**Gustavo Velez**

**gustavo@gavd.net**

Bootcamp SharePoint Saturday - Madrid, noviembre 11, 2020

**Usando Microsoft Graph con SPFx**

Tutorial

Contenido

[1. Requisitos 2](#_Toc55245449)

[2. Ejemplo 01 - WebPart SPFx básica (JavaScript) 2](#_Toc55245450)

[2.1. Generar la WebPart 2](#_Toc55245451)

[2.2. Comprobar la WebPart 3](#_Toc55245452)

[2.3. El código de la WebPart 4](#_Toc55245453)

[2.4. Agregue los permisos de Graph 7](#_Toc55245454)

[2.5. Desplegar la WebPart en el tenant 8](#_Toc55245455)

[2.6. Aprobar los permisos de la WebPart en Azure AD 9](#_Toc55245456)

[2.7. Testear la WebPart en SharePoint 10](#_Toc55245457)

[3. Ejemplo 02 - WebPart SPFx (React) 13](#_Toc55245458)

[3.1. Generar la WebPart 13](#_Toc55245459)

[3.2. Comprobar la WebPart 14](#_Toc55245460)

[3.3. El código de la WebPart 14](#_Toc55245461)

[3.4. Desplegar la WebPart en el tenant 21](#_Toc55245462)

[3.5. Aprobar los permisos de la WebPart en Azure AD 22](#_Toc55245463)

[3.6. Testear la WebPart en SharePoint 23](#_Toc55245464)

## Requisitos

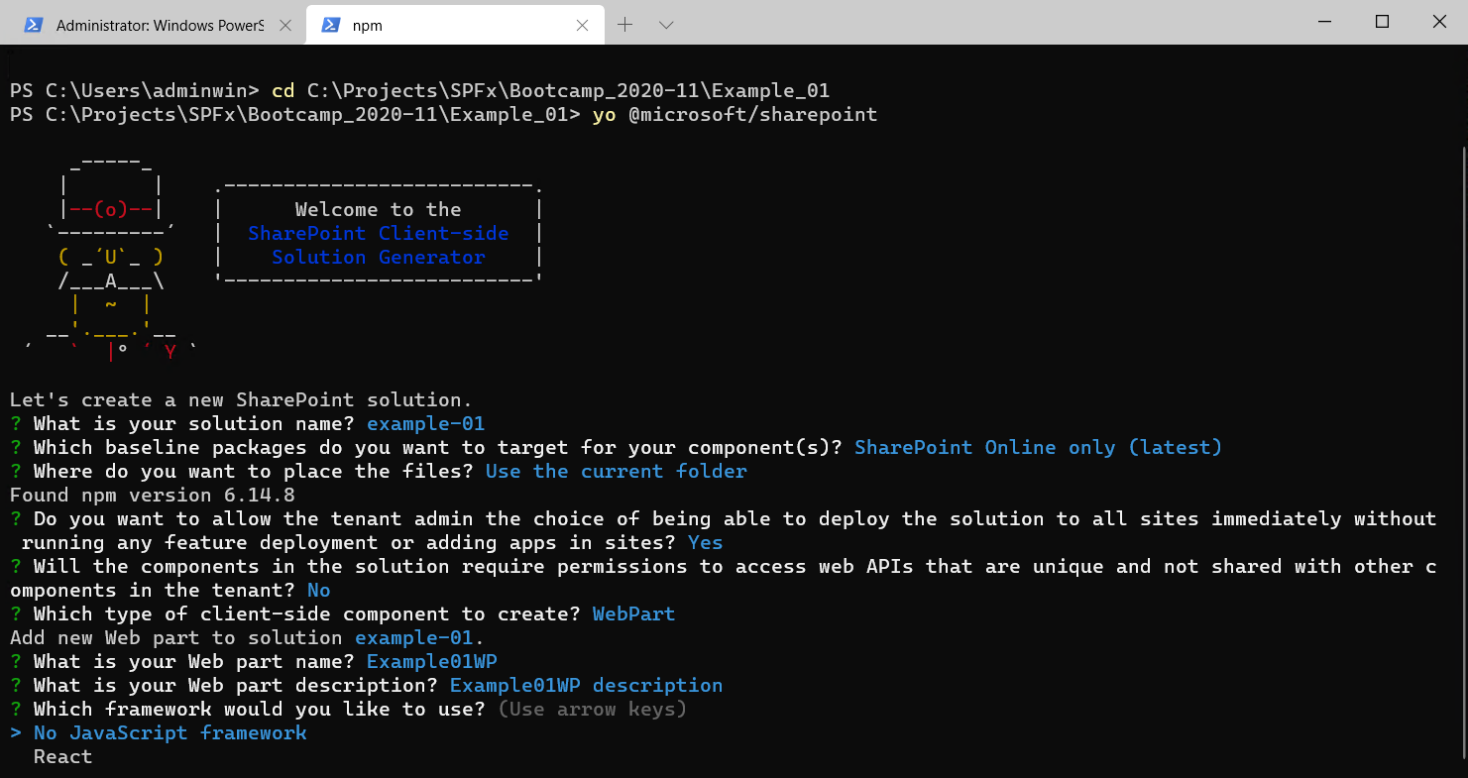
* Computador (o Máquina Virtual) con las últimas versiones disponibles instaladas de
  + NodeJS (debe tener versión 10.23.0 instalada, no la última versión)
  + Npm
  + Yeoman
  + Gulp
  + Yeoman SharePoint Generator
  + Visual Studio Code
* Instrucciones sobre configuración de un computador para programar componentes SPFx se puede encontrar en el sitio de Microsoft <https://docs.microsoft.com/en-us/sharepoint/dev/spfx/set-up-your-development-environment>

## Ejemplo 01 - WebPart SPFx básica (JavaScript)

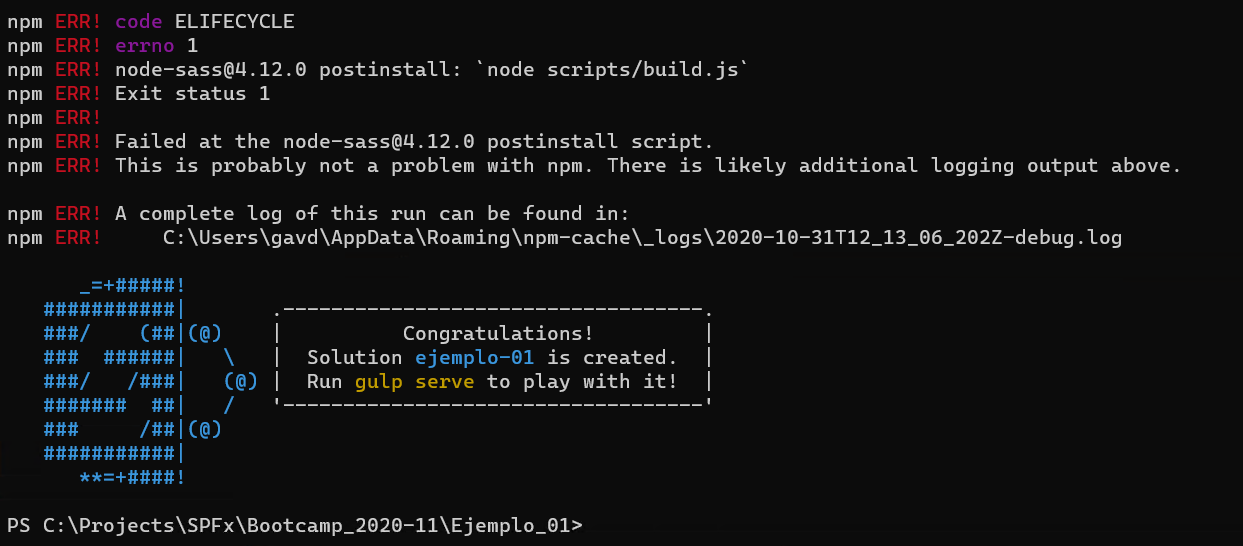
Esta WebPart muestra como ejecutar comandos de Microsoft Graph para obtener datos sobre el usuario actual del sistema. Utiliza JavaScript como plataforma de desarrollo, sin ningún framework.

### Generar la WebPart

* Cree un directorio en Windows para los archivos necesarios para la WebPart
* Abra una consola de PowerShell como administrador. Mueva la consola al directorio de la nueva WebPart:
  + *cd [ruta al directorio]*
* Ejecute el siguiente comando para generar la WebPart
  + *yo @microsoft/sharepoint*
* Acepte el nombre sugerido de la WebPart, o cree uno nuevo si es necesario
* Seleccione **SharePoint Online only (latest)**
* Seleccione **Use the current folder**
* Escriba **y** cuando salga la opción de **Do you want to allow the tenant…**
* Escriba **n** cuando salga la opción de **Will the components in the solution…**
* Seleccione **WebPart** en el nuevo dialogo
* Escriba **Example01WP** cuando salga el dialogo **What is your Web part name**
* Acepte la descripción sugerida
* Seleccione **No JavaScript framework** en el nuevo dialogo

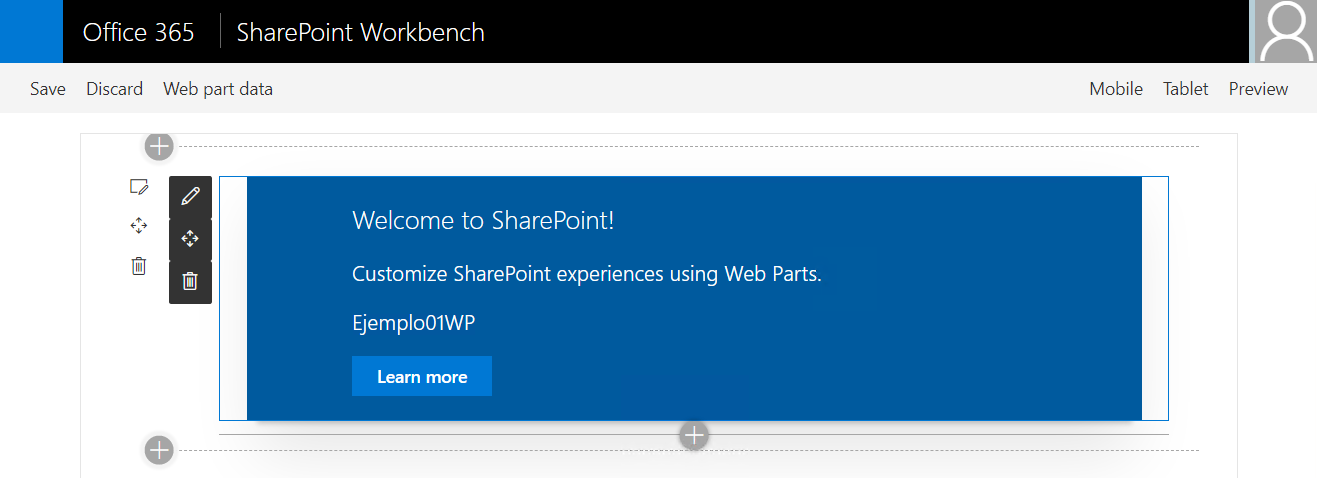


* La WebPart es generada. Dependiendo de la conexión de internet, puede tomar algún tiempo antes de terminar



### Comprobar la WebPart

* Cuando la WebPart este generada, ejecute el siguiente comando para iniciar Visual Studio Code con el código
  + *code .*
* Despliegue la WebPart al Workbench local usando el siguiente comando:
  + *gulp serve*



* Para detener el servidor de gulp, use:
  + *Ctrl+C*

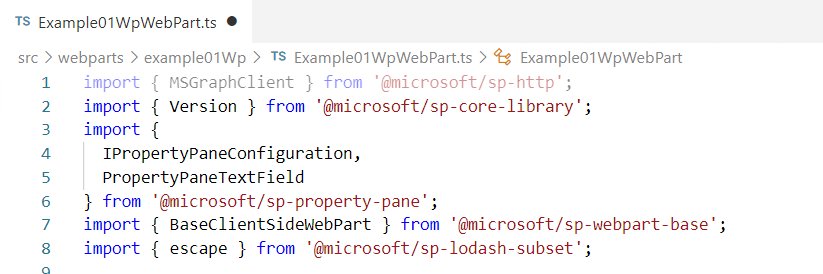
### El código de la WebPart

* En el código fuente de la WebPart (usando Visual Studio Code), abra el archivo **src\webparts\example01Wp\Example01WpWebPart.ts** y agregue la siguiente línea de código al principio de la página:

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

Esto implementa los métodos para consumir Microsoft Graph. Hay dos opciones: usando el objeto **AadHttpClient** o usando el objeto **MSGraphClient**

**AadHttpClient** es útil para consumir cualquier API de REST. Puede usarlo para consumir Microsoft Graph o cualquier otro API REST. **MSGraphClient** solo puede consumir Microsoft Graph e internamente usa **AadHttpClient**.



* Cambie todo el método **Render** original por el siguiente código:

public render(): void {

  this.context.msGraphClientFactory

    .getClient()

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

      // Get information about the current user from the Microsoft Graph

      client

        .api('/me')

        .get((error, response: any, rawResponse?: any) => {

          // The response

          console.log(JSON.stringify(response));

      this.domElement.innerHTML = `

        <div class="${ styles.example01Wp }">

          <div class="${ styles.container }">

            <div class="${ styles.row }">

              <div class="${ styles.column }">

                <p class="${ styles.description }">DisplayName - ${response.displayName}</p>

                <p class="${ styles.description }">Email - ${response.mail}</p>

                <p class="${ styles.description }">Phone Number - ${response.businessPhones[0]}</p>

              </div>

            </div>

          </div>

        </div>`;

      });

    });

  }



* La consulta a Graph por medio del **msGraphClientFactory** puede ser más compleja e incluir, por ejemplo, statments para filtrar y seleccionar:

this.context.msGraphClientFactory

.getClient()

.then((client: MSGraphClient) => {

client

.api("users")

.version("v1.0")

.select("displayName,mail,userPrincipalName")

.filter(`(givenName eq '${escape(this.state.searchFor)}') or (surname eq '${escape(this.state.searchFor)}') or (displayName eq '${escape(this.state.searchFor)}')`)

.get((err, res) => { ...

* La entidad que recibe el **response** contiene las siguientes propiedades (con las que se puede expandir el **innerHTML**, si es necesario)

{

"businessPhones": [

"businessPhones-value"

],

"displayName": "displayName-value",

"givenName": "givenName-value",

"jobTitle": "jobTitle-value",

"mail": "mail-value",

"mobilePhone": "mobilePhone-value",

"officeLocation": "officeLocation-value",

"preferredLanguage": "preferredLanguage-value",

"surname": "surname-value",

"userPrincipalName": "userPrincipalName-value",

"id": "id-value"

}

### Agregue los permisos de Graph

* Abra el archivo **config\package-solution.json** en Visual Studio Code. Agregue una coma al final de la propiedad **solution** y luego el siguiente código:

    "webApiPermissionRequests": [{

      "resource": "Microsoft Graph",

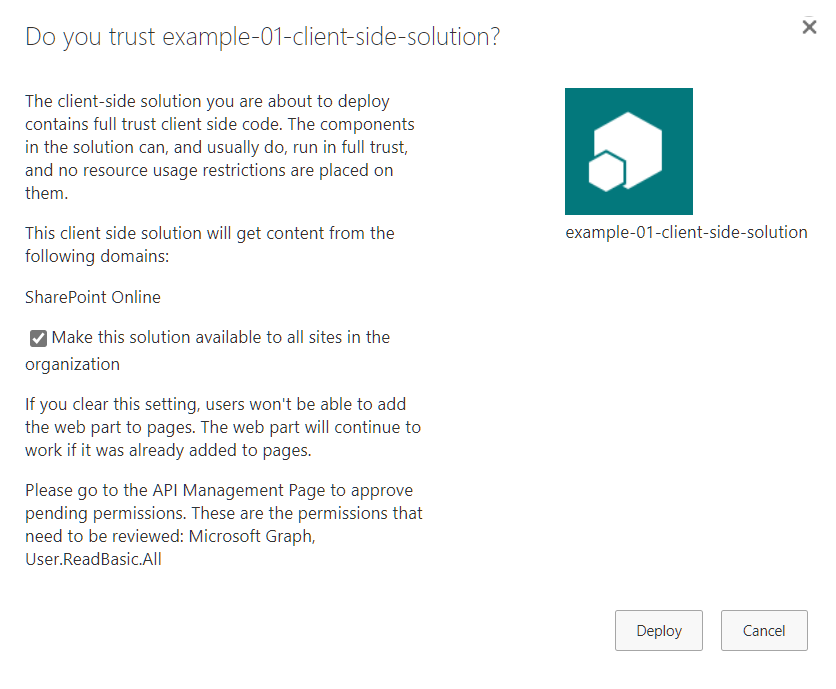
      "scope": "User.ReadBasic.All"

    }]



### Desplegar la WebPart en el tenant

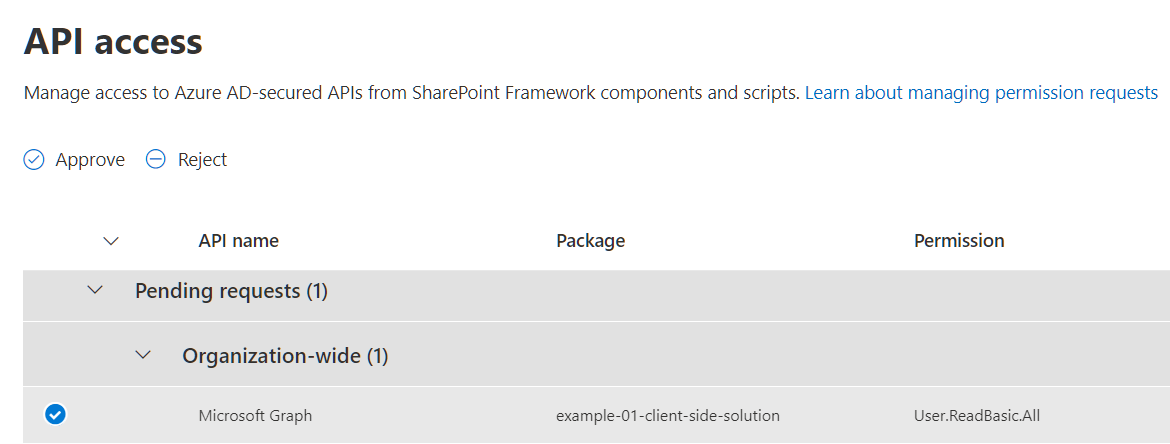
* Ejecute los dos siguientes comandos desde la consola de PowerShell para compilar y empacar la solución
  + *gulp bundle --ship*
  + *gulp package-solution --ship*
* Suba el archivo **…\Example\_01\sharepoint\solution\example-01.sppkg** al Catálogo de aplicaciones de SharePoint en el tenant de Office 365. El siguiente mensaje aparece:



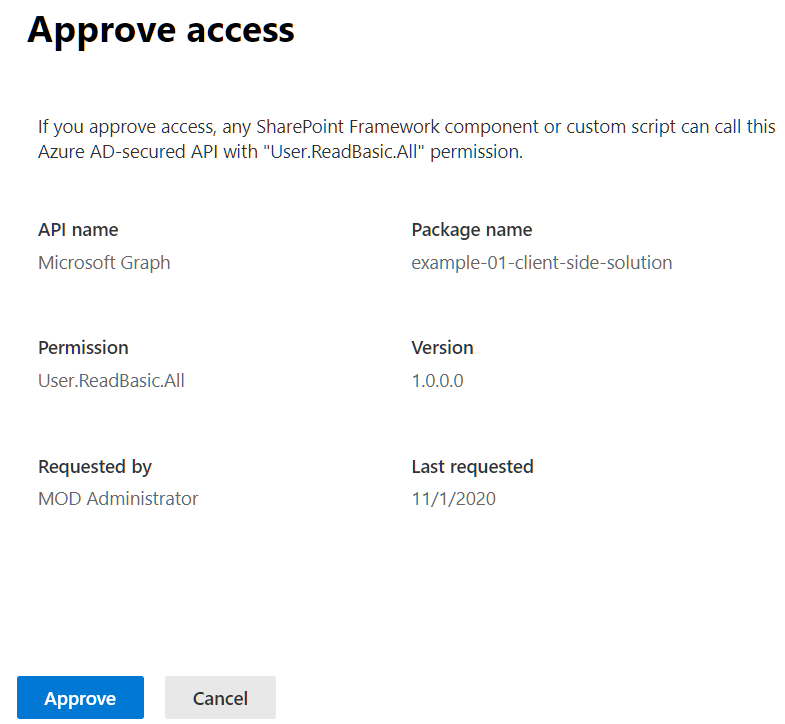
* Seleccione la casilla de **Make this solution…** y use el botón de **Deploy**

### Aprobar los permisos de la WebPart en Azure AD

* Para aceptar que la WebPart puede acceder a los datos de Graph, vaya a la administración central de SharePoint y en la sección de **Advanced** - **API Access** debe aparecer la WebPart que se ha instalado en el App Catalog:

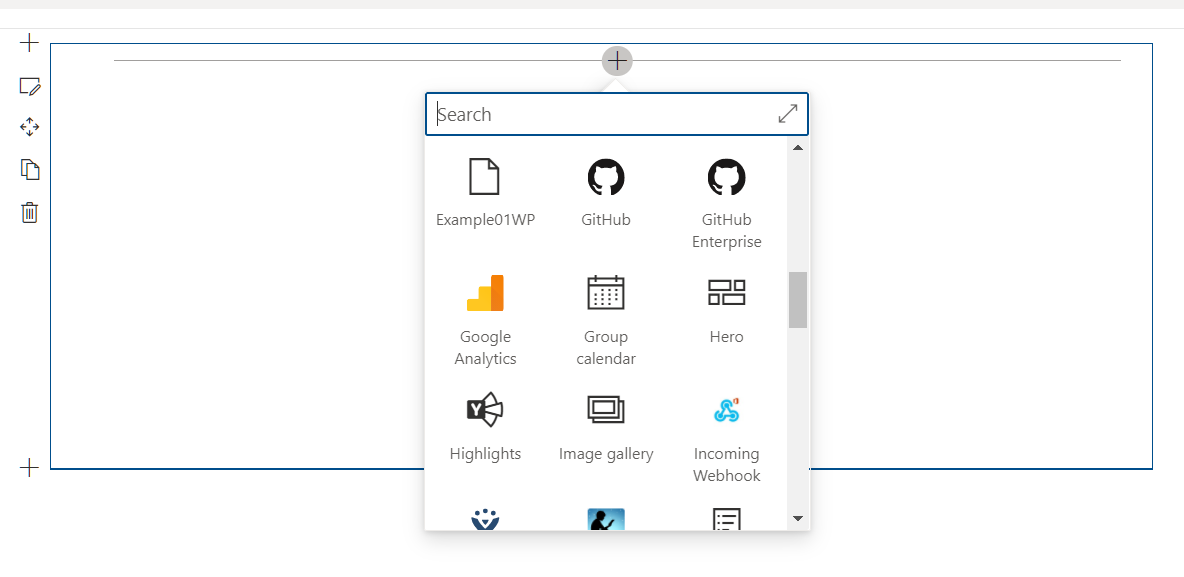


* Seleccione la entrada correspondiente y use el botón de **Approve** en la cabecera de la página
* Un dialogo de confirmación aparece en la página. Utilice el botón de **Approve** de nuevo

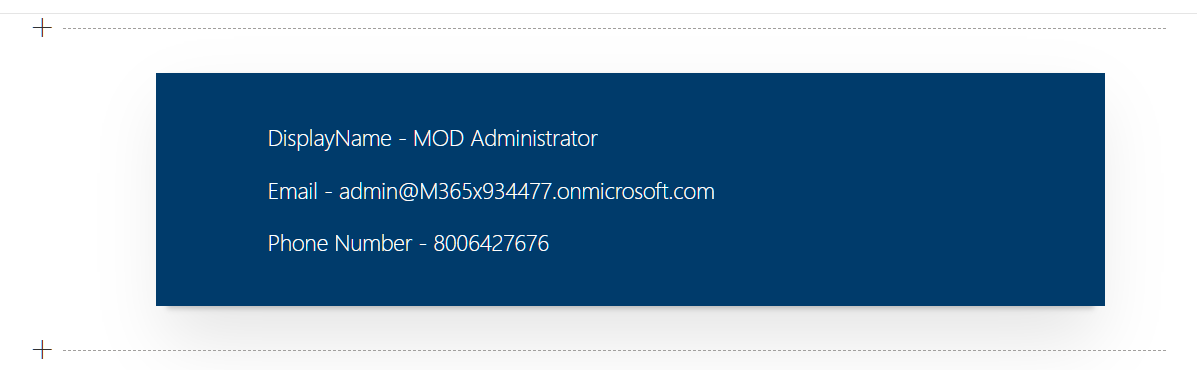


### Testear la WebPart en SharePoint

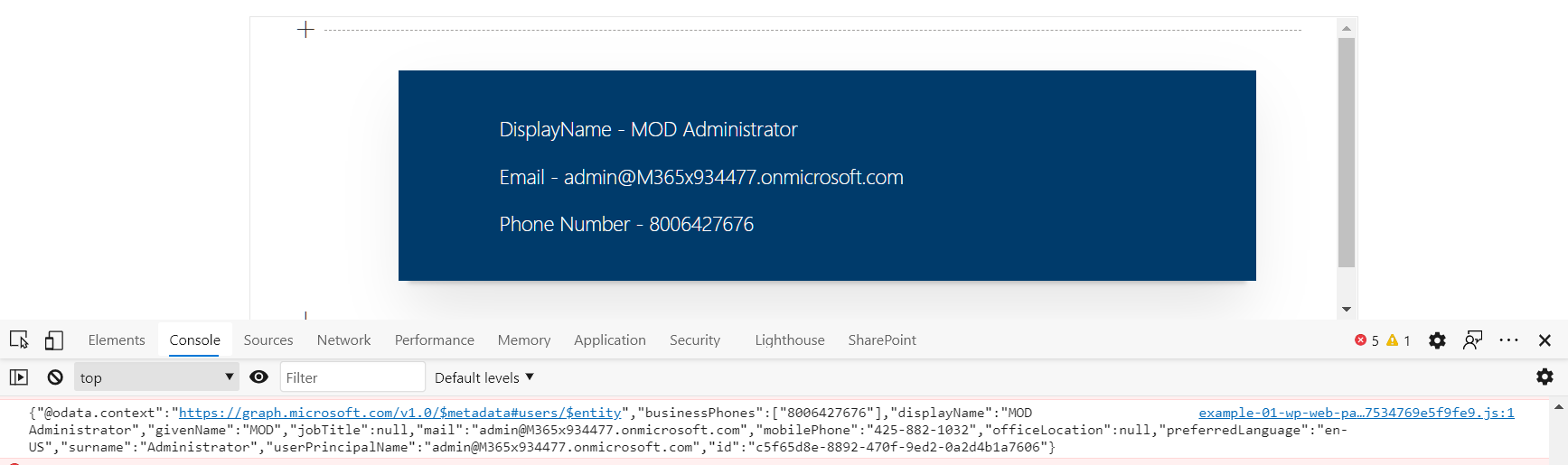
* Abra un sitio de SharePoint
* Abra el Workbench en el sitio agregando al final del URL del sitio:
  + */\_layouts/15/workbench.aspx*
* Instale la WebPart en el Workbench



* La WebPart debe renderizar los datos del usuario actual



* La línea de código "console.log(JSON.stringify(response));" en la página .ts imprime todo el **response** en la consola de JavaScript del navegador. Abra las herramientas de desarrollador del navegador, y en la pestaña **Console** podrá ver el resultado:



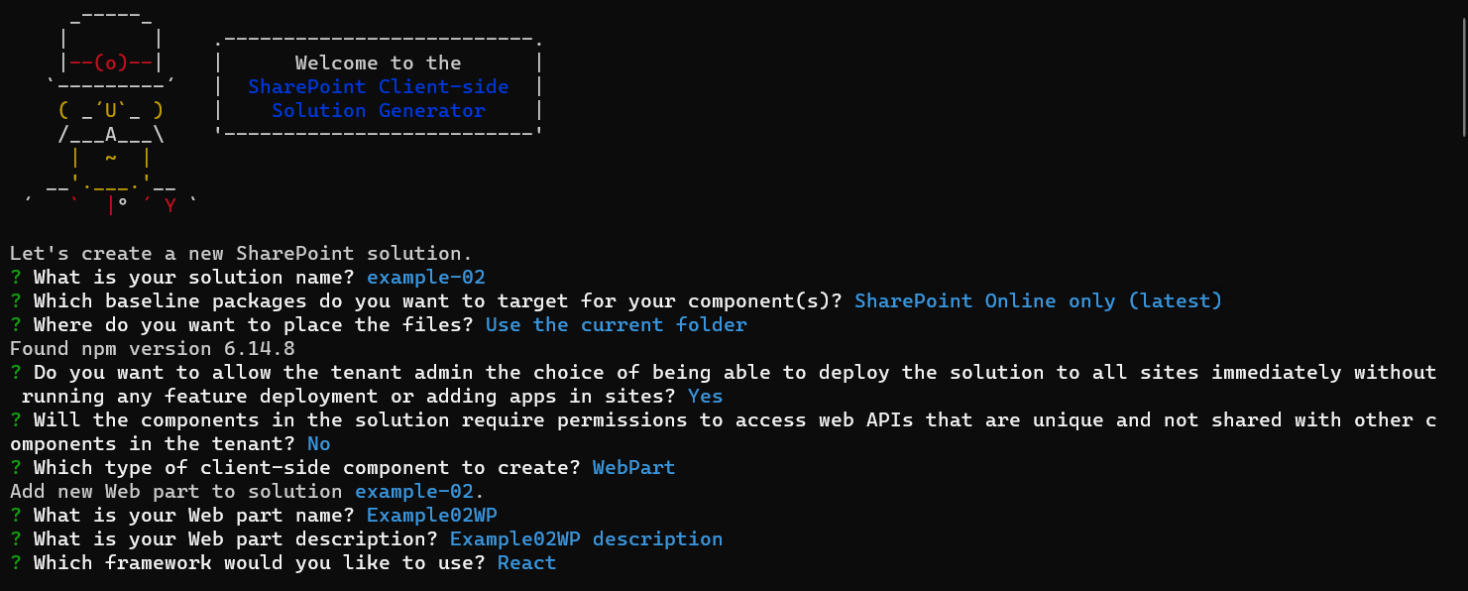
## Ejemplo 02 - WebPart SPFx (React)

Esta WebPart interactúa con los comandos de Teams con Microsoft Graph para mostrar los Teams y Channels que un usuario puede utilizar en un tenant de Office 365, y enviar mensajes a uno de los Channels. Utiliza React como JavaScript framework.

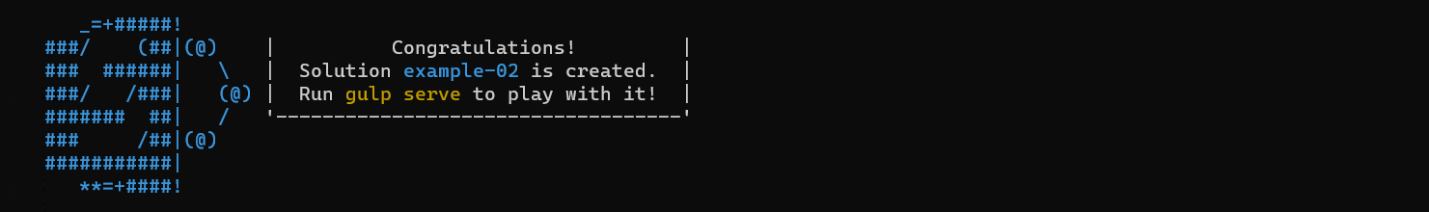
## 

### Generar la WebPart

* Cree un directorio en Windows para los archivos necesarios para la WebPart
* Abra una consola de PowerShell como administrador. Mueva la consola al directorio de la nueva WebPart:
  + *cd [ruta al directorio]*
* Ejecute el siguiente comando para generar la WebPart
  + *yo @microsoft/sharepoint*
* Acepte el nombre sugerido de la WebPart, o cree uno nuevo si es necesario
* Seleccione **SharePoint Online only (latest)**
* Seleccione **Use the current folder**
* Escriba **y** cuando salga la opción de **Do you want to allow the tenant…**
* Escriba **n** cuando salga la opción de **Will the components in the solution…**
* Seleccione **WebPart** en el nuevo dialogo
* Escriba **Example02WP** cuando salga el dialogo **What is your Web part name**
* Acepte la descripción sugerida
* Seleccione **React** en el nuevo dialogo

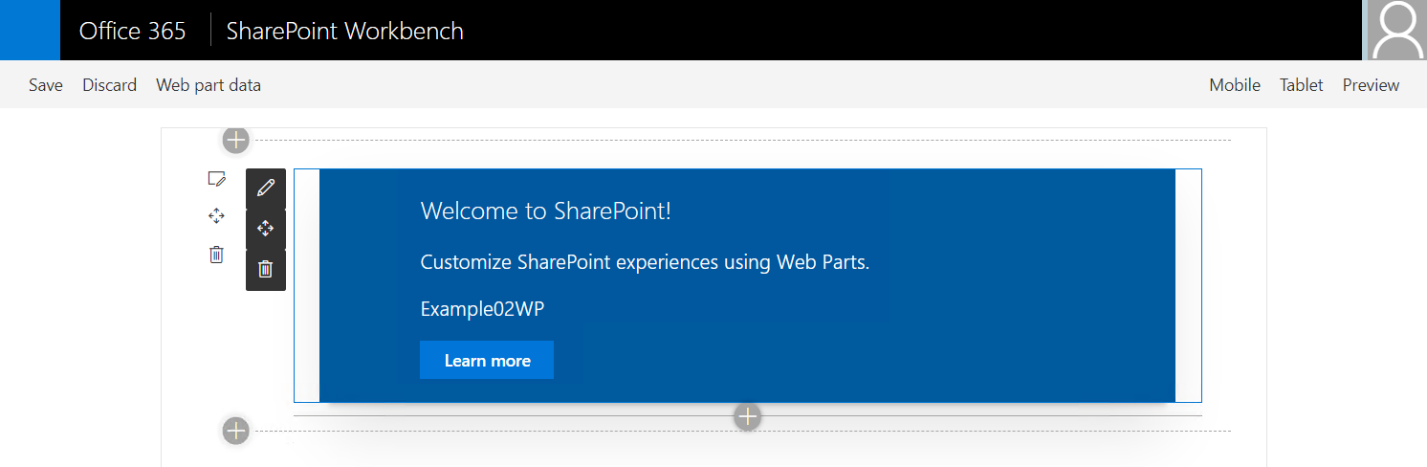


* La WebPart es generada. Dependiendo de la conexión de internet, puede tomar algún tiempo antes de terminar



### Comprobar la WebPart

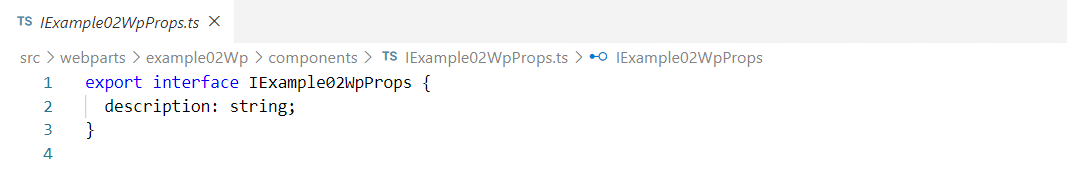
* Cuando la WebPart este generada, ejecute el siguiente comando para iniciar Visual Studio Code con el código
  + *code .*
* Despliegue la WebPart al Workbench local usando el siguiente comando:
  + *gulp serve*



* Para detener el servidor de gulp, use:
  + *Ctrl+C*

### El código de la WebPart

* En el código fuente de la WebPart (usando Visual Studio Code), abra el archivo **src\webparts\example02Wp\components/IExample02WpProps.ts** y agregue las siguientes líneas de código al principio de la página. Código original:



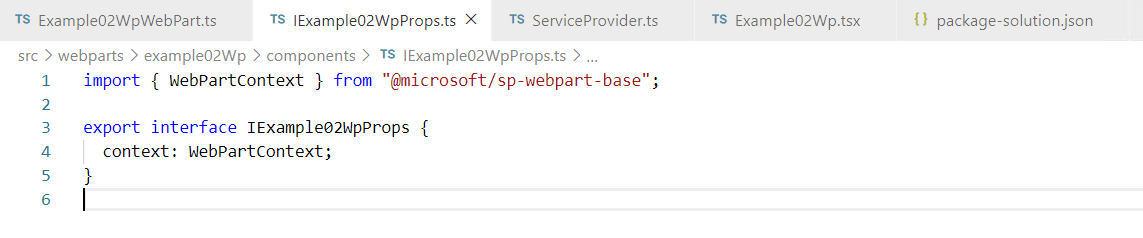
Modifique el Código para que quede:

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

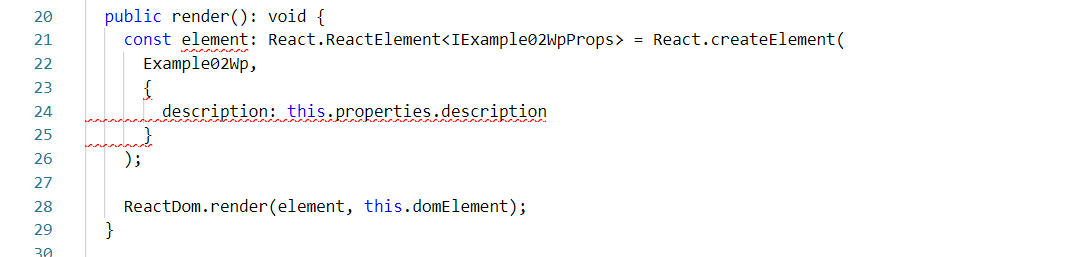
export interface IExample02WpProps {

  context: WebPartContext;

}

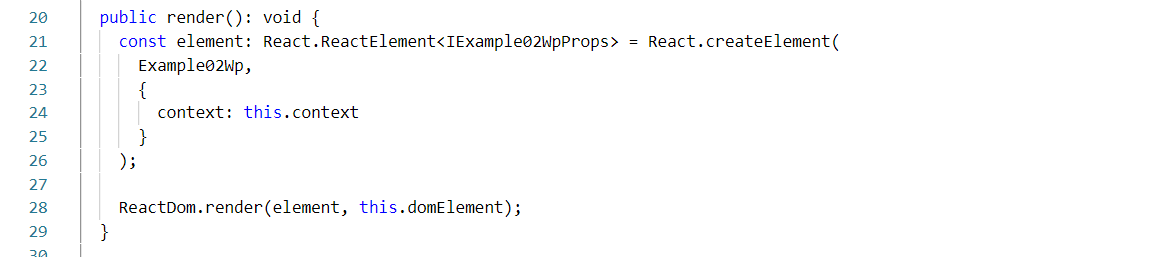


* Abra el archivo **src\webparts\example02Wp\Example02WpWebPart.ts**. Agregue el contexto creado en las propiedades en el método **Render** original. Código original:



Reemplace la propiedad **description** con el siguiente Código para el contexto. El error indicado en Visual Studio Code desaparece:

context: this.context



* Cree un directorio nuevo en la ruta **..src\shared\services\** y agregue un archivo de texto llamado **ServiceProvider.ts**. En el archivo agregue el siguiente código:

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

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

export class ServiceProvider {

  public graphClient: MSGraphClient;

  private spcontext: WebPartContext;

  public constructor(spcontext: WebPartContext) {

    this.spcontext = spcontext;

  }

  public getmyTeams = async (): Promise<[]> => {

    this.graphClient = await this.spcontext.msGraphClientFactory.getClient();

    let myTeams: [] = [];

    try {

      const teamsResponse = await this.graphClient.api('me/joinedTeams').version('v1.0').get();

      myTeams = teamsResponse.value as [];

    } catch (error) {

      console.log('Unable to get teams', error);

    }

    return myTeams;

  }

  public getChannel = async (teamID): Promise<[]> => {

    this.graphClient = await this.spcontext.msGraphClientFactory.getClient();

    let myTeams: [] = [];

    try {

      const teamsResponse = await this.graphClient.api('teams/' + teamID + '/channels').version('v1.0').get();

      myTeams = teamsResponse.value as [];

    } catch (error) {

      console.log('unable to get channels', error);

    }

    return myTeams;

  }

  public sendMessage = async (teamId, channelId, message): Promise<[]> => {

    this.graphClient = await this.spcontext.msGraphClientFactory.getClient();

    try {

      var content = {

        "body": {

          "content": message

        }

      };

      const messageResponse = await this.graphClient.api('/teams/' + teamId + '/channels/' + channelId + "/messages/")

        .version("beta").post(content);

      return messageResponse;

    } catch (error) {

      console.log('Unable to send message', error);

      return null;

    }

  }

}

En esta página de código se han creado tres métodos para hacer llamadas por medio de Graph y obtener una lista de los Teams a los que el usuario tiene acceso, una lista de los Channels de un Team, y para enviar un mensaje a un Channel.

* Abra el archivo **src\webparts\example02Wp\components\Example02Wp.tsx** y reemplace todo el código original con el siguiente fragmento:

|  |
| --- |
| import \* as React from 'react';  import styles from './Example02Wp.module.scss';  import { IExample02WpProps } from './IExample02WpProps';  import { escape } from '@microsoft/sp-lodash-subset';  import { ServiceProvider } from '../../../shared/services/ServiceProvider';    export interface IExample02WpState {    myteams: any[];    selectedTeam: any;    teamChannels: any;    selectedChannel: any;  }  export default class Example02Wp extends React.Component<IExample02WpProps, IExample02WpState> {    private serviceProvider;    private messageTextRef;      public constructor(props: IExample02WpProps, state: IExample02WpState) {      super(props);      this.serviceProvider = new ServiceProvider(this.props.context);      this.state = {        myteams: [],        selectedTeam: null,        selectedChannel: null,        teamChannels: []      };    }      public render(): React.ReactElement<IExample02WpProps> {      return (        <React.Fragment>           <div className={ styles.example02Wp }>          <h1>Using Graph API in SPFx WebParts</h1>            <div>            <button className={styles.buttons} onClick={() => this.GetmyTeams()}>Get My Teams</button>          </div>          { this.state.myteams.length>0 &&          <React.Fragment>           <h3>Your Teams</h3>           <h4>Select one Team</h4>          </React.Fragment>          }          {this.state.myteams.map(            (team: any, index: number) => (              <React.Fragment>                <input className={styles.radio} onClick={() => this.setState({ selectedTeam: team })} type="radio" id={team.id} name="myteams" value={team.id} />                <label >{team.displayName}</label><br />              </React.Fragment>            )          )          }            {this.state.selectedTeam &&          <React.Fragment>              <br></br>              <button  className={styles.buttons} onClick={() => this.getChannels()}>Get Channels</button>            </React.Fragment>          }          { this.state.teamChannels.length>0 &&          <React.Fragment>          <h3>Channels in the selected Team : {this.state.selectedTeam.displayName}</h3>          <h4>Select one Channel. Enter a message and click on 'Send Message'</h4>          </React.Fragment>          }            {this.state.teamChannels.map(            (channel: any, index: number) => (              <React.Fragment>                <input className={styles.radio} onClick={() => this.setState({ selectedChannel: channel })} type="radio" id={channel.id} name="teamchannels" value={channel.id} />                <label >{channel.displayName}</label><br />              </React.Fragment>            )          )          }            {this.state.selectedChannel &&            <React.Fragment>            <br></br>            <div>              <input className={styles.textbox}  ref={(elm) => { this.messageTextRef = elm; }} type="text" id="message" name="message" />              <br></br>              <br></br>              <button  className={styles.buttons} onClick={() => this.sendMesssage()}>Send Message</button>              </div>            </React.Fragment>          }          </div>        </React.Fragment>      );    }      private GetmyTeams() {      this.serviceProvider.        getmyTeams()        .then(          (result: any[]): void => {            console.log(result);            this.setState({ myteams: result });          }        )        .catch(error => {          console.log(error);        });    }    private getChannels() {      this.serviceProvider.        getChannel(this.state.selectedTeam.id)        .then(          (result: any[]): void => {            console.log(result);            this.setState({ teamChannels: result });          }        )        .catch(error => {          console.log(error);        });    }      private sendMesssage() {      this.serviceProvider.        sendMessage(this.state.selectedTeam.id, this.state.selectedChannel.id, this.messageTextRef.value)        .then(          (result: any[]): void => {            alert("message posted sucessfully");          }        )        .catch(error => {          console.log(error);        });    }  } |

Esta es la interfaz de usuario de la WebPart. Contiene los elementos para mostrar la información, y los botones y sus eventos para activar las llamadas a Graph.

* Para modificar los estilos de la interfaz, abra el archivo **src\webparts\controls\components\example02Wp.module.scss** y reemplace todo el codigo del archivo con el siguiente fragmento:

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

.example02Wp {

  label{

    font-size: 16px;

    line-height: 1.5;

  }

  .radio{

    width: 16px;

    height: 16px;

  }

  .textbox{

    height: 30px;

    width: 300px;

  }

  h4{

    font-style: italic;

    font-weight: normal;

  }

  .buttons{

    background-color: $ms-color-themePrimary;

    width: 200px;

    height: 40px;

    color: white;

    font-size: 18px;

   }

  }

* Para configurar los permisos que Graph necesita, abra el archivo config\package-solution.json y agregue una coma al final de la propiedad solution y luego copie el siguiente fragmento en la siguiente línea:

    "webApiPermissionRequests": [{

      "resource": "Microsoft Graph",

      "scope": "User.Read.All"

      }, {

        "resource": "Microsoft Graph",

        "scope": "User.ReadWrite.All"

    }, {

        "resource": "Microsoft Graph",

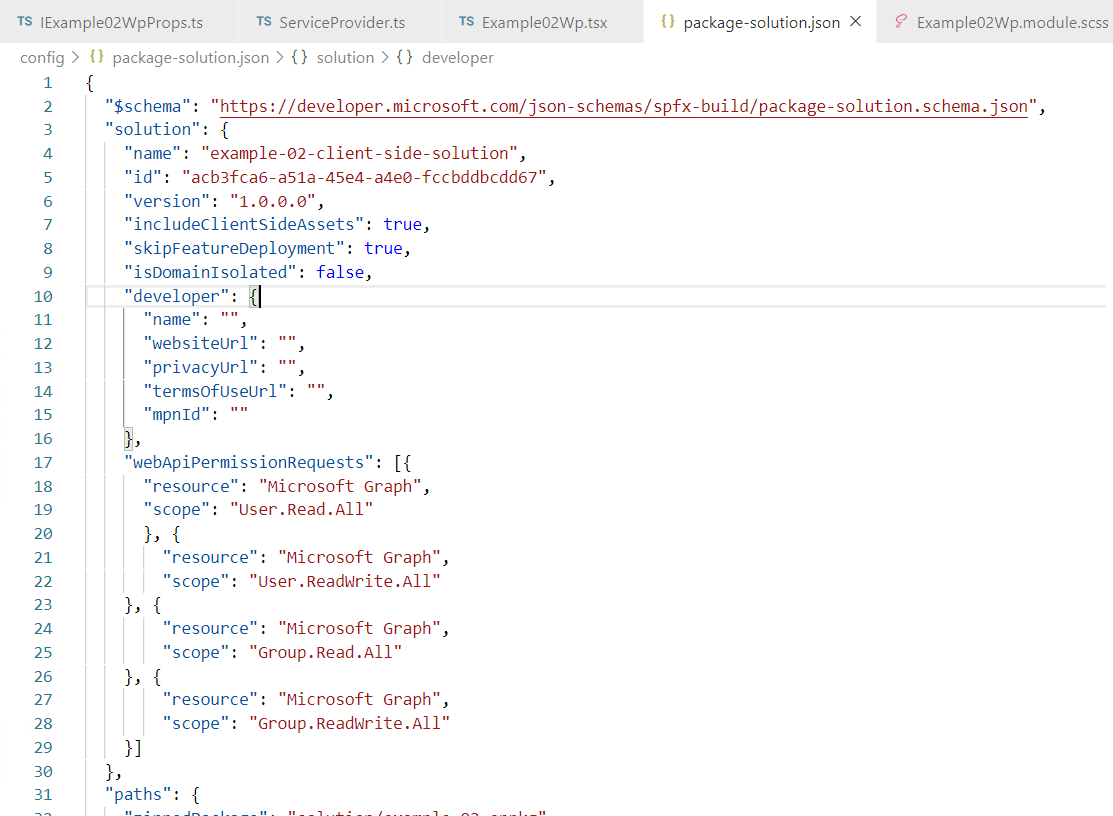
        "scope": "Group.Read.All"

    }, {

        "resource": "Microsoft Graph",

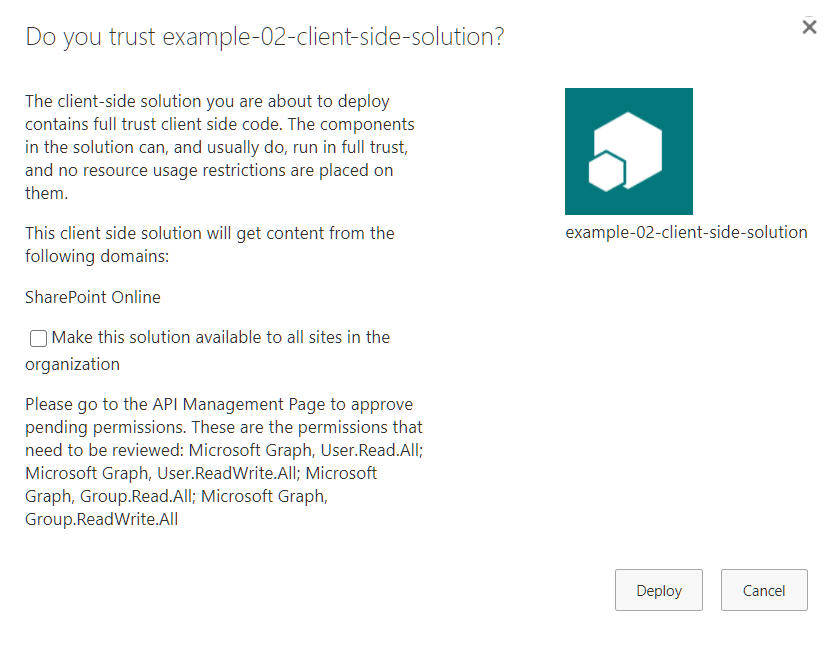
        "scope": "Group.ReadWrite.All"

    }]



### Desplegar la WebPart en el tenant

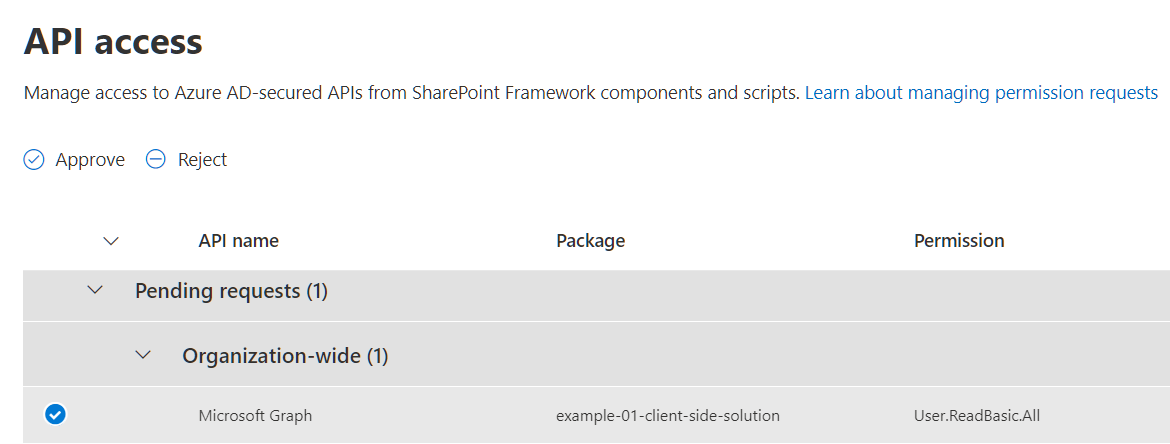
* Ejecute los dos siguientes comandos desde la consola de PowerShell para compilar y empacar la solución
  + *gulp bundle --ship*
  + *gulp package-solution --ship*
* Suba el archivo **…\Example\_02\sharepoint\solution\example-02.sppkg** al Catálogo de aplicaciones de SharePoint en el tenant de Office 365. El siguiente mensaje aparece:



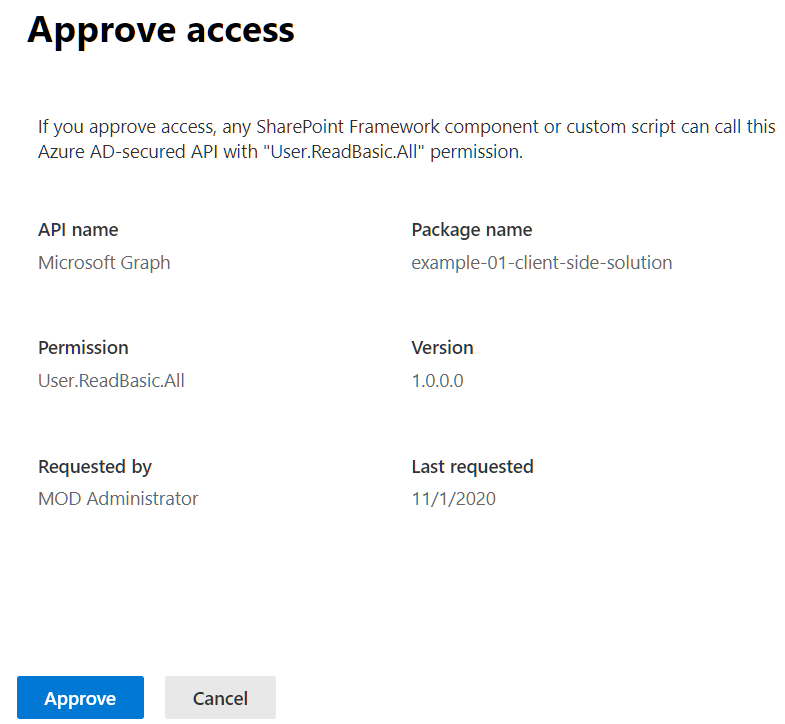
* Seleccione la casilla de **Make this solution…** y use el botón de **Deploy**

### Aprobar los permisos de la WebPart en Azure AD

* Para aceptar que la WebPart puede acceder a los datos de Graph, vaya a la administración central de SharePoint y en la sección de **Advanced** - **API Access** debe aparecer la WebPart que se ha instalado en el App Catalog, en este caso con cuatro entradas porque la aplicación requiere cuatro diferentes tipos de permisos:

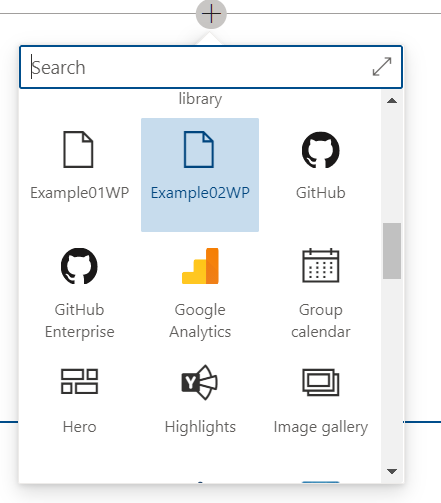


* Seleccione cada una de las entradas y use el botón de **Approve** en la cabecera de la página
* Un dialogo de confirmación aparece en la página. Utilice el botón de **Approve** de nuevo

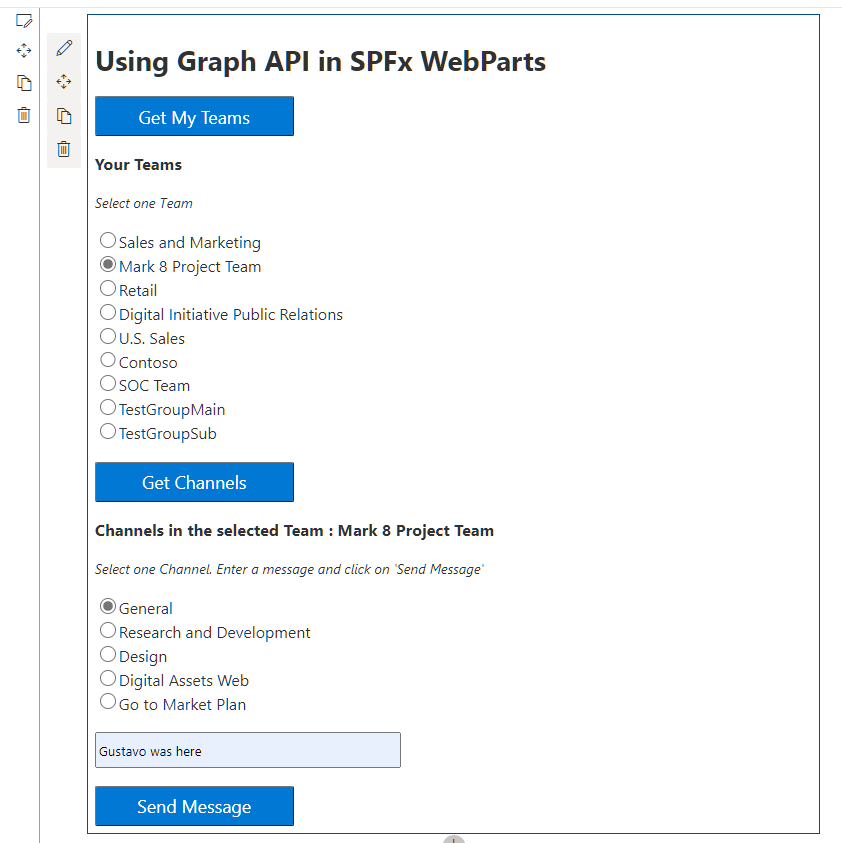


### Testear la WebPart en SharePoint

* Abra un sitio de SharePoint
* Abra el Workbench en el sitio agregando al final del URL del sitio:
  + */\_layouts/15/workbench.aspx*
* Instale la WebPart en el Workbench



* La WebPart debe renderizar los datos del usuario actual



* El mensaje debe aparecer en el Channel del Team seleccionado:

