**SharePoint Framework (SPFx) - Graph API React JS - Read SharePoint List Items.**

Microsoft Graph offers a wide range of APIs to access the content and services provided by Office 365. In this article, we will see how we can read SharePoint list items into SharePoint framework – SPFx ReactJS web part using Graph API.

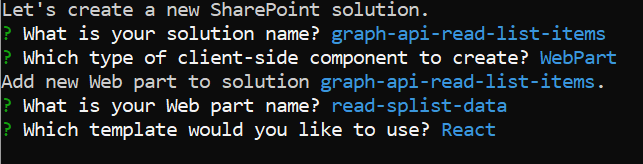
**Step 1:**Create a folder with the name GraphAPI-ReadListItems on your local drive.

C:\Users\ABC\Documents\SPFx\ **GraphAPI-ReadListItems**

**Step 2:**Open the location in the command prompt using cd command. Scaffold the SPFx solution using Yeoman Generator

*yo @microsoft/sharepoint*

**Step 3:**Give the webpart name and other details as shown below



**Step 4:**Install MS Graph types node package by using command:

*npm install @microsoft/microsoft-graph-types --save-dev*

**Step 5:** Open the solution in VS Code. You can use the "Code ." command in the command prompt directly to open the solution in VS Code. Update React component properties interface file **IReadSplistDataProps.ts.** Add property for **WebPartContext**

import { WebPartContext } from '@microsoft/sp-webpart-base';

export interface IReadSplistDataProps{

  context: WebPartContext;

}

**Step 6:** Create a new file under components folder **IListItem.ts** and paste below code – interface for our sp list columns.

export interface IListItem {

    Title: string;

    SoftwareName: string;

    SoftwareVendor: string;

}

**Step 7:** Create another file under component folder **IGraphConsumersplistState.ts** and paste below code – interface for react state to handle multiple records

import { IListItem } from './IListItem';

export interface IGraphConsumersplistState {

    lists: Array<IListItem>;

}

**Step 8:** Update web part render() method and add context property.

context: this.context

**Step 9:** Open **ReadSplistData.tsx** file under component folder add below codes

Import below files

import { IGraphConsumersplistState } from './IGraphConsumersplistState';

import { IListItem } from './IListItem';

import { MSGraphClient } from "@microsoft/sp-http";

import {

  PrimaryButton,

  TextField,

  Label,

  DetailsList,

  DetailsListLayoutMode,

  CheckboxVisibility,

  SelectionMode

} from 'office-ui-fabric-react';

then Configure the columns for the DetailsList component

let \_listItemColumns = [

  {

    key: 'Title',

    name: 'Title',

    fieldName: 'Title',

    minWidth: 50,

    maxWidth: 200,

    isResizable: true

  },

  {

    key: 'SoftwareName',

    name: 'Software Name',

    fieldName: 'SoftwareName',

    minWidth: 50,

    maxWidth: 200,

    isResizable: true

  },

  {

    key: 'SoftwareVendor',

    name: 'Software Vendor',

    fieldName: 'SoftwareVendor',

    minWidth: 50,

    maxWidth: 200,

    isResizable: true

  }

];

This array is used in the settings of the DetailsList component, as you can see in the render() method of the React component.

To render the graph api out put to Detail list column, replace below code to the render() method

**Step 10:** Update the React component type declaration and add a constructor

constructor(props: IReadSplistDataProps, state: IGraphConsumersplistState) {

    super(props);

    // Initialize the state of the component

    this.state = {

      lists: []

    };

  }

**Step 11:** Get List Items using Microsoft API replace the ComponentDidMount() method and all these methods support the asynchronous development model of Type Script, you can handle their result with promises.

You get an instance of the **MSGraphClient** type by calling the **context.msGraphClientFactory.getClient()** method.

You then use the fluent API of the Microsoft Graph SDK to define the OData query that runs against the target Microsoft Graph endpoint.

The result is a JSON response that you have to decode and map to the typed result.

public componentDidMount() {

    // Log the current operation

    console.log("Using \_searchWithGraph() method");

    this.props.context.msGraphClientFactory

      .getClient()

      .then((client: MSGraphClient): void => {

        client

          .api("sites('root')/lists('Test')/items?expand=fields")

          .version("v1.0")

          .get((err, res) => {

            if (err) {

              console.error(err);

              return;

            }

            // Prepare the output array

            var lists: Array<IListItem> = new Array<IListItem>();

            // Map the JSON response to the output array

            res.value.map((item: any) => {

              lists.push({

                Title: item.fields.Title,

                SoftwareName: item.fields.SoftwareName,

                SoftwareVendor: item.fields.SoftwareVendor

              });

            });

            // Update the component state accordingly to the result

            this.setState(

              {

                lists: lists,

              }

            );

          });

      });

  }

**Step 12:** To consume Microsoft Graph or any other third-party REST API, you need to explicitly declare the permission requirements from an OAuth perspective in the manifest of your solution.

Copy the declaration of the **webApiPermissionRequests** property in **cofig\package-solution.json**

    "webApiPermissionRequests": [{

      "resource": "Microsoft Graph",

      "scope": "User.ReadBasic.All"

    },

    {

        "resource": "Microsoft Graph",

        "scope": "Sites.Read.All"

    },

    {

        "resource": "Microsoft Graph",

        "scope": "Sites.ReadWrite.All"

    }

]

Notice the webApiPermissionRequests, which is an array of webApiPermissionRequest items. Each item defines the resource and the scope of the permission request. Don’t add the scope together, it will throw error while approve the API permission

**Step 13:** Deploy the solution and grant permissions

Run the gulp commands

*gulp build*

*gulp bundle --ship*

*gulp package-solution –ship*

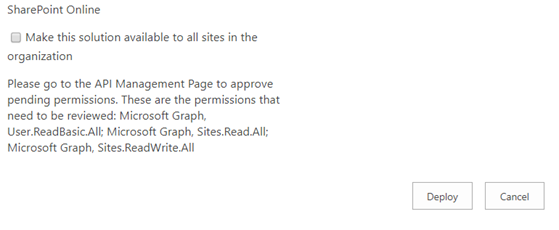
Also verify the **package-solution.json** file under config folder

*"includeClientSideAssets": true,*

*"isDomainIsolated": true,*

Deploy the webpart as Isolated webpart to maintain the security, when grant permission via API access.

**Step 14:** Deploy the package solution. Browse to the app catalog of your tenant and upload the solution package.



A message in the lower part of the screen tells you that the solution package requires permissions approval. This is because of the **webApiPermissionRequests** property in the **package-solution.json** file.

**Step 15:** API Permission Management

1. After deploying the web part, follow the below steps to approve API requests.
2. Open SharePoint Admin Center ([https://[tenant]-admin.sharepoint.com](https://[tenant]-admin.sharepoint.com/)).
3. From left navigation, click “Advanced” –> API access.
4. Approve the pending requests, refer below screenshot.

