Developing React WebParts

Lab Time: 60 minutes

Lab Folder: C:\Student\Modules\05_ReactWebparts\Lab

Lab Overview: In this lab, you will begin by creating a simple a SharePoint Framework project with a simple React Webpart. This will give you an chance to work with the React component and apply styling using support from the Office UI Fabric. Next, you will move through the steps developing a React webpart which uses the SharePoint REST API to program against the current SharePoint site.

Exercise 1: Create a React WebPart Styled using the Office UI Fabric

In this exercise you will create and test a simple React webpart.

1. Create a new SharePoint Framework project named **react-webparts-lab**.

    a) From the Node.JS command prompt, run the following command to set your current folder to the folder for this lab.

    ```
    cd C:\Student\Modules\05_ReactWebparts\Lab
    ```
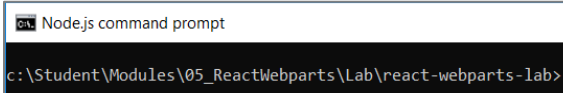
    b) Type the following command and execute it by pressing **Enter** to create a new folder for your project.

    ```
    md react-webparts-lab
    ```

    c) Type the following command and execute it by pressing **Enter** to move to the current directory to the new folder.

    ```
    cd react-webparts-lab
    ```

    d) The current directory for the console should now be located at the new folder you just created named **react-webparts-lab**.
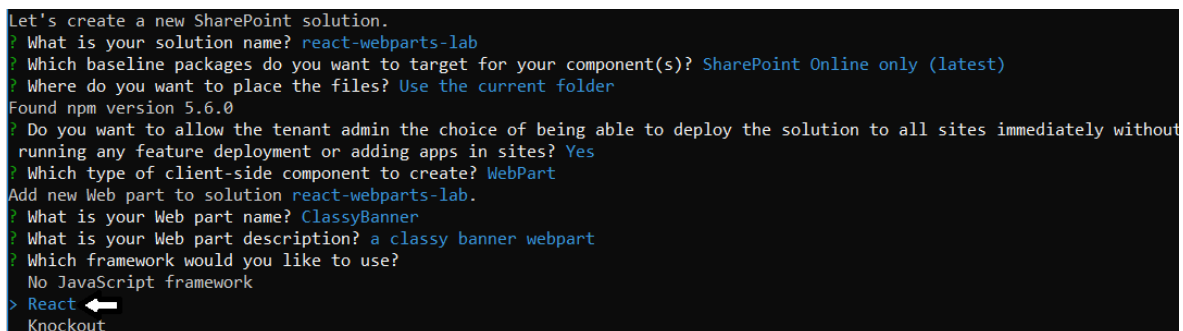
    ```
    Node.js command prompt

    c:\Student\Modules\05_ReactWebparts\Lab\react-webparts-lab>
    ```

    e) Type the following command and execute it to launch the Yeoman generator with the SPFx project template.

    ```
    yo @microsoft/sharepoint
    ```

    f) When prompted with **What is your solution name?**, press **Enter** to accept the default value which is the name of the folder.

    g) When prompted with **Which baseline packages do you want to target for your component(s)?**, press **Enter** to accept the default value of **SharePoint Online only (latest)**.

    h) When prompted **Where do you want to place the files?**, press **Enter** to accept the default value of **Use the current folder**.

    i) When prompted **Do you want to allow the tenant admin the choice of being able to deploy to all sites immediately without running any feature deployment or adding apps in sites (y/N)?**, type "**y**" and press **Enter** to accept the option.

    j) When prompted with **Which type of client-side component to create?**, press **Enter** to accept the default value of **WebPart**.

    k) When prompted with **What is your Web part name?**, type **ClassyBanner** and press **Enter** to submit your value.

    l) When prompted with **What is your Web part description?**, type in a short description and press **Enter**.

    m) When prompted with **Which framework would you like to use?**, select **React** and press **Enter** to create the new project.

    ```
    Let's create a new SharePoint solution.
    ? What is your solution name? react-webparts-lab
    ? Which baseline packages do you want to target for your component(s)? SharePoint Online only (latest)
    ? Where do you want to place the files? Use the current folder
    Found npm version 5.6.0
    ? Do you want to allow the tenant admin the choice of being able to deploy the solution to all sites immediately without
     running any feature deployment or adding apps in sites? Yes
    ? Which type of client-side component to create? WebPart
    Add new Web part to solution react-webparts-lab.
    ? What is your Web part name? ClassyBanner
    ? What is your Web part description? a classy banner webpart
    ? Which framework would you like to use?
      No JavaScript framework
    > React  ⬅
      Knockout
    ```

Once you have answered all the questions, the Yeoman generator will run and add the starter files to your project folder.

a) Wait until the Yeoman generator completes it work and display a message indicating the new solution has been created.

```
added 1837 packages in 40.783s

    _=+#####!
 ###########|
 ###/    (##|(@)    .-----------------------------------.
 ### #####|   \    |          Congratulations!         |
 ###/   /###|   (@) |  Solution react-webparts-lab is created. |
 #######  ##|  /    |    Run gulp serve to play with it!  |
 ###     /##|(@)    '-----------------------------------'
 ###########|
    **=+####!

c:\Student\Modules\05_ReactWebparts\Lab\react-webparts-lab>
```
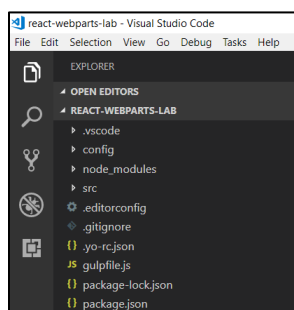
2. Open the project with Visual Studio Code.

a) Type the following command and execute it by pressing **Enter** to open your new project in Visual Studio Code.

```
code .
```

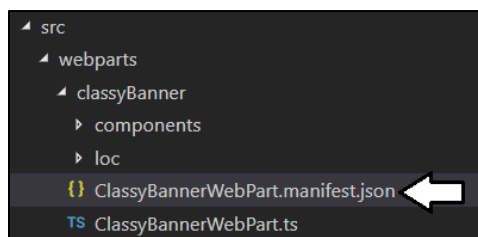b) As the command execute, it should open your new project folder with Visual Studio Code.
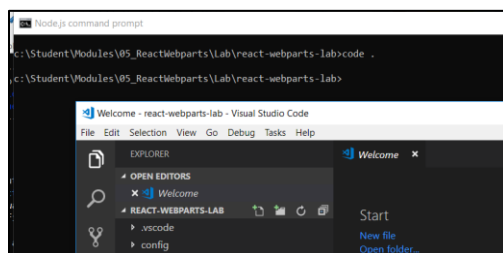
c) Take a moment to familiarize yourself with the files and folders at the root of the **react-web-parts** project.

3. Modify the webpart manifest.

a) Expand the **src/webparts/classBanner** folder.

b) Open the webpart manifest named **ClassyBannerWebPart.manifest.json**.

    c)   Move down inside the **preconfiguredEntries** section of the webpart manifest.

    d)   Update the **title** property to **"Classy Banner"**.

    e)   Update the **officeFabricIconFontName** property to **"News"**.

```
"preconfiguredEntries": [{
  "groupId": "5c03119e-3074-46fd-976b-c60198311f70",
  "group": { "default": "Other" },
  "title": { "default": "Classy Banner" },
  "description": { "default": "a classy banner webpart" },
  "officeFabricIconFontName": "News",
  "properties": {
    "description": "ClassyBanner"
  }
}]
```

> Note that the **description** property is automatically added by the Yeoman webpart template. The **description** property not be used in this exercise and you can remove it if you'd like.

    f)   Save and close **ClassyBannerWebPart.manifest.json**.

4.   Inspect (but do not update) the webpart implementation class named **ClassyBannerWebPart**.

    a)   Open the webpart implementation file named **ClassyBannerWebPart.ts.**
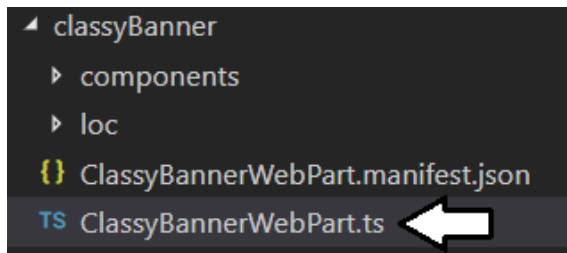
    b)   Take a look at the **render** method to examine how it creates an instance of the React component named **ClassyBanner**.

```
public render(): void {
  const element: React.ReactElement<IClassyBannerProps > = React.createElement(
    ClassyBanner,
    {
      description: this.properties.description
    }
  );

  ReactDom.render(element, this.domElement);
}
```

> In simple scenarios like the one in this lab exercise, there is no need to modify the webpart class because all the changes you need to make can be made to the React component that the webpart instantiates. However, as the design of your webpart becomes less trivial, it is often required to update the webpart class to pass data such as persistent webpart properties to the React component.

    c)   Once you have examined the **render** method, close **ClassyBannerWebPart.ts** without saving any changes.

> Over the next steps, you will configure your project to use the Office UI Fabric. However, you cannot do this unless you uninstall the SharePoint Framework Fabric Core package that layers on top of the Office UI Fabric. Therefore, your first step is to uninstall the SharePoint Framework Fabric Core package using the **npm** utility from the console window.

5.   Uninstall the SharePoint Framework Fabric Core package.

    a)   Navigate to the Terminal console.

    b)   Type and execute the following command to uninstall the SPFx Fabric Core library which is not needed.

```
npm uninstall @microsoft/sp-office-ui-fabric-core
```

    c)   The SPFx Fabric Core should have been removed from your project.

6. Update the CSS styles used by the React webpart in the CSS module named **ClassyBanner.module.scss**.

    a) Expand the **src/webparts/classyBanner/components** folder.

    b) Open the source file with the React component named **ClassyBanner.module.scss**.

    c) You should see that the first line in **ClassyBanner.module.scss** includes an **@import** statement.

    ```
    @import '~@microsoft/sp-office-ui-fabric-core/dist/sass/SPFabricCore.scss';
    ```

    d) Delete the existing **@import** statement and replace it with the following an **@import** statement.

    ```
    @import '~office-ui-fabric-react/dist/sass/_References.scss';
    ```

    e) Delete all the code underneath the **@import** statement and replace it with the following code.

    ```
    @import '~office-ui-fabric-react/dist/sass/_References.scss';

    .classyBanner {
      .container {
        max-width: 800px;
        height: 122px;;
        margin: 0px auto;
        border: 1px solid black;
        border-radius: 8px;
        @include ms-Grid;
      }

      .row {
        @include ms-Grid-row;
      }

      .body {
        @include ms-Grid-col;
        @include ms-lg10;
        background-color: $ms-color-themeLight;

      }

      .title {
          @include ms-font-xl;
          @include ms-fontColor-white;
          background-color: $ms-color-themeDark;
          margin: 4px;
          padding: 4px;
          text-align: center;
          border-radius: 6px;
      }

      .image {
        height: 120px;
        @include ms-Grid-col;
        @include ms-lg2;
        background-color: red;
        border-left: 1px solid black;
      }

    }
    ```

> You should take note of how this SCSS code uses the **@include** statement to import CSS styles into your CSS class from the classes defined by Microsoft in in the Office UI Fabric library.

    f) Save your changes to **ClassyBanner.module.scss**.

7. Run a gulp build command to rebuild the CSS module named **ClassyBanner.module.scss**.

    a) Use the **View > Terminal** menu command in Visual Studio Code to display the Terminal console.

    b) Run the following command to rebuild the project which will have the effect of rebuilding the CSS module as well.

    ```
    gulp build
    ```

> The reason to run the **gulp build** command is to rebuild the CSS module which will, in turn, make the new styles you have defined inside **ClassyBanner.module.scss** appear in IntelliSense when you apply the styles in **ClassyBanner.tsx**.

8. Update the HTML layout generated by the webpart's React component in **ClassyBanner.tsx**.

   a) Inside the **src/webparts/classyBanner/components** folder, open **ClassyBanner.tsx**.

   b) Delete the existing code inside **ClassyBanner.tsx** and replace it with the following code.

```
import * as React from 'react';
import styles from './ClassyBanner.module.scss';

export default class ClassyBanner extends React.Component<any, any> {
  public render(): React.ReactElement<any> {
    return (
      <div className={styles.classyBanner}>
        <div className={styles.container}>
          <div className={styles.row}>
            <div className={styles.body}>
              <div className={styles.title}>
                I am a Modern Developer using the SharePoint Framework
              </div>
            </div>
            <div className={styles.image} />
          </div>
        </div>
      </div>
    );
  }
}
```

   c) Save your changes to **ClassyBanner.tsx**.

9. Test out the **react-webparts-lab** project by running it in the local SharePoint workbench

   a) Navigate to the Terminal console.

   b) Execute the **gulp serve** command to start up the project and test it out using the local workbench.

```
gulp serve
```

   c) The browser should launch and display a page for adding modern webparts like the one shown in the following screenshot. Click on the button with the **+** sign in the middle of the page to add your webpart to the page so you can test it.
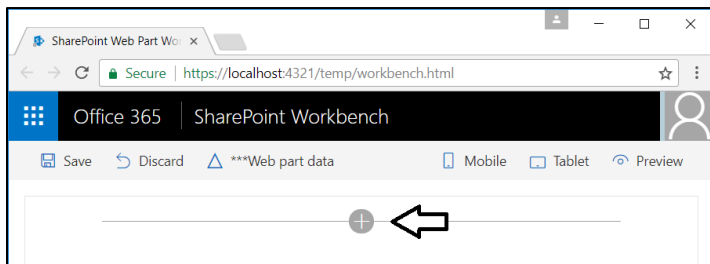
   d) Select the **Classy Banner** to add it to the page as a new webpart instance.
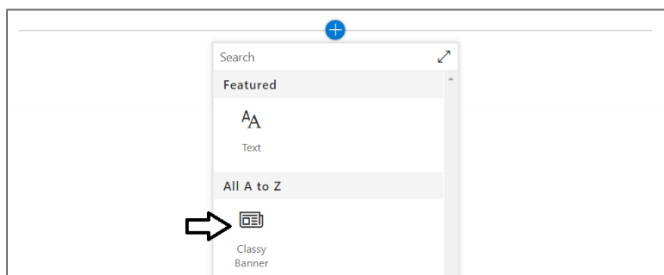
e) The webpart should appear as the one shown in the following screenshot.
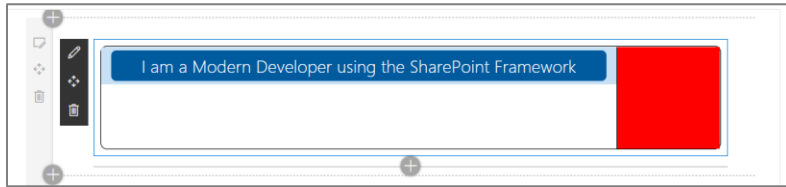
Leave this page with the Classy Banner webpart open in the browser as you continue to work on this exercise. Whenever you save changes to either **ClassyBanner.module.scss** or **ClassyBanner.tsx**, the project will automatically rebuild and refresh the webpart in the browser so you can quickly see the effects of your changes.

10. Add new CSS styles to **ClassyBanner.module.scss** to style content in the body of the banner.

a) Open **ClassyBanner.module.scss** in an editor window

b) Add the following three new CSS classes into **ClassyBanner.module.scss** just below the **.title** class

```
.bodyContent {
    margin-left: 16px;
}

.bodyContent p {
    color: black;
}

.bodyContent p i {
    margin-right: 8px;
}
```

c) Save your changes to **ClassyBanner.module.scss**.

d) Run the **gulp build** command to rebuild the CSS module.

11. Update **ClassyBanner.tsx** to add content to the banner body.

a) Return to the editor window for **ClassyBanner.tsx**.

b) Locate the **div** element with the **className** of **styles.title**.

```
<div className={styles.title}>
  I am a Modern Developer using the SharePoint Framework
</div>
```

c) Below the **div** element with the **className** of **styles.title**, add the following TSX code.

```
<div className={styles.bodyContent}>
  <p>
    <i className="ms-Icon ms-Icon--Emoji2" aria-hidden="true"></i>
    Programming in Typescript makes me happy!
  </p>
  <p>
    <i className="ms-Icon ms-Icon--Permissions" aria-hidden="true"></i>
    Developing with SPFx is the key to my future!
  </p>
</div>
```

d) Save your changes to **ClassyBanner.tsx**.

e) Return to the browser and your webpart should match the webpart shown in the following screenshot.
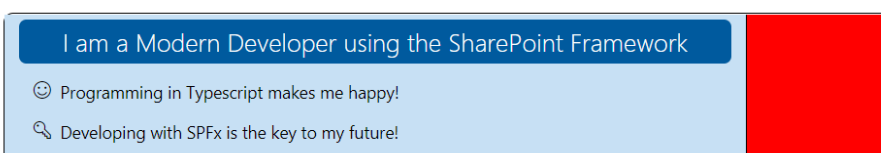
12. Add an image file to the project to use in the webpart display.

   a) Using Windows Explorer, locate the image file named **BannerImage.jpg** at the following location.

   ```
   C:\Student\Modules\05_ReactWebparts\Lab\StarterFiles\BannerImage.jpg
   ```

   b) Using Windows Explorer, copy the **BannerImage.jpg** file to the Windows clipboard.

   c) Stay in Windows Explorer, navigate to the webpart **components** folder inside your project at the following path.

   ```
   C:\Student\Modules\05_ReactWebparts\Lab\react-webparts-lab\src\webparts\classyBanner\components
   ```

   d) Paste the image file named **BannerImage.jpg** from the Windows clipboard into the **components** folder.

   e) Return to Visual Studio Code and verify that you can see **BannerImage.jpg** in the components folder.
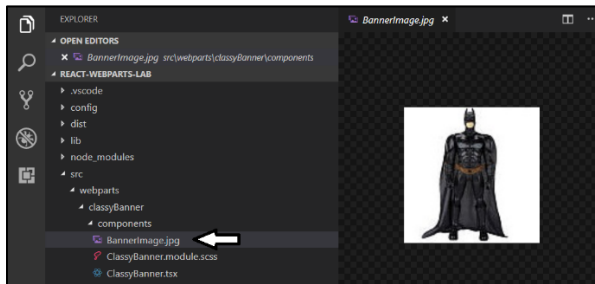
13. Reference the image file from a CSS class in **ClassyBanner.module.scss**.

   a) Return to the editor window for **ClassyBanner.module.scss**.

   b) Locate the **.image** class at the bottom of **ClassyBanner.module.scss**.

   ```
   .image {
     height: 120px;
     @include ms-Grid-col;
     @include ms-lg2;
     background-color: red;
     border-left: 1px solid black;
   }
   ```

   c) Extend the **.image** class by adding the **background** style and the **background-repeat** style as shown in the following listing.

   ```
   .image {
     height: 120px;
     @include ms-Grid-col;
     @include ms-lg2;
     background-color: red;
     border-left: 1px solid black;
     background: url('./BannerImage.jpg');
     background-repeat: no-repeat;
   }
   ```

   d) Save your changes to **ClassyBanner.module.scss**.

   e) Return to the browser and your webpart should now display the image as shown in the following screenshot.
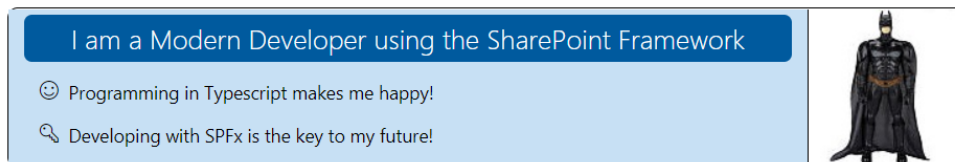
   f) Close the browser window with the webpart, return to Visual Studio Code and stop the debugging session.

Leave the **react-webparts-lab** project open in the Visual Studio Code because you will continue to work on it in the next exercise.

Exercise 2: Create a React Webpart with a Synchronized Property

In this lab, you will create a second React webpart in the **react-webparts-lab** project to display the data from items in a SharePoint list.
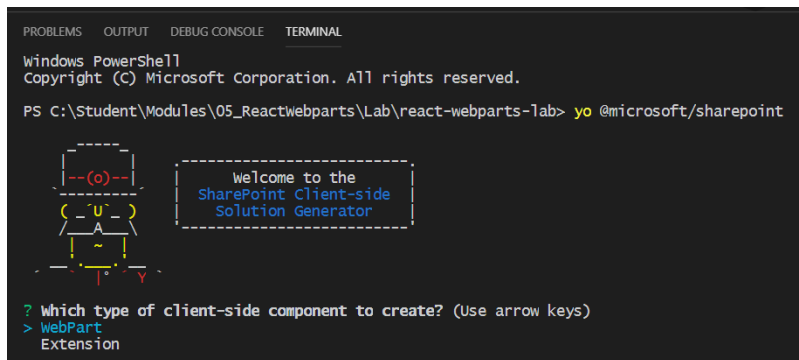
1. Add a new webpart to the **react-webparts-lab** project.

   a) Return to the **react-webparts-lab** project in Visual Studio Code.

   b) Navigate to the Terminal console.

   c) Type the following command and execute it to launch the Yeoman generator with the SPFx project template.

   ```
   yo @microsoft/sharepoint
   ```

   d) Make sure to execute this command in the context of the top-level folder for the **react-webparts-lab** project.

   Since you are running this command inside the context of a folder that already contains a SharePoint Framework project, the Yeoman project wizard for SharePoint Framework projects does not prompt you will all the same questions because all the core project files have already been added. Instead, the Yeoman project wizard begins by asking what type of component you want to create.

   e) When prompted with **Which type of client-side component to create?**, press **Enter** to accept the default value of **WebPart**.
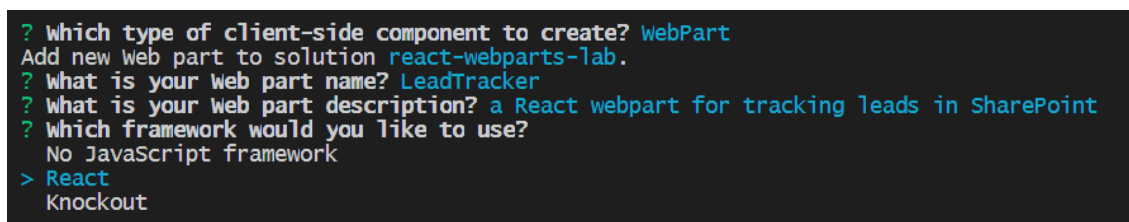
   f) When prompted with **What is your Web part name?**, type **LeadTracker** and press **Enter** to submit your value.

   g) When prompted with **What is your Web part description?**, type in a short description and press **Enter**.

   h) When prompted with **Which framework would you like to use?**, press **Enter** to accept **React**.

   Once you have answered all the questions, the Yeoman generator will run and add the new webpart files to your project folder.
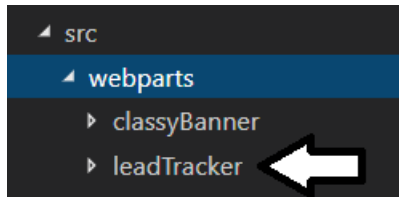
   i) Wait until the Yeoman generator completes it work and displays a message indicating the new solution has been created.

    j)   Look inside the **src/webparts** folder and verify you see the new folder named **leadTracker** for the webpart you just created.

```
⊿ src
   ⊿ webparts
      ▷ classyBanner
      ▷ leadTracker   ⇐
```

2. Modify the interface named **ILeadTrackerProps** which defines the React component properties.

   a)   Inside the **components** folder, click on **ILeadTrackerProps.ts** to open the file in an editor window.

```
⊿ leadTracker
   ⊿ components
      TS ILeadTrackerProps.ts   ⇐
      🎨 LeadTracker.module.scss
```
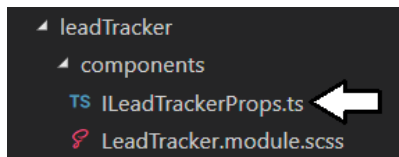
    b)   Update the **ILeadTrackerProps** interface using the following definition.

```
export interface ILeadTrackerProps {
  targetListDefault: string;
}
```

    c)   Save your changes and close **ILeadTrackerProps.ts**.

3. Add a new interface named **ILeadTrackerState** to provide the definition of the React component state.

    a)   Inside the components folder, create a new source file named **ILeadTrackerState.ts**.

```
⊿ leadTracker
   ⊿ components
      TS ILeadTrackerProps.ts
      TS ILeadTrackerState.ts   ⇐
      🎨 LeadTracker.module.scss
      ⚛ LeadTracker.tsx
```
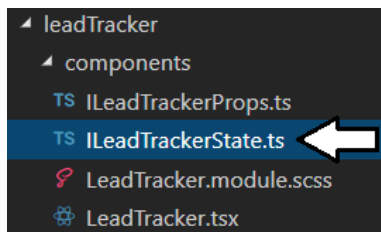
    b)   Add the following code into **ILeadTrackerState.ts** to define the **ILeadTrackerState** interface.

```
export interface ILeadTrackerState {
  targetList: string;
  loading: boolean;
}
```

4. Modify the webpart class in **LeadTrackerWebPart.ts**.

    a)   Inside the **leadTracker** folder, click on **LeadTrackerWebPart.ts** to open the file in an editor window.

```
⊿ leadTracker
   ▷ components
   ▷ loc
   {} LeadTrackerWebPart.manifest.json
   TS LeadTrackerWebPart.ts   ⇐
```
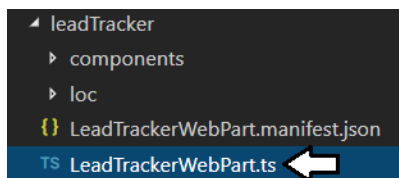
    b)   Inside **LeadTrackerWebPart.ts**, locate the existing definition for the interface named **ILeadTrackerWebPartProps**.

```
export interface ILeadTrackerWebPartProps {
  description: string;
}
```

c) Update the interface to contain a single property named **targetList**.

```
export interface ILeadTrackerWebPartProps {
  targetList: string;
}
```

d) Move down inside the **LeadTrackerWebPart** class.

e) Add a private field to the class named **leadTracker** based on the **LeadTracker** type.

```
export default class LeadTrackerWebPart extends BaseClientSideWebPart<ILeadTrackerWebPartProps> {

  private leadTracker: LeadTracker;
```

f) Replace the current implementation of **render** with the following code.

```
public render(): void {
  const element: React.ReactElement<ILeadTrackerProps> = React.createElement(
    LeadTracker, { targetListDefault: this.properties.targetList }
  );
  this.leadTracker = <LeadTracker>ReactDom.render(element, this.domElement);
}
```

There are two important changes that have been made to the **render** method. First, the webpart class is now initializing the **targetListDefault** property of the React component using its persistent webpart property named **targetList**. Second, the **render** method is now assigning the return value from **ReactDom.render** to the private file named **leadTracker**. This design is important because it gives the webpart class the ability to directly call methods on the React component such as **setState**.

g) Move down inside the **LeadTrackerWebPart** class and locate the method named **onDispose**.

h) Just below the **onDispose** method, add a new method named **onPropertyPaneFieldChanged** using the following code.

```
rotected onPropertyPaneFieldChanged(propertyPath: string, oldValue: any, newValue: any): void {
  super.onPropertyPaneFieldChanged(propertyPath, oldValue, newValue);
  if (propertyPath === 'targetList' && newValue) {
    this.leadTracker.setState({ targetList: newValue });
  }
}
```

i) Move down inside the **LeadTrackerWebPart** class and locate the **getPropertyPaneConfiguration** method.

j) Replace the implementation of **getPropertyPaneConfiguration** with the following code.

```
protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {
  return {
    pages: [
      { header: { description: "List Tracker Properties" },
        groups: [ {
            groupName: "Data source",
            groupFields: [
              PropertyPaneTextField('targetList', { label: "Target List"})
            ]
          }
        ]
      }
    ]
  };
}
```

5. Modify webpart properties in the webpart manifest file named **LeadTrackerWebPart.manifest.json**.

a) Click on the file named **LeadTrackerWebPart.manifest.json** to open it in an editor window.
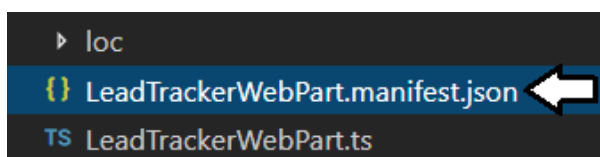
b) Remove all the comments from **LeadTrackerWebPart.manifest.json** to get rid of the red underlining.

c) Inspect the properties inside the **preconfiguredEntries** section.

d) Update the **title** and **officeFabricIconFontName** properties and add the **targetList** property to match the following code.

```
"preconfiguredEntries": [{
  "groupId": "5c03119e-3074-46fd-976b-c60198311f70",
  "group": { "default": "Other" },
  "title": { "default": "Lead Tracker" },
  "description": { "default": "a lead tracker webpart" },
  "officeFabricIconFontName": "ContactCard",
  "properties": {
    "targetList": "Leads"
  }
}]
```

e) Save your changes and close **LeadTrackerWebPart.manifest.json**.

6. Update the CSS styles used by the React webpart in the CSS module named **LeadTracker.module.scss**.

a) Expand the **src/webparts/leadTracker/components** folder.

b) Open the source file with the React component named **LeadTracker.module.scss**.

c) You should see that the first line in **LeadTracker.module.scss** includes an **@import** statement.

```
@import '~@microsoft/sp-office-ui-fabric-core/dist/sass/SPFabricCore.scss';
```

d) Delete the existing **@import** statement and replace it with the following an **@import** statement.

```
@import '~office-ui-fabric-react/dist/sass/_References.scss';
```

e) Delete all the code underneath the **@import** statement and replace it with the following code.

```
@import '~@microsoft/sp-office-ui-fabric-core/dist/sass/SPFabricCore.scss';

.leadTracker {
    border: 1px solid darkblue;
    background-color: lightyellow;
    padding: 8px;
}
```

f) Save your changes to **LeadTracker.module.scss** but leave the file open so you can continue to edit it.

7. Update the HTML layout generated by the webpart's React component in **LeadTracker.tsx**.

a) Inside the **src/webparts/leadTracker/components** folder, open **LeadTracker.tsx**.

b) Delete the existing code inside **LeadTracker.tsx** and replace it with the following code.

```
import * as React from 'react';
import styles from './LeadTracker.module.scss';
import { ILeadTrackerProps } from './ILeadTrackerProps';
import { ILeadTrackerState } from './ILeadTrackerState';
import { escape } from '@microsoft/sp-lodash-subset';

export default class LeadTracker extends React.Component<ILeadTrackerProps, ILeadTrackerState> {

  public state: ILeadTrackerState = {
    targetList: this.props.targetListDefault,
    loading: false
  };

  public render(): React.ReactElement<ILeadTrackerProps> {
    return (
      <div className={styles.leadTracker}>
        <p>Target List: <strong>{escape(this.state.targetList)}</strong></p>
      </div>
    );
  }
}
```

    c)   Save your changes to **LeadTracker.tsx** but leave the file open so you can continue to edit it.

8.   Test out the **LeadTracker** webpart by running it in the local SharePoint workbench

    a)   Make sure you have saved your changes to all files within the **react-webparts-lab** project.

    b)   Navigate to the Terminal console.

    c)   Execute the **gulp serve** command to start up the project and test it out using the local SharePoint Workbench.

```
gulp serve
```

    d)   The browser should launch and display a page for adding modern webparts like the one shown in the following screenshot. Click on the button with the **+** sign in the middle of the page to add your webpart to the page so you can test it.
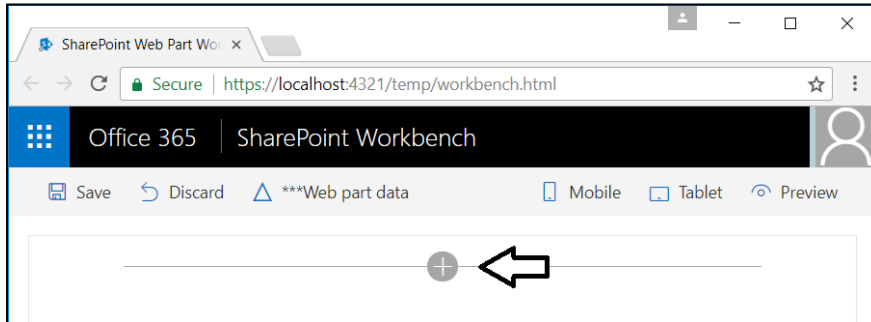
    e)   Select the **Lead Tracker** webpart to add it to the page as a new webpart instance.
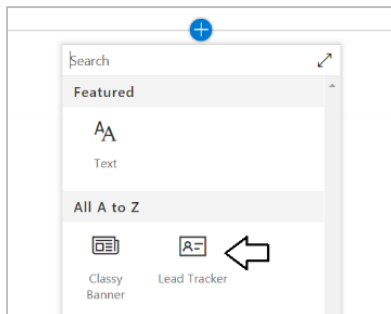
    f)   The **ListTracker** webpart should appear as the one shown in the following screenshot.
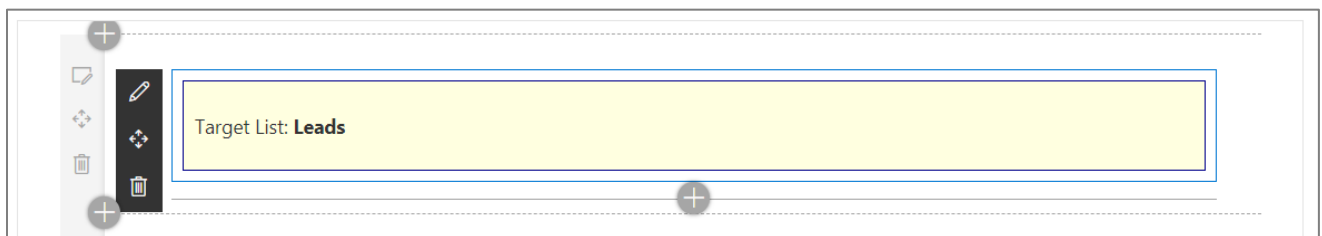
    g)   Click on the button with the pen icon to move the webpart into edit mode.
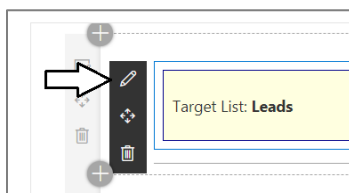
h) You should see the properties pane on the right with a textbox holding the current value of the **targetList** property.
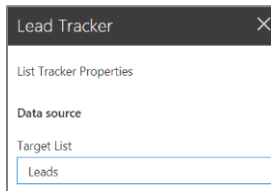
i) Edit the **targetList** property value and verify that your edits are automatically reflected in the webpart display.
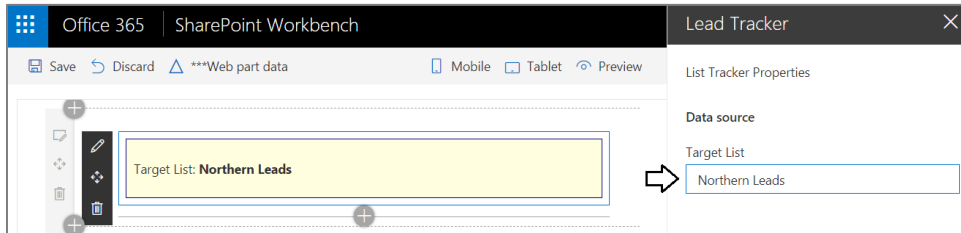
j) Close the browser window with the webpart, return to Visual Studio Code and stop the debugging session.

Leave the **react-webparts-lab** project open in the Visual Studio Code because you will continue to work on it in the next exercise.

Exercise 3: Extend The React Webpart using the DetailsList React Component

In this exercise you will begin to build a user interface experience to display a set of customer leads within the **LeadsTracker** webpart. You will begin by building set of interfaces and a mock class with sample data. Along the way, you will extend the React component for the **LeadTracker** webpart with the **DetailsList** React component available through the Office UI Fabric React component library.

1. Create a new set of interfaces for retrieve customer lead data from a SharePoint contacts list.

   a) In the **src** folder, add a new child folder named **models**.

Make sure you create the **models** folder directly in the **src** folder so it is at the same level as the **webparts** folder. The reason for adding the **models** folder directly in the **src** folder is to create a place for project-wide interface definitions that can be used across all types of components in the project including webparts.

   b) Add three new source files inside the **models** folder named **ILead.ts**, **IList.ts** and **ILeadsService.ts**.
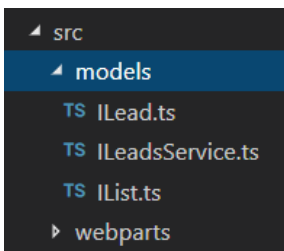
   c) Add the following interface definition to **ILead.ts**.

```
export default interface ILead {
  id: string;
  firstName: string;
  lastName: string;
  company: string;
  emailAddress: string;
}
```

   d) Save your changes and close **ILead.ts**.

e) Add the following interface definition to **IList.ts**.

```
export default interface IList {
  id: string;
  title: string;
}
```

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

g) Add the following interface definition to **ILeadsService.ts**.

```
import ILead from "./ILead";
import IList from "./IList";

export default interface ILeadsService {
  getLeads(targetList: string): Promise<ILead[]>;
  getLeadsLists(): Promise<IList[]>;
}
```

h) Save your changes and close **ILeadsService.ts**.

2. Add a new service class named **MockLeadsService** to provide sample customer lead 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 **MockLeadsService.ts**.
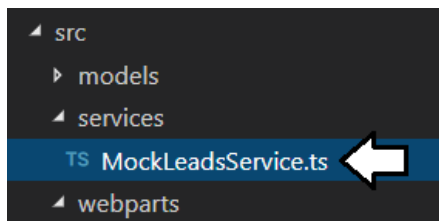
In the next step you will copy-and-paste the code for the **MockLeadsService** class. This class definition is large because it contains a large amount of hard-coded customer lead data in a JSON format. Rather than have you copy-and-paste the code for the **MockLeadsService** class from this document, you will open a separate text file in the **StarterFiles** folder named **MockLeadsService.ts.txt**. This will make it easier to copy and paste the code you need for the **MockLeadsService** class.

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

```
C:\Student\Modules\05_ReactWebparts\Lab\StarterFiles\MockLeadsService.ts.txt
```

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

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

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

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

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

3. Update the **ILeadTrackerState** interface by adding a new property named **leads**.

   a) Inside the **src/webparts/leadTracker/components** folder, open **ILeadTrackerState.ts**.

   b) Replace the existing **ILeadTrackerState** interface with the following interface definition which adds a new **leads** property.

```
import ILead from '../../../models/ILead'

export interface ILeadTrackerState {
  targetList: string;
  loading: boolean;
  leads: ILead[];
}
```

   c) Save your changes and close **ILeadTrackerState.ts**.

4. Update the React component to retrieve lead data from the **MockLeadsService** class.

   a) Inside the **src/webparts/leadTracker/components** folder, open **LeadTracker.tsx**.

   b) Underneath the existing **import** statements, add the following **import** statements for the new interfaces and mock data class.

```
import ILead from '../../../models/ILead'
import IList from '../../../models/IList'
import ILeadsService from '../../../models/ILeadsService'
import MockLeadsService from '../../../services/MockLeadsService'
```

   c) Below the **import** statements you just added, add another **import** statement for the React **DetailsList** components.

```
import {
  DetailsList,
  IColumn,
  DetailsListLayoutMode
} from 'office-ui-fabric-react';
```

Over the next few steps, you will update the **LeadTracker** component using an Office UI Fabric React component named **DetailsList** which will be used to display customer lead data. The other two imported types named **IColumn** and **DetailsListLayoutMode** are an interface type and an enumeration type that will be used to configure the **DetailsList** component.

   d) After the **import** statements and above the **LeadTracker** class, add a constant named **leadColumns** using the following code.

```
const leadColumns: IColumn[] = [
  { key: 'id', fieldName: 'id', name: 'ID', minWidth: 12, maxWidth: 24 },
  { key: 'firstName', fieldName: 'firstName', name: 'First Name', minWidth: 24, maxWidth: 64 },
  { key: 'lastName', fieldName: 'lastName', name: 'Last Name', minWidth: 24, maxWidth: 64 },
  { key: 'company', fieldName: 'company', name: 'Company', minWidth: 64, maxWidth: 120 },
  { key: 'emailAddress', fieldName: 'emailAddress', name: 'Email', minWidth: 100, maxWidth: 240 }
];
```

The **leadColumns** constant contains an array of **IColumn** objects that will be used to initialize the **DetailsList** component.

   e) Move down and place your cursor inside **LeadTracker** class before any other code.

   f) Add a private field named **leadsService** based on the **ILeadsService** interface.

   g) Initialize the **leadsService** field with a new instance of the **MockLeadsService** class.

```
export default class LeadTracker extends React.Component<ILeadTrackerProps, ILeadTrackerState> {

  private leadsService: ILeadsService = new MockLeadsService();
```

   h) Move down to the **state** initializer for the **LeadTracker** component.

   i) Add the **leads** property to the **state** initializer and set its value to an empty array.

```
public state: ILeadTrackerState = {
  targetList: this.props.targetListDefault,
  loading: false,
  leads: []
};
```

j) Replace the existing implementation of **render** with the following code which displays leads using the **DetailsList** component.

```
public render(): React.ReactElement<ILeadTrackerProps> {
  return (
    <div className={styles.leadTracker}>
      <DetailsList
        items={this.state.leads}
        columns={leadColumns}
        setKey='set'
        layoutMode={DetailsListLayoutMode.fixedColumns}
      />
    </div>
  );
}
```

k) Underneath the **render** method, add an implementation of the **componentDidMount** method using the following code.

```
componentDidMount() {
  this.leadsService.getLeads(this.state.targetList).then((leads: ILead[]) => {
    this.setState({ leads: leads });
  })
}
```

l) Save your changes to **LeadTracker.tsx**.

5. Update the CSS styles inside **LeadTracker.module.scss**.

a) Inside the **src/webparts/leadTracker/components** folder, open **LeadTracker.module.scss**.

b) Replace the contents of **LeadTracker.module.scss** with the following code.

```
@import '~office-ui-fabric-react/dist/sass/_References.scss';

.leadTracker {
    border: 1px solid #333;
}
```
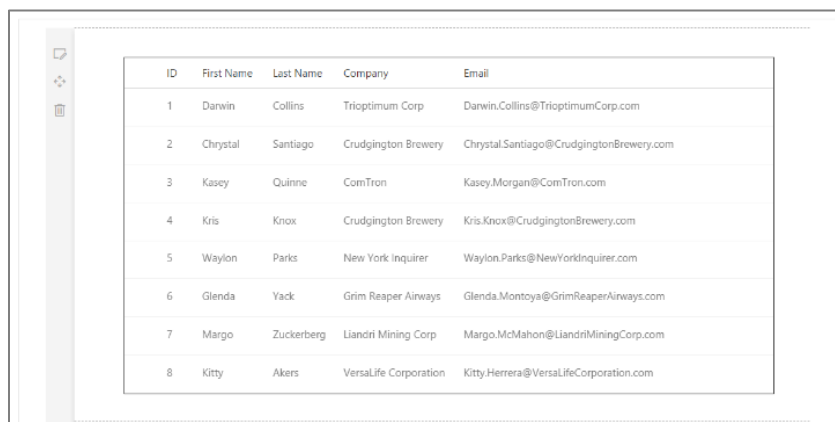
c) Save your changes and close **LeadTracker.module.scss**.

6. Test out the **LeadTracker** webpart by running it in the local SharePoint workbench

a) Make sure you have saved your changes to all files within the **react-webparts-lab** project.

b) Navigate to the Terminal console and execute **gulp serve** to start up the project and test it out using the local workbench.

```
gulp serve
```

c) The browser should launch and display the local SharePoint workbench.

d) Click on the button with the **+** sign in the middle of the page to add your webpart to the page.

e) Select the **Lead Tracker** webpart to add it to the page as a new webpart instance.

f) The **ListTracker** webpart should appear as the one shown in the following screenshot.

Exercise 4: Call the SharePoint REST API from a SharePoint Framework Webpart

In this exercise, you will write a service class which retrieves customer lead data from a SharePoint list which has been created using the standard out-of-the-box **Contacts** list template available whenever creating a SharePoint list. However, before creating the webpart, you must first create a pair of new **Contacts** lists in the SharePoint Online site where you will conduct your testing.

1. Create a new **Contacts** list in your SharePoint Online site and add a few items for testing purposes.

    a) In the browser, navigate to the root SharePoint site for your Office 365 tenant.

    b) Drop down the **Site Actions** menu and select the **Add an app** command.
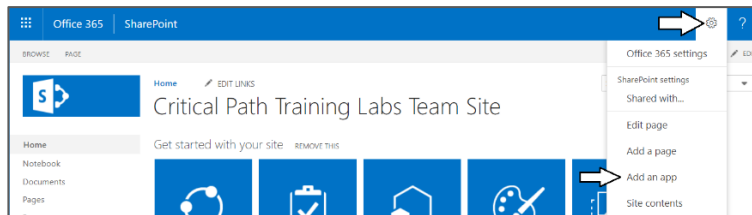
    c) Create a new list based on the **Contacts** list type.
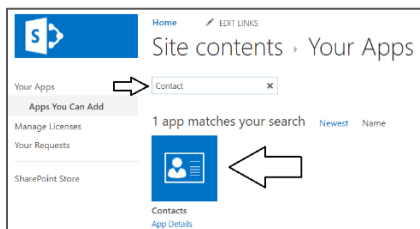
    d) When you are prompted to **Pick a name** for your new list, give it a name of **Northern Leads** and click **Create**.
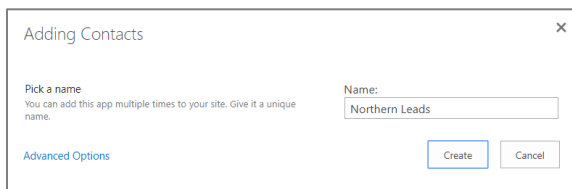
When you enter the **Name** for a new SharePoint list, it becomes the list **Title** property. Later in this exercise, you will reference this list by its Title when you program against it to retrieve items. Therefore, it is important you add the name exactly as "Northern Leads".
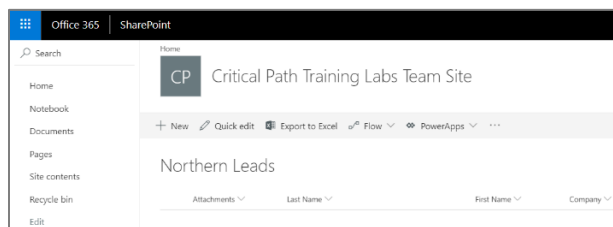
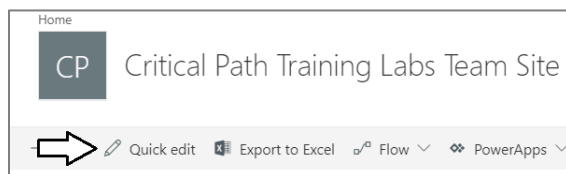   e) Once the **Northern Leads** list has been created, navigate to it from the link on the **Site Contents** page.
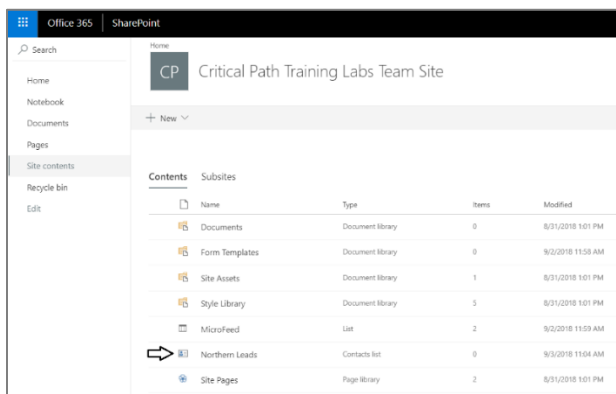
f) If you're prompted by the **Welcome to the new list experience** dialog, close by clicking the **X** in the upper, right corner.
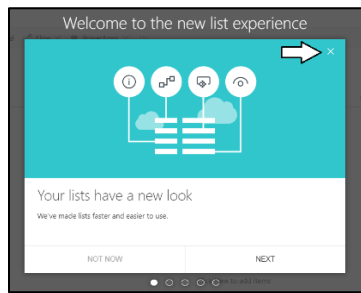
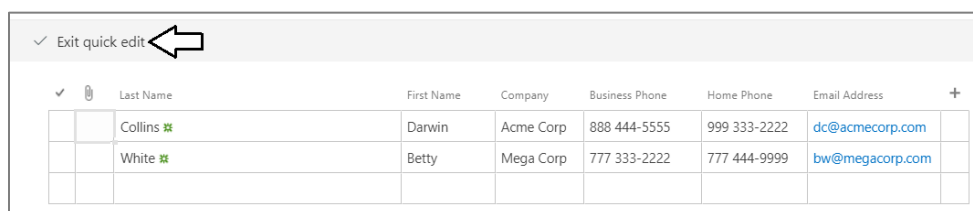g) You should now see the default view for the Northern Leads list.

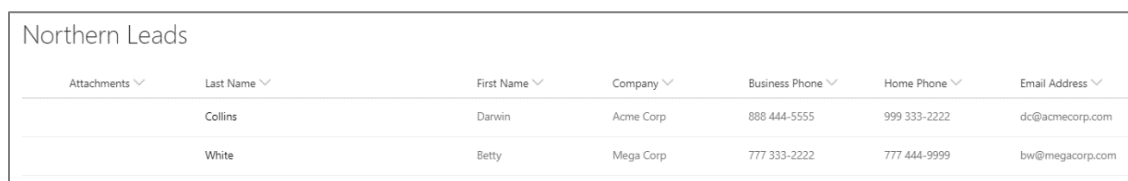h) Click the **Quick edit** button to enter quick edit mode.

i) You should now be able to enter data directory into the grid to create new items.

j) Create 2 or 3 samples items with sample data like the leads shown in the following screenshot and then click **Exit quick edit**.

k) You should now have a list with sample data to test your new webpart.

2. Create a second **Contacts** list named **Southern Leads**.

   a) Follow the exact steps as you did to create the **Northern Leads** list to create another **Contacts** list named **Southern Leads**.

   b) After creating the **Southern Leads** list, add a few sample items of data.

   c) You should now have at least two **Contacts** lists in your test site.

Now it's time to extend the **LeadTracker** webpart to retrieve items from a SharePoint **Contacts** lists. You'll begin by creating a service class named **SharePointLeadsService** which will retrieve list items by calling to the SharePoint REST API using the **SPHttpClient**.

3. Create new service class named **SharePointLeadsService**.

   a) In the **src/service** folder, create a new source file named **SharePointLeadsService.ts**.
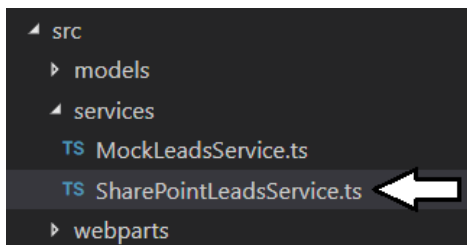
   b) Add the following starter code for the **SharePointLeadsService** class.

```
import ILead from "../models/ILead";
import IList from "../models/IList";
import ILeadsService from "../models/ILeadsService";

import {
  SPHttpClient,
  SPHttpClientResponse
} from '@microsoft/sp-http';

export default class SharePointLeadsService implements ILeadsService {

  constructor(private spHttpClient: SPHttpClient, private siteUrl: string) {
  }

  public getLeads(targetList: string): Promise<ILead[]> {
  }

  public getLeadsLists(): Promise<IList[]> {
  }

}
```

Note that this starter class contains a constructor that accepts an **SPHttpClient** parameter and a string parameter with the **siteUrl**. These values must be passed from the webpart to this service class in order to call into SharePoint via the SharePoint REST API.

c) Replace the implementation of the **getLeads** method with the following code.

```
public getLeads(targetList: string): Promise<ILead[]> {

    let restUrl = this.siteUrl +
        `/_api/web/lists/getByTitle('${targetList}')/items/` +
        "?$select=Id,FirstName,Title,Company,Email";

    return this.spHttpClient.get(restUrl, SPHttpClient.configurations.v1)
        .then(response => response.json())
        .then(response => {
            return response.value.map(lead => <ILead>({
                id: lead.Id,
                firstName: lead.FirstName,
                lastName: lead.Title,
                company: lead.Company,
                emailAddress: lead.Email
            }));
        });
}
```

Note that the **targetList** parameter is being used find the correct list by its title when calling the SharePoint REST API. You should also observe how this method uses the **map** function to rename item properties to match the casing of the properties in the **ILead** interface.

d) Replace the implementation of the **getLeadsLists** method with the following code.

```
public getLeadsLists(): Promise<IList[]> {

    let restUrl = this.siteUrl + "/_api/web/lists/" +
        "?$select=Id,Title&$filter=BaseTemplate+eq+105";

    return this.spHttpClient.get(restUrl, SPHttpClient.configurations.v1)
        .then(response => response.json())
        .then(response => {
            return response.value.map(list => <IList>({
                id: list.Id,
                title: list.Title
            }));
        });
}
```

Note that this method limits the lists returned by filtering the **BaseTemplate** property to be **105** which indicates it is a **Contacts** list.

e) Save your changes and close **SharePointLeadsService.ts**.

4. Update the React component properties for the **LeadetTracker** component.

a) In the **src/webparts/leadTracker/components** folder, open the source file named **ILeadTrackerProps.ts**.
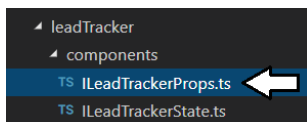
b) Replace the contents of **ILeadTrackerProps.ts** with the following code.

```
import { SPHttpClient } from '@microsoft/sp-http';

export interface ILeadTrackerProps {
    targetListDefault: string;
    siteUrl: string;
    spHttpClient: SPHttpClient | undefined;
}
```

c) Save your changes and close **ILeadTrackerProps.ts**.

Now that you have added two new properties to the React component named **LeadTracker**, you must update the webpart class named **LeadTrackerWebPart** to pass those properties when it creates the React component in the **render** method.

5. Update the **LeadetTrackerWebPart** webpart class.

   a) in the **src/webparts/leadTracker** folder, open the source file named **LeadTrackerWebPart.ts**.

   b) Under the existing **import** statements, add a new **import** statement for **SPHttpClient**.

   ```
   import { SPHttpClient } from '@microsoft/sp-http';
   ```

   c) Move down inside the **LeadetTrackerWebPart** class and locate the **render** method.

   d) The current implementation of **render** creates the **LeadTracker** component with a single property named **targetListDefault**.

   ```
   public render(): void {
     const element: React.ReactElement<ILeadTrackerProps> = React.createElement(
       LeadTracker, { targetListDefault: this.properties.targetList }
     );
     this.leadTracker = <LeadTracker>ReactDom.render(element, this.domElement);
   }
   ```

   e) Update the **render** method to pass the **siteUrl** and **spHttpClient** properties when creating the **LeadTracker** component.

   ```
   public render(): void {
     const element: React.ReactElement<ILeadTrackerProps> = React.createElement(
       LeadTracker, {
         targetListDefault: this.properties.targetList,
         siteUrl: this.context.pageContext.web.absoluteUrl,
         spHttpClient: <SPHttpClient>this.context.spHttpClient
       }
     );
     this.leadTracker = <LeadTracker>ReactDom.render(element, this.domElement);
   }
   ```

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

6. Update the React component named **LeadTracker** to use **SharePointLeadsService** instead of **MockLeadsService**.

   a) In the **src/webparts/leadTracker/components** folder, open the source file named **LeadTracker.tsx**.

   b) Underneath the import statement for **MockLeadsService**, add a new import statement for **SharePointLeadsService**.

   ```
   import ILead from '../../../models/ILead';
   import IList from '../../../models/IList';
   import ILeadsService from '../../../models/ILeadsService';
   import MockLeadsService from '../../../services/MockLeadsService';
   import SharePointLeadsService from '../../../services/SharePointLeadsService';
   ```

   c) Move down inside the **LeadTracker** class and locate the declaration of the private field named **leadsService**.

   ```
   private leadsService: ILeadsService = new MockLeadsService();
   ```

   d) Modify the code to initialize the **leadsService** field with an instance of the **SharePointLeadsService** class.

   ```
   private leadsService: ILeadsService =
           new SharePointLeadsService(this.props.spHttpClient, this.props.siteUrl);
   ```

   e) Save your changes to **LeadTracker.tsx**.

7. Check the default value of the webpart's **targetList** property to make sure it matches the title of an existing list.

   a) Inside the **src/webparts/leadTracker folder**, open the file named file named **LeadTrackerWebPart.manifest.json**.

   b) Make sure the default value for **targetList** matches the title of a list you created earlier in this exercise.

   c) If you have updated **LeadTrackerWebPart.manifest.json**, make sure to save your changes.

It's once again time to test out your webpart. However, you can no longer test your webpart in the local SharePoint Workbench because that does not provide the SharePoint context required to call the SharePoint REST API. Therefore, you will now have to conduct your testing in the hosted SharePoint Workbench in your SharePoint Online test site.

8. Test out the **LeadTracker** webpart by running it in the hosted SharePoint Workbench in SharePoint Online.

    a) Make sure you have saved your changes to all files within the **react-webparts-lab** project.

    b) Navigate to the Terminal console and execute **gulp serve** with the **--nobrowser** parameter to start up the project.

```
gulp serve --nobrowser
```

    c) In the Chrome browser, navigate to your test site and log in.

    d) Once you have successfully logged in, navigate to the hosted SharePoint Workbench at the following URL.

```
https://[YOUR_TENANT_NAME].sharepoint.com/_layouts/15/workbench.aspx
```

    e) The hosted SharePoint Workbench should appear and allow you to add a webpart.

    f) Click on the button with the **+** sign in the middle of the page to add your webpart to the page.

    g) Select the **Lead Tracker** webpart to add it to the page as a new webpart instance.
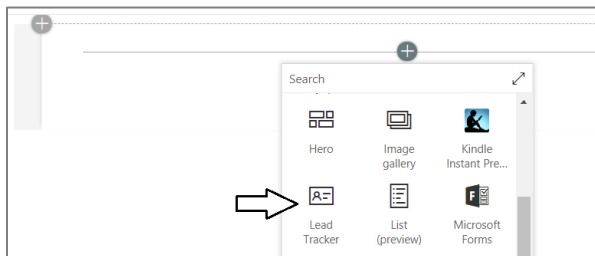
    h) The **ListTracker** webpart should display leads from the SharePoint list as the one shown in the following screenshot.


    i) Close the browser window with the webpart, return to Visual Studio Code and stop the debugging session.

Leave the **react-webparts-lab** project open in the Visual Studio Code because you will continue to work on it in the next exercise.

Exercise 5: Extend The Webpart to Dynamically Switch Between Contacts Lists

In the final exercise, you will extend the webpart so that a user can select a **Contacts** list in the webpart property pane from a set of options that include all the list in the current site that have been created from the **Contacts** list type.

1. Update the webpart class in **LeadTrackerWebPart.ts**.

    a) Inside the **src/webparts/leadTracker** folder, open the source file for the webpart class named **LeadTrackerWebPart.ts**.
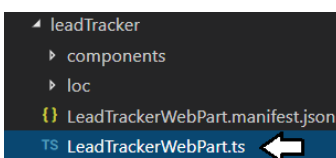
b) Locate the existing **import** statement for the types from the **@microsoft/sp-webpart-base** package.

```
import {
  BaseClientSideWebPart,
  IPropertyPaneConfiguration,
  PropertyPaneTextField
} from '@microsoft/sp-webpart-base';
```

c) Update this **import** statement to import 2 more types named **PropertyPaneDropdown** and **IPropertyPaneDropdownOption**.

```
import {
  BaseClientSideWebPart,
  IPropertyPaneConfiguration,
  PropertyPaneTextField,
  PropertyPaneDropdown,
  IPropertyPaneDropdownOption
} from '@microsoft/sp-webpart-base';
```

d) Underneath the other **import** statements, add three new **import** statements for **ILead**, **IList** and **SharePointLeadsService**.

```
import IList from '../../models/IList';
import ILeadsService from '../../models/ILeadsService';
import SharePointLeadsService from '../../services/SharePointLeadsService';
```

e) Move down in the **LeadTrackerWebPart** class definition and locate the declaration of the private field named **leadTracker**.

f) After the declaration of the private field named **leadTracker**, add two new private fields named **listOptions** and **listsFetched**.

```
private leadTracker: LeadTracker;

private listOptions: IPropertyPaneDropdownOption[];
private listsFetched: boolean = false;
```

g) Move down in the class declaration and add a method named **fetchListOptions** using the following code.

```
private fetchListOptions(): Promise<IPropertyPaneDropdownOption[]> {

  let leadsService: ILeadsService =
    new SharePointLeadsService(
      this.context.spHttpClient,
      this.context.pageContext.web.absoluteUrl
    );

  return leadsService.getLeadsLists().then((lists: IList[]) => {
    var options: Array<IPropertyPaneDropdownOption> = new Array<IPropertyPaneDropdownOption>();
    lists.map((list: IList) => {
      options.push({ key: list.title, text: list.title });
    });
    return options;
  });
}
```

h) Move down below the **onDispose** method and add the **onPropertyPaneConfigurationStart** method using the following code.

```
protected onPropertyPaneConfigurationStart(): void {
    if (this.listsFetched) {
      return;
    }
    this.fetchListOptions().then((options: IPropertyPaneDropdownOption[]) => {
      this.listOptions = options;
      this.listsFetched = true;
      this.context.propertyPane.refresh();
      this.render();
    });
  }
```

The **onPropertyPaneConfigurationStart** method is called automatically whenever a user navigate into edit mode for your webpart. That means that **onPropertyPaneConfigurationStart** executes whenever the property pane is displayed which makes it possible to perform actions to initialize the user interface in the properties pane such as filling a dropdown combo box with a set of options

    i) Move down inside the **LeadTrackerWebPart** class and locate the **getPropertyPaneConfiguration** method.

    j) Replace the existing implementation of **getPropertyPaneConfiguration** with the following code.

```
protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {
  return {
    pages: [
      {
        header: { description: "List Tracker Properties" },
        groups: [{
          groupName: "Data source",
          groupFields: [
            PropertyPaneDropdown(
              "targetList", {
                label: "Select a Contacts list",
                options: this.listOptions,
                disabled: !this.listsFetched
              }),
          ]
        }]
      }
    ]
  };
}
```

    k) Save your changes and close **LeadTrackerWebPart.ts**.

With your latest changes, the user is able to dynamically select a **Contacts** list for the webpart from the webpart properties pane. Therefore, it is no longer needed to hard code a list name into the default value of the **targetList** properties. In the next step you will update the default value for the **targetList** property to an empty string.

2. Change the default value of the webpart's **targetList** property to be an empty string.

    a) Inside the **src/webparts/leadTracker** folder, open the file named **LeadTrackerWebPart.manifest.json**.

    b) Set the default value for the **targetList** property to be an empty string.

    c) Save your changes and close **LeadTrackerWebPart.manifest.json**.

Now you just need to update the HTML and CSS for the React component.

3. Add an image file to the project to use to indicate when the webpart is loading data from across the network.

    a) Using Windows Explorer, locate the image file named **loading.gif** at the following location.

```
C:\Student\Modules\05_ReactWebparts\Lab\StarterFiles\loading.gif
```

    b) Using Windows Explorer, copy the **loading.gif** file to the Windows clipboard.

    c) Stay in Windows Explorer and navigate to the **src** folder at the following path.

```
C:\Student\Modules\05_ReactWebparts\Lab\react-webparts-lab\src\
```

    d) Using Windows Explorer, create a new child folder inside the **src** folder named **assets**.

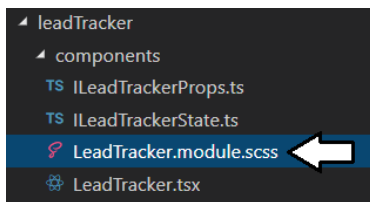   e) Inside the **assets** folder, paste the image file named **loading.gif.**
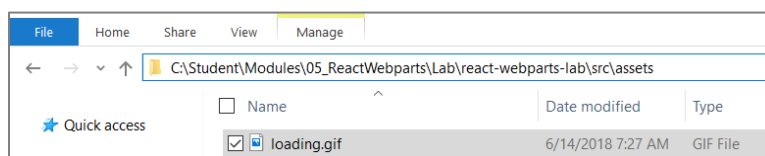
f) Return to Visual Studio Code and verify that you can see **loading.gif** in the **assets** folder.
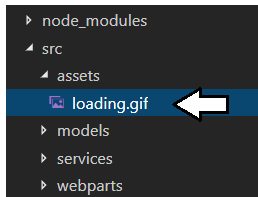
4. Update the styles in **LeadTracker.module.scss**.

a) Open the source file named **LeadTracker.module.scss** in an editor window.

b) Replace the contents of **LeadTracker.module.scss** with the following code.

```scss
@import '~office-ui-fabric-react/dist/sass/_References.scss';

.leadTracker {
    border: 1px solid #333;

    .loadingContainer {
        background-image: url('../../../assets/loading.gif');
        background-position: center;
        background-position-y: 100px;
        background-repeat: no-repeat;
        font-size: 24px;
        color: red;
    }

    .messageContainer {
        color: blue;
        font-size: 18px;

    }

    .loadingContainer, .messageContainer  {
        height: 240px;
        padding: 24px;
        border: 1px solid black;
        text-align: center;
    }
}
```

c) Save your changes and close **LeadTracker.module.scss**.

In the next step you will modify the user experience created by the **LeadTracker** component in two ways. First, the webpart will show an informational message indicating the user must select a list when the **targetList** property value is empty. Second, the **LeadTracker** component will display a loading indicator whenever the webpart is calling out across the network to the SharePoint REST API.

5. Modify the user interface experienced created by the **LeadTracker** component.

a) Open **LeadTracker.tsx** in an editor window.
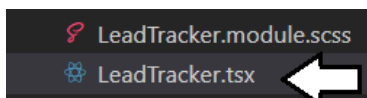
b) Replace the existing implementation of the **render** method using the following code.

```
public render(): React.ReactElement<ILeadTrackerProps> {
  return (
    <div className={styles.leadTracker}>

      {(this.state.targetList === "") ?
        <div className={styles.messageContainer} >Select a list from the web part property pane</div> :
        (this.state.loading) ?
          <div className={styles.loadingContainer} >Calling to the SharePoint REST API</div> :
          <DetailsList
            items={this.state.leads}
            columns={leadColumns}
            setKey='set'
            layoutMode={DetailsListLayoutMode.fixedColumns}
          />
      }

    </div>
  );
}
```

There is one last thing you need to do. Whenever the user selects a new list in the properties pane, the webpart class calls **setState** on the React component to set the **targetList** property to reference the new list. However, there is nothing yet in your implementation that will trigger the React component to call across the network to SharePoint whenever the list is changed from one **Contacts** list to another. Now you will implement this triggering behavior by adding the **componentDidUpdate** method to the React component class.

c) Move down under the **componentDidMount** method and add the **componentDidUpdate** method using the following code.

```
public componentDidUpdate(prevProps: ILeadTrackerProps, prevState: ILeadTrackerState, prevContext:
any): void {
  if (prevState.targetList != this.state.targetList) {
    this.setState({ loading: true });
    this.leadsService.getLeads(this.state.targetList).then((leads: ILead[]) => {
      this.setState({ leads: leads, loading: false });
    });
  }
}
```

d) Save your changes to **LeadTracker.tsx**.

6. Test out the **LeadTracker** webpart by running it in the hosted SharePoint Workbench in SharePoint Online.

a) Make sure you have saved your changes to all files within the **react-webparts-lab** project.

b) Navigate to the Terminal console and execute **gulp serve** with the **--nobrowser** parameter to start up the project.

```
gulp serve --nobrowser
```

c) In the Chrome browser, navigate to your test site and log in.

d) Once you have successfully logged in, navigate to the hosted SharePoint Workbench at the following URL.

```
https://[YOUR_TENANT_NAME].sharepoint.com/_layouts/15/workbench.aspx
```
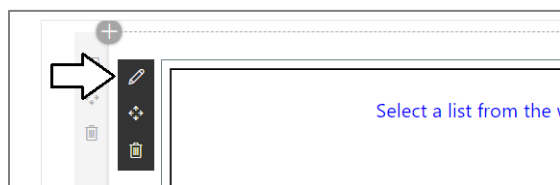
e) Click on the button with the **+** sign in the middle of the page to add your webpart to the page.

f) Select the **Lead Tracker** webpart to add it to the page as a new webpart instance.
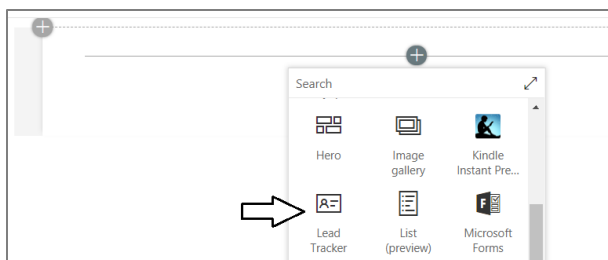
g) The **ListTracker** webpart should now display a message instructing the user to select a list from the properties pane.
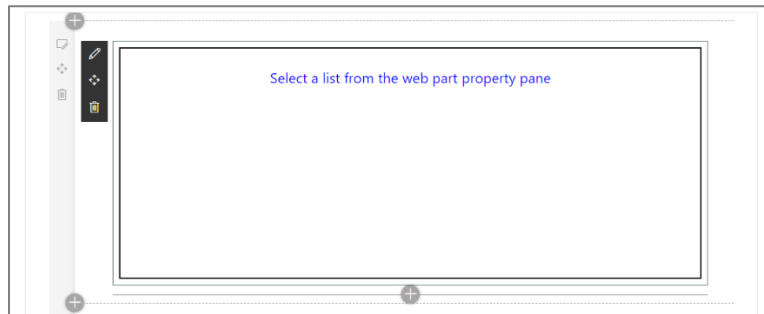

h) Click the button with pen icon to display the webpart properties pane.

i) The **targetList** property is now presented to the user as a dropdown combo box instead of a textbox. Currently, there is no list selected because the **targetList** property has the value of an empty string.
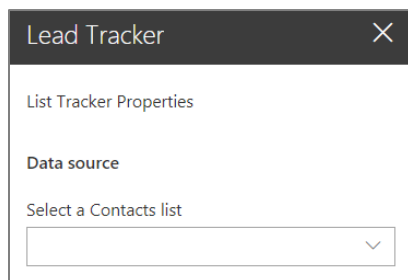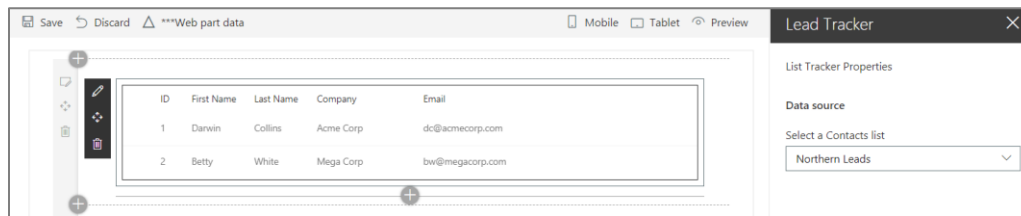

j) Drop down the combo box menu and you should see all the **Contacts** lists in the current site.
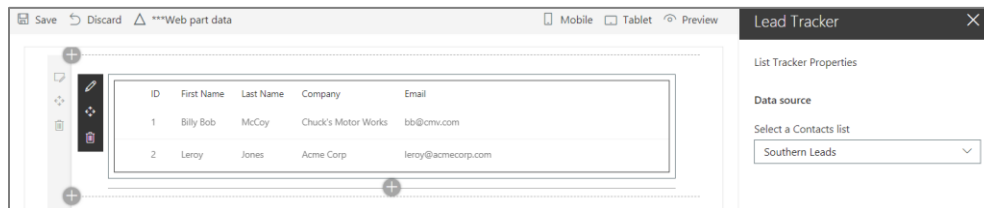
k) Select a list such as **Northern Leads**.
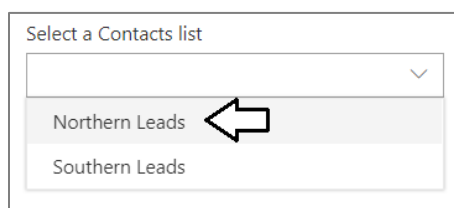
l) The webpart should display a loading message while it calls into the SharePoint REST API.

m) The webpart should then display the items from that **Contacts** list.

n) Change to a different **Contacts** list in the properties pane dropdown menu and the webpart should automatically update.

o) Close the browser window with the webpart, return to Visual Studio Code and stop the debugging session.

Congratulations. You have now reached the end of this lab.