# Exercise 5: Implementing authorization to consume an API

This exercise demonstrates how a JavaScript single-page application (SPA) can:

* Sign in personal accounts, as well as work and school accounts.
* Acquire an access token.
* Call the Microsoft Graph API or other APIs that require access tokens from the Microsoft identity platform endpoint.

The sample application used in this exercise enables a JavaScript SPA to query the Microsoft Graph API or a web API that accepts tokens from the Microsoft identity platform endpoint. In this scenario, after a user signs in, an access token is requested and added to HTTP requests through the authorization header. Token acquisition and renewal are handled by the Microsoft Authentication Library (MSAL).

Graph of sample project solution project implementation.

Graph of sample project solution project implementation.

## Task 1: Download sample project

1. Download the sample project for Node.js:
   1. To run the project by using a local web server, such as Node.js, download the project files to your **C:/Labfiles** directory. Visit <https://github.com/Azure-Samples/active-directory-javascript-graphapi-v2/releases> and select the latest release: **Source code (zip)**.
2. Navigate to where the download zip file is and unblock file.
   1. Right-select and select **Properties**.
   2. Select the **Unblock** checkbox and select **OK**.
3. Extract the zipped file.
4. Open the project in Visual Studio Code.
   1. Launch Visual Studio Code as Administrator and browse for the **active-directory-javascript-graphapi-v2-quickstart** root directory and open.

## Task 2: Register an application

1. On the left menu, navigate to **App registrations**.
2. Select **New Registration**.
3. From the **Register an application** pane, perform the following actions:
   1. In the **Name** text box, enter a meaningful application name that will be displayed to users of the app, for example **JavaScript-SPA-App**.
   2. Under **Supported account type**, select **Accounts in any organizational directory and personal Microsoft accounts**.

### For setting a redirect URL for the Node.js project

Follow these steps if you choose to use the Node.js project. For Node.js, you can set the web server port in the server.js file. This project uses port 30662, but you can use any other available port.

1. To set up a redirect URL in the application registration information, switch back to the **Application Registration** pane, and do either of the following:
   1. Set **http://localhost:30662/** as the **Redirect URL**.
   2. If you’re using a custom TCP port, use **http://localhost:[port]/** (where **[port]** is the custom TCP port number).
2. Select **Register**.
3. On the app **Overview** page, note the **Application (client) ID** value for later use.
4. This sample project requires the **Implicit grant flow** to be enabled. In the left pane of the registered application, select **Authentication**.
5. In **Advanced** settings, under **Implicit grant**, select the **ID tokens** and **Access tokens** check boxes. ID tokens and access tokens are required because this app must sign in users and call an API.
6. Select **Save**.

## Task 3: Permission and scope setup

### Update the App API permissions

1. In the left navigation, navigate to **API Permissions**.

* Navigate to API permissions
* Navigate to API permissions

1. Select **Add a permission**.
2. Next, select **Microsoft Graph** from **Microsoft APIs**.

* Select Microsoft Graph from Microsoft APIs.
* Select Microsoft Graph from Microsoft APIs.

1. Select **Delegated Permissions** and under **Select permissions**, search for **calendars** and select **Calendars.Read**.

* Select Calendars.Read
* Select Calendars.Read

1. Search for people and select **People.Read** permission and provide admin consent.
2. Select **Add permissions**.
   1. Wait for **Preparing for consent** to finish then select **Grant admin consent for Contoso**.
   2. From the Permissions requested dialog, select **Yes**.
   3. Your **JavaScript-SPA-App** app is now configured and authorized to use the Calendars.Read and People.Read permissions for Microsoft Graph.

### Update solution code

1. Navigate back to Visual Studio Code and open the **index.html** file.
2. Locate and select the following code:

* <div class="leftContainer">  
   <p id="WelcomeMessage">Welcome to the Microsoft Authentication Library For Javascript Quickstart</p>  
   <button id="SignIn" onclick="signIn()">Sign In</button>  
  </div>

1. Replace the selected code with the following, to add a new button to the application:

* <div class="leftContainer">  
   <p id="WelcomeMessage">Welcome to the Microsoft Authentication Library For Javascript Quickstart</p>  
   <button id="SignIn" onclick="signIn()">Sign In</button>  
  <button id="Share" onclick="acquireTokenPopupAndCallMSGraphInc()">Share</button>  
  </div>

1. Locate the object **requestIncObj** and change the scopes to **people.read**.

* var requestIncObj = {  
   scopes: ["people.read"]  
   };

1. Update the graph API URL to call the people object.

* var graphConfig = {  
   graphMeEndpoint: https://graph.microsoft.com/v1.0/me,  
  graphPeopleEndpoint: "https://graph.microsoft.com/v1.0/people"  
   };

1. Create a new endpoint for **people.read** and replace the method **acquireTokenPopupAndCallIncMSGraph** with the code below.

* function acquireTokenPopupAndCallIncMSGraph() {  
   //Always start with acquireTokenSilent to obtain a token in the signed in user from cache  
   myMSALObj.acquireTokenSilent(requestIncObj).then(function (tokenResponse) {  
   callMSGraph(graphConfig.graphMePeopleEndpoint, tokenResponse.accessToken, graphAPICallback);  
   }).catch(function (error) {  
   console.log(error);  
   // Upon acquireTokenSilent failure (due to consent or interaction or login required ONLY)  
   // Call acquireTokenPopup(popup window)  
   if (requiresInteraction(error.errorCode)) {  
   myMSALObj.acquireTokenPopup(requestIncObj).then(function (tokenResponse) {  
   callMSGraph(graphConfig.graphPeopleEndpoint, tokenResponse.accessToken, graphAPICallback);  
   }).catch(function (error) {  
   console.log(error);  
   });  
   }  
   });  
   }

1. Find the **var msalConfig** code block and replace the following:
   1. <Enter\_the\_Application\_Id\_here> is the **Application (client) ID** for the application you registered.
   2. <Enter\_the\_Tenant\_info\_here> is set to one of the following options:
      * If your application supports **Accounts in this organizational directory**, replace this value with the **Tenant ID** or **Tenant** **name** (for example, **contoso.microsoft.com**).
      * If your application supports **Accounts in any organizational directory**, replace this value with **organizations**.
      * If your application supports **Accounts in any organizational directory and personal Microsoft accounts**, replace this value with **common**. To restrict support to Personal Microsoft accounts only, replace this value with **consumers**.

## Task 4: Run the application

1. From **Visual Studio Code**, open **Terminal**. Type the following and hit ENTER:

* npm install

1. From the Terminal, type the following and hit ENTER.

* node server.js

1. From the browser, launch **http://localhost:30662**.

* Call localhost 30662 showing sign in screen.
* Call localhost 30662 showing sign in screen.

1. Select **Sign In**.
2. If the **Permissions requested** dialog opens, select **Accept**.

* Call localhost 30662 showing sign in screen.
* Call localhost 30662 showing sign in screen.

1. You should now be authenticated successfully.

* Logged in successfully to sample project.
* Logged in successfully to sample project.

1. Go back to the Visual Studio Code project. Stop the running project by selecting **Ctrl + C**.
2. Exit Visual Studio Code.

## Review

In this exercise, you implemented authorization and incremental consent using the Microsoft Identity.