Authorization with non-Microsoft identity providers

Article • 03/21/2023

There are many popular identity providing services, in addition to the Microsoft identity platform, that you can use in your add-in. They give users, and applications such as your Office Add-in, access to the users' accounts in other applications.

The industry standard framework for enabling web application access to an online service is **OAuth 2.0**. In most situations, you don't need to know the details of how the framework works to use it in your add-in. Many libraries are available that simplify the details for you.

A fundamental idea of OAuth is that an application can be a security principal unto itself, just like a user or a group, with its own identity and set of permissions. In the most typical scenarios, when the user takes an action in the Office Add-in that requires the online service, the add-in sends the service a request for a specific set of permissions to the user's account. The service then prompts the user to grant the add-in those permissions. After the permissions are granted, the service sends the add-in a small encoded *access token*. The add-in can use the service by including the token in all its requests to the service's APIs. But the add-in can act only within the permissions that the user granted it. The token also expires after a specified time.

Several OAuth patterns, called *flows* or *grant types*, are designed for different scenarios. The following two patterns are the most commonly implemented.

- **Implicit flow**: Communication between the add-in and the online service is implemented with client-side JavaScript. This flow is commonly used in single-page applications (SPAs).
- **Authorization Code flow**: Communication is *server-to-server* between your add-in's web application and the online service. So, it is implemented with server-side code.

The purpose of an OAuth flow is to secure the identity and authorization of the application. In the Authorization Code flow, you're provided a *client secret* that needs to be kept hidden. An application that has no server-side backend, such as an SPA, has no way to protect the secret, so we recommend that you use the Implicit flow in SPAs.

You should be familiar with the pros and cons of the Implicit flow and the Authorization Code flow. For more information about these two flows, see Authorization Code and Implicit.

> **Note**
>
> You also have the option of using a middleman service to perform authorization and pass the access token to your add-in. For details about this scenario, see the **Middleman services** section later in this article.

Use the Implicit flow in Office Add-ins

The best way to find out if an online service supports the Implicit flow is to consult the service's documentation.

For information about libraries that support the Implicit flow, see the **Libraries** section later in this article.

Use the Authorization Code flow in Office Add-ins

Many libraries are available for implementing the Authorization Code flow in various languages and frameworks. For more information about some of these libraries, see the **Libraries** section later in this article.

Libraries

Libraries are available for many languages and platforms, for both the Implicit flow and the Authorization Code flow. Some libraries are general purpose, while others are for specific online services.

Facebook: Search Facebook for Developers for "library" or "sdk".

General OAuth 2.0: A page of links to libraries for over a dozen languages is maintained by the IETF OAuth Working Group at: OAuth Code. Note that some of these libraries are for implementing an OAuth compliant service. The libraries of interest to you as a an add-in developer are called *client* libraries on this page because your web server is a client of the OAuth compliant service.

Middleman services

Your add-in can use a middleman service such as OAuth.io or Auth0 to perform authorization. A middleman service may either provide access tokens for popular online

services or simplify the process of enabling social login for your add-in, or both. With very little code, your add-in can use either client-side script or server-side code to connect to the middleman service and it will send your add-in any required tokens for the online service. All of the authorization implementation code is in the middleman service.

We recommend that the UI for authentication/authorization in your add-in use our Dialog APIs to open a login page. See Authenticate with the Office dialog API for more information. When you open an Office dialog in this way, the dialog has a completely new and separate instance of the browser and JavaScript engine from the instance in the parent page (e.g., the add-in's task pane or FunctionFile). A token, and any other information that can be converted to a string, is passed back to the parent using an API called `messageParent`. The parent page can then use the token to make authorized calls to the resource. Because of this architecture, you must be careful how you use the APIs provided by a middleman service. Often the service will provide an API set in which your code creates some kind of context object which both gets a token and uses that token in making subsequent calls to the resource. Often the service has a single API method that makes the initial call *and* creates the context object. An object like this cannot be completely stringified, so it cannot be passed from the Office dialog to the parent page. Typically, the middleman service provides a second API set, at a lower level of abstraction, such as a REST API. This second set will have an API that gets a token from the service, and other APIs that pass the token to service when using it to get authorized access to the resource. You need to work with an API at this lower level of abstraction so that you can get the token in the Office dialog and then use `messageParent` to pass it to the parent page.

What is CORS?

CORS stands for Cross Origin Resource Sharing. For information about how to use CORS inside add-ins, see Addressing same-origin policy limitations in Office Add-ins.

See also

- Overview of authentication and authorization in Office Add-ins.