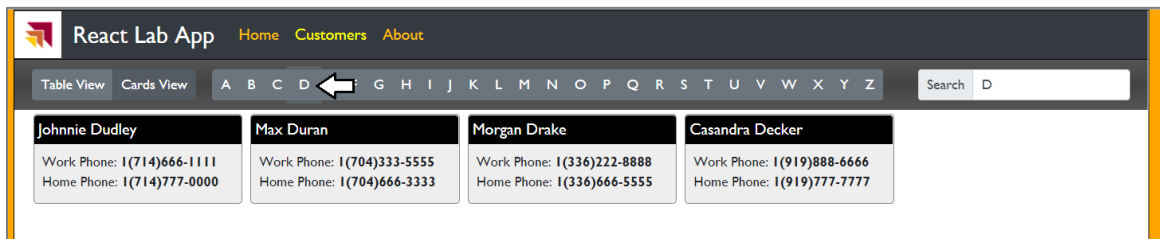
Filtering and searching should continue to work when switching between views.

a) After you have tested the application, close the browser and stop the webpack dev server from running.

Exercise 5: Retrieving Data from an OData Web Service using the React Fetch API

In this lab you will create a new service class named **CustomersService** that will replace the **MockCustomersService** service class. You will write the **CustomersService** class to implement the **ICustomersService** interface so that it will be easy to substitute this component into the application where the **MockCustomersService** class is currently being used.

1. Create a new class named **CustomersService** class which implements the **ICustomersService** interface..

    a) Inside the **services** folder, add a new source file named **CustomersService.ts**.
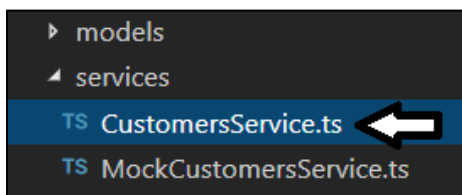
    b) Add the following code into **CustomersService.ts** to provide a starting point for the **CustomersService** class.

```
import ICustomer from "../models/ICustomer";
import ICustomerDetail from "../models/ICustomerDetail";
import ICustomersService from "../models/ICustomersService";

export default class CustomersService implements ICustomersService {

  getCustomers(): Promise<ICustomer[]> { }

  getCustomersByLastName = (lastNameSearch: string): Promise<ICustomer[]> => { }

  getCustomer = (customerId: string): Promise<ICustomerDetail> => { }

}
```

c) Replace the **getCustomers** method with the following code.

```
getCustomers(): Promise<ICustomer[]> {

  const restUrl =
    "http://subliminalsystems.com/api/Customers/?" +
    "$select=CustomerId,LastName,FirstName,EmailAddress,WorkPhone,HomePhone,Company" +
    "&$filter=(CustomerId+le+12)&$top=200";

  return fetch(restUrl)
    .then(response => response.json())
    .then(response => {
      return response.value;
    });
}
```

d) Replace the **getCustomersByLastName** method with the following code.

```
getCustomersByLastName = (lastNameSearch: string): Promise<ICustomer[]> => {

  const restUrl =
    "http://subliminalsystems.com/api/Customers/?" +
    "$select=CustomerId,LastName,FirstName,EmailAddress,WorkPhone,HomePhone,Company" +
    `&$filter=startswith(tolower(LastName),tolower('${lastNameSearch}'))&$orderby=LastName,FirstName`;

  return fetch(restUrl)
    .then(response => response.json())
    .then(response => {
      return response.value;
    });
}
```

e) Replace the **getCustomer** method with the following code.

```
getCustomer = (customerId: string): Promise<ICustomerDetail> => {

  const restUrl = "http://subliminalsystems.com/api/Customers(" + customerId + ")";

  return fetch(restUrl)
    .then(response => response.json())
    .then(response => {
      return response;
    });
}
```

f) Save your changes and close **CustomersService.ts**.

2. Modify the **ViewCustomers** component to use the **CustomersService** class instead of the **MockCustomersService** class.

   a) Open **ViewCustomer.tsx** in an editor window if it's not already open.

   b) Add an **import** statement for the **CustomersService** class along with the **other** import statements.

```
import * as React from 'react';
import ICustomer from '../../models/ICustomer';
import ICustomersService from '../../models/ICustomersService';
import MockCustomersService from "./../../services/MockCustomersService";
import CustomersService from "./../../services/CustomersService";
import CustomersToolbar from './CustomersToolbar'
import CustomersTable from './CustomersTable';
import CustomerCard from './CustomerCard';
import './ViewCustomers.css';
```

   c) Locate the code inside the **ViewCustomers** class which initializes the component's state.

```
state: ViewCustomersState = {
  viewType: 'table',
  customerService: new MockCustomersService(),
  customers: [],
  loading: false
}
```
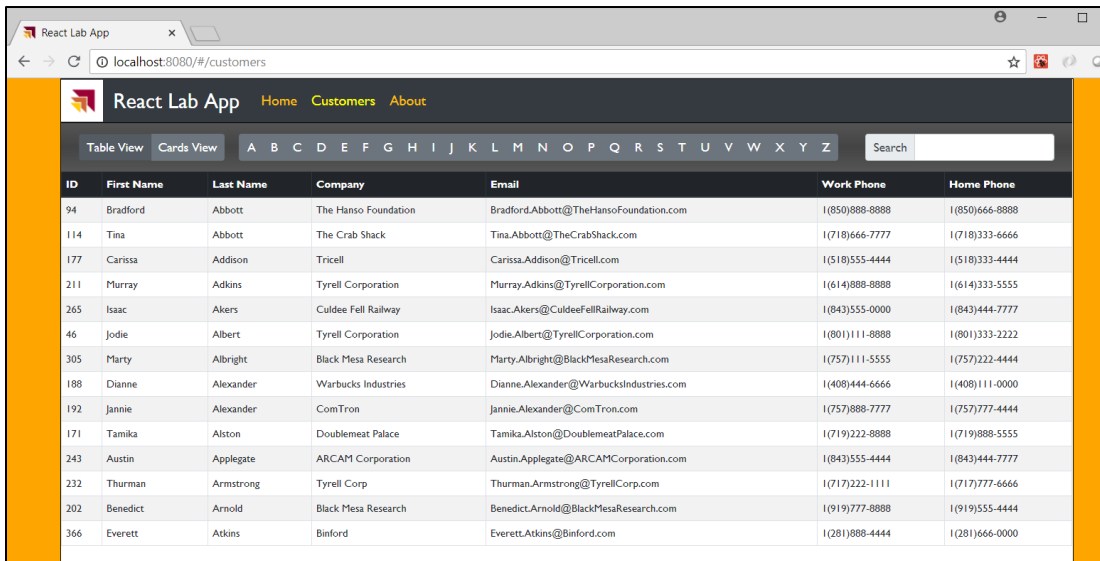
d) Replace the **MockCustomersService** class with the **CustomersService** class.

```
state: ViewCustomersState = {
  viewType: 'table',
  customerService: new CustomersService(),
  customers: [],
  loading: false
}
```
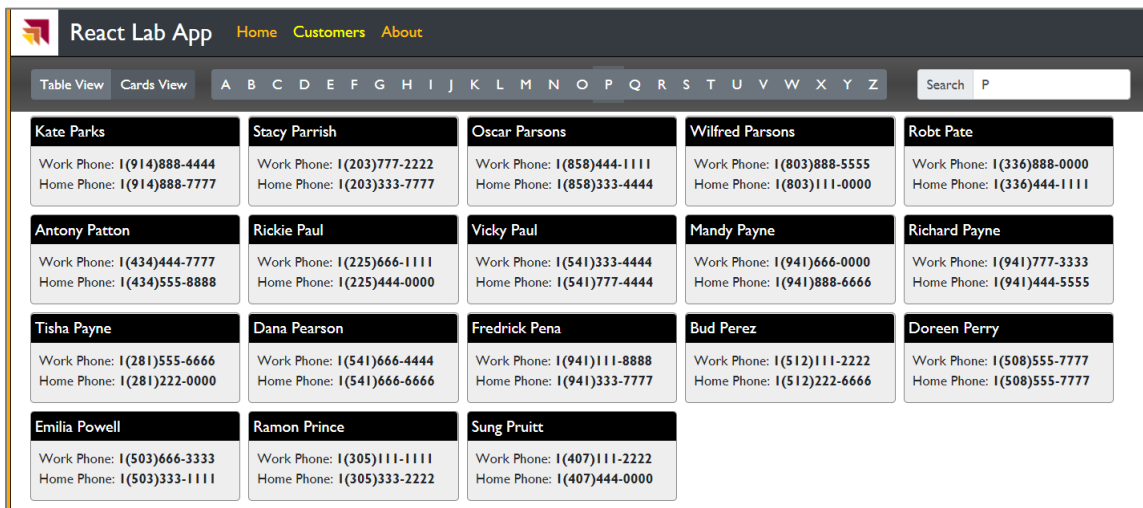
e) Save your changes to **ViewCustomers.tsx**.

3. Run and test the application.

   a) Make sure you have saved your changes to all source files in the project.

   b) Execute the **npm run start** command from the Integrated Terminal console.

   c) When the application starts up, click the **Customer** link to test the **ViewCustomers** component.

   d) The application should now displays a different set of customers which have been retrieved from across the network.

e) You should still be able to switch between views, search and filter just as you did before.

f) Leave the application running and the browser window open for the next step.

4. Use Fiddler to monitor the web service calls made using the React Fetch API.

   a) Open the Fiddler to monitor your local HTTP traffic.

   b) Return to your React application and click a few filter buttons to call across the network.

   c) Use Fiddler to examine the HTTP requests that your application is making.

   d) Select the **Inspectors** tab to see details for each request and response pair.

   e) Select the **WebForms** view for the request and the **JSON** tab for the response.

   f) You should be able to see the OData query string parameter values of the request and the JSON payload of the resonse.
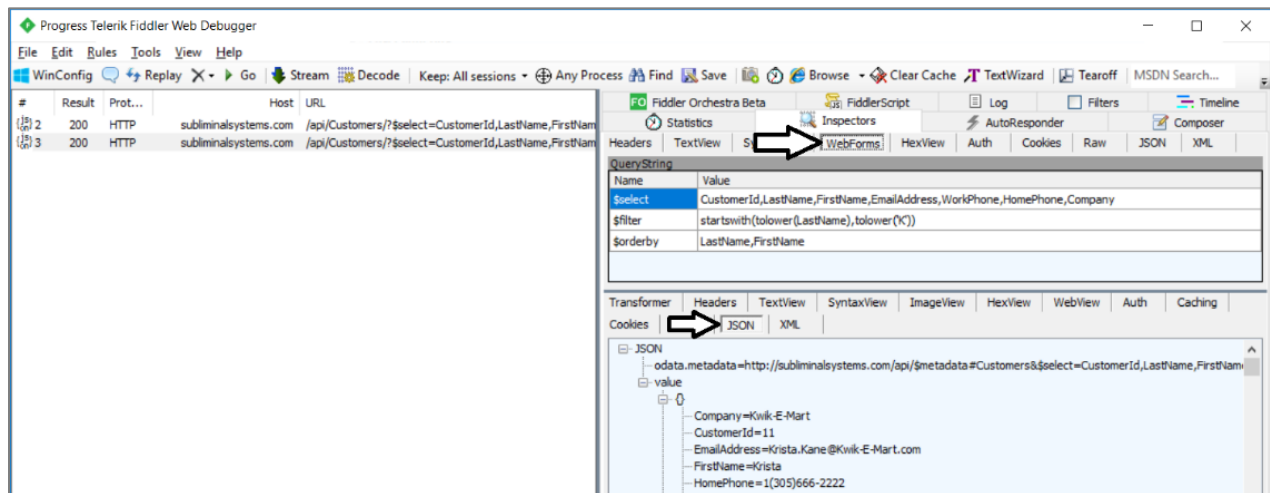
Congratulations. You have now reached the end of this lab.