

Developing Webparts with the Microsoft Graph API

Lab Time: 60 minutes

Lab Folder: C:\Student\Modules\08_MicrosoftGraph\Lab

Lab Overview: In this module you will work through the process of developing a SPFx webpart that calls to the Microsoft Graph API. This lab assumes you are working with the SharePoint Framework project template with a version of 1.6 or later.

Exercise 1: Create a React Webpart that Calls the Microsoft Graph API

In this exercise you will create a new SPFx project with a single client-side web part that uses React, the Office UI Fabric React component library and the Microsoft Graph API to display the current user's personal details in the React Persona card.

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\08_MicrosoftGraph\Lab
```

- b) Type the following command and execute it by pressing **Enter** to create a new folder for your project.

```
md microsoft-graph-lab
```

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

```
cd microsoft-graph-lab
```

- d) The current directory for the console should now be located at the new folder you just created named **microsoft-graph-lab**.



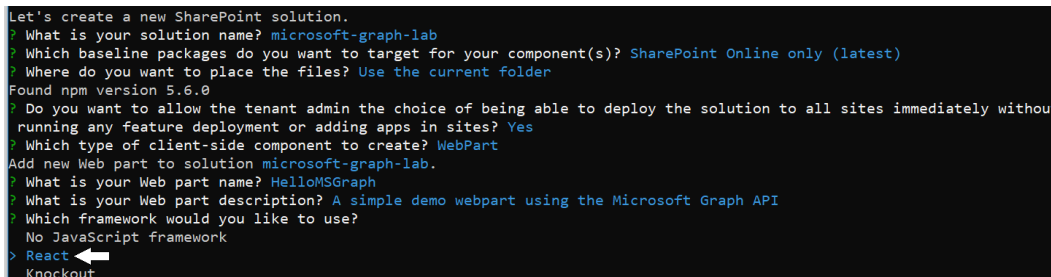
```
Node.js command prompt
c:\Student\Modules\08_MicrosoftGraph\Lab\microsoft-graph-lab>_
```

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

```
yo @microsoft/sharepoint
```

Use the following to complete the prompts that are displayed by the SPFx project template:

- f) What is your solution name?: **microsoft-graph-lab**
 - g) Which baseline packages do you want to target for your component(s)?: **SharePoint Online only (latest)**
 - h) Where do you want to place the files?: **Use the current folder**
 - i) 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?: **y**
 - j) Which type of client-side component to create?: **WebPart**
 - k) What is your Web part name?: **HelloMSGraph**
 - l) What is your Web part description?: **A simple demo webpart using the Microsoft Graph API**
 - m) Which framework would you like to use?: **React**



```
Let's create a new SharePoint solution.
? What is your solution name? microsoft-graph-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 microsoft-graph-lab.
? What is your Web part name? HelloMSGraph
? What is your Web part description? A simple demo webpart using the Microsoft Graph API
? Which framework would you like to use?
No JavaScript framework
> React
Knockout
```

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

```
added 1851 packages in 49.812s

_+#####!
#####!
##/  (##) (@)  [-----]
##/  ##### \   | Congratulations!
##/  /###  (@)  | Solution microsoft-graph-lab is created.
#####  ##   | Run gulp serve to play with it!
##/  /##| (@)  |
#####  |
*+=#####!

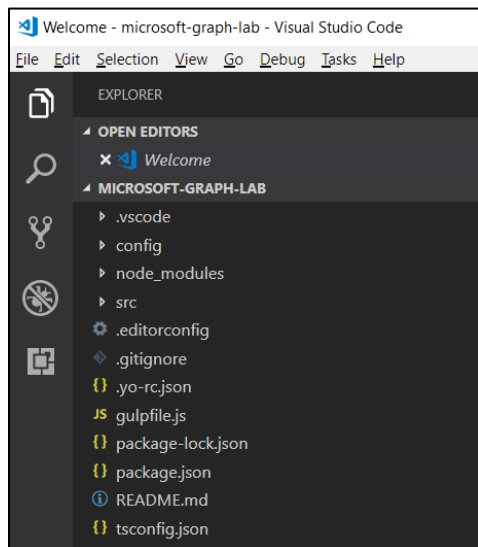
c:\Student\Modules\08_MicrosoftGraph\Lab\microsoft-graph-lab>
```

2. Open the project with Visual Studio Code.

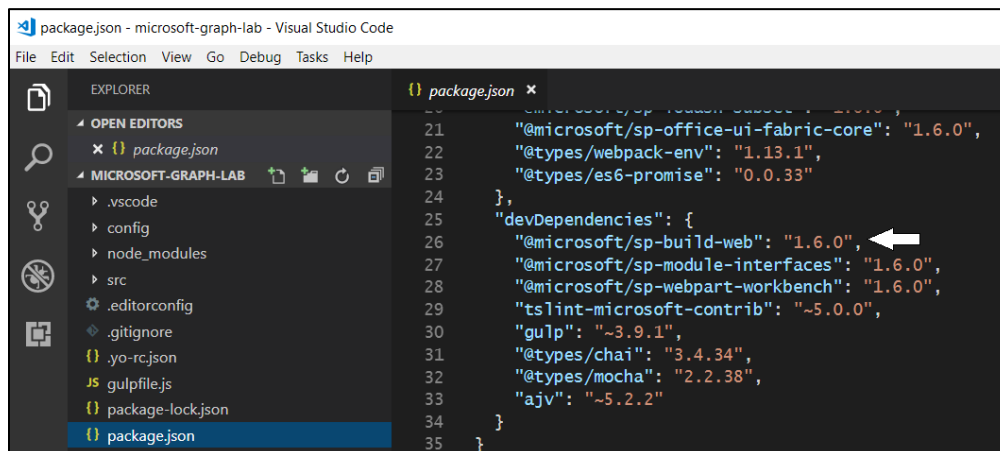
- a) Type the following command and execute it by pressing **Enter** to open your new project in Visual Studio Code.

```
code .
```

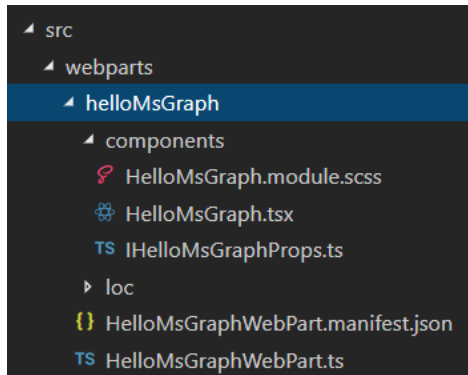
- b) Review the structure of the new project you've just created.



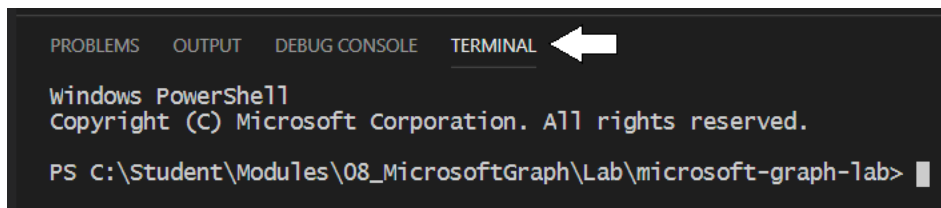
- c) Open package.json to verify the SharePoint Framework version.



- d) Expand the **src/webparts/helloMsGraph** folder and the inner **components** folder.
- e) Examine the names of the sources files associated with the new webpart.



- 3. Install the Node package for the Microsoft Graph API type definitions for TypeScript.
 - a) Use the **View > Terminal** menu command in Visual Studio Code to display the Integrated Terminal.
 - b) The Terminal should provide a console with its current directory located at your project folder.



- c) Install the Microsoft Graph TypeScript Types package by executing the following statement on the command line:

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

In order to take advantage of the Office UI Fabric and its react component library, it is necessary to uninstall the SharePoint Framework package named **sp-office-ui-fabric-core**.

- 4. Uninstall the Node package named **sp-office-ui-fabric-core** to remove it from the project
 - a) Execute the following command from the Terminal console to uninstall the **sp-office-ui-fabric-core** package.

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

- 5. Lock down the versions of the packages your project is using.
 - a) Execute the following npm command to lock in the version for each package in the current project..

```
npm shrinkwrap
```

Now that you have uninstalled the **sp-office-ui-fabric-core** package, the next step is to update the SCSS module for the React component named **helloMSGraph** in order to import the correct Office UI Fabric style sheet.

- 6. Configure the included components styles to use the Fabric Core CSS from the Fabric React project.
 - a) Inside the **helloMSGraph\components** folder, locate and open **helloMSGraph.module.scss**.
 - b) Currently, the first line in **helloMSGraph.module.scss** should match the following code listing.

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

- c) Remove the entire contents of **helloMSGraph.module.scss** with the following **@import** statement.

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

- d) Below the `@import` statement, add the following SCSS code.

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

.HelloMSGraph {
  background-color: $ms-color-themeLighter;
  border: 1px solid $ms-color-neutralPrimaryAlt;
  border-radius: 12px;
  padding: 8px;
  ul {
    margin-top: 2px;
    border-top: 1px solid black;
    padding-left: 24px;
    li {
      margin-top: 4px;
      color: black;
    }
  }
}
```

- e) Save your changes and close **helloMSGraph.module.scss**.

Over the next few steps you will update the interface that defines the properties of the React component so that the webpart class can pass an **MSGraphClient** instance when it creates the React components in the **render** method.

7. Update the interface for the React component to include a new property based on **MSGraphClient**.
- a) In the folder named **src\webparts\helloMSGraph\components**, open the file named **IHelloMsGraphProps.ts**.
 - b) Replace the contents of **IHelloMsGraphProps.ts** with the following code to update the React component properties:

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

export interface IHelloMsGraphProps {
  graphClient: MSGraphClient;
}
```

8. Update the default web part to pass into the React component an instance of the Microsoft Graph client API:
- a) In the **src\webparts\helloMSGraph** folder, open the web part file **HelloMsGraphWebPart.ts**.
 - b) Add the following **import** statements underneath all the other existing **import** statements:

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

Currently, the webpart **render** method creates the React element by passing the **description** property. Now, you must update **render** to pass an **MSGraphClient** instance instead of the **description** property.

- c) Replace the contents of the **render** method with the following code.

```
public render(): void {
  this.context.msGraphClientFactory
    .getClient()
    .then((client: MSGraphClient): void => {
      // create React component by passing MSGraphClient
      const element: React.ReactElement<IHelloMsGraphProps> = React.createElement(
        HelloMsGraph, { graphClient: client }
      );
      ReactDOM.render(element, this.domElement);
    });
}
```

Take a moment to understand what this code in the **render** method is doing. First, it creates an instance of the **MSGraphClient** class by calling the asynchronous method on the **msGraphClientFactory** named **getClient**. After creating a **MSGraphClient** instance, the **render** method then passes this **MSGraphClient** instance to the React component when calling **React.createElement**. This common design allows you to write the client code against the **MSGraphClient** instance in the React component instead of having to write the client code which uses the **MSGraphClient** instance inside the webpart class.

9. Save your changes to **HelloMsGraphWebPart.ts**.

10. Update the React component defined inside of **HelloMsGraph.tsx**.

- Inside the **src\webparts\helloMSGraph\components** folder, open **HelloMsGraph.tsx** in an editor window.
- Underneath all the other **import** statements, add the following **import** statements.

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

import {
  Persona,
  PersonaSize
} from 'office-ui-fabric-react/lib/components/Persona';
```

Next you will create a new interface named **IHelloMsGraphState** to define the React component's state. While you could define this new interface in its own file, in this lab you will define it inside **HelloMsGraph.tsx** just above the **HelloMsGraph** class definition.

- Underneath the existing **import** statements, add the interface named **IHelloMsGraphState**.

```
export interface IHelloMsGraphState {
  name: string;
  email: string;
  phone: string;
  image: string;
}
```

Now that you have defined the **IHelloMsGraphState** interface, you must update the React component class definition to use it.

- Locate the top of the **HelloMSGraph** class definition and notice the second parameter is passed as an empty object **{}**:

```
export default class HelloMSGraph extends React.Component<IHelloMSGraphProps, {}>
```

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

```
export default class HelloMSGraph extends React.Component<IHelloMSGraphProps, IHelloMsGraphState>
```

- Add the following state initializer to the top of the **HelloMSGraph** class definition to initialize component state at creation time.

```
public state = {
  name: '',
  email: '',
  phone: '',
  image: ''
};
```

- Modify the **render** method with the following code to return JSX using the **Persona** component from the Fabric React library.

```
public render(): React.ReactElement<IHelloMSGraphProps> {
  return (
    <div className={styles.HelloMSGraph} >
      <Persona primaryText={this.state.name}
        secondaryText='Personal Details:'
        onRenderTertiaryText={ () => (
          <ul>
            <li>email: {this.state.email}</li>
            <li>Phone: {this.state.phone}</li>
          </ul>
        )}
        imageUrl={this.state.image}
        size={PersonaSize.size100}
      />
    </div>
  );
}
```

The code in the **Persona** component include in inline function for tertiary text which returns HTML using JSX/TSX syntax. The last step is to update the React component to call into the Microsoft Graph API after the element for the component has mounted.

- h) Underneath the **render** method, add the following implementation for the **componentDidMount** lifecycle method.

```
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 });  
        });  
}
```

Note the syntax of calling into the Microsoft Graph API using the **MSGraphClient** using **.api()** and **.get()**.

- i) Save your changes to **HelloMSGraph.tsx**.

Exercise 2: Deploy the Solution Package and Grant Microsoft Graph API Permissions

In this exercise you will add permission request and build the solution package. Once you have built the solution package, you will then deploy it and step through the process of granting SharePoint Framework solution permissions for the Microsoft Graph API.

1. Update the manifest file for the Walmart Greeter webpart.
 - a) Inside **src/webparts/helloMSGraph**, open the webpart manifest file named **HelloMsGraphWebPart.manifest.json**.
 - b) Remove all the comments from **HelloMsGraphWebPart.manifest.json** until the red underlining is gone.
 - c) At the bottom of file named **HelloMsGraphWebPart.manifest.json**, locate the **preconfiguredEntries** section.
 - d) Inside the **preconfiguredEntries** section, modify the **default** value of **title** from **HelloMSGraph** to **Hello MS Graph API**.

```
"title": { "default": "Hello MS Graph API" },
```

- e) Modify the value of **officeFabricIconFontName** from **Page** to **TestUserSolid**.

```
"officeFabricIconFontName": "TestUserSolid",
```

- f) Modify the value of **properties** to be an empty object.

```
"properties": {}
```

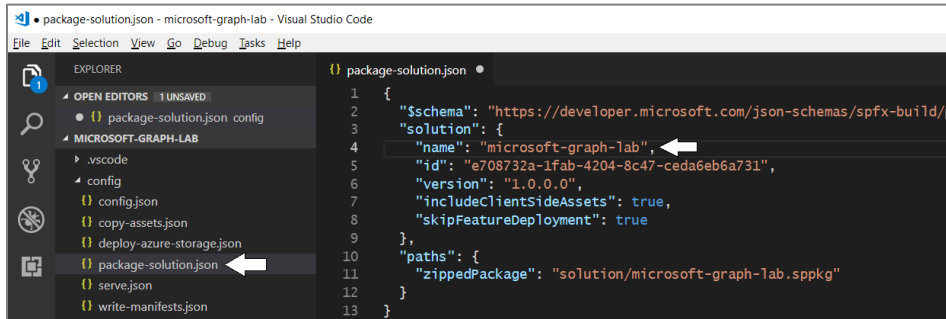
- g) Your edits should match what is shown in the following screenshot.

```
"preconfiguredEntries": [{  
    "groupId": "5c03119e-3074-46fd-976b-c60198311f70",  
    "group": { "default": "Critical Path Training" },  
    "title": { "default": "Hello MS Graph API" },  
    "description": { "default": "A simple demo webpart using the  
    "officeFabricIconFontName": "TestUserSolid",  
    "properties": {}  
    }  
}]
```

- h) Save your changes and close **HelloMsGraphWebPart.manifest.json**.

2. Update the SPFx Package Permission Requests

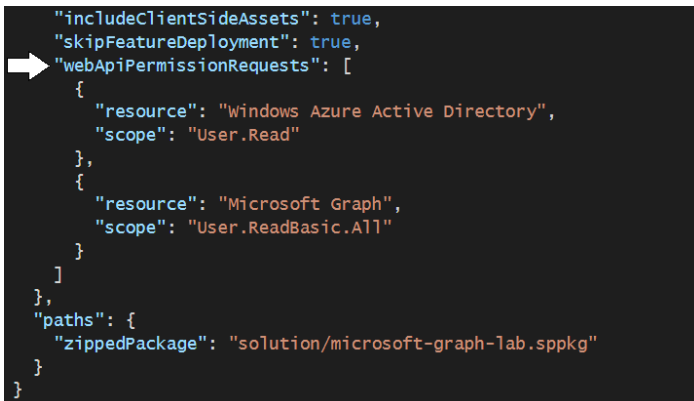
- Open the **config\package-solution.json** file.
- Shorten name from **microsoft-graph-lab-client-side-solution** to **microsoft-graph-lab**.



- Examine the JSON code inside **package-solution.json** and locate the **solution** node.
- Add the following **webApiPermissionRequests** property after the **includeClientSideAssets** and **skipFeatureDeployment**.

```
"includeClientSideAssets": true,
"skipFeatureDeployment": true,
"webApiPermissionRequests": [
  {
    "resource": "Windows Azure Active Directory",
    "scope": "User.Read"
  },
  {
    "resource": "Microsoft Graph",
    "scope": "User.ReadBasic.All"
  }
]
```

- Make sure your **webApiPermissionRequests** property in **package-solution.json** matches the following screenshot

A screenshot of the 'webApiPermissionRequests' array in the 'package-solution.json' file. A white arrow points to the start of the array. The array contains two objects: one for 'Windows Azure Active Directory' with scope 'User.Read', and another for 'Microsoft Graph' with scope 'User.ReadBasic.All'. The rest of the file content is visible below the array.

```
"includeClientSideAssets": true,
"skipFeatureDeployment": true,
"webApiPermissionRequests": [
  {
    "resource": "Windows Azure Active Directory",
    "scope": "User.Read"
  },
  {
    "resource": "Microsoft Graph",
    "scope": "User.ReadBasic.All"
  }
],
"paths": {
  "zippedPackage": "solution/microsoft-graph-lab.sppkg"
}
```

Note that requirement for the **Windows Azure Active Directory** permission will likely be removed in the near future.

- Save your changes to **package-solution.json**.

Now it's time to build your solution and install it so you can grant permissions that allows your code to call the Microsoft Graph API.

3. Create the SharePoint package for deployment.

- Navigate to the Terminal console so you can execute **gulp** commands.
- Build the solution by executing the **gulp build** command.

```
gulp build
```

- c) Bundle the solution by executing the **gulp bundle --ship** command.

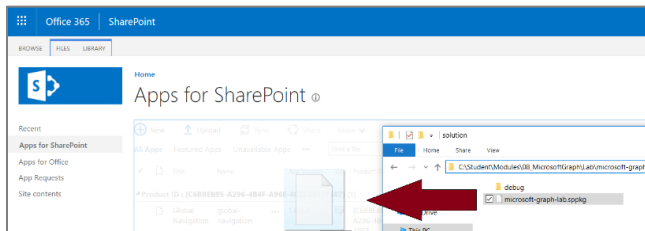
```
gulp bundle --ship
```

- d) Package the solution by executing the **gulp package-solution --ship** command.

```
gulp package-solution --ship
```

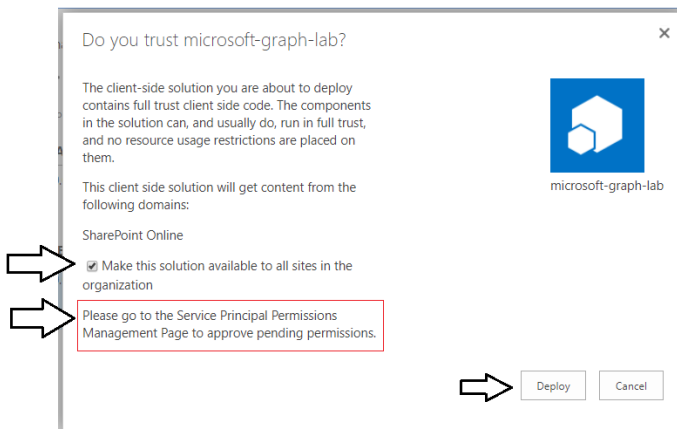
4. Deploy and trust the SharePoint package:

- a) In the browser, navigate to your SharePoint Online Tenant App Catalog.
b) Select the **Apps for SharePoint** link in the navigation.
c) In Windows Explorer, locate the solution package file **microsoft-graph-lab.sppkg** in the **\sharepoint\solution** folder.
d) Drag the generated SharePoint package named **microsoft-graph-lab.sppkg** into the **Apps for SharePoint** library.



At this point, you should be promoted by a dialog with the title **Do you trust microsoft-graph-lab?**

- e) Make sure to check the **Make this solution available to all sites in the organization?** option.
f) Read the text that tells you to go to the **Service Principal Permissions Management Page**.
g) Click the **Deploy** button to complete the installation.

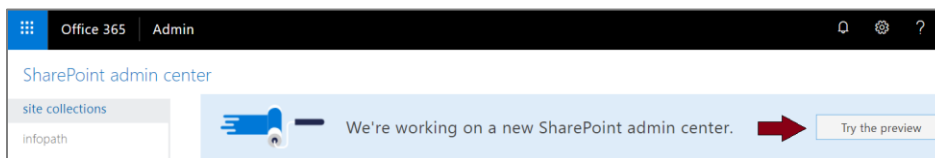


5. Approve the API permission request:

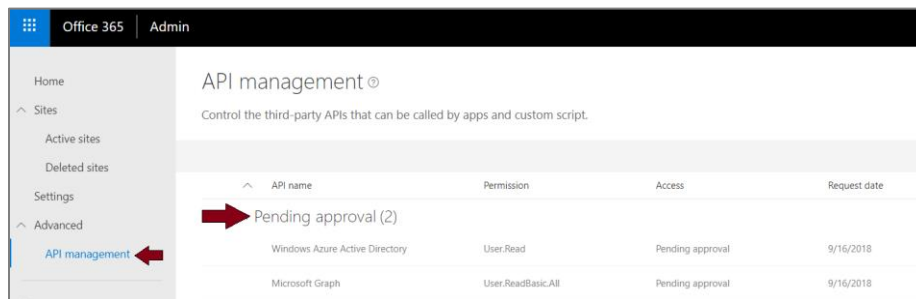
- a) Navigate to SharePoint Admin Portal using the following URL (swap [YOUR_TENANT_NAME] with your tenant name).

```
https://[YOUR_TENANT_NAME]-admin.sharepoint.com/_layouts/15/online/AdminHome.aspx
```

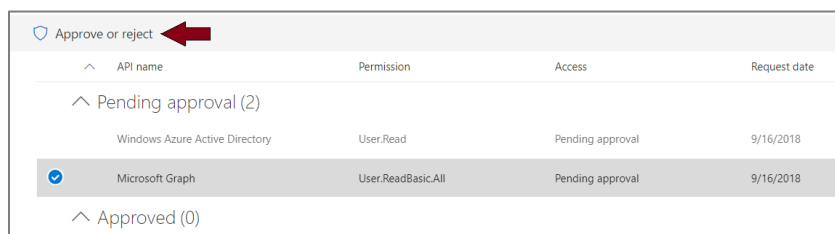
- b) Click **Try the preview** to switch over to the new SharePoint admin center which is still in preview as of September 2018.



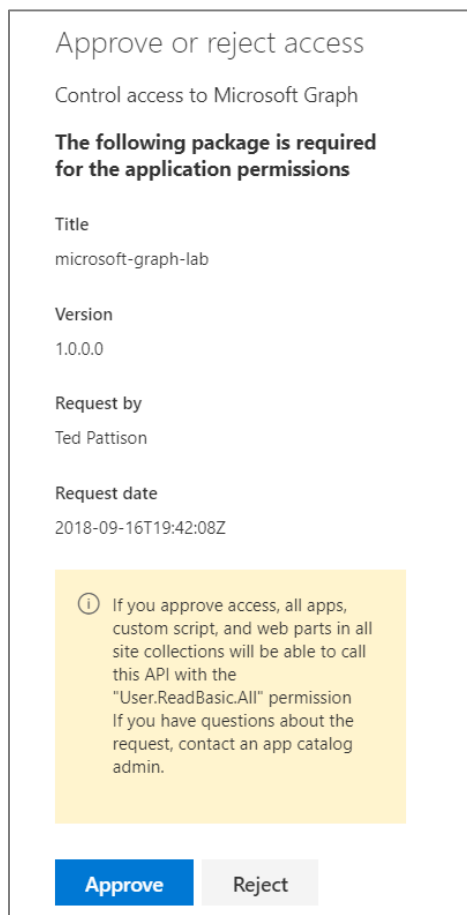
- c) In the navigation, select **Advanced > API Management**:



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



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



- f) Follow the same steps to approve the **User.Read** permission from **Windows Azure Active Directory**.
- g) When you are done, the permissions you need should be located in the **Approved** section.

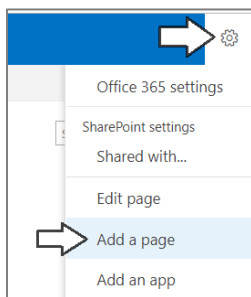
^ Approved (2)		
Microsoft Graph	User.ReadBasic.All	Approved
Windows Azure Active Directory	User.Read	Approved

The SharePoint Framework includes the hosted SharePoint workbench in SharePoint Online for testing custom solutions. However, it is currently required to run the code the first time in a real modern pages to successfully get through the authentication process with the Microsoft Graph API proxy in SharePoint Online. Therefore, you will now run the webpart on a real modern page. Once you get through this step and your browser has cached the authentication results, you can then leverage local webserver and hosted SharePoint workbench for testing the solution.

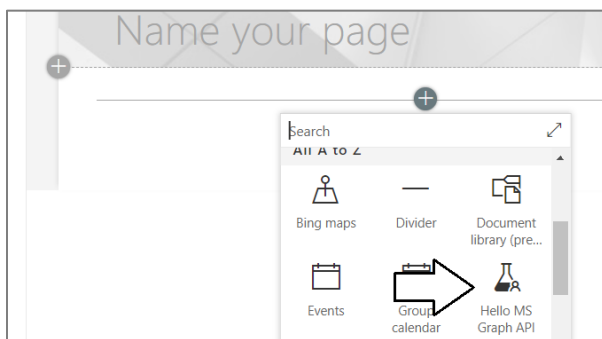
6. Create a modern page to test the **HelloMSGraph** webpart.
- a) In a browser, navigate to the root SharePoint site for your Office 365 tenant.

[https://\[YOUR_TENANT_NAME\].sharepoint.com](https://[YOUR_TENANT_NAME].sharepoint.com)

- b) Drop down the **Site Actions** menu and select the **Add a page** menu command.



- c) In the browser, select the Web part icon button to open the list of available web parts:
- d) Locate the **Hello MS Graph API** webpart and select it



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

