Exercise - Retrieve and display list   
data with the SharePoint REST API

In this exercise, you'll create a SharePoint Framework (SPFx) web part that will get and display data from a SharePoint list using the SharePoint REST API.

## Step 1 - Create a SharePoint list to use for testing

*NOTE: In the main folder of the workshop content is an Excel file called Countries. Instead of creating the list below manually, you can upload Countries.xlsx into the document folder of your Dev Site and create a list from the file. Then skip to Step 2.*

Before you can get display data from a SharePoint list using the SharePoint REST API, you'll first need to create a list and populate it with some data.

In a browser, navigate to a site collection in SharePoint Online. Select **Site Contents** in the left-hand navigation and then select **New > List** in the toolbar.

Set the list name to **Countries** and select **Create**.

Add items to the list by entering the names of different countries as shown in the following image.

## Step 2 - Create the web part to display data using the SharePoint REST API

Open a command prompt and change to the folder where you want to create the project.

Important: The instructions below assume you are using v1.11.0 of the SharePoint Framework Yeoman generator.

Create new folder for your project and run the SharePoint Yeoman generator by executing the following command:

yo @microsoft/sharepoint

Use the following to complete the prompt that is displayed (if additional options are presented, accept the default answer):

* **What is your solution name?**: SpFxHttpClientDemo
* **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
* **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?**: No
* **Will the components in the solution require permissions to access web APIs that are unique and not shared with other components in the tenant?**: No
* **Which type of client-side component to create?**: Web Part
* **What is your Web Part name?**: SpFxHttpClientDemo
* **What is your Web Part description?**: SpFxHttpClientDemo description
* **Which framework would you like to use?** React

After provisioning the folders required for the project, the generator will install all the dependency packages by running npm install automatically. When NPM completes downloading all dependencies, test the default project provisioned by the generator.

In order to use the SharePoint REST APIs, we need to install a dependency for sp-http. At the command line, enter the following:

npm install @microsoft/sp-http --save

Open the project in **Visual Studio Code**.

### Create an interface for the SharePoint list items

Locate the **./src** folder in the project. Create a new subfolder **models** in the **src** folder.

Create a new file **ICountryListItem.ts** in the **models** folder and add the following code to it:

export interface ICountryListItem {

Id: string;

Title: string;

}

### Create a type for a button click event handler

Create a new file **ButtonClickedCallback.ts** in the **models** folder and add the following code to it:

export type ButtonClickedCallback = () => void;

### Create a barrel for the types implemented in the models folder

Create a new file **index.ts** in the **models** folder and add the following code to it:

export \* from './ButtonClickedCallback';

export \* from './ICountryListItem';

### Update the public interface for the React component

Locate and open the file **./src/webparts/restDemo/components/ISpFxHttpClientDemoProps.ts**. This is the interface for the public properties of the React component. It will need to display a list of items, so update the signature to accept a list of items.

Add the following code to the top of the file:

import { ButtonClickedCallback, ICountryListItem } from '../../../models';

Update the interface to replace the existing description property to be a collection of items to be passed in and add an event when a button is selected:

export interface ISpFxHttpClientDemoProps {

spListItems: ICountryListItem[];

onGetListItems?: ButtonClickedCallback;

}

### Implement the user interface for the web part

Locate and open the file **./src/webparts/spFxHttpClientDemo/components/SpFxHttpClientDemo.module.scss**.

Add the following classes to the bottom of the file, immediately before the closing }:

.button {

padding: 0 5px;

margin: 0 5px;

}

.list {

color: $ms-color-themeDark;

background-color: $ms-color-themeLight;

font-family: 'Segoe UI Regular WestEuropean', 'Segoe UI', Tahoma, Arial, sans-serif;

font-size: 14px;

font-weight: normal;

box-sizing: border-box;

margin: 0 70px;

padding: 10px 0 100px 0;

line-height: 50px;

list-style-type: none;

}

.item {

color: $ms-color-themeDark;

background-color: $ms-color-themeLighterAlt;

vertical-align: center;

font-family: 'Segoe UI Regular WestEuropean', 'Segoe UI', Tahoma, Arial, sans-serif;

font-size: 14px;

font-weight: normal;

box-sizing: border-box;

margin: 0;

padding: 0;

box-shadow: none;

\*zoom: 1;

padding: 0 15px;

position: relative;

box-shadow: 0 2px 4px 0 rgba(0, 0, 0, 0.2), 0 25px 50px 0 rgba(0, 0, 0, 0.1);

}

Locate and open the file **./src/webparts/SpFxHttpClientDemo/components/** **SpFxHttpClientDemo.tsx**.

Update the markup returned by the render() method to the following code. This will create a list displaying the data contained in the spListItems property rendered using the CSS classes added in the previous step. Also notice that there's an anchor tag <a> that acts as a button and has an onClick handler wired up to it.

<div className={ styles.spFxHttpClientDemo }>

<div className={ styles.container }>

<div className={ styles.row }>

<div className={ styles.column }>

<p className={ styles.title }>SharePoint Content!</p>

<a href="#" className={ styles.button } onClick={ this.onGetListItemsClicked }>

<span className={ styles.label }>Get Counties</span>

</a>

</div>

</div>

<div className={ styles.row }>

<ul className={ styles.list }>

{ this.props.spListItems &&

this.props.spListItems.map((list) =>

<li key={list.Id} className={styles.item}>

<strong>Id:</strong> {list.Id}, <strong>Title:</strong> {list.Title}

</li>

)

}

</ul>

</div>

</div>

</div>

Add the following event handler to the SpFxHttpClientDemo class to handle the click event on the button. This code will prevent the default action of the <a> element from navigating away from (or refreshing) the page and call the callback set on the public property, notifying the consumer of the component an event occurred.

private onGetListItemsClicked = (event: React.MouseEvent<HTMLAnchorElement>): void => {

event.preventDefault();

this.props.onGetListItems();

}

### Retrieve data from the SharePoint REST API

Locate and open the **./src/webparts/SpFxHttpClientDemo/SpFxHttpClientDemo WebPart.ts** file.

After the existing import statements at the top of the file, add the following import statements:

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

import { ICountryListItem } from '../../models';

Locate the class SpFxHttpClientDemoWebPart and add the following private member to it:

private \_countries: ICountryListItem[] = [];

Locate the render() method. Notice that this method is creating an instance of the component SpFxHttpClientDemo and then setting its public properties. The default code sets the description property, however this was removed from the interface ISpFxHttpClientDemoProps in the previous steps. Update the properties to set the list of countries to the private member added above and attach to the event handler:

spListItems: this.\_countries,

onGetListItems: this.\_onGetListItems

Add the following method as an event handler to the SpFxHttpClientDemoWebPart class. This method calls another method (that you'll add in the next step) that returns a collection of list items. Once those items are returned via a JavaScript Promise, the method updates the internal \_countries member and re-renders the web part. This will bind the collection of countries returned by the \_getListItems() method to the public property on the React component that will render the items.

private \_onGetListItems = (): void => {

this.\_getListItems()

.then(response => {

this.\_countries = response;

this.render();

});

}

Add the following method to the SpFxHttpClientDemoWebPart class. The method retrieves list items from the **Countries** list using the SharePoint REST API. It will use the spHttpClient object to query the SharePoint REST API. When it receives the response, it calls response.json() that will process the response as a JSON object and then returns the value property in the response to the caller. The value property is a collection of list items that match the interface created previously.

private \_getListItems(): Promise<ICountryListItem[]> {

return this.context.spHttpClient.get(

this.context.pageContext.web.absoluteUrl + `/\_api/web/lists/getbytitle('Countries')/items?$select=Id,Title`,

SPHttpClient.configurations.v1)

.then(response => {

return response.json();

})

.then(jsonResponse => {

return jsonResponse.value;

}) as Promise<ICountryListItem[]>;

}

## Step 3 - Test the web part

Run the following command to start the local web server:

gulp serve

The browser will load the local workbench, but you cannot use this for testing because there's no SharePoint context in the local workbench. Instead, navigate to the SharePoint Online site where you created the **Countries** list, and load the hosted workbench at **https://[sharepoint-online-site]/\_layouts/workbench.aspx**.

Add the web part to the page: Select the **Add a new web part** control...

...then select the expand toolbox icon in the top-right...

...and select the **SPFxHttpClientDemo** web part to add the web part to the page.

The web part will appear on the page with a single button and no data in the list.

Select the **Get Countries** button and notice the list will display the data from the SharePoint REST API.

Stop the local web server by pressing CTRL+C in the console/terminal window.

# Step 4 – Add write operations using the SharePoint Framework APIs and SharePoint REST API

Locate and open the **./src/webparts/spFxHttpClientDemo/components/ISpFxHttpClientDemoProps.ts** file.

Add the following properties to the ISpFxHttpClientDemoProps interface.

onAddListItem?: ButtonClickedCallback;

onUpdateListItem?: ButtonClickedCallback;

onDeleteListItem?: ButtonClickedCallback;

Locate and open the **./src/webparts/spFxHttpClientDemo/components/SpFxHttpClientDemo.tsx** file.

Within the render() method in the SpFxHttpClientDemo class, locate the button **Get Countries**. Add the following markup to add three more buttons to the user interface:

<a href="#" className={ styles.button } onClick={ this.onAddListItemClicked }>

<span className={ styles.label }>Add List Item</span>

</a>

<a href="#" className={ styles.button } onClick={ this.onUpdateListItemClicked }>

<span className={ styles.label }>Update List Item</span>

</a>

<a href="#" className={ styles.button } onClick={ this.onDeleteListItemClicked }>

<span className={ styles.label }>Delete List Item</span>

</a>

Add the following event handlers to the SpFxHttpClientDemo class:

private onAddListItemClicked = (event: React.MouseEvent<HTMLAnchorElement>): void => {

event.preventDefault();

this.props.onAddListItem();

}

private onUpdateListItemClicked = (event: React.MouseEvent<HTMLAnchorElement>): void => {

event.preventDefault();

this.props.onUpdateListItem();

}

private onDeleteListItemClicked = (event: React.MouseEvent<HTMLAnchorElement>): void => {

event.preventDefault();

this.props.onDeleteListItem();

}

Locate and open the **./src/webparts/spFxHttpClientDemo/SpFxHttpClientDemoWebpart.ts** file.

Within the render() method in the SpFxHttpClientDemoWebPart class, locate the code where the public properties are set on the React component SpFxHttpClientDemo. It will look like this:

{

spListItems: this.\_countries,

onGetListItems: this.\_onGetListItems

}

Update the public properties to add handlers for the events when buttons are pressed in the component:

{

spListItems: this.\_countries,

onGetListItems: this.\_onGetListItems,

onAddListItem: this.\_onAddListItem,

onUpdateListItem: this.\_onUpdateListItem,

onDeleteListItem: this.\_onDeleteListItem

}

Implement the three event handlers you just added.

private \_onAddListItem = (): void => {

this.\_addListItem()

.then(() => {

this.\_getListItems()

.then(response => {

this.\_countries = response;

this.render();

});

});

}

private \_onUpdateListItem = (): void => {

this.\_updateListItem()

.then(() => {

this.\_getListItems()

.then(response => {

this.\_countries = response;

this.render();

});

});

}

private \_onDeleteListItem = (): void => {

this.\_deleteListItem()

.then(() => {

this.\_getListItems()

.then(response => {

this.\_countries = response;

this.render();

});

});

}

These event handlers will call different methods, which you'll add in the remainder of this exercise. Each one will add, update, or delete an item in the SharePoint list, call the existing \_getListItems() method you created in the previous exercise, and refresh the web part by calling render().

Add the following methods to the SpFxHttpClientDemoWebPart class to add a new item to the list:

private \_getItemEntityType(): Promise<string> {

return this.context.spHttpClient.get(

this.context.pageContext.web.absoluteUrl + `/\_api/web/lists/getbytitle('Countries')?$select=ListItemEntityTypeFullName`,

SPHttpClient.configurations.v1)

.then(response => {

return response.json();

})

.then(jsonResponse => {

return jsonResponse.ListItemEntityTypeFullName;

}) as Promise<string>;

}

private \_addListItem(): Promise<SPHttpClientResponse> {

return this.\_getItemEntityType()

.then(spEntityType => {

const request: any = {};

request.body = JSON.stringify({

Title: new Date().toUTCString(),

'@odata.type': spEntityType

});

return this.context.spHttpClient.post(

this.context.pageContext.web.absoluteUrl + `/\_api/web/lists/getbytitle('Countries')/items`,

SPHttpClient.configurations.v1,

request);

}

) ;

}

The method \_getItemEntityType() will get the type of data that the **Countries** list expects. This is done by:

* Using the spHttpClient API's get() method to issue an HTTP GET request to the SharePoint REST API. This method requires two parameters: (1) the endpoint to query and (2) the configuration to use.
* After processing the response body as JSON...
* It returns the ListItemEntityTypeFullName as a single string value to the caller.

The method \_addListItem() first obtains the data type supported by the list needed when creating a new item. Then it creates a new item by:

* Creating a new object with a new Title, set to the current UTC timestamp, and the @odata.type property that is set to the value obtained in the \_getItemEntityType() method.
* This new object is set to the body property as a string on a request object that will be sent in the HTTP POST.
* Then, using the spHttpClient API's post() method, set the endpoint to the list's items collection, with the specified configuration and then set the request object as the third parameter for the post() method. This will tell the spHttpClient API to send the new object as part of the body in the HTTP request.

Add the following method to the SpFxHttpClientDemoWebPart class to update an item in the list:

private \_updateListItem(): Promise<SPHttpClientResponse> {

// get the first item

return this.context.spHttpClient.get(

this.context.pageContext.web.absoluteUrl + `/\_api/web/lists/getbytitle('Countries')/items?$select=Id,Title&$filter=Title eq 'United States'`,

SPHttpClient.configurations.v1)

.then(response => {

return response.json();

})

.then(jsonResponse => {

return jsonResponse.value[0];

})

.then((listItem: ICountryListItem) => {

// update item

listItem.Title = 'USA';

// save it

const request: any = {};

request.headers = {

'X-HTTP-Method': 'MERGE',

'IF-MATCH': (listItem as any)['@odata.etag']

};

request.body = JSON.stringify(listItem);

return this.context.spHttpClient.post(

this.context.pageContext.web.absoluteUrl + `/\_api/web/lists/getbytitle('Countries')/items(${listItem.Id})`,

SPHttpClient.configurations.v1,

request);

});

}

This method will update an existing item in the list by doing the following steps:

* It first gets a reference to a single item in the list with the Title = **United States**. This is done using the OData $filter operator on the URL parameters of the HTTP GET request endpoint.
* Upon receiving the response, after processing the response as JSON, it gets the first item in the value collection, which is an array. This will contain the single item in our query results.
* Once the item is retrieved, the Title property is changed to **USA**.
* A new request object is created to submit to the SharePoint REST API:
  + The headers are set to instruct the REST API you wish to do a **MERGE** operation and...
  + The item that will be updated on the server should have the same version, as indicated by the @odata.etag property, as the item that is submitted in the HTTP request.
* Similar to the add operation, using the spHttpClient API's post() method, the specific item in the SharePoint list is updated by using the endpoint of the specific item, the wanted configuration and the request object this method constructed.

Add the following method to the SpFxHttpClientDemoWebPart class to delete the last item in the list:

private \_deleteListItem(): Promise<SPHttpClientResponse> {

// get the last item

return this.context.spHttpClient.get(

this.context.pageContext.web.absoluteUrl + `/\_api/web/lists/getbytitle('Countries')/items?$select=Id,Title&$orderby=ID desc&$top=1`,

SPHttpClient.configurations.v1)

.then(response => {

return response.json();

})

.then(jsonResponse => {

return jsonResponse.value[0];

})

.then((listItem: ICountryListItem) => {

const request: any = {};

request.headers = {

'X-HTTP-Method': 'DELETE',

'IF-MATCH': '\*'

};

request.body = JSON.stringify(listItem);

return this.context.spHttpClient.post(

this.context.pageContext.web.absoluteUrl + `/\_api/web/lists/getbytitle('Countries')/items(${listItem.Id})`,

SPHttpClient.configurations.v1,

request);

});

}

This method will delete the last item in the list by doing the following steps:

* It first gets a reference to the last item in the list by sorting the list in descending order by the ID property and taking just the first result. This is done using the OData $orderby and $top operators on the URL parameters of the HTTP GET request endpoint.
* A new request object is created to submit to the SharePoint REST API:
  + The headers are set to instruct the REST API you wish to do a **DELETE** operation and...
  + The item that will be deleted can match any version.
* Using the spHttpClient API's post() method, the specific item in the SharePoint list is deleted by using the endpoint of the specific item, the wanted configuration and the request object this method constructed.

## Test the web part

Start the local web server and test the web part in the hosted workbench:

gulp serve

The browser will loads the local workbench, but you cannot use this for testing because there is no SharePoint context in the local workbench. Instead, navigate to the SharePoint Online site where you created the **Countries** list, and load the hosted workbench at **https://[sharepoint-online-site]/\_layouts/workbench.aspx**.

Add the web part to the page: Select the **Add a new web part** control...

...then select the expand toolbox icon in the top-right...

...and select the **SPFxHttpClientContent** web part to add the web part to the page.

The web part will appear on the page with a single button and no data in the list.

Select the **Get Countries** button and examine the results returned. Scroll to the bottom of the list and notice there's no entry with a timestamp for the **Title**.

Select the **Add List item** button and scroll to the end of the results returned. Notice the new item that appears with a timestamp as the **Title**.

Ensure there's an item in the list with the title equal to **United States**. Test the update process by selecting the **Update List Item** button.

Notice after selecting the button, the title has changed from **United States** to **USA**.

Note the title of the last item in the list. Test the delete process by selecting the **Delete List Item** button.

Notice after selecting the button, the last item in the list has been removed.

Stop the local web server by pressing CTRL+C in the console/terminal window.

## Summary

In this exercise, you extended the SharePoint Framework to interact with data using the SharePoint REST API.