

✓ 100 XP ▶

# Introduction

3 minutes

The Microsoft Authentication Library (MSAL) enables developers to acquire tokens from the Microsoft identity platform in order to authenticate users and access secured web APIs.

After completing this module, you'll be able to:

- Explain the benefits of using MSAL and the application types and scenarios it supports
- Instantiate both public and confidential client apps from code
- Register an app with the Microsoft identity platform
- Create an app that retrieves a token by using the MSAL.NET library

## Next unit: Explore the Microsoft Authentication Library

[Continue >](#)

How are we doing? ☆ ☆ ☆ ☆ ☆



✓ 100 XP

▶

# Explore the Microsoft Authentication Library

3 minutes

The Microsoft Authentication Library (MSAL) can be used to provide secure access to Microsoft Graph, other Microsoft APIs, third-party web APIs, or your own web API. MSAL supports many different application architectures and platforms including .NET, JavaScript, Java, Python, Android, and iOS.

MSAL gives you many ways to get tokens, with a consistent API for a number of platforms. Using MSAL provides the following benefits:

- No need to directly use the OAuth libraries or code against the protocol in your application.
- Acquires tokens on behalf of a user or on behalf of an application (when applicable to the platform).
- Maintains a token cache and refreshes tokens for you when they are close to expire. You don't need to handle token expiration on your own.
- Helps you specify which audience you want your application to sign in.
- Helps you set up your application from configuration files.
- Helps you troubleshoot your app by exposing actionable exceptions, logging, and telemetry.

## Application types and scenarios

Using MSAL, a token can be acquired from a number of application types: web applications, web APIs, single-page apps (JavaScript), mobile and native applications, and daemons and server-side applications. MSAL currently supports the platforms and frameworks listed in the table below.

Library	Supported platforms and frameworks
<a href="#">MSAL for Android</a>	Android
<a href="#">MSAL Angular</a>	Single-page apps with Angular and Angular.js frameworks

Library	Supported platforms and frameworks
<a href="#">MSAL for iOS and macOS</a>	iOS and macOS
<a href="#">MSAL Go (Preview)</a>	Windows, macOS, Linux
<a href="#">MSAL Java</a>	Windows, macOS, Linux
<a href="#">MSAL.js</a>	JavaScript/TypeScript frameworks such as Vue.js, Ember.js, or Durandal.js
<a href="#">MSAL.NET</a>	.NET Framework, .NET Core, Xamarin Android, Xamarin iOS, Universal Windows Platform
<a href="#">MSAL Node</a>	Web apps with Express, desktop apps with Electron, Cross-platform console apps
<a href="#">MSAL Python</a>	Windows, macOS, Linux
<a href="#">MSAL React</a>	Single-page apps with React and React-based libraries (Next.js, Gatsby.js)

## Authentication flows

Below are some of the different authentication flows provided by Microsoft Authentication Library (MSAL). These flows can be used in a variety of different application scenarios.

Flow	Description
Authorization code	Native and web apps securely obtain tokens in the name of the user
Client credentials	Service applications run without user interaction
On-behalf-of	The application calls a service/web API, which in turns calls Microsoft Graph
Implicit	Used in browser-based applications

Flow	Description
Device code	Enables sign-in to a device by using another device that has a browser
Integrated Windows	Windows computers silently acquire an access token when they are domain joined
Interactive	Mobile and desktops applications call Microsoft Graph in the name of a user
Username/password	The application signs in a user by using their username and password

## Public client, and confidential client applications

Security tokens can be acquired by multiple types of applications. These applications tend to be separated into the following two categories. Each is used with different libraries and objects.

- **Public client applications:** Are apps that run on devices or desktop computers or in a web browser. They're not trusted to safely keep application secrets, so they only access web APIs on behalf of the user. (They support only public client flows.) Public clients can't hold configuration-time secrets, so they don't have client secrets.
- **Confidential client applications:** Are apps that run on servers (web apps, web API apps, or even service/daemon apps). They're considered difficult to access, and for that reason capable of keeping an application secret. Confidential clients can hold configuration-time secrets. Each instance of the client has a distinct configuration (including client ID and client secret).

## Next unit: Initialize client applications

[Continue >](#)

How are we doing? ☆ ☆ ☆ ☆ ☆

# Initialize client applications

3 minutes

With MSAL.NET 3.x, the recommended way to instantiate an application is by using the application builders: `PublicClientApplicationBuilder` and `ConfidentialClientApplicationBuilder`. They offer a powerful mechanism to configure the application either from the code, or from a configuration file, or even by mixing both approaches.

Before initializing an application, you first need to register it so that your app can be integrated with the Microsoft identity platform. After registration, you may need the following information (which can be found in the Azure portal):

- The client ID (a string representing a GUID)
- The identity provider URL (named the instance) and the sign-in audience for your application. These two parameters are collectively known as the authority.
- The tenant ID if you are writing a line of business application solely for your organization (also named single-tenant application).
- The application secret (client secret string) or certificate (of type `X509Certificate2`) if it's a confidential client app.
- For web apps, and sometimes for public client apps (in particular when your app needs to use a broker), you'll have also set the `redirectUri` where the identity provider will contact back your application with the security tokens.

## Initializing public and confidential client applications from code

The following code instantiates a public client application, signing-in users in the Microsoft Azure public cloud, with their work and school accounts, or their personal Microsoft accounts.


C#

📄 Copy

```
IPublicClientApplication app =  
PublicClientApplicationBuilder.Create(clientId).Build();
```

In the same way, the following code instantiates a confidential application (a Web app located at <https://myapp.azurewebsites.net>) handling tokens from users in the Microsoft Azure public cloud, with their work and school accounts, or their personal Microsoft accounts. The application is identified with the identity provider by sharing a client secret:

C#

 Copy

```
string redirectUri = "https://myapp.azurewebsites.net";
IConfidentialClientApplication app =
    ConfidentialClientApplicationBuilder.Create(clientId)
        .WithClientSecret(clientSecret)
        .WithRedirectUri(redirectUri)
        .Build();
```

## Builder modifiers

In the code snippets using application builders, a number of `.With` methods can be applied as modifiers (for example, `.WithAuthority` and `.WithRedirectUri`).

- `.WithAuthority` modifier: The `.WithAuthority` modifier sets the application default authority to an Azure Active Directory authority, with the possibility of choosing the Azure Cloud, the audience, the tenant (tenant ID or domain name), or providing directly the authority URL.

C#

 Copy

```
var clientApp = PublicClientApplicationBuilder.Create(client_id)
    .WithAuthority(AzureCloudInstance.AzurePublic, tenant_id)
    .Build();
```

- `.WithRedirectUri` modifier: The `.WithRedirectUri` modifier overrides the default redirect URI. In the case of public client applications, this will be useful for scenarios which require a broker.

C#

 Copy

```
var clientApp = PublicClientApplicationBuilder.Create(client_id)
    .WithAuthority(AzureCloudInstance.AzurePublic, tenant_id)
    .WithRedirectUri("http://localhost")
    .Build();
```

# Modifiers common to public and confidential client applications

The table below lists some of the modifiers you can set on a public, or client confidential client.

Modifier	Description
<code>.WithAuthority()</code>	Sets the application default authority to an Azure Active Directory authority, with the possibility of choosing the Azure Cloud, the audience, the tenant (tenant ID or domain name), or providing directly the authority URI.
<code>.WithTenantId(string tenantId)</code>	Overrides the tenant ID, or the tenant description.
<code>.WithClientId(string)</code>	Overrides the client ID.
<code>.WithRedirectUri(string redirectUri)</code>	Overrides the default redirect URI. In the case of public client applications, this will be useful for scenarios requiring a broker.
<code>.WithComponent(string)</code>	Sets the name of the library using MSAL.NET (for telemetry reasons).
<code>.WithDebugLoggingCallback()</code>	If called, the application will call <code>Debug.Write</code> simply enabling debugging traces.
<code>.WithLogging()</code>	If called, the application will call a callback with debugging traces.
<code>.WithTelemetry(TelemetryCallback telemetryCallback)</code>	Sets the delegate used to send telemetry.

## Modifiers specific to confidential client applications

The modifiers you can set on a confidential client application builder are:

Modifier	Description
----------	-------------



Modifier	Description
<code>.WithCertificate(X509Certificate2 certificate)</code>	Sets the certificate identifying the application with Azure Active Directory.
<code>.WithClientSecret(string clientSecret)</code>	Sets the client secret (app password) identifying the application with Azure Active Directory.

## Next unit: Exercise: Implement interactive authentication by using MSAL.NET

[Continue >](#)

How are we doing? ☆ ☆ ☆ ☆ ☆

# Exercise: Implement interactive authentication by using MSAL.NET

10 minutes

In this exercise you'll learn how to perform the following actions:

- Register an application with the Microsoft identity platform
- Use the `PublicClientApplicationBuilder` class in MSAL.NET
- Acquire a token interactively in a console application

## Prerequisites

- An **Azure account** with an active subscription. If you don't already have one, you can sign up for a free trial at <https://azure.com/free>
- **Visual Studio Code**: You can install Visual Studio Code from <https://code.visualstudio.com>

## Register a new application

1. Sign in to the portal: <https://portal.azure.com>
2. Search for and select **Azure Active Directory**.
3. Under **Manage**, select **App registrations > New registration**.
4. When the **Register an application** page appears, enter your application's registration information:

Field	Value
Name	az204appreg

Field	Value
Supported account types	Select Accounts in this organizational directory only
Redirect URI (optional)	Select Public client/native (mobile & desktop) and enter http://localhost in the box to the right.

Below are more details on the **Supported account types**.

## 5. Select Register.

Register an application - Microsoft

https://portal.azure.com

Microsoft Azure Search resources, services, and docs (G+)

meganb@contoso.com CONTOSO AD (DEV)

Home > Contoso AD (dev) >

### Register an application

\* Name

The user-facing display name for this application (this can be changed later).

Supported account types

Who can use this application or access this API?

- ☒ Accounts in this organizational directory only (Contoso AD (dev) only - Single tenant)
- ☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- ☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
- ☐ Personal Microsoft accounts only

[Help me choose...](#)

Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Web e.g. https://myapp.com/auth


[By proceeding, you agree to the Microsoft Platform Policies](#)

Register


Azure Active Directory assigns a unique application (client) ID to your app, and you're taken to your application's **Overview** page.

# Set up the console application


1. Launch Visual Studio Code and open a terminal by selecting **Terminal** and then **New Terminal**.
2. Create a folder for the project and change in to the folder.

ps	 Copy
md az204-auth cd az204-auth	

3. Create the .NET console app.

ps	 Copy
dotnet new console	

4. Open the *az204-auth* folder in VS Code.

ps	 Copy
code . -r	

## Build the console app

In this section you will add the necessary packages and code to the project.

## Add packages and using statements


1. Add the `Microsoft.Identity.Client` package to the project in a terminal in VS Code.

ps	 Copy
dotnet add package Microsoft.Identity.Client	

2. Open the *Program.cs* file and add `using` statements to include `Microsoft.Identity.Client` and to enable async operations.

	
--	---

C#

 Copy

```
using System.Threading.Tasks;  
using Microsoft.Identity.Client;
```

3. Change the Main method to enable async.

C#

 Copy

```
public static async Task Main(string[] args)
```

## Add code for the interactive authentication

1. We'll need two variables to hold the Application (client) and Directory (tenant) IDs. You can copy those values from the portal. Add the code below and replace the string values with the appropriate values from the portal.

C#

 Copy

```
private const string _clientId = "APPLICATION_CLIENT_ID";  
private const string _tenantId = "DIRECTORY_TENANT_ID";
```

2. Use the `PublicClientApplicationBuilder` class to build out the authorization context.

C#

 Copy

```
var app = PublicClientApplicationBuilder  
    .Create(_clientId)  
    .WithAuthority(AzureCloudInstance.AzurePublic, _tenantId)  
    .WithRedirectUri("http://localhost")  
    .Build();
```

Code	Description
<code>.Create</code>	Creates a <code>PublicClientApplicationBuilder</code> from a <code>clientId</code> .
<code>.WithAuthority</code>	Adds a known Authority corresponding to an ADFS server. In the code we're specifying the Public cloud, and using the tenant for the app we registered.

## Acquire a token

When you registered the *az204appreg* app it automatically generated an API permission `user.read` for Microsoft Graph. We'll use that permission to acquire a token.

1. Set the permission scope for the token request. Add the following code below the `PublicClientApplicationBuilder`.

C#

 Copy

```
string[] scopes = { "user.read" };
```

2. Add code to request the token and write the result out to the console.

C#

 Copy

```
AuthenticationResult result = await  
app.AcquireTokenInteractive(scopes).ExecuteAsync();  
  
Console.WriteLine($"Token:\t{result.AccessToken}");
```

## Review completed application

The contents of the *Program.cs* file should resemble the example below.

C#

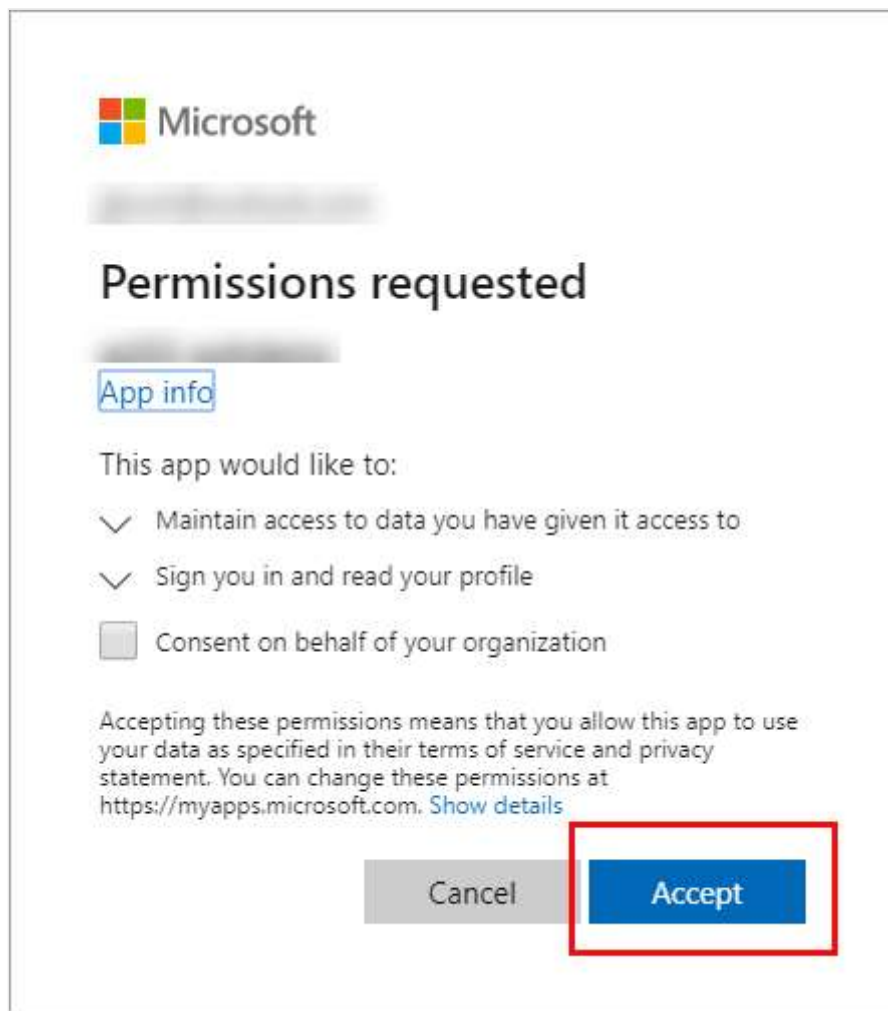
 Copy

```
using System;  
using System.Threading.Tasks;  
using Microsoft.Identity.Client;  
  
namespace az204_auth  
{  
    class Program  
    {  
        private const string _clientId = "APPLICATION_CLIENT_ID";  
        private const string _tenantId = "DIRECTORY_TENANT_ID";  
  
        public static async Task Main(string[] args)  
        {  
            var app = PublicClientApplicationBuilder  
                .Create(_clientId)  
                .WithAuthority(AzureCloudInstance.AzurePublic, _tenantId)  
                .WithRedirectUri("http://localhost")
```


```
        .Build();  
        string[] scopes = { "user.read" };  
        AuthenticationResult result = await  
app.AcquireTokenInteractive(scopes).ExecuteAsync();  
  
        Console.WriteLine($"Token:\t{result.AccessToken}");  
    }  
}
```

## Run the application

1. In the VS Code terminal run `dotnet build` to check for errors, then `dotnet run` to run the app.
2. The app will open the default browser prompting you to select the account you want to authenticate with. If there are multiple accounts listed select the one associated with the tenant used in the app.
3. If this is the first time you've authenticated to the registered app you will receive a **Permissions requested** notification asking you to approve the app to read data associated with your account. Select **Accept**.



4. You should see the results similar to the example below in the console.

	 Copy
Token: eyJ0eXAiOiJKV1QiLCJub25jZSI6I1VhU.....	

## Next unit: Knowledge check

[Continue >](#)

How are we doing? ☆ ☆ ☆ ☆ ☆



[< Previous](#)

Unit 5 of 6 ▾

[Next >](#)

✓ 200 XP ▶

# Knowledge check

3 minutes

## Check your knowledge

1. Which of the following MSAL libraries supports single-page web apps?

☐ MSAL Node

☒ MSAL.js

✓ That's correct. MSAL.js supports single-page applications.

☐ MSAL.NET

## Next unit: Summary

[Continue >](#)

How are we doing? ☆ ☆ ☆ ☆ ☆



[< Previous](#)

Unit 6 of 6

✓ 100 XP

# Summary

3 minutes

In this module, you learned how to:

- Explain the benefits of using MSAL and the application types and scenarios it supports
- Instantiate both public and confidential client apps from code
- Register an app with the Microsoft identity platform
- Create an app that retrieves a token by using the MSAL.NET

**Module complete:**

[Unlock achievement](#)

How are we doing? ☆ ☆ ☆ ☆ ☆

