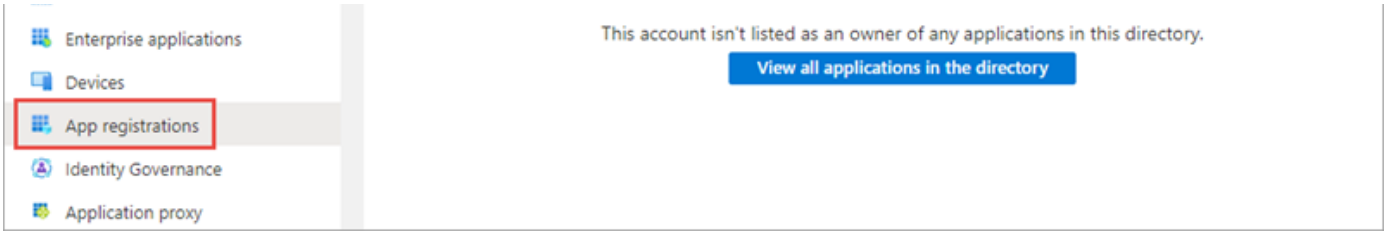
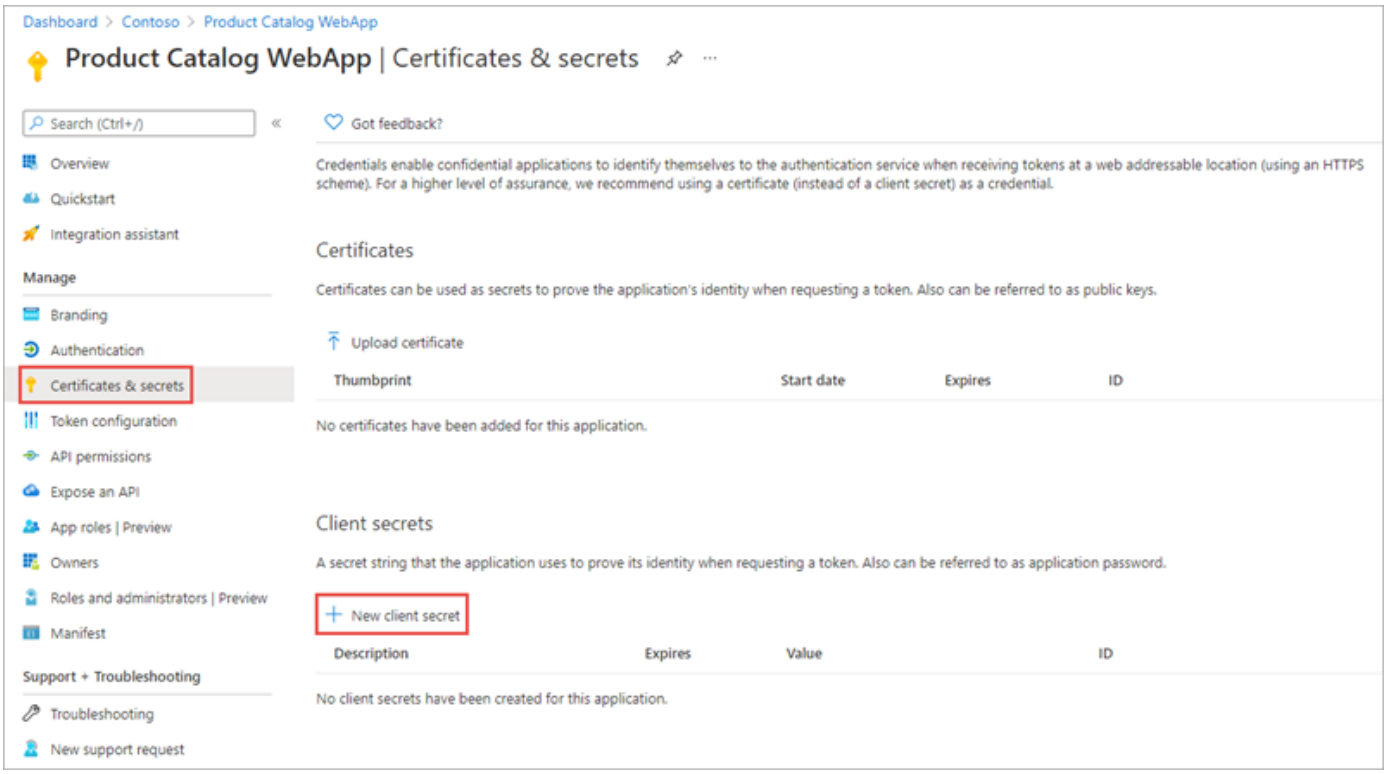


The important part of the app registration is the creation of an app secret. This secret, along with the client ID of the app, are used by the web application to authenticate with Microsoft identity when it requests an access token.



Web apps that call web APIs are *confidential client* applications. That's why they register a secret (an application password or certificate) with Azure AD.

Configure web applications to call secured APIs

The next step is to create the web application that will allow the user to sign in and then access the secured web API.

Create a new ASP.NET Core MVC application using the following command, followed by installing a few NuGet packages to support and Microsoft identity.

```
Console
```

```
dotnet new mvc --auth SingleOrg
dotnet add package Microsoft.Identity.Web
dotnet add package Microsoft.Identity.Web.UI
```

Update web app to support user sign-in for Microsoft identity

Now update the project to associate it with the Azure AD app you registered for the web app. Open the **appsettings.json** file and set the details of the registered Azure AD application you previously created. This includes details such as:

- **Domain**: the domain of your Azure AD tenant where you registered the Azure AD application
- **TenantId**: the ID of your Azure AD tenant where you registered the Azure AD application
- **ClientId**: the ID of your Azure AD application
- **ClientSecret**: the secret of your Azure AD application

Next, configure the web app's authentication by updating the `ConfigureServices()` method in the `Startup` class for the project. This code will configure the web app's middleware to support Azure AD for authentication and to obtain an ID token:

C#

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddAuthentication(OpenIdConnectDefaults.AuthenticationScheme)
      .AddMicrosoftIdentityWebApp(Configuration.GetSection("AzureAd"))
      .EnableTokenAcquisitionToCallDownstreamApi(Constants.ProductCatalogAPI.SCOPES)
      .AddInMemoryTokenCaches();

  services.AddControllersWithViews(options =>
  {
    var policy = new AuthorizationPolicyBuilder()
              .RequireAuthenticatedUser()
              .Build();
    options.Filters.Add(new AuthorizeFilter(policy));
  });
  services.AddRazorPages()
        .AddMicrosoftIdentityUI();
}
```

At this point, the web app is associated with the registered Azure AD app and configured to support users signing in with their Microsoft identity account.

Add token acquisition

The next step is to create controllers to implement the pages on the site.

When the user requests one of these pages, the web app already has an access token for the user. This token is used, in addition to the web app's Azure AD app details, to obtain a new access token. The resulting access token is intended to be used with the web API but is for the web app to use.

C#

```
public CategoriesController(ITokenAcquisition tokenAcquisition)
{
  this.tokenAcquisition = tokenAcquisition;
}
```

C#

```
[AuthorizeForScopes(Scopes = new[] { Constants.ProductCatalogAPI.CategoryReadScope
})]
public async Task<ActionResult> Index()
{
  var client = new HttpClient();

  var accessToken = await
tokenAcquisition.GetAccessTokenForUserAsync(Constants.ProductCatalogAPI.SCOPES);
  client.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", accessToken);

  var json = await client.GetStringAsync(url);

  var serializerOptions = new JsonSerializerOptions
  {
    PropertyNamingPolicy = JsonNamingPolicy.CamelCase
  };
  var categories = JsonSerializer.Deserialize(json, typeof(List<Category>), serial-
izerOptions) as List<Category>;
  return View(categories);
}
```

Summary

In this unit, you learned how to create server-side web apps that enable users to sign in and grant the app permissions.

Next unit: Exercise - Call secured APIs from web applications

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆