

Authorize to Microsoft Graph from an Office Add-in

Article • 01/23/2024

Your add-in can get authorization to Microsoft Graph data by obtaining an access token to Microsoft Graph from the Microsoft identity platform. Use either the Authorization Code flow or the Implicit flow just as you would in other web applications but with one exception: The Microsoft identity platform doesn't allow its sign-in page to open in an iframe. When an Office Add-in is running in *Office on the web*, the task pane is an iframe. This means you'll need to open the sign-in page in a dialog box by using the Office dialog API. This affects how you use authentication and authorization helper libraries. For more information, see [Authentication with the Office dialog API](#).

ⓘ Note

If you're implementing SSO and plan to access Microsoft Graph, see [Authorize to Microsoft Graph with SSO](#).

For information about programming authentication using the Microsoft identity platform, see [Microsoft identity platform documentation](#). You'll find tutorials and guides in that documentation set, as well as links to relevant samples. Once again, you may need to adjust the code in the samples to run in the Office dialog box to account for the Office dialog box that runs in a separate process from the task pane.

After your code obtains the access token to Microsoft Graph, either it passes the access token from the dialog box to the task pane, or it stores the token in a database and signals the task pane that the token is available. (See [Authentication with the Office dialog API](#) for details.) Code in the task pane requests data from Microsoft Graph and includes the token in those requests. For more information about calling Microsoft Graph and the Microsoft Graph SDKs, see [Microsoft Graph documentation](#).

Recommended libraries and samples

We recommend that you use the following libraries when accessing Microsoft Graph.

- For add-ins using a server-side with a .NET-based framework such as .NET Core or ASP.NET, use [MSAL.NET](#) [↗](#).
- For add-ins using a NodeJS-based server-side, use [Passport Azure AD](#) [↗](#).
- For add-ins using the Implicit flow, use [msal.js](#) [↗](#).


For more information about recommended libraries for working with Microsoft Identity Platform (formerly AAD v.2.0), see [Microsoft identity platform authentication libraries](#).

The following samples get Microsoft Graph data from an Office Add-in.



- [Office Add-in Microsoft Graph ASP.NET](#) 
- [Outlook Add-in Microsoft Graph ASP.NET](#) 
- [Office Add-in Microsoft Graph React](#) 

Google Chrome 3rd party cookie support

Google Chrome is phasing out 3rd party cookies in 2024 and introducing a feature named Tracking Prevention. If Tracking Prevention is enabled in the Chrome browser, your add-in will not be able to use any 3rd party cookies. Your add-in may encounter issues when authenticating the user, such as multiple sign-on requests, or errors.

For improved authentication experiences, see [Using device state for an improved SSO experience on browsers with blocked third-party cookies](#) .

For more information about the Google Chrome rollout, see the following resources:

- [The next step toward phasing out third-party cookies in Chrome](#) .
- [The Privacy Sandbox Timeline for the Web](#) 

Enable automatic password saving in Microsoft Edge WebView2

Article • 04/11/2025

Most browsers can automatically save passwords on behalf of the user when they sign in. This helps users manage passwords in a secure environment. Microsoft Edge WebView2 also supports automatic password saving. When your add-in is loaded in Microsoft Office on Windows, Webview2 hosts your add-in. To enable automatic password saving, add HTML input controls for the username and password, as shown in the following HTML.

HTML

```
<div>
  <label for="username">Username:</label><br/>
  <input type="text" id="username" name="username" /><br/>

  <label for="password">Password:</label><br/>
  <input type="password" id="password" name="password" /><br/>

  <button id="btn" type="button">Sign in</button>
</div>
```

In the button click event handler for the sign-in button, call the authentication library of your choice to sign in the user. Once the sign-in is complete, redirect to a new web page. When WebView2 sees the redirect, and the username and password, it prompts the user to offer to automatically save the credentials. The following code shows how to handle the sign-in button click event.

JavaScript

```
async function btnSignIn() {
  // Get the username and password credentials entered by the user.
  const username = document.getElementById("username").value;
  const pwd = document.getElementById("password").value;

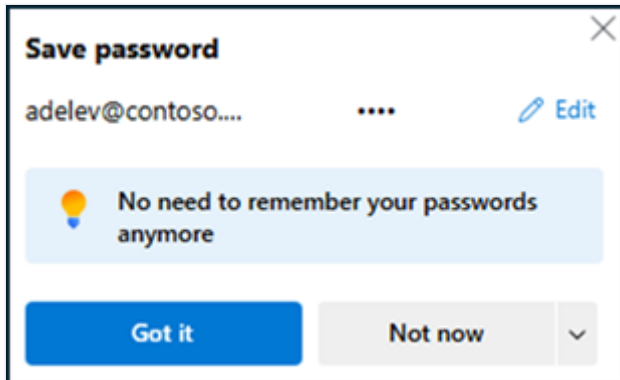
  try {
    // Sign in the user. This is a placeholder for the actual sign-in logic.
    await signInUser(username, pwd);

    // Redirect to a success page to trigger the password autosave.
    window.location.href = "/home.html";
  }
  catch (error) {
    console.error("Sign in failed: " + error);
    return;
  }
}
```

```
}  
}
```

How the user manages passwords

When the user enters a new password in your add-in, and your add-in redirects to a new web page, WebView2 asks the user if they want to save their username and password. The next time your add-in prompts for credentials, WebView2 automatically enters the user's account information.



Users remove saved passwords by deleting The WebView2 local cache folder at `%LOCALAPPDATA%\Microsoft\Office\16.0\Wef\webview2\`. If your add-in relies on automatically saving passwords, you should document this folder location so users can remove their passwords.

Related content

- [Microsoft Edge WebView2](#)
- [Browsers and webview controls used by Office Add-ins](#)