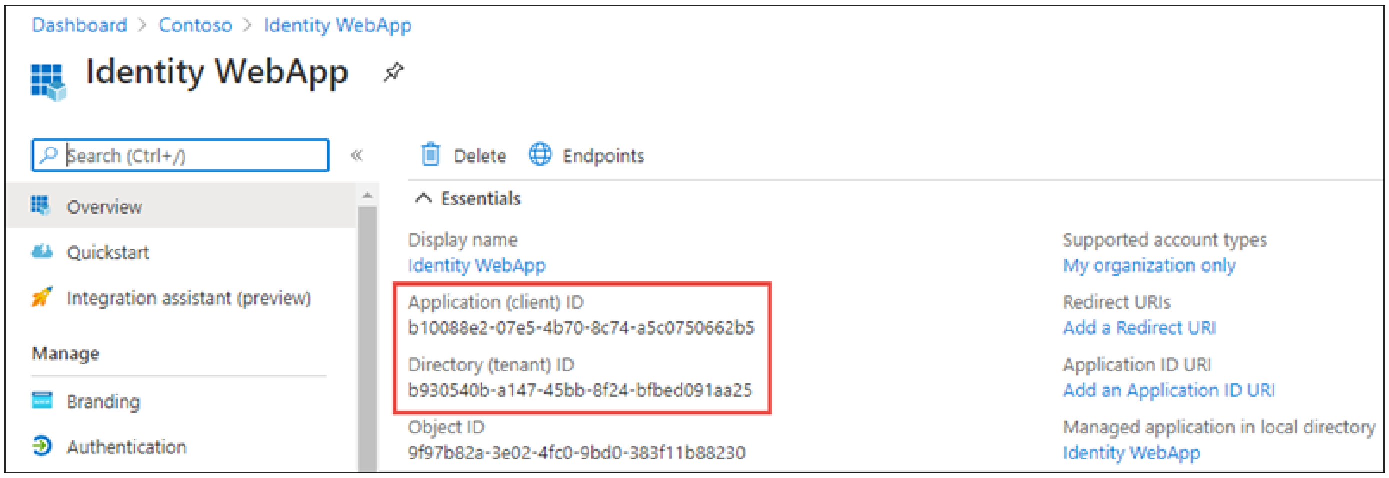


Display name
Identity WebApp

Application (client) ID
b10088e2-07e5-4b70-8c74-a5c0750662b5

Directory (tenant) ID
b930540b-a147-45bb-8f24-bfbed091aa25

Object ID
9f97b82a-3e02-4fc0-9bd0-383f11b88230

Supported account types
My organization only

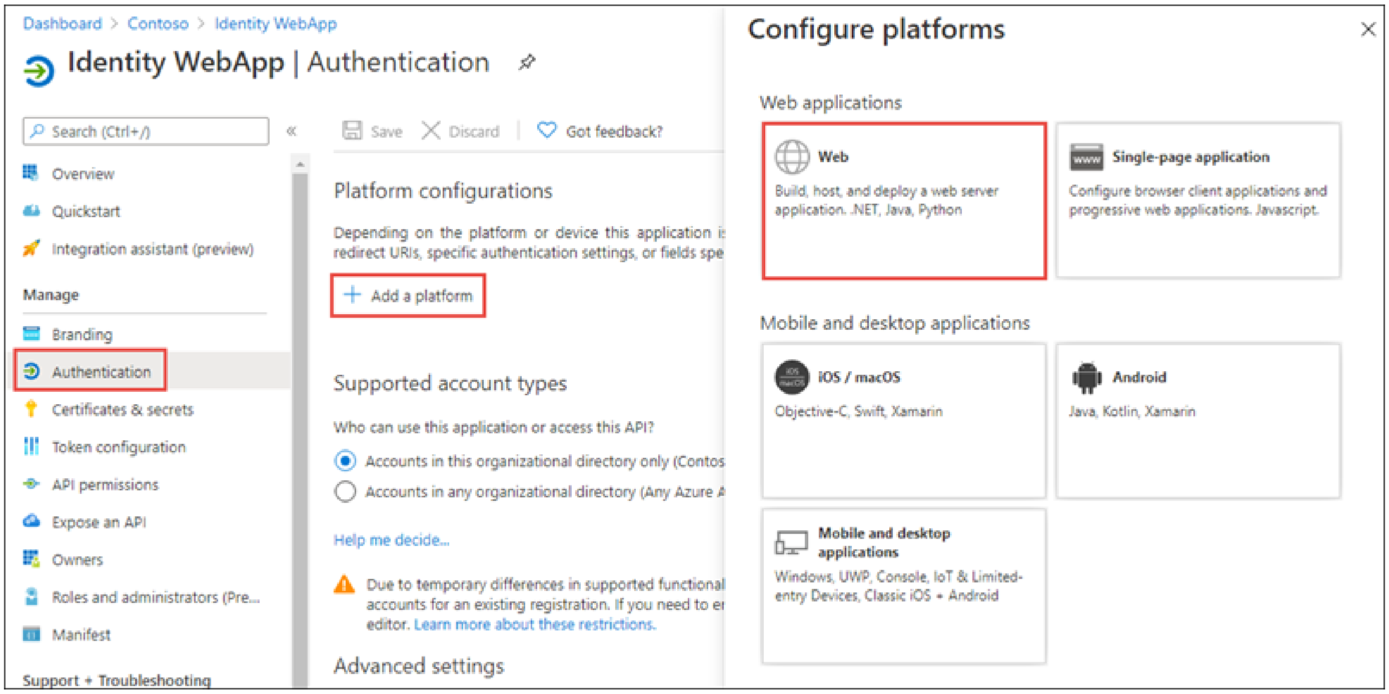
Redirect URIs
Add a Redirect URI

Application ID URI
Add an Application ID URI

Managed application in local directory
Identity WebApp

Select **Manage** > **Authentication** in the left-hand navigation.

On the **Authentication** page, select **Add a platform**. When the **Configure platforms** panel appears, select **Web**.



Platform configurations

Depending on the platform or device this application i redirect URIs, specific authentication settings, or fields spe

Supported account types

Who can use this application or access this API?

● Accounts in this organizational directory only (Contos

○ Accounts in any organizational directory (Any Azure A

Due to temporary differences in supported functional accounts for an existing registration. If you need to e editor. Learn more about these restrictions.

Advanced settings

Web applications

Web
Build, host, and deploy a web server application. .NET, Java, Python

Single-page application
Configure browser client applications and progressive web applications. Javascript.

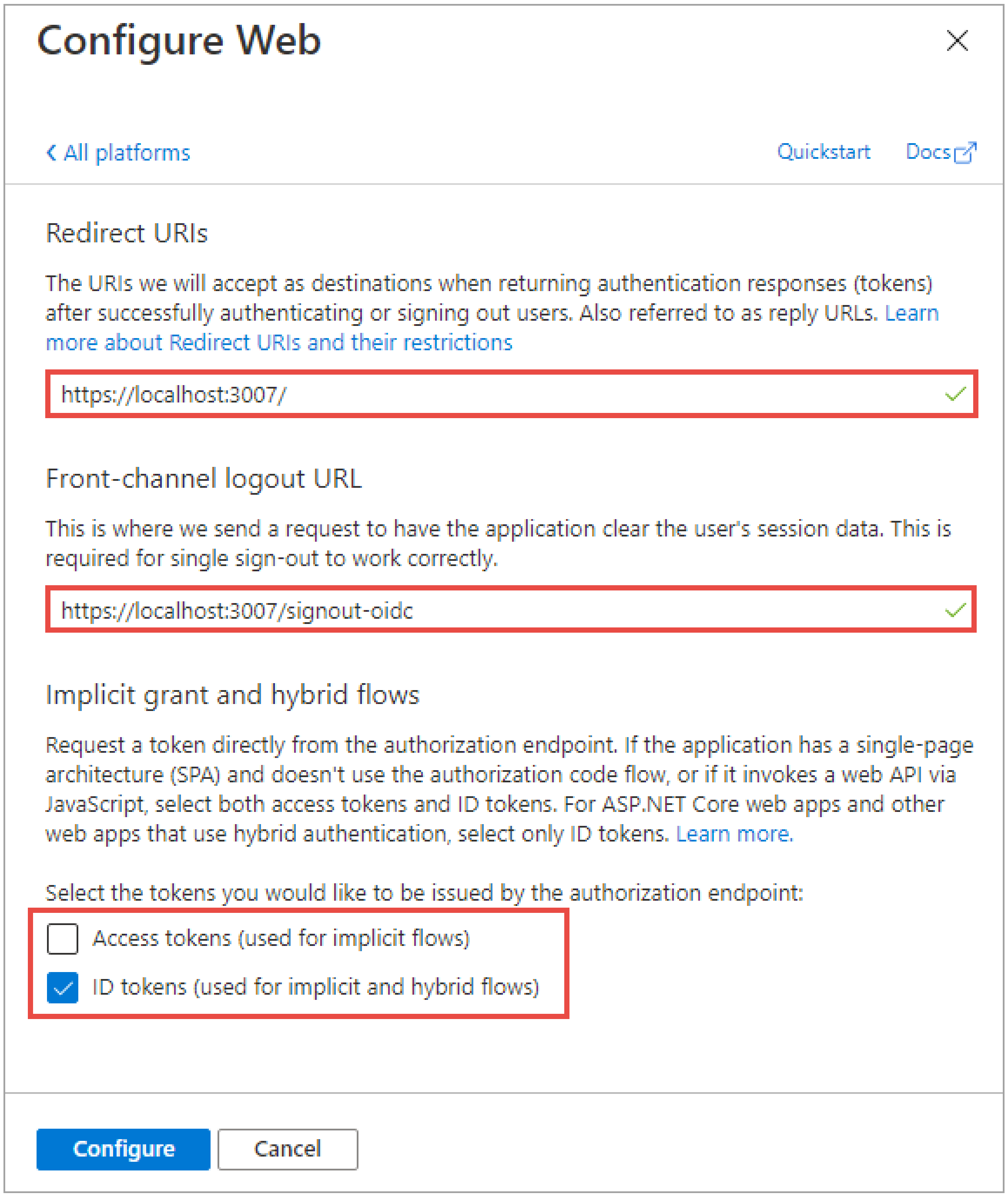
Mobile and desktop applications

iOS / macOS
Objective-C, Swift, Xamarin

Android
Java, Kotlin, Xamarin

Mobile and desktop applications
Windows, UWP, Console, IoT & Limited-entry Devices, Classic iOS + Android

In the **Configure Web** panel, add **https://localhost:3007** under **Redirect URIs**, add **https://localhost:3007/signout-oidc** under **Logout URL**, select **ID tokens (used for implicit and hybrid flows)** under **Implicit grant and hybrid flows**, and select **Configure**.



Configure Web

< All platforms   Quickstart   Docs

Redirect URIs

The URIs we will accept as destinations when returning authentication responses (tokens) after successfully authenticating or signing out users. Also referred to as reply URLs. Learn more about Redirect URIs and their restrictions

https://localhost:3007/ ✓

Front-channel logout URL

This is where we send a request to have the application clear the user's session data. This is required for single sign-out to work correctly.

https://localhost:3007/signout-oidc ✓

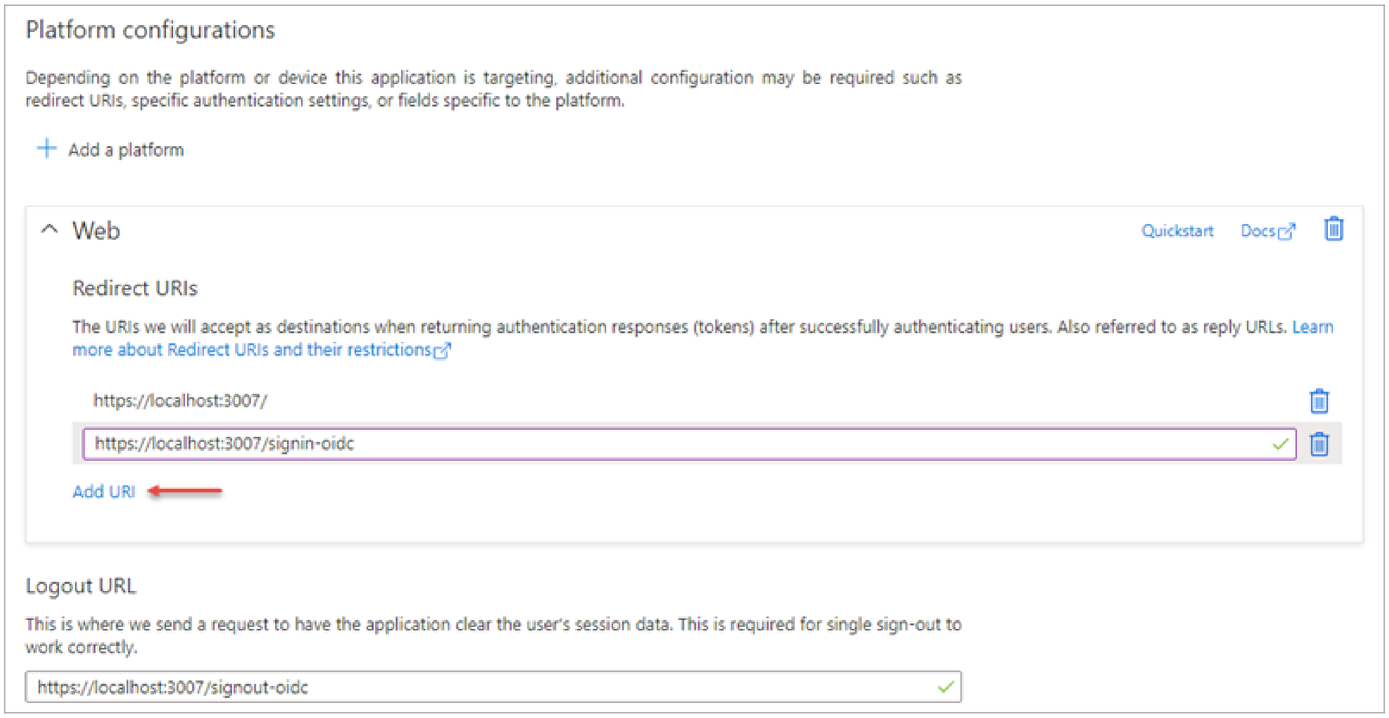
Implicit grant and hybrid flows

Request a token directly from the authorization endpoint. If the application has a single-page architecture (SPA) and doesn't use the authorization code flow, or if it invokes a web API via JavaScript, select both access tokens and ID tokens. For ASP.NET Core web apps and other web apps that use hybrid authentication, select only ID tokens. Learn more.

Select the tokens you would like to be issued by the authorization endpoint:

- [ ] Access tokens (used for implicit flows)
- [x] ID tokens (used for implicit and hybrid flows)

When the **Authentication** page refreshes, select **Add URI**, add **https://localhost:3007/signin-oidc**, and select **Save** in the top menu to save your changes.



Platform configurations

Depending on the platform or device this application is targeting, additional configuration may be required such as redirect URIs, specific authentication settings, or fields specific to the platform.

Redirect URIs

The URIs we will accept as destinations when returning authentication responses (tokens) after successfully authenticating users. Also referred to as reply URLs. Learn more about Redirect URIs and their restrictions

https://localhost:3007/

https://localhost:3007/signin-oidc

Logout URL

This is where we send a request to have the application clear the user's session data. This is required for single sign-out to work correctly.

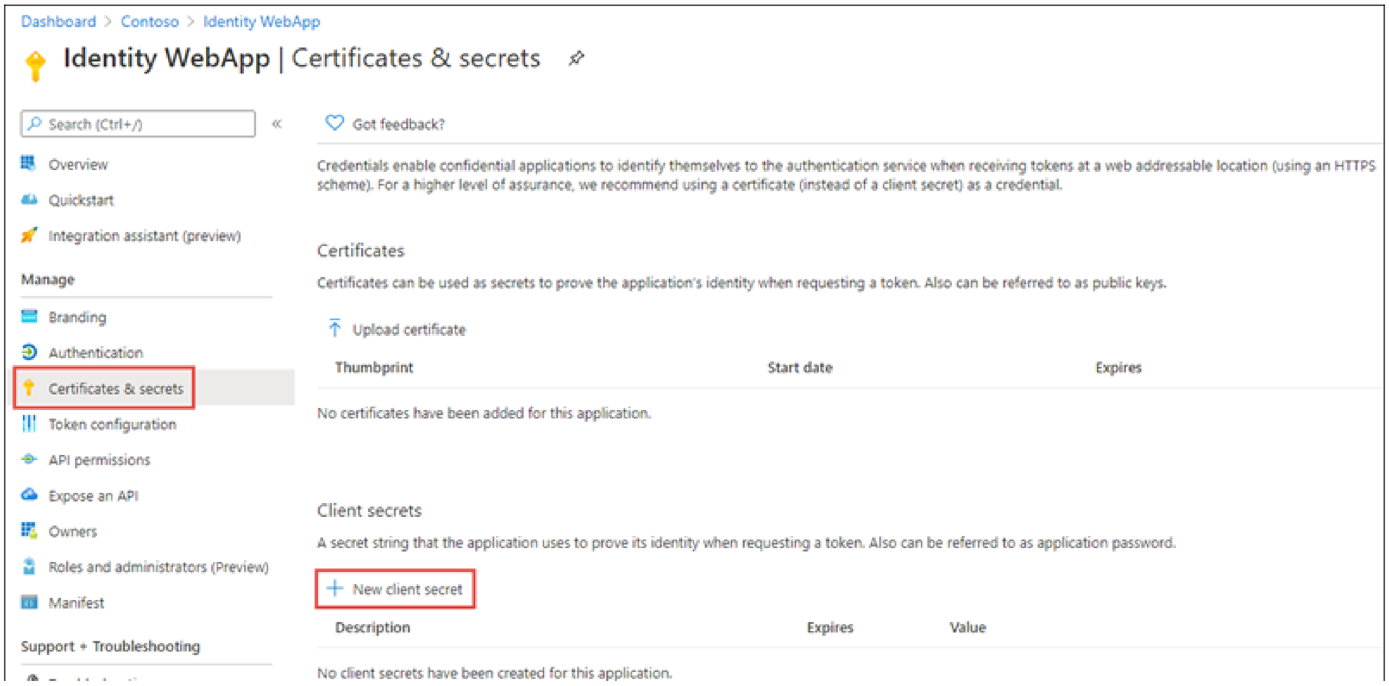
https://localhost:3007/signout-oidc

Create a client secret for the app

In order for the app to run without user involvement, it will sign in to Azure AD with an application ID and either a certificate or secret. In this exercise, you'll use a secret.

Select **Certificates & secrets** from the left-hand navigation panel.

Select the **New client secret** button:



Dashboard > Contoso > Identity WebApp
Identity WebApp | Certificates & secrets
Search (Ctrl+/)
Overview
Quickstart
Integration assistant (preview)
Manage
Branding
Authentication
Certificates & secrets
Token configuration
API permissions
Expose an API
Owners
Roles and administrators (Preview)
Manifest
Support + Troubleshooting
Got feedback?

Credentials enable confidential applications to identify themselves to the authentication service when receiving tokens at a web addressable location (using an HTTPS scheme). For a higher level of assurance, we recommend using a certificate (instead of a client secret) as a credential.

Certificates

Certificates can be used as secrets to prove the application's identity when requesting a token. Also can be referred to as public keys.

Thumbprint	Start date	Expires

No certificates have been added for this application.

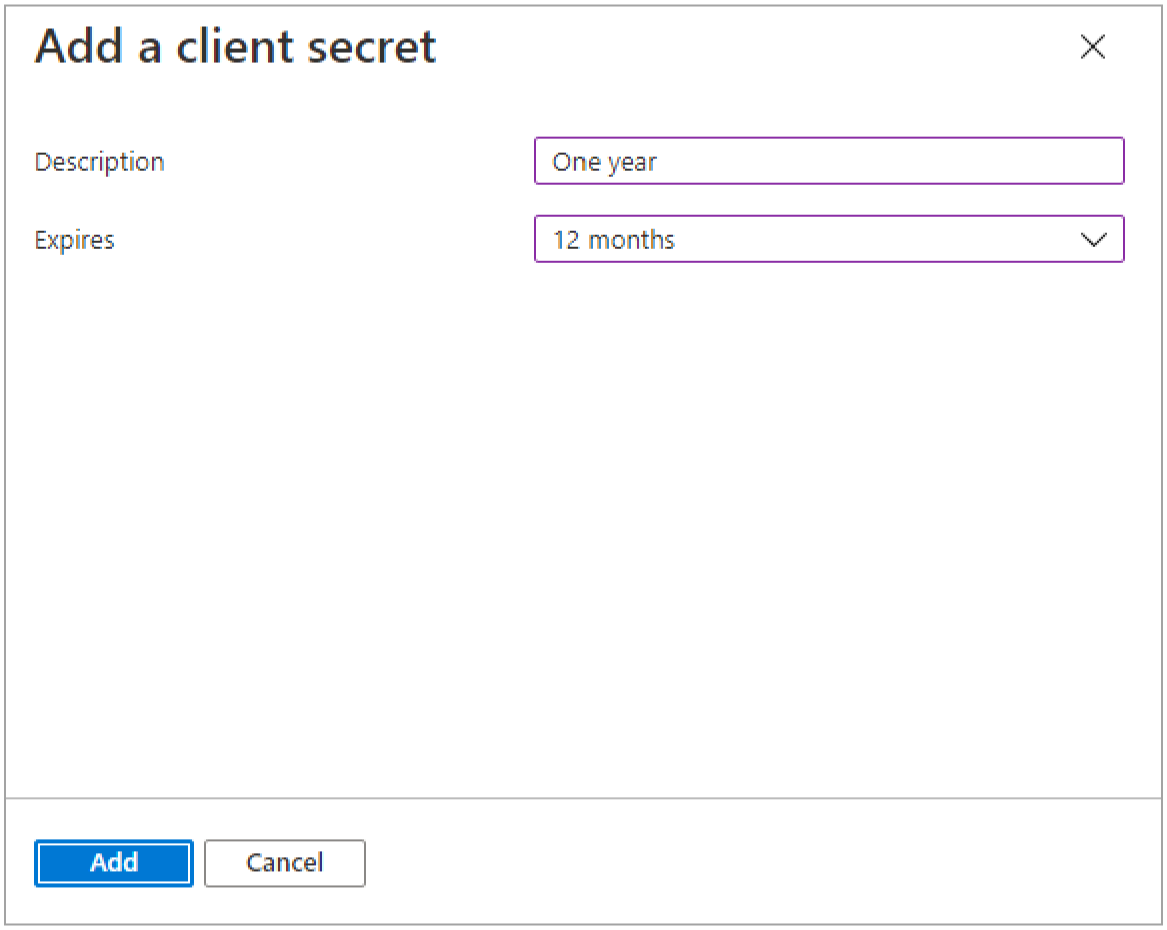
Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

Description	Expires	Value

No client secrets have been created for this application.

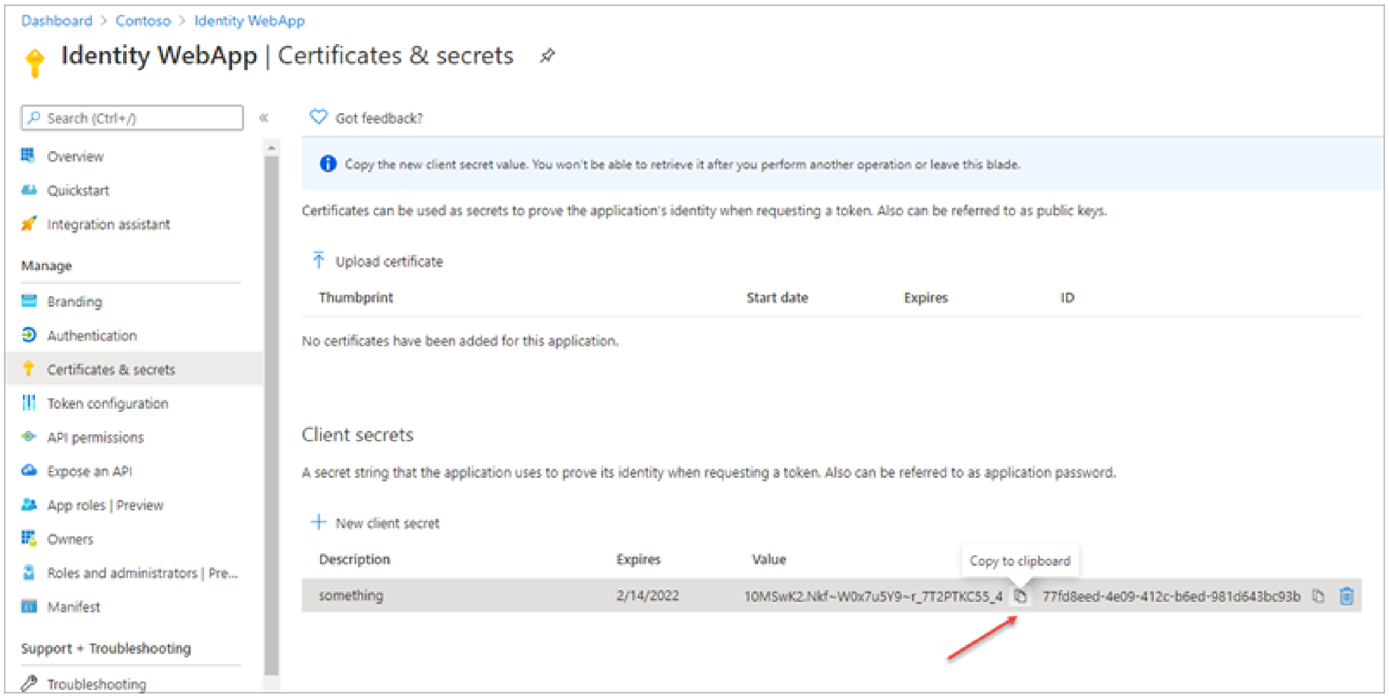
When prompted, give the secret a description and select one of the expiration duration options provided and select **Add**. *What you enter and select doesn't matter for the exercise.*


Add a client secret

Description: One year

Expires: 12 months

The **Certificate & Secrets** page will display the new secret. It's important you copy this value as it's only shown this one time; if you leave the page and come back, it will only show as a masked value.



Copy the new client secret value. You won't be able to retrieve it after you perform another operation or leave this blade.

Certificates can be used as secrets to prove the application's identity when requesting a token. Also can be referred to as public keys.

Thumbprint	Start date	Expires	ID

No certificates have been added for this application.

Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

Description	Expires	Value	
something	2/14/2022	10MSwK2.Nkf~W0x7u5Y9~r_7T2PTKC55_4	77fd8eed-4e09-412c-b6ed-981d643bc93b

Create a single organization ASP.NET web application

Note

The instructions below assume you are using .NET 5. They were last tested using v5.0.202 of the .NET 5 SDK.

Open your command prompt, navigate to a directory where you want to save your work, create a new folder, and change directory into that folder.

Execute the following command to create a new ASP.NET Core MVC web application:

Console

```
dotnet new mvc --auth SingleOrg -o IdentityWeb
```

After creating the application, run the following commands to ensure your new project runs correctly.

Console

```
cd IdentityWeb
dotnet add package Microsoft.Identity.Web
dotnet add package Microsoft.Identity.Web.UI
dotnet add package Microsoft.Identity.Web.MicrosoftGraph
```

Open the application in Visual Studio Code using the following command:

Console Copy

```
code .
```

When a dialog box asks if you want to add required assets to the project, select **Yes**.

Update the web application's launch configuration

Locate and open the **./Properties/launchSettings.json** file in the ASP.NET Core project.

Set the **iisSettings.iisExpress.applicationUrl** property to **https://localhost:3007**.

Set the **iisSettings.iisExpress.sslPort** property to **3007**.

Configure the web application with the Azure AD application

Locate and open the **./appsettings.json** file in the ASP.NET Core project.

Set the **AzureAd.Domain** property to the domain of your Azure AD tenant where you created the Azure AD application (*for example: contoso.onmicrosoft.com*).

Set the **AzureAd.TenantId** property to the **Directory (tenant) ID** you copied when creating the Azure AD application in the previous section.

Set the **AzureAd.ClientId** property to the **Application (client) ID** you copied when creating the Azure AD application in the previous section.

Create a new property, **ClientSecret**, immediately after the **ClientId**. Set the value of this to the client secret you created when creating the Azure AD application in the previous section.

Locate and open the **./Startup.cs** file in the ASP.NET Core project.

Within the `ConfigureServices()` method, locate the following line:

C# Copy

```
services.AddAuthentication(OpenIdConnectDefaults.AuthenticationScheme)
    .AddMicrosoftIdentityWebApp(Configuration.GetSection("AzureAd"));
```

Update the line to the following. This will configure the web app's middleware to add support for the Microsoft Graph:

C# Copy

```csharp
services.AddAuthentication(OpenIdConnectDefaults.AuthenticationScheme)
    .AddMicrosoftIdentityWebApp(Configuration.GetSection("AzureAd"))
    .EnableTokenAcquisitionToCallDownstreamApi(new string[] { "User.Read" })
    .AddMicrosoftGraph("https://graph.microsoft.com/v1.0", "User.Read")
    .AddInMemoryTokenCaches();
```

Add a User controller and view to the web app

The last step is to add a controller and view to the web app that will display the current user's name from a Microsoft Graph request.

Add a new file **UserController.cs** to the **Controllers** folder. Add the following code to it:

C# Copy

```csharp
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using Microsoft.Graph;
using Microsoft.Identity.Web;

namespace IdentityWeb.Controllers
{
  [Authorize]
  public class UserController : Controller
  {
    private readonly ILogger<UserController> _logger;
    private readonly GraphServiceClient _graphServiceClient;

    public UserController(ILogger<UserController> logger, GraphServiceClient graph-
ServiceClient)
    {
      _logger = logger;
      _graphServiceClient = graphServiceClient;
    }


    [AuthorizeForScopes(Scopes = new[] { "User.Read" })]
    public async Task<IActionResult> Index()
    {
      var user = await _graphServiceClient.Me.Request().GetAsync();

      return View(user);
    }
  }
}
```

This controller's default method, `Index()`, submits a request to Microsoft Graph for the current user's details. This is done using the `GraphServiceClient` added as a singleton to the ASP.NET Core dependency injection (DI) configuration in the previous step.

Now create the view to display the user's name.

Add a new folder **User** to the **Views** folder. Add a new file, **Index.cshtml**, to the new **User** folder and add the following code to it. This will display the currently signed-in user's name on the page:

HTML Copy

```html
@{
  ViewData["Title"] = "User Page";
}

<div class="text-center">
  <h1 class="display-4">Welcome @Model.DisplayName</h1>
</div>
```

Build and test the web app

Execute the following command in a command prompt to compile and run the application:

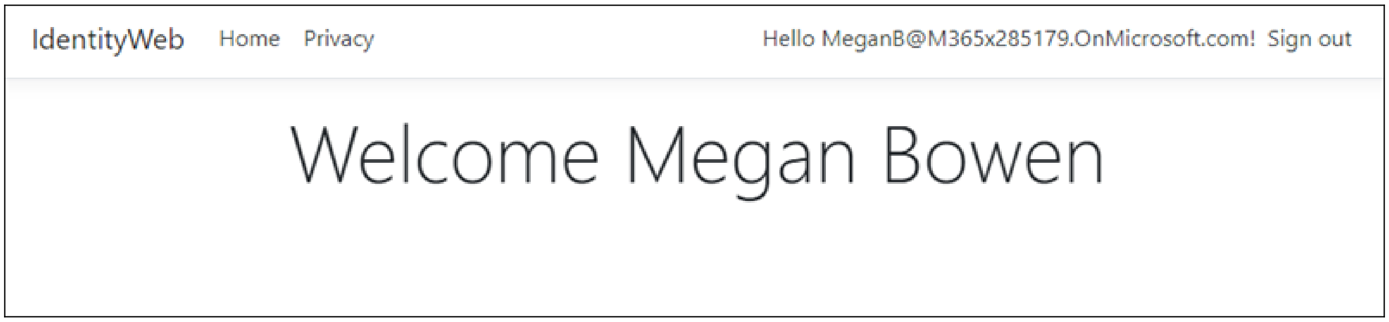
Console Copy

```
dotnet build
dotnet run
```

Open a browser and navigate to the url **https://localhost:5001**. The web application will redirect you to the Azure AD sign-in page.

Sign in using a Work and School account from your Azure AD directory. Azure AD will redirect you back to the web application.

Update the URL to **https://localhost:5001/User** to navigate to the **User** controller. Notice the name of the currently signed in user is displayed on the page:



Welcome Megan Bowen

Summary

In this unit, you learned how to create a server-side web app that allows users to sign in and grant the app permissions to act on the user's behalf. Once the user has authenticated and granted the app consent to act on their behalf, the web application will use data returned from Microsoft Graph by using the OAuth 2.0 auth code grant flow.

Test your knowledge

1. Which of the following unique characteristics of the authorization code grant flow makes it a preferred option for securing web apps?

- ○ The web app never has access to the user's credentials because the user signs in with Azure AD, not with the app.
- ○ All communication between the web app and Azure AD is secured.
- ○ The web app authenticates with Azure AD using a x509 certificate when obtaining an access token.

2. What elements are required to create, configure and/or collect when registering an Azure AD app for use with the OAuth 2.0 authorization code grant flow?

- ○ Tenant ID, application ID, application secret, and the redirect URI

- ○ Application ID, application secret, and the redirect URI.

- ○ Tenant ID, application ID, and the application secret.

Check your answers

How are we doing? ☆ ☆ ☆ ☆ ☆