Developing SPFX Web Parts using the Microsoft Graph API

Lab Time: 60 minutes

Lab Folder: C:\Student\Modules\02_AzureWebApp\Lab

Lab Overview: In this module you…

Exercise 1: Access the Microsoft Graph API from a React Webpart

In this exercise you will create a new SPFx project with a single client-side web part that uses React, Fabric React and the Microsoft Graph to display the currently logged in user's personal details in a familiar office Persona card.

1. Create the SPFx Solution
   a) Open a command prompt and change to the folder where you want to create the project.
   b) Run the SharePoint Yeoman generator by executing the following command:

   ```
   yo @microsoft/sharepoint --plusbeta
   ```

   c) Use the following to complete the prompt that is displayed:
   d) **What is your solution name?**: MSGraphSPFx
   e) **Which baseline packages do you want to target for your component(s)?**: SharePoint Online only (latest)
   f) **Where do you want to place the files?**: Use the current folder
   g) **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
   h) **Which type of client-side component to create?**: WebPart
   i) **What is your Web part name?**: GraphPersona
   j) **What is your Web part description?**: Display current user's persona details in a Fabric React Persona card
   k) **Which framework would you like to use?**: React
   l) After provisioning the folders required for the project, the generator will install all the dependency packages using NPM.
   m) When NPM completes downloading all dependencies, open the project in Visual Studio Code.
2. Update Solution Dependencies
   a) Install the Microsoft Graph Typescript type declarations by executing the following statement on the command line:

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

   b) The web part will use the Fabric React controls to display user interface components. Configure the project to use Fabric React:
   c) Execute the following on the command line to uninstall the SPFx Fabric Core library which is not needed as it is included in Fabric React:

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

3. Configure the included components styles to use the Fabric Core CSS from the Fabric React project.
   a) Open the **src\webparts\graphPersona\components\GraphPersona.module.scss**
   b) Replace the first line:

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

   c) With the following:

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

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

a) Open the web part file **src\webparts\graphPersona\GraphPersonaWebPart.ts**.

b) Add the following `import` statements after the existing `import` statements:

```
import { MSGraphClient } from '@microsoft/sp-client-preview';
import * as MicrosoftGraph from '@microsoft/microsoft-graph-types';
```

c) 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.

d) Replace the contents of the **`render`** method with the following code to create an initialize a new instance of the Microsoft Graph client:

```
const element: React.ReactElement<IGraphPersonaProps> = React.createElement(
  GraphPersona,
  {
    graphClient: this.context.serviceScope.consume(MSGraphClient.serviceKey)
  }
);

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

Implement the React Component

5. 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:

a) Open the **src\webparts\graphPersona\components\IGraphPersonaProps.tsx**

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

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

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

c) Create a new interface that will keep track of the state of the component's state:

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

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

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

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

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

a) Open the **src\webparts\graphPersona\components\GraphPersona.tsx**

b) Add the following `import` statements after the existing `import` statements:

```
import { IGraphPersonaState } from './IGraphPersonaState';

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

7. Update the public signature of the component to include the state:
   a) Locate the class `GraphPersona` declaration.
   b) At the end of the line, notice there is generic type with two parameters, the second is an empty object {}:

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

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

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

   d) 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
  };
}
```

   e) 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} />
  );
}
```

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

```
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 the Microsoft Graph to get details on the current user as well as their photo. When each of these results complete, they will update the component's state which will trigger the component to rerender.

   g) 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(rawResponse.xhr.response);
      this.setState({ image: blobUrl });
    });
}
```

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.

8. Update the SPFx Package Permission Requests
   a) Open the **config\package-solution.json** file.
   b) Locate the `solution` section.
   c) Add the following permission request element just after the property `includeClientSideAssets`:

```
"webApiPermissionRequests": [
  {
    "resource": "Microsoft Graph",
    "scope": "User.ReadBasic.All"
  }
]
```

Now it's time to Test the Solution

9. Create the SharePoint package for deployment:
   a) Build the solution by executing the following on the command line:

```
gulp build
```

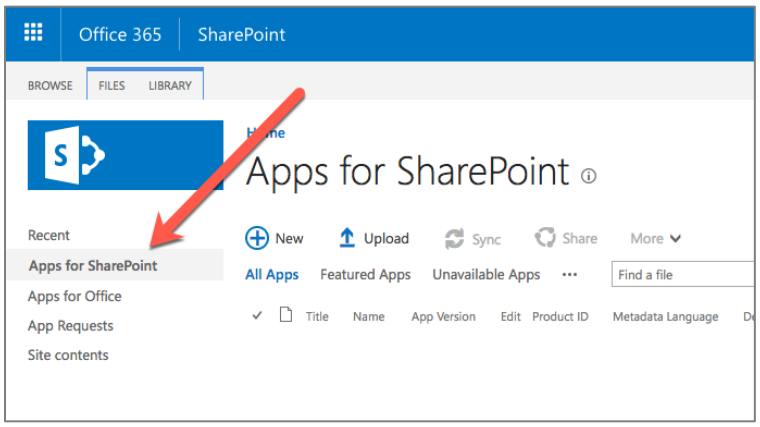
   b) Bundle the solution by executing the following on the command line:

```
gulp bundle --ship
```

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

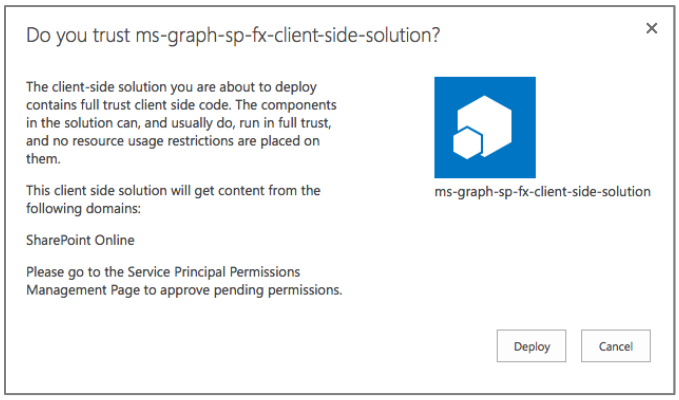
```
gulp package-solution --ship
```

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

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



11. Approve the API permission request:

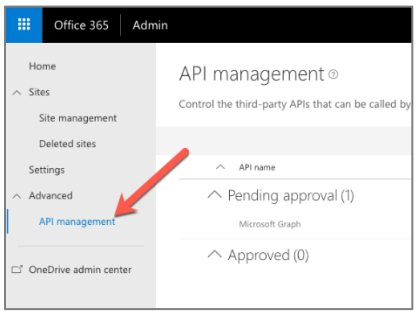
a) Navigate to the SharePoint Admin Portal located at

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

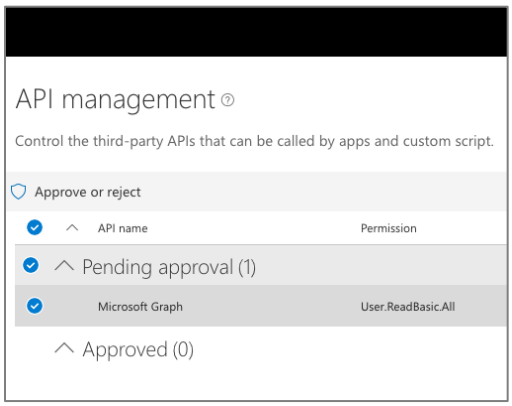
b) Replacing the domain with your SharePoint Online's administration tenant URL.

As of September 2018, this feature is only in the SharePoint Online preview portal.

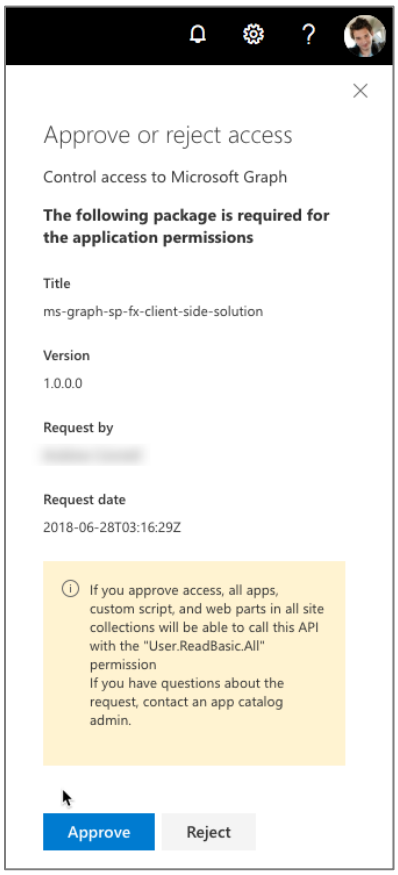
c) In the navigation, select **Advanced > API Management**:



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



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



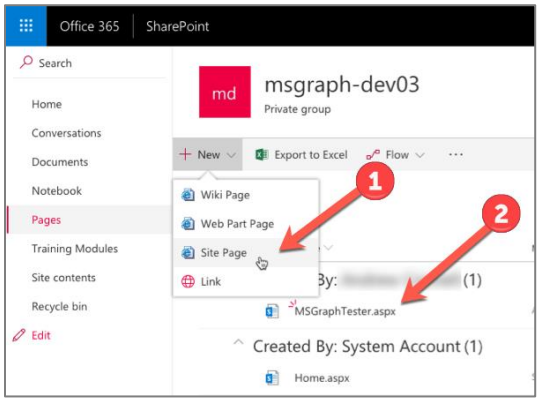
The SharePoint Framework includes a locally hosted & SharePoint Online hosted workbench for testing custom solutions. However, the workbench will not work the first time when testing solutions that utilize the Microsoft due to nuances with how the workbench operates and authentication requirements. Therefore, 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.

12. Setup environment to test the web part on a real SharePoint Online modern page:
    a) In a browser, navigate to a SharePoint Online site.

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

c) 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.



13. Setup environment to test the from the local webserver and hosted workbench:

a) In the command prompt for the project, execute the following command to start the local web server:

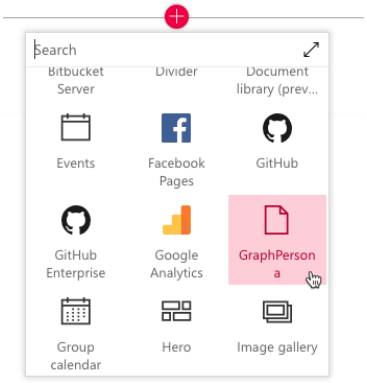
```
gulp serve --nobrowser
```

b) In a browser, navigate to one of your SharePoint Online site's hosted workbench located at **/_layouts/15/workbench.aspx**

c) Add the web part to the page and test:

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

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



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

Exercise 2: Show calendar events from Microsoft Graph in SPFx client-side web part

In this exercise you add a client-side web part that uses React, Fabric React and the Microsoft Graph to an existing SPFx project that will display a list of the current user's calendar events using the List component.

1. Add a second webpart to the existing SharePoint Framework solution.
   a) Open a command prompt and change to the folder of the existing SPFx solution.
   b) Run the SharePoint Yeoman generator by executing the following command:

```
yo @microsoft/sharepoint --plusbeta
```

Use the following to complete the prompt that is displayed:

   c) **Which type of client-side component to create?**: WebPart
   d) **What is your Web part name?**: GraphEventsList
   e) **What is your Web part description?**: Display current user's calendar events in a Fabric React List
   f) **Which framework would you like to use?**: React

2. The project will use a library to assist in working with dates.
   a) Add this by executing the following command in the command prompt:

```
npm install date-fns --save
```

   b) Open the project in Visual Studio Code.

First, you will update the default web part to pass into the React component an instance of the Microsoft Graph client API:

   c) Open the web part file **src\webparts\graphEventsList\GraphEventsListWebPart.ts**.
   d) Add the following `import` statements after the existing `import` statements:

```
import { MSGraphClient } from '@microsoft/sp-client-preview';
import * as MicrosoftGraph from '@microsoft/microsoft-graph-types';
```

   1. Locate the `render` method.

The **render** 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.

   e) Replace the contents of the `render` method with the following code to create an initialize a new instance fo the Microsoft Graph client:

```
const element: React.ReactElement<IGraphEventsListProps> = React.createElement(
  GraphEventsList,
  {
    graphClient: this.context.serviceScope.consume(MSGraphClient.serviceKey)
  }
);

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

Implement the React Component

3. After updating the public signature of the **GraphEventsList** component, the public property interface of the component needs to be updated to accept the Microsoft Graph client:
   a) Open the **src\webparts\graphEventsList\components\IGraphEventsListProps.tsx**
   b) Replace the contents with the following code to change the public signature of the component:

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

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

4. Create a new interface that will keep track of the state of the component's state:
   a) Create a new file **IGraphEventsListState.ts** and save it to the folder: **src\webparts\graphEventsList\components**.
   b) Add the following code to define a new state object that will be used by the component:

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

export interface IGraphEventsListState {
  events: MicrosoftGraph.Event[];
}
```

5. Update the component's references to add the new state interface, support for the Microsoft Graph, Fabric React List and other necessary controls.
   a) Open the **src\webparts\graphEventsList\components\GraphEventsList.tsx**
   b) Add the following `import` statements after the existing `import` statements:

```
import { IGraphEventsListState } from './IGraphEventsListState';

import { MSGraphClient } from '@microsoft/sp-client-preview';
import * as MicrosoftGraph from '@microsoft/microsoft-graph-types';

import { List } from 'office-ui-fabric-react/lib/List';
import { format } from 'date-fns';
```

6. Update the public signature of the component to include the state:
   a) Locate the class `GraphEventsList` declaration.
   b) At the end of the line, notice there is generic type with two parameters, the second is an empty object `{}`:

```
export default class GraphEventsList extends React.Component<IGraphEventsListProps, {}>
```

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

```
export default class GraphEventsList extends React.Component<IGraphEventsListProps, IGraphEventsListState>
```

   d) Add the following constructor to the `GraphEventsList` class to initialize the state of the component:

```
constructor(props: IGraphEventsListProps) {
  super(props);

  this.state = {
    events: []
  };
}
```

   e) Add the Fabric React List to the `render()` method's return statement:

```
public render(): React.ReactElement<IGraphEventsListProps> {
  return (
    <List items={this.state.events}
          onRenderCell={this._onRenderEventCell} />
```

```
  );
}
```

f) The code in the List card references a utility methods to control rendering of the list cell. Add the following to method to the `GraphEventsList` class that will be used to render the cell accordingly:

```
private _onRenderEventCell(item: MicrosoftGraph.Event, index: number | undefined): JSX.Element {
  return (
    <div>
      <h3>{item.subject}</h3>
      {format( new Date(item.start.dateTime),
               'MMMM Mo, YYYY h:mm A')} - {format( new Date(item.end.dateTime), 'h:mm A')}
    </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 the Microsoft Graph to get current user's calendar events. When each of these results complete, they will update the component's state which will trigger the component to rerender.

g) Add the following method to the `GraphEventsList` class:

```
public componentDidMount(): void {
  this.props.graphClient
    .api('/me/events')
    .get((error: any, eventsResponse: any, rawResponse?: any) => {
      const calendarEvents: MicrosoftGraph.Event[] = eventsResponse.value;
      console.log('calendarEvents', calendarEvents);
      this.setState({ events: calendarEvents });
    });
}
```

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 calendar events.

7. Update the SPFx Package Permission Requests
   a) Open the **config\package-solution.json** file.
   b) Locate the `webApiPermissionRequests` property. Add the following permission request element just after the existing permission:

```
{
  "resource": "Microsoft Graph",
  "scope": "Calendars.Read"
}
```

Test the Solution

8. Create the SharePoint package for deployment:
   a) Build the solution by executing the following on the command line:

```
gulp build
```

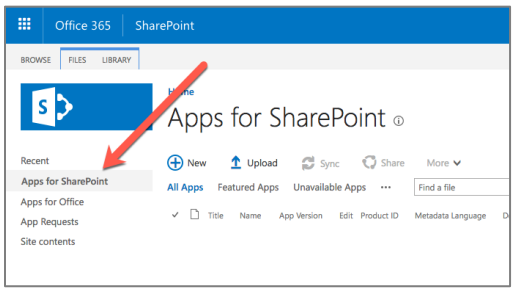
   b) Bundle the solution by executing the following on the command line:

```
gulp bundle --ship
```

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

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

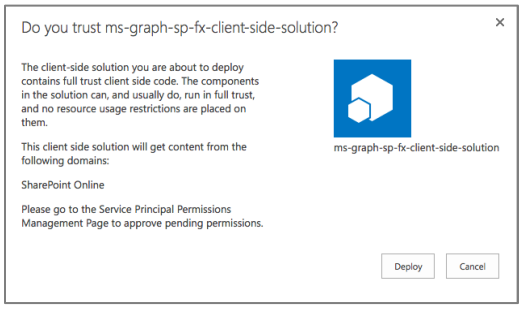
9. Deploy and trust the SharePoint package:
   a) In the browser, navigate to your SharePoint Online Tenant App Catalog.
   b) Select the **Apps for SharePoint** link in the navigation:



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

d) If you previously uploaded the same package, as in the case from exercise 1, if the **A file with the same name already exists** dialog, select the **Replace It** button.

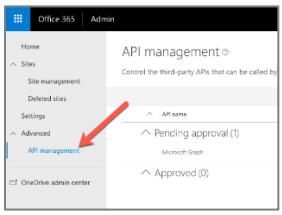
e) In the **Do you trust ms-graph-sp-fx-client-side-solution?** dialog, select **Deploy**.



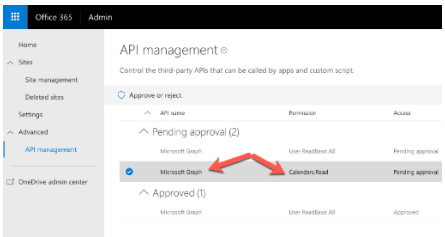
10. Approve the API permission request:

a) 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.

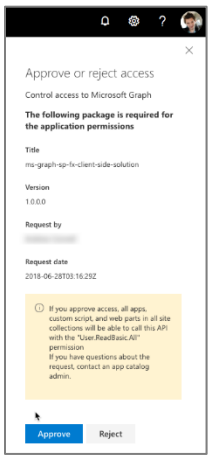
b) In the navigation, select **Advanced** > **API Management**:



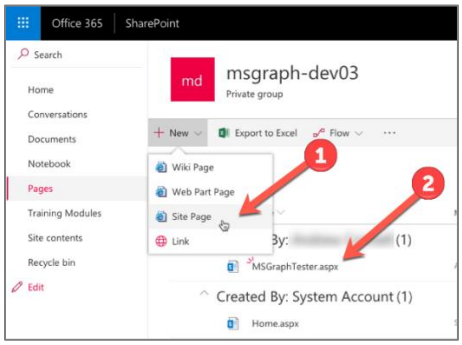
c) Select the **Pending approval** for the **Microsoft Graph** permission **Calendars.Read**.



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



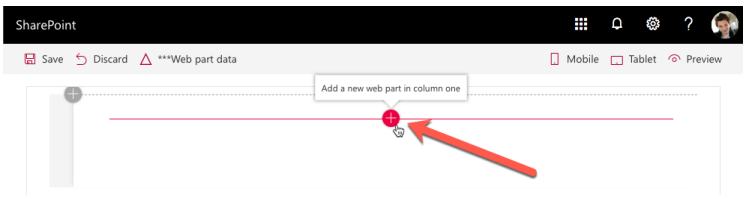
11. Test the web part:
    a) Setup environment to test the web part on a real SharePoint Online modern page:
    b) In a browser, navigate to a SharePoint Online site.
    c) In the site navigation, select the **Pages** library.
    d) Select an existing page (**option 2 in the following image**), or create a new page (**option 1 in the following image**) in the library to test the web part on.



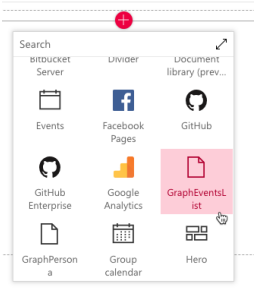
12. Setup environment to test the from the local webserver and hosted workbench:
    a) In the command prompt for the project, execute the following command to start the local web server:

```
gulp serve --nobrowser
```

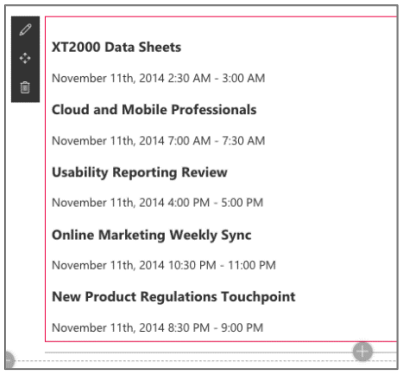
    b) In a browser, navigate to one of your SharePoint Online site's hosted workbench located at **/_layouts/15/workbench.aspx**
    c) Add the web part to the page and test:
    d) In the browser, select the Web part icon button to open the list of available web parts:



    e) Locate the **GraphEventList** web part and select it



f) When the page loads, notice after a brief delay, it will display the current user's calendar events in the list



Exercise 3: Show Planner tasks from Microsoft Graph in SPFx client-side web part

In this exercise you add a client-side web part that uses React, Fabric React and the Microsoft Graph to an existing SPFx project that will display a list of the current user's tasks from Planner using the List component.

13. Add SPFx Component to Existing SPFx Solution
    a) Open a command prompt and change to the folder of the existing SPFx solution.
    b) Run the SharePoint Yeoman generator by executing the following command:

```
yo @microsoft/sharepoint --plusbeta
```

Use the following to complete the prompt that is displayed:

    c) **Which type of client-side component to create?**: WebPart
    d) **What is your Web part name?**: GraphTasks
    e) **What is your Web part description?**: Display current user's tasks from Planner in a Fabric React List
    f) **Which framework would you like to use?**: React

After provisioning the folders required for the project, the generator will install all the dependency packages using NPM.

    g) Open the project in Visual Studio Code.

14. Update the default Webpart to pass into the React component an instance of the Microsoft Graph client API.
    a) Open the web part file **src\webparts\graphTasks\GraphTasksWebPart.ts**.
    b) Add the following `import` statements after the existing `import` statements:

```
import { MSGraphClient } from '@microsoft/sp-client-preview';
import * as MicrosoftGraph from '@microsoft/microsoft-graph-types';
```

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

```
const element: React.ReactElement<IGraphTasksProps> = React.createElement(
  GraphPersona,
  {
    graphClient: this.context.serviceScope.consume(MSGraphClient.serviceKey)
  }
);

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

15. Implement the React Component

a) After updating the public signature of the **GraphTasks** component, the public property interface of the component needs to be updated to accept the Microsoft Graph client:

b) Open the **src\webparts\graphTasks\components\IGraphTasksProps.tsx**

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

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

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

d) Create a new interface that will keep track of the state of the component's state:

e) Create a new file **IGraphTasksState.ts** and save it to the folder: **src\webparts\graphTasks\components**.

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

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

export interface IGraphTasksState {
  tasks: MicrosoftGraph.PlannerTask[];
}
```

Now you must update the component's references to add the new state interface, support for the Microsoft Graph, Fabric React List and other necessary controls.

16. Update the component's references.

a) Open the **src\webparts\graphTasks\components\GraphTasks.tsx**

b) Add the following `import` statements after the existing `import` statements:

```
import { IGraphTasksState } from './IGraphTasksState';

import { MSGraphClient } from '@microsoft/sp-client-preview';
import * as MicrosoftGraph from '@microsoft/microsoft-graph-types';

import { List } from 'office-ui-fabric-react/lib/List';
import { format } from 'date-fns';
```

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

a) Locate the class `GraphTasks` declaration.

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

```
export default class GraphTasks extends React.Component<IGraphTasksProps, {}>
```

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

```
export default class GraphTasks extends React.Component<IGraphTasksProps, IGraphTasksState>
```

d) Add the following constructor to the `GraphTasks` class to initialize the state of the component:

```
constructor(props: IGraphTasksProps) {
  super(props);
```

```
  this.state = {
    tasks: []
  };
}
```

e) Add the Fabric React List to the `render()` method's return statement:

```
public render(): React.ReactElement<IGraphTasksProps> {
  return (
    <List items={this.state.tasks}
          onRenderCell={this._onRenderEventCell} />
  );
}
```

f) The code in the List card references a utility methods to control rendering of the list cell. Add the following to method to the `GraphTasks` class that will be used to render the cell accordingly:

```
private _onRenderEventCell(item: MicrosoftGraph.PlannerTask, index: number | undefined): JSX.Element {
  return (
    <div>
      <h3>{item.subject}</h3>
      <strong>Due:</strong> {format( new Date(item.dueDateTime), 'MMMM Mo, YYYY at h:mm A')}
    </div>
  );
}
```

g) Add the following method to the `GraphEventsList` class:

```
public componentDidMount(): void {
  this.props.graphClient
    .api('/me/planner/tasks')
    .get((error: any, tasksResponse: any, rawResponse?: any) => {
      console.log('tasksResponse', tasksResponse);
      const plannerTasks: MicrosoftGraph.PlannerTask[] = tasksResponse.value;
      this.setState({ tasks: plannerTasks });
    });
}
```

18. Update the SPFx Package Permission Requests

a) Open the **config\package-solution.json** file.

b) Locate the `webApiPermissionRequests` property. Add the following permission request element just after the existing permission:

```
{
  "resource": "Microsoft Graph",
  "scope": "Group.Read.All"
}
```

Note: There are multiple "**task**" related permissions (scopes) used with the Microsoft Graph. Planner tasks are accessible via the **`Groups.Read.All`** scope while Outlook/Exchange tasks are accessible via the `Tasks.Read` scope.

19. Create the SharePoint package for deployment:

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

```
gulp build
```

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

```
gulp bundle --ship
```

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

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

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

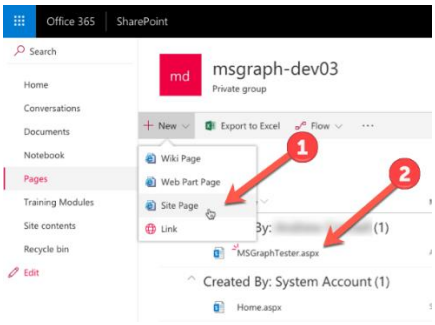
If you previously uploaded the same package, as in the case from exercise 1, if the **A file with the same name already exists** dialog, select the **Replace It** button.

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

21. Approve the API permission request:
    a) 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.
    b) In the navigation, select **Advanced > API Management**:
    c) Select the **Pending approval** for the **Microsoft Graph** permission **Group.Read.All**.
    d) Select the **Approve or Reject** button, followed by selecting **Approve**.

Test the web part:

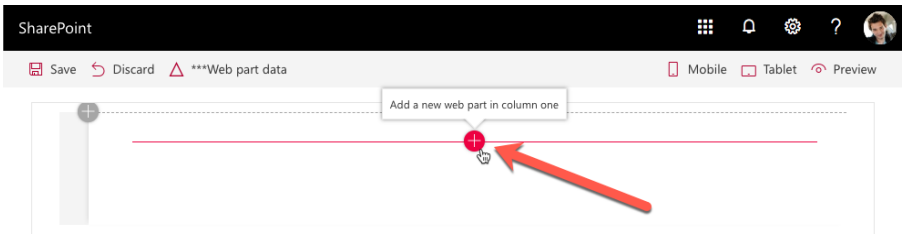
22. Setup environment to test the web part on a real SharePoint Online modern page:
    a) In a browser, navigate to a SharePoint Online site.
        i) In the site navigation, select the **Pages** library.
    b) Select an existing page (**option 2 in the following image**), or create a new page (**option 1 in the following image**) in the library to test the web part on.



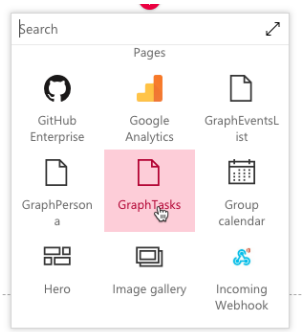
23. Setup environment to test the from the local webserver and hosted workbench:
    a) In the command prompt for the project, execute the following command to start the local web server:

```
gulp serve --nobrowser
```

    b) In a browser, navigate to one of your SharePoint Online site's hosted workbench located at **/_layouts/15/workbench.aspx**
    c) Add the web part to the page and test:
    d) In the browser, select the Web part icon button to open the list of available web parts:



e) Locate the **GraphTasks** web part and select it



f) When the page loads, notice after a brief delay, it will display the current user's tasks in the list: