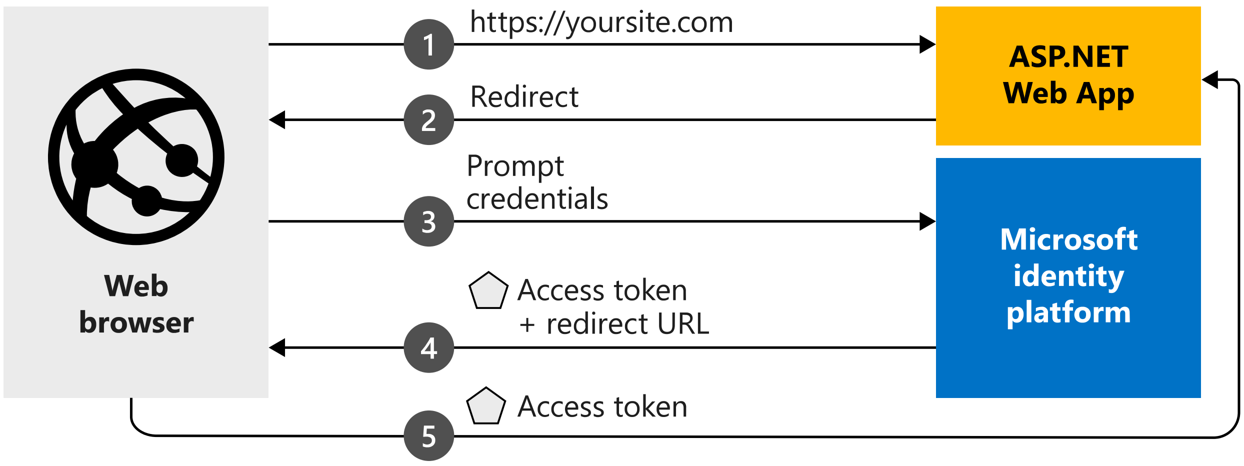
Quickstart: Add Microsoft identity platform sign-in to an ASP.NET web app

09/25/2020 • 6 minutes to read • +19

In this article

In this quickstart, you download and run a code sample that demonstrates how an ASP.NET web app can sign in users from any Azure Active Directory (Azure AD) organization.

The following diagram shows how the sample app works:



Prerequisites

- An Azure account with an active subscription. Create an account for free .
- Visual Studio 2019
- .NET Framework 4.7.2+

Register and download the app

You have two options to start building your application: automatic or manual configuration.

Automatic configuration

If you want to automatically configure your app and then download the code sample, follow these steps:

1. Go to the Azure portal page for app registration .
2. Enter a name for your application and select **Register**.
3. Follow the instructions to download and automatically configure your new application in one click.

Manual configuration

If you want to manually configure your application and code sample, use the following procedures.

Step 1: Register your application

1. Sign in to the Azure portal .
2. If you have access to multiple tenants, use the **Directory + subscription** filter on the top menu to select the tenant in which you want to register the application.
3. Search for and select **Azure Active Directory**.
4. Under **Manage**, select **App registrations** > **New registration**.
5. For **Name**, enter a name for your application. For example, enter **ASPNET-Quickstart**. Users of your app will see this name, and you can change it later.
6. Add **https://localhost:44368/** in **Redirect URI**, and select **Register**.
7. Under **Manage**, select **Authentication**.
8. In the **Implicit grant and hybrid flows** section, select **ID tokens**.
9. Select **Save**.

Step 2: Download the project

Download the Visual Studio 2019 solution

Tip

To avoid errors caused by path length limitations in Windows, we recommend extracting the archive or cloning the repository into a directory near the root of your drive.

Step 3: Run your Visual Studio project

1. Extract the .zip file to a local folder that's close to the root folder. For example, extract to *C:\Azure-Samples*.

   We recommend extracting the archive into a directory near the root of your drive to avoid errors caused by path length limitations on Windows.

2. Open the solution in Visual Studio (*AppModelv2-WebApp-OpenIDConnect-DotNet.sln*).

3. Depending on the version of Visual Studio, you might need to right-click the project **AppModelv2-WebApp-OpenIDConnect-DotNet** and then select **Restore NuGet packages**.

4. Open the Package Manager Console by selecting **View** > **Other Windows** > **Package Manager Console**. Then run `Update-Package Microsoft.CodeDom.Providers.DotNetCompilerPlatform -r`.

5. Edit *Web.config* and replace the parameters `ClientId`, `Tenant`, and `redirectUri` with:

   XML Copy

   ```xml
   <add key="ClientId" value="Enter_the_Application_Id_here" />
   <add key="Tenant" value="Enter_the_Tenant_Info_Here" />
   <add key="redirectUri" value="https://localhost:44368/" />
   ```

   In that code:

   - `Enter_the_Application_Id_here` is the application (client) ID of the app registration that you created earlier. Find the application (client) ID on the app's **Overview** page in **App registrations** in the Azure portal.
   - `Enter_the_Tenant_Info_Here` is one of the following options:
     - If your application supports **My organization only**, replace this value with the directory (tenant) ID or tenant name (for example, `contoso.onmicrosoft.com`). Find the directory (tenant) ID on the app's **Overview** page in **App registrations** in the Azure portal.
     - If your application supports **Accounts in any organizational directory**, replace this value with `organizations`.
     - If your application supports **All Microsoft account users**, replace this value with `common`.
   - `redirectUri` is the **Redirect URI** you entered earlier in **App registrations** in the Azure portal.

More information

This section gives an overview of the code required to sign in users. This overview can be useful to understand how the code works, what the main arguments are, and how to add sign-in to an existing ASP.NET application.

OWIN middleware NuGet packages

You can set up the authentication pipeline with cookie-based authentication by using OpenID Connect in ASP.NET with OWIN middleware packages. You can install these packages by running the following commands in Package Manager Console within Visual Studio:

PowerShell Copy

```powershell
Install-Package Microsoft.Owin.Security.OpenIdConnect
Install-Package Microsoft.Owin.Security.Cookies
Install-Package Microsoft.Owin.Host.SystemWeb
```

OWIN startup class

The OWIN middleware uses a *startup class* that runs when the hosting process starts. In this quickstart, the *startup.cs* file is in the root folder. The following code shows the parameters that this quickstart uses:

C# Copy

```csharp
public void Configuration(IAppBuilder app)
{

app.SetDefaultSignInAsAuthenticationType(CookieAuthenticationDefaults.Authen
ticationType);

    app.UseCookieAuthentication(new CookieAuthenticationOptions());
    app.UseOpenIdConnectAuthentication(
        new OpenIdConnectAuthenticationOptions
        {
            // Sets the client ID, authority, and redirect URI as obtained
from Web.config
            ClientId = clientId,
            Authority = authority,
            RedirectUri = redirectUri,
            // PostLogoutRedirectUri is the page that users will be redi-
rected to after sign-out. In this case, it's using the home page
            PostLogoutRedirectUri = redirectUri,
            Scope = OpenIdConnectScope.OpenIdProfile,
            // ResponseType is set to request the code id_token, which con-
tains basic information about the signed-in user
            ResponseType = OpenIdConnectResponseType.CodeIdToken,
            // ValidateIssuer set to false to allow personal and work ac-
counts from any organization to sign in to your application
            // To only allow users from a single organization, set Vali-
dateIssuer to true and the 'tenant' setting in Web.config to the tenant name
            // To allow users from only a list of specific organizations,
set ValidateIssuer to true and use the ValidIssuers parameter
            TokenValidationParameters = new TokenValidationParameters()
            {
                ValidateIssuer = false // Simplification (see note below)
            },
            // OpenIdConnectAuthenticationNotifications configures OWIN to
send notification of failed authentications to the OnAuthenticationFailed
method
            Notifications = new OpenIdConnectAuthenticationNotifications
            {
                AuthenticationFailed = OnAuthenticationFailed
            }
        }
    );
}
```

Where	Description
`ClientId`	The application ID from the application registered in the Azure portal.
`Authority`	The security token service (STS) endpoint for the user to authenticate. It's usually `https://login.microsoftonline.com/{tenant}/v2.0` for the public cloud. In that URL, *{tenant}* is the name of your tenant, your tenant ID, or `common` for a reference to the common endpoint. (The common endpoint is used for multitenant applications.)
`RedirectUri`	The URL where users are sent after authentication against the Microsoft identity platform.
`PostLogoutRedirectUri`	The URL where users are sent after signing off.
`Scope`	The list of scopes being requested, separated by spaces.
`ResponseType`	The request that the response from authentication contains an authorization code and an ID token.

Where	Description
`TokenValidationParameters`	A list of parameters for token validation. In this case, `ValidateIssuer` is set to `false` to indicate that it can accept sign-ins from any personal, work, or school account type.
`Notifications`	A list of delegates that can be run on `OpenIdConnect` messages.

Note

Setting `ValidateIssuer = false` is a simplification for this quickstart. In real applications, validate the issuer. See the samples to understand how to do that.

Authentication challenge

You can force a user to sign in by requesting an authentication challenge in your controller:

C# Copy

```
public void SignIn()
{
    if (!Request.IsAuthenticated)
    {
        HttpContext.GetOwinContext().Authentication.Challenge(
            new AuthenticationProperties{ RedirectUri = "/" },
            OpenIdConnectAuthenticationDefaults.AuthenticationType);
    }
}
```

Tip

Requesting an authentication challenge by using this method is optional. You'd normally use it when you want a view to be accessible from both authenticated and unauthenticated users. Alternatively, you can protect controllers by using the method described in the next section.

Attribute for protecting a controller or a controller actions

You can protect a controller or controller actions by using the `[Authorize]` attribute. This attribute restricts access to the controller or actions by allowing only authenticated users to access the actions in the controller. An authentication challenge will then happen automatically when an unauthenticated user tries to access one of the actions or controllers decorated by the `[Authorize]` attribute.

Help and support

If you need help, want to report an issue, or want to learn about your support options, see Help and support for developers.

Next steps

For a complete step-by-step guide on building applications and new features, including a full explanation of this quickstart, try out the ASP.NET tutorial.

Add sign-in to an ASP.NET web app

Is this page helpful?

Yes No