# Exercise - Call Microsoft Graph from SharePoint Framework components

In this exercise, you'll create a new SharePoint Framework project with a single client-side web part that uses React and Microsoft Graph to display the currently logged in user's personal details in a familiar office [Persona](https://developer.microsoft.com/fabric#/components/persona) card. You'll use the Microsoft Graph HTTP client API included in the SharePoint Framework to authenticate and call the Microsoft Graph REST API.

## Create the Persona SharePoint Framework solution

Important

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

Open a command prompt and change to the folder where you want to create the project. Run the SharePoint Yeoman generator by executing the following command:

shell Copy

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?**: MSGraphSPFx
* **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?**: WebPart
* **What is your Web part name?**: GraphPersona
* **What is your Web part description?**: Display current user's persona details in a Fabric React Persona card
* **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, open the project in **Visual Studio Code**.

## Update the solution dependencies

Install the Microsoft Graph TypeScript type declarations by executing the following statement on the command line:

npm install @microsoft/sp-http --save

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

## Update the Persona web part

Update the default web part to pass the React component an instance of the Microsoft Graph client API.

Open the web part file **src/webparts/graphPersona/GraphPersonaWebPart.ts**.

Add the following import statement after the existing import statements:

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

Locate the render() method. This method creates a new instance of a React element by passing in the component class and the properties to bind to it. The only property being set is the description property.

Replace the contents of the render() method with the following code to create and initialize a new instance of the Microsoft Graph client:

this.context.msGraphClientFactory.getClient()

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

const element: React.ReactElement<IGraphPersonaProps> = React.createElement(

GraphPersona,

{

graphClient: client

}

);

ReactDom.render(element, this.domElement);

});

## Implement the GraphPersona React component

After updating the public signature of the GraphPersona component, the public property interface of the component needs to be updated to accept the Microsoft Graph client.

Open the **src/webparts/graphPersona/components/IGraphPersonaProps.ts**.

Replace the contents with the following code to change the public signature of the component:

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

export interface IGraphPersonaProps {

graphClient: MSGraphClient;

}

Create a new interface that will keep track of the state of the component's state.

Create a new file **IGraphPersonaState.ts** and save it to the folder: **src/webparts/graphResponse/components/**.

Add the following code to define a new state object that will be used by the component:

export interface IGraphPersonaState {

name: string;

email: string;

phone: string;

image: string;

}

Update the component's references to add the new state interface, support for the Microsoft Graph, Fabric React Persona control, and other necessary controls.

Open the **src/webparts/graphPersona/components/GraphPersona.tsx**.

Add the following import statements after the existing import statements:

import { IGraphPersonaState } from './IGraphPersonaState';

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

import \* as MicrosoftGraph from '@microsoft/microsoft-graph-types';

import {

Persona,

PersonaSize

} from 'office-ui-fabric-react/lib/components/Persona';

import { Link } from 'office-ui-fabric-react/lib/components/Link';

Update the public signature of the component to include the state.

Locate the class GraphPersona declaration.

At the end of the line, notice there's generic type with two parameters, the second is an empty object {}:

export default class GraphPersona extends React.Component<IGraphPersonaProps, {}>

Update the second parameter to be the state interface previously created:

export default class GraphPersona extends React.Component<IGraphPersonaProps, IGraphPersonaState>

Add the following constructor to the GraphPersona class to initialize the state of the component:

constructor(props: IGraphPersonaProps) {

super(props);

this.state = {

name: '',

email: '',

phone: '',

image: null

};

}

Add the Fabric React Persona card to the render() method's return statement:

public render(): React.ReactElement<IGraphPersonaProps> {

return (

<Persona primaryText={this.state.name}

secondaryText={this.state.email}

onRenderSecondaryText={this.\_renderMail}

tertiaryText={this.state.phone}

onRenderTertiaryText={this.\_renderPhone}

imageUrl={this.state.image}

size={PersonaSize.size100} />

);

}

The code in the Persona card references two utility methods to control rendering of the secondary and tertiary text. Add the following to methods to the GraphPersona class that will be used to render the text:

private \_renderMail = () => {

if (this.state.email) {

return <Link href={`mailto:${this.state.email}`}>{this.state.email}</Link>;

} else {

return <div />;

}

}

private \_renderPhone = () => {

if (this.state.phone) {

return <Link href={`tel:${this.state.phone}`}>{this.state.phone}</Link>;

} else {

return <div />;

}

}

The last step is to update the loading, or mounting, phase of the React component. When the component loads on the page, it should call Microsoft Graph to get details on the current user and their photo. When each of these results complete, they'll update the component's state, which will trigger the component to rerender.

Add the following method to the GraphPersona class:

public componentDidMount(): void {

this.props.graphClient

.api('me')

.get((error: any, user: MicrosoftGraph.User, rawResponse?: any) => {

this.setState({

name: user.displayName,

email: user.mail,

phone: user.businessPhones[0]

});

});

this.props.graphClient

.api('/me/photo/$value')

.responseType('blob')

.get((err: any, photoResponse: any, rawResponse: any) => {

const blobUrl = window.URL.createObjectURL(photoResponse);

this.setState({ image: blobUrl });

});

}

## Update the package permission requests

The last step before testing is to notify SharePoint that upon deployment to production, this app requires permission to the Microsoft Graph to access the user's persona details.

Open the **config/package-solution.json** file.

Locate the solution section. Add the following permission request element just after the property isDomainIsolated:

"webApiPermissionRequests": [

{

"resource": "Microsoft Graph",

"scope": "User.ReadBasic.All"

}

]

## Create the SharePoint package for deployment

Build the solution by executing the following command on the command line:

gulp build

Bundle the solution by executing the following command on the command line:

gulp bundle --ship

Package the solution by executing the following command on the command line:

gulp package-solution --ship

## Deploy and trust the SharePoint package

Note

The SharePoint Framework includes a locally hosted and a SharePoint Online hosted workbench for testing custom solutions. However, the workbench will not work the first time when testing solutions that utilize the Microsoft Graph due to nuances with how the workbench operates and authentication requirements. Therefore, the first time you test a Microsoft Graph enabled SPFx solution, you will need to test them in a real modern page.

Once this has been done and your browser has been cookied by the Azure AD authentication process, you can leverage local webserver and SharePoint Online-hosted workbench for testing the solution.

In the browser, navigate to your SharePoint Online Tenant App Catalog.

Select the **Apps for SharePoint** link in the navigation:

Drag the generated SharePoint package from **/sharepoint/solution/ms-graph-sp-fx.sppkg** into the **Apps for SharePoint** library.

In the **Do you trust ms-graph-sp-fx-client-side-solution?** dialog, select **Deploy**.

## Approve the API permission request

Navigate to the SharePoint Admin Portal located at **https://{{REPLACE\_WITH\_YOUR\_TENANTID}}-admin.sharepoint.com/\_layouts/15/online/AdminHome.aspx**, replacing the domain with your SharePoint Online's administration tenant URL.

In the navigation, select **Advanced > API access**:

Select the **Pending approval** for the **Microsoft Graph** permission **User.ReadBasic.All**.

Select the **Approve or Reject** button, followed by selecting **Approve**.

## Test the web part

### Add the web part to your site collection

In a browser, navigate to a SharePoint Online site.

In the Office 365 gear, select **Add an App**.

In site navigation, select **From your Organization**.

Select ms-graph-sp-fx-client-side-solution to add your web part.

### Test the web part on a SharePoint Online modern page

In the site navigation, select the **Pages** library.

Select an existing page (option 2 in the following image), or create a new page (option 1 in the following image) in the library to test the web part on.

Add the web part to the page and test.

In the browser, select the Web part icon button to open the list of available web parts:

Locate the **GraphPersona** web part and select it

When the page loads, notice after a brief delay, it will display the current user's details on the Persona card:

## Summary

In this exercise, you created a new SharePoint Framework project with a single client-side web part that uses React and Microsoft Graph to display the currently logged in user's personal details in a familiar office [Persona](https://developer.microsoft.com/fabric#/components/persona) card. You used the Microsoft Graph HTTP client API included in the SharePoint Framework to authenticate and call the Microsoft Graph REST API.