**Organizational Accounts for ASP .NET Core 3.1**

By Shahed C on April 13, 2020

[5 Replies](https://wakeupandcode.com/organizational-accounts-for-asp-net-core-3-1/#comments)



This is the fifteenth of a new [series of posts](https://wakeupandcode.com/aspnetcore/#aspnetcore2020) on ASP .NET Core 3.1 for 2020. In this series, we’ll cover 26 topics over a span of 26 weeks from January through June 2020, titled **ASP .NET Core A-Z!** To differentiate from the [2019 series](https://wakeupandcode.com/aspnetcore/#aspnetcore2019), the 2020 series will mostly focus on a growing single codebase ([NetLearner!](https://wakeupandcode.com/netlearner-on-asp-net-core-3-1/)) instead of new unrelated code snippets week.

Previous post:

* [.NET 5.0, VS2019 Preview and C# 9.0 for ASP .NET Core developers](https://wakeupandcode.com/net-5-0-vs2019-preview-and-c-9-0-for-asp-net-core-developers/)

**NetLearner on GitHub**:

* Repository: <https://github.com/shahedc/NetLearnerApp>
* v0.15-alpha release: <https://github.com/shahedc/NetLearnerApp/releases/tag/v0.15-alpha>

**NOTE**: The NetLearner suite of apps won’t be updated to use organizational authentication in the main branch, so you can check out the new sample code in the experimental subfolder, merged from a [branch](https://github.com/shahedc/NetLearnerApp/tree/orgauth/experimental/NetLearner.OrgAuth):

* Org Auth: <https://github.com/shahedc/NetLearnerApp/tree/main/experimental/NetLearner.OrgAuth>

**In this Article:**

* O is for Organizational Accounts
* Adding Authentication
* Configuring App Registration
* Using Authentication in your Code
* Running the Samples
* References

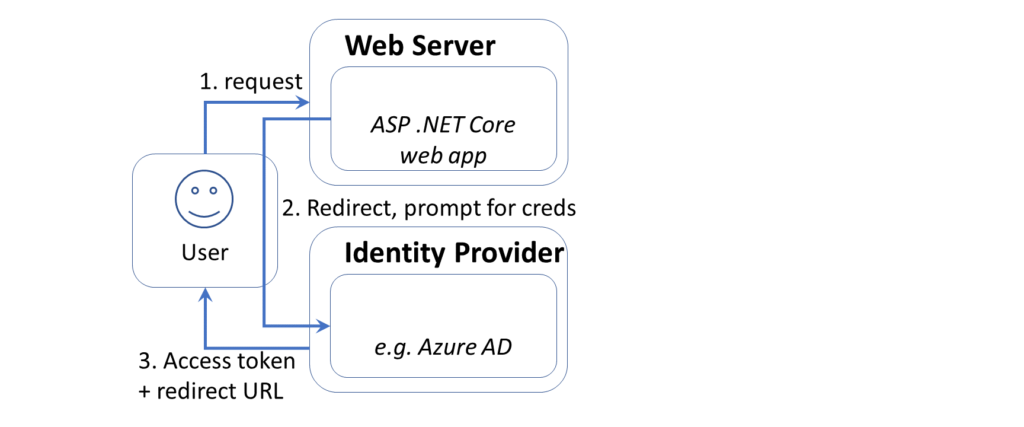
**O is for Organizational Accounts**

If you’ve created new ASP .NET Core projects, you’ve probably seen an option to add authentication upon creation. In Visual Studio, the IDE provides radio buttons to select a specific type of Authentication. Using CLI commands (e.g. in the VS Code terminal) you can use the ‐‐auth flag to choose the type of authentication you’d like to add.

The possible values are:

* **None** – No authentication (Default).
* **Individual** – Individual authentication (accounts stored in-app).
* **IndividualB2C** – Individual authentication with Azure AD B2C.
* **SingleOrg** – Organizational authentication for a single tenant.
* **MultiOrg** – Organizational authentication for multiple tenants.
* **Windows** – Windows authentication..

In this article, we will focus on the option for Work or School Accounts. This option can be used to authenticate users with AD (Active Directory, Azure AD or Office 365. This authentication process is illustrated in the diagram shown below:

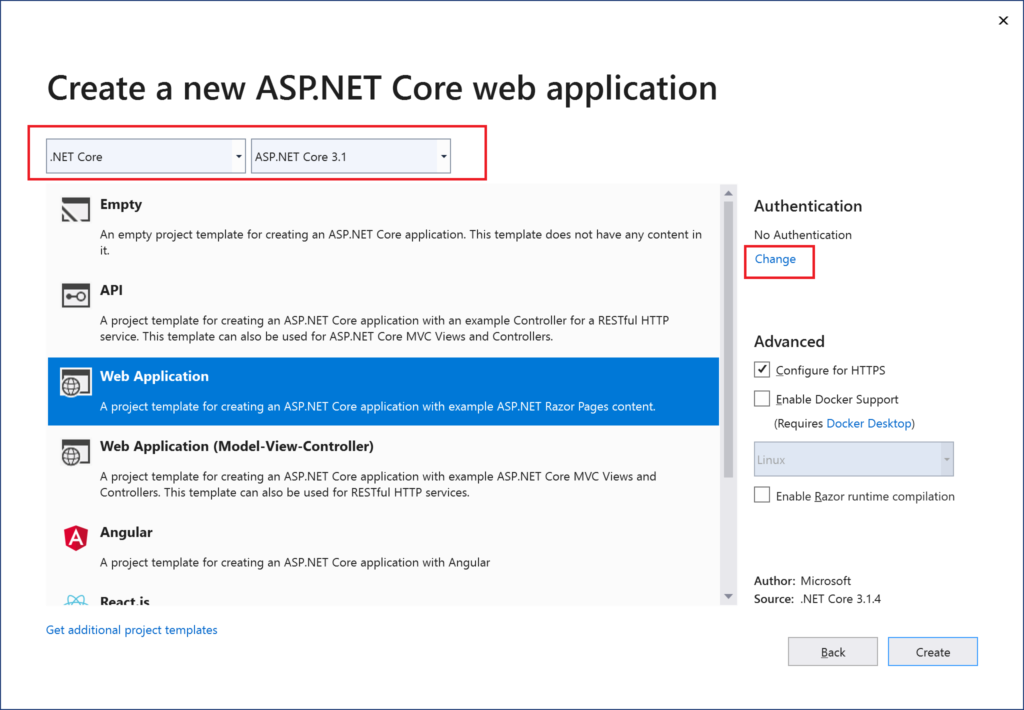
ASP .NET Core web authentication via AzureAD

**Adding Authentication**

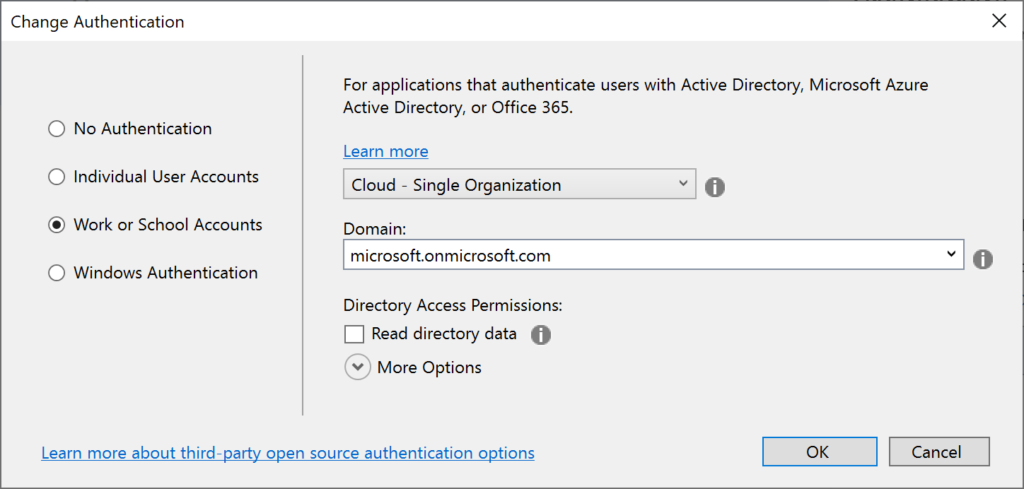
To add authentication to a new project quickly, here are the instructions for Visual Studio 2019.

If you choose to use the new splash screen:

1. Click “Create a new project”
2. Select “ASP .NET Core Web Application” or “Blazor” (server-side)
3. Click Next
4. Enter Project Name, Location, Solution Name
5. Optional: check the checkbox to place in the same directory
6. Click Create
7. Select .NET Core and then ASP .NET Core 3.1 from dropdowns
8. Select a project type, e.g. Empty, Web Application (Razor Pages or MVC), Blazor App.
9. Click the “Change” action link in the Authentication section

Creating a new web app project with authentication

This should allow you to change the authentication type to “Work or School Accounts” so that you may your organizational domain info. As always, you may select the little info buttons (lowercase i) to learn more about each field. Talk to your system administrator if you need more help on what domains to use.

Work/School account selected for authentication method

**NOTE**: If you need help creating new .NET Core 3.1 project types in Visual Studio, take a look at this [previous blog post in this series on .NET Core 3.0](https://wakeupandcode.com/hello-asp-net-core-v3-1/) to get some help on how to enable it.

* Hello ASP .NET Core 3.1: <https://wakeupandcode.com/hello-asp-net-core-v3-1/>

If you wish to *skip* the Splash Screen instead upon launching VS2019:

1. Click “Continue without code” in the lower right area of the splash screen.
2. In the top menu, click File | New | Project (or Ctrl-Shift-N)
3. Follow the remaining steps outlined earlier in this section

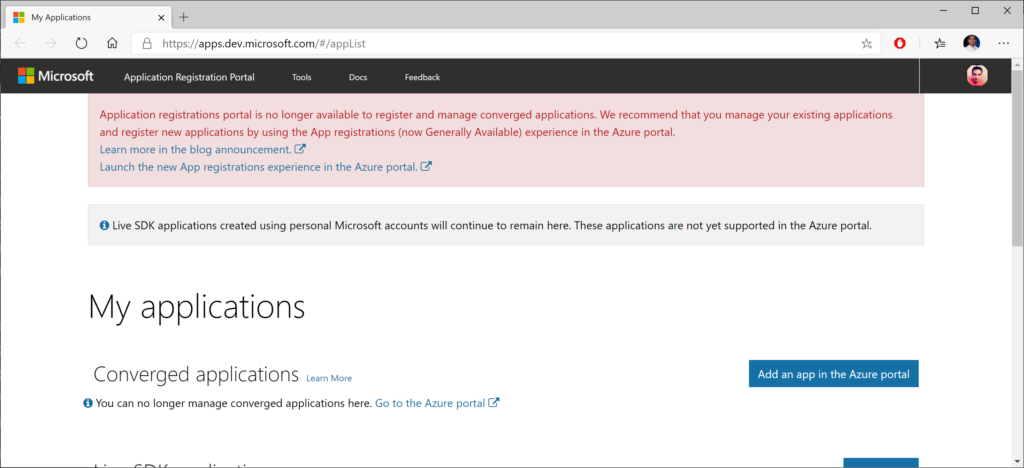
To use CLI Commands in a Terminal window, use the **dotnet new** command followed by the ‐‐auth flag. The authentication type can be any of the aforementioned authentication types, e.g. Individual.

dotnet new mvc --auth Individual -o myproj

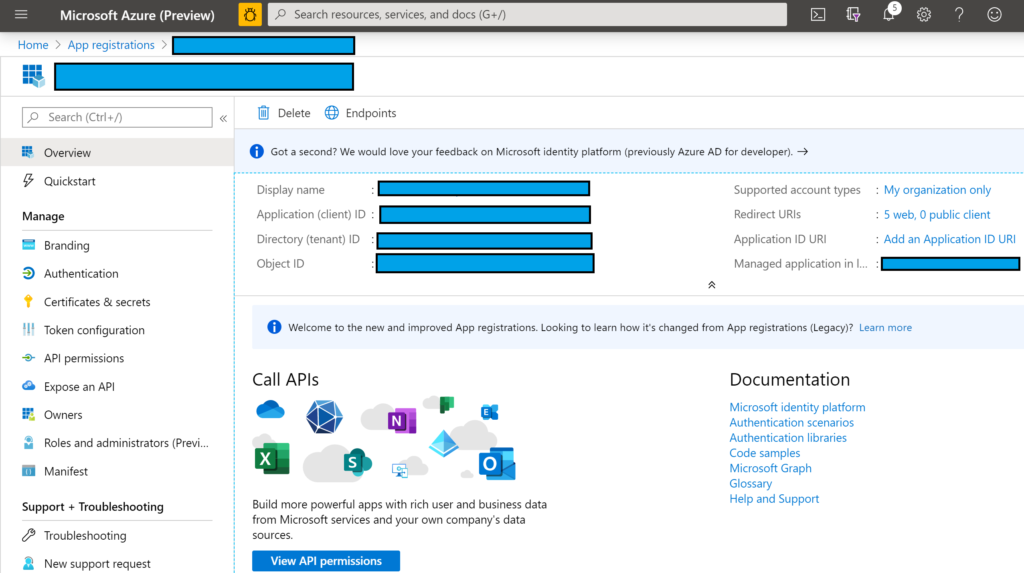
**Configuring App Registration**

If you’ve used “Individual User Accounts” before, you’ve probably used a database to store user data, either on-premises or in the cloud. If you’ve used “Work or School Accounts” (i.e. organizational accounts), you may have the old App portal at the following URL:

* (OLD) Application Registration Portal: <https://apps.dev.microsoft.com>

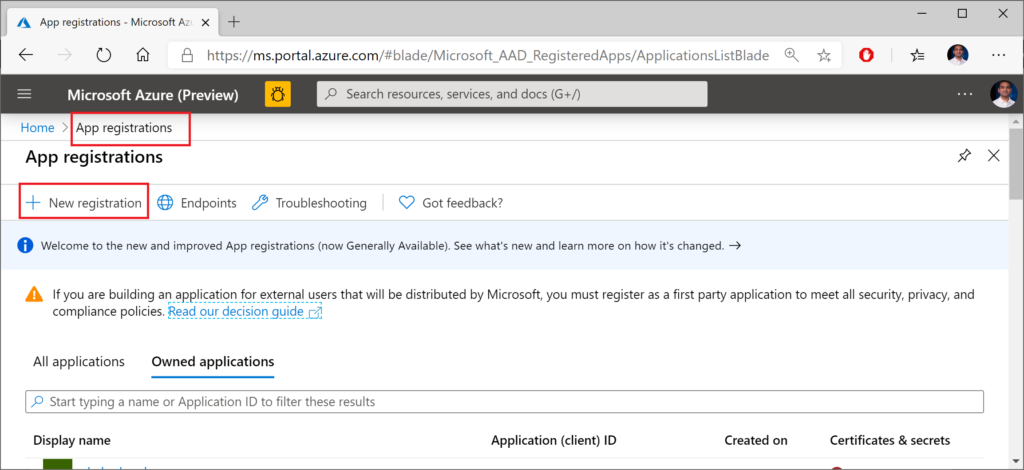
Old App Registration Portal

On the old site, you may see a message suggesting that you should go to the Azure Portal to use the newer (current) App Registrations feature. This feature was previously in preview, and you can currently use it directly in the Azure Portal. If you click the link from the old site, it should take you directly to the App Registrations page, and may prompt you to log in first.

App Registration screen showing overview

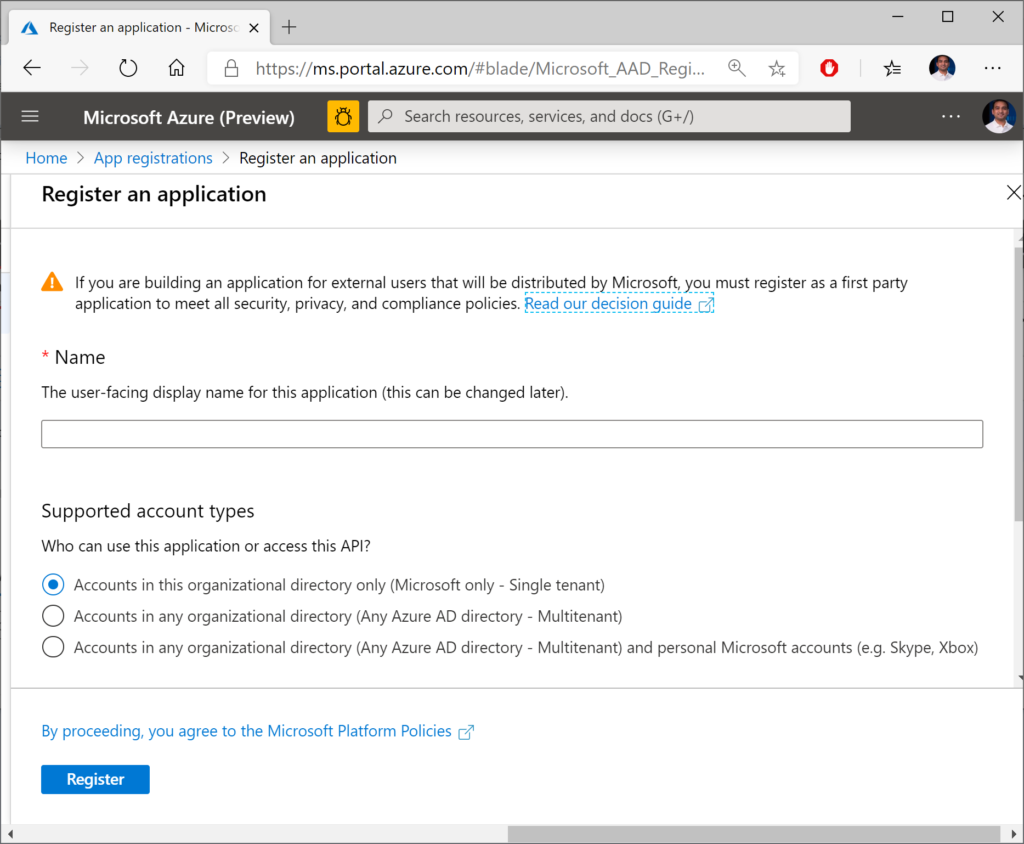
If you’re not clear about how you got to this screen or how to come back to it later, here’s a set of steps that may help.

1. Log in to the [Azure Portal](https://portal.azure.com/)
2. Search for App Registrations
3. Arrive at the App Registrations page
4. When a newer preview version becomes available, you may have to click the banner that takes you to a preview experience.

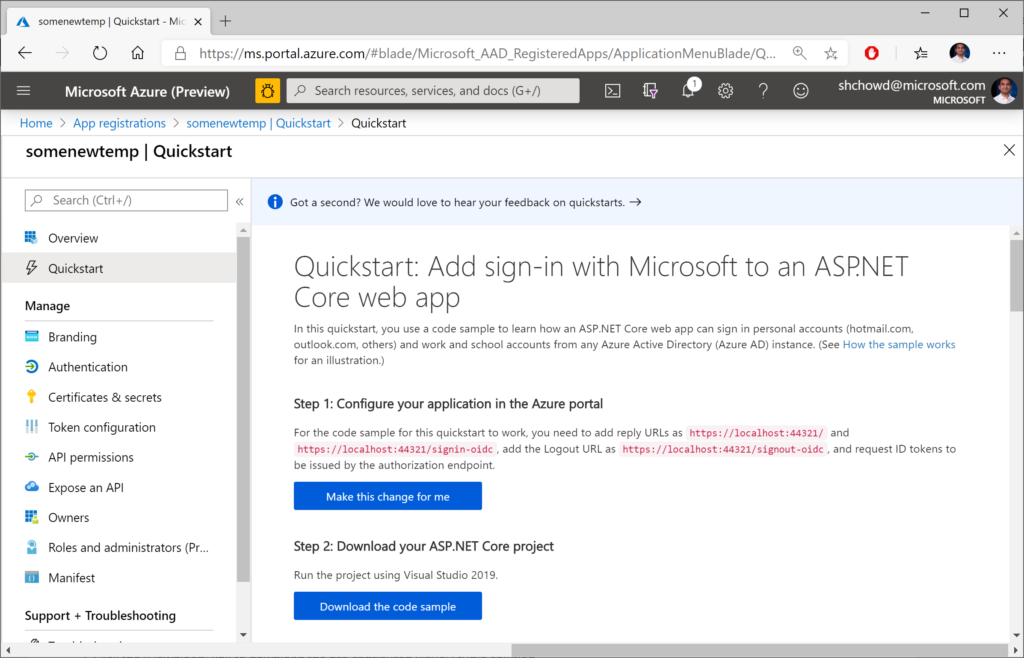
New App Registration in Azure Portal

In the form that follows, fill out the values for:

* **Name** (which you can change later)
* **Account Type** (your org, any org, any org + personal MSA)
* **Redirect URI** (where users will return after authentication)

 Register an application for authentication

Now you should have options to configure your app and also download a pre-written application to get started. In the QuickStart section for your newly registered application (after selecting  the guide for “ASP .NET Core”, you should see a button to make the changes for you and also download a configured Visual Studio application.

Quickstart screen for app registration

In the steps that follow:

1. Click the “Make the changes for me” button to make the necessary configuration changes.
2. Optional: Click the “Download” link to download the pre-configured Visual Studio solution, if you don’t already have a project.

At the time of this writing, the downloadable sample project is a VS2019 application for ASP .NET Core 2.2. You can download it to inspect it, but I would recommend creating a new project manually in VS2019. There may be some subtle differences between projects created by VS2019 with authentication turned on, versus what you get with the downloaded project.

For further customization using the Manifest file available to you, check out the official documentation on the Azure AD app manifest:

* Understanding the Azure Active Directory app manifest: <https://docs.microsoft.com/en-us/azure/active-directory/develop/reference-app-manifest>

**Using Authentication in Your Code**

When creating a new project in VS2019, you get the following lines of code in your **ConfigureServices()** method, including calls to .**AddAuthentication**(). For different project types (MVC, Razor Pages, Blazor), you should also see some additional code setting up the authorization policy.

// contents of ConfigureServices() when created in VS2019  
  
services.AddAuthentication(AzureADDefaults.AuthenticationScheme)  
.AddAzureAD(options => Configuration.Bind("AzureAd", options));

The string value “AzureAd” is referenced in your appsettings file, e.g. appsettings.json in [MVC](https://github.com/shahedc/NetLearnerApp/blob/main/experimental/NetLearner.OrgAuth/OrgAuth.Mvc/appsettings.json), [Razor Pages](https://github.com/shahedc/NetLearnerApp/blob/main/experimental/NetLearner.OrgAuth/OrgAuth.Pages/appsettings.json), or [Blazor](https://github.com/shahedc/NetLearnerApp/blob/main/experimental/NetLearner.OrgAuth/OrgAuth.Blazor/appsettings.json) projects. The properties for this entry include the following:

* **Instance**: URL for login page, e.g. https://login.microsoftonline.com
* **Domain**: your AD tenant’s domain, e.g. microsoft.onmicrosoft.com
* **TenantId**: your Tenant Id value, usually a GUID
* **ClientId**: Your app’s Client Id, obtained from app registration
* **Callback Path**: partial path for sign-in, e.g. /signin-oidc

{  
 "AzureAd": {  
 "Instance": "https://login.microsoftonline.com/",  
 "Domain": "<REPLACE\_DOMAIN\_NAME>",  
 "TenantId": "<REPLACE\_TENANT\_ID>",  
 "ClientId": "<REPLACE\_CLIENT\_ID>",  
 "CallbackPath": "/signin-oidc"  
 },  
 "Logging": {  
 "LogLevel": {  
 "Default": "Information",  
 "Microsoft": "Warning",  
 "Microsoft.Hosting.Lifetime": "Information"  
 }  
 },  
 "AllowedHosts": "\*"  
}

You can see more information on appsettings.json values in the official docs. You can also open the apssettings.json from the portal-downloaded project to get your own app’s Id values. The following documentation is specifically written for v2, but offers explains for important fields such as **ClientId** and **TenantId**.

* Microsoft identity platform ASP.NET Core web app quickstart: [https://docs.microsoft.com/en-us/azure/active-directory/develop/quickstart-v2-aspnet-core-webapp](https://docs.microsoft.com/en-us/azure/active-directory/develop/quickstart-v2-aspnet-core-webapp?WT.mc_id=-blog-shchowd)

To get a refresher on Authentication and Authorization in ASP .NET Core 3.1, check out the following post on from earlier in this blog series.

* Authentication and Authorization in ASP .NET Core 3.1: <https://wakeupandcode.com/authentication-authorization-in-asp-net-core-3-1/>

For more on assigning users to specific roles, check out the official documentation at the following URL:

* Role-based authorization in ASP.NET Core: <https://docs.microsoft.com/en-us/aspnet/core/security/authorization/roles>

Finally, take a look at the Login.partial.cshtml partial view to observe the way a user’s identity is detected and shown in an [MVC](https://github.com/shahedc/NetLearnerApp/blob/main/experimental/NetLearner.OrgAuth/OrgAuth.Mvc/Views/Shared/_LoginPartial.cshtml) or [Razor Pages](https://github.com/shahedc/NetLearnerApp/blob/main/experimental/NetLearner.OrgAuth/OrgAuth.Pages/Pages/Shared/_LoginPartial.cshtml) web app. Here is a snippet from the samples:

@if (User.Identity.IsAuthenticated)  
{  
 <li class="nav-item">  
 <span class="nav-text text-dark">Hello @User.Identity.Name!</span>  
 </li>  
...  
}

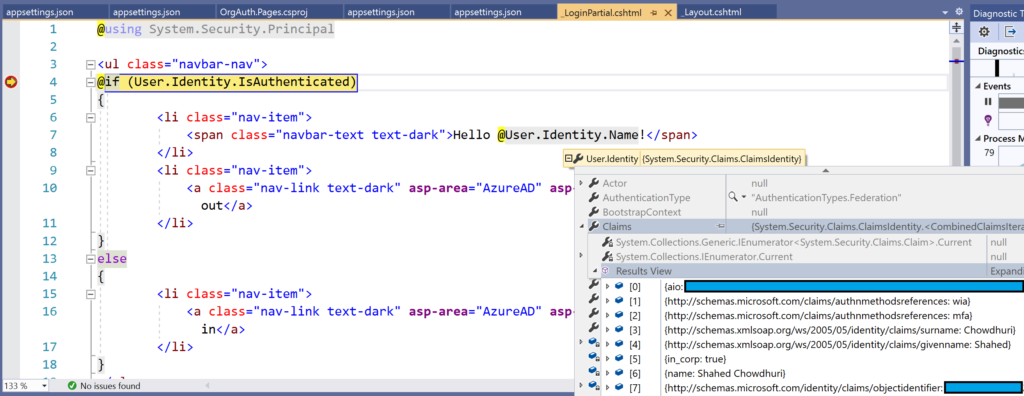
**NOTE**: The [Blazor](https://github.com/shahedc/NetLearnerApp/blob/main/experimental/NetLearner.OrgAuth/OrgAuth.Blazor/Shared/LoginDisplay.razor) web app wraps the authorized content in an <**Authorized**> tag within an <**AuthorizedView**. For more on the Blazor project, take a look at the Blazor post:

* Blazor full-stack web dev: <https://wakeupandcode.com/blazor-full-stack-web-dev-in-asp-net-core-3-1/>

Depending on what you have access to, the User.Identity object may not contain everything you expect it to. Here are some things to take note of:

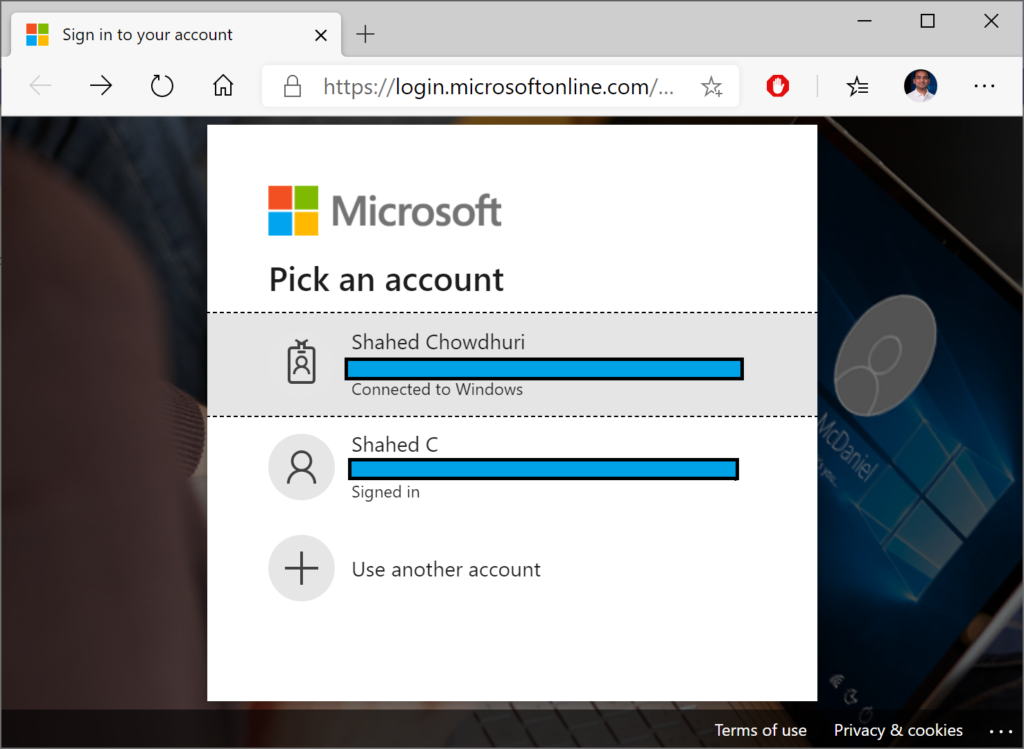
* User.Identity should be null when *not* logged in
* User.Identity should be non-null when logged in…
* … however, User.Identity.Name may be null even when logged in
* If User.Identity.Name is null, also check User.Identity.Claims
* User.Identity.Claims should have more than 0 values when logged in

The following screenshot shows an example of the user information in my debugging environment when logged in:

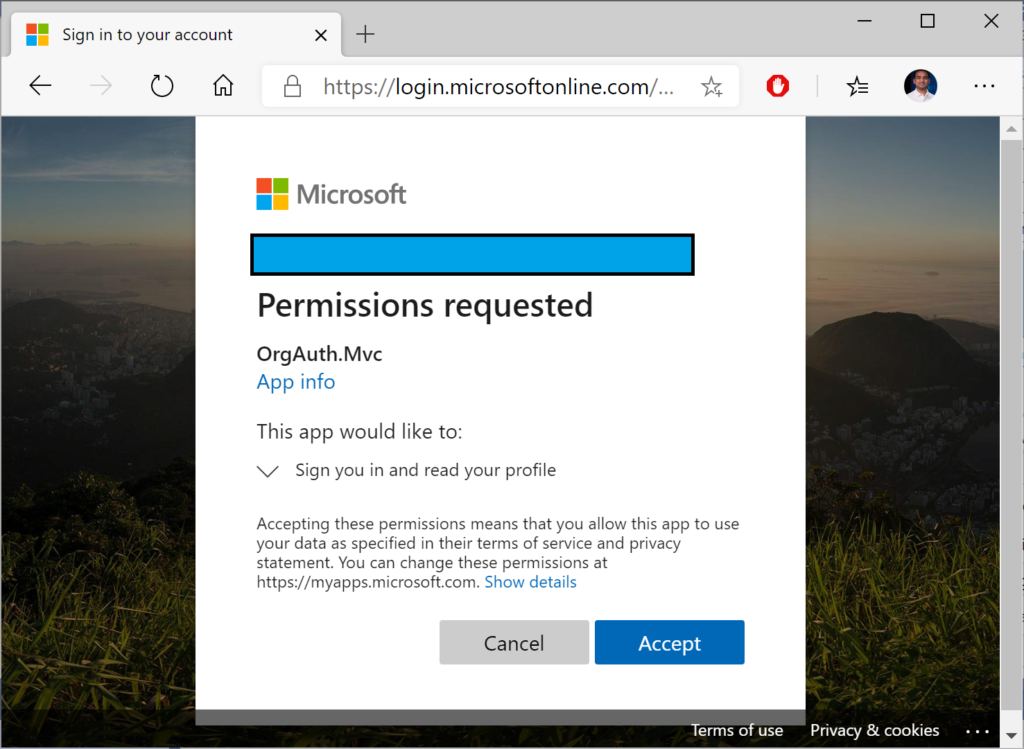
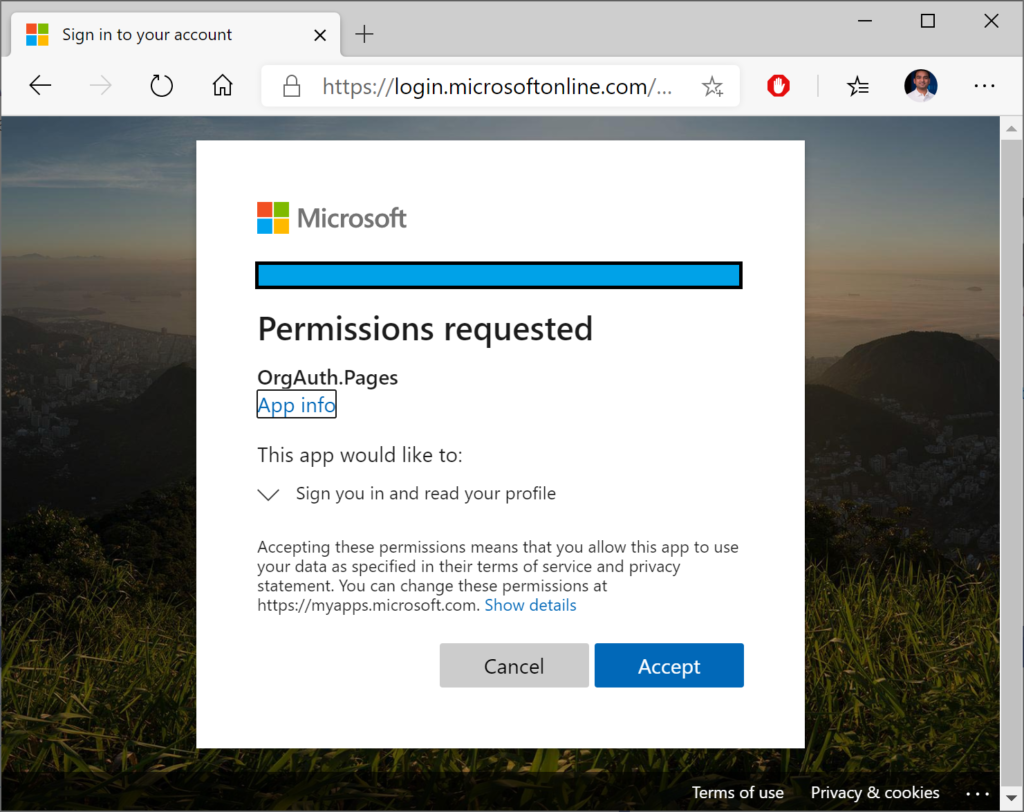
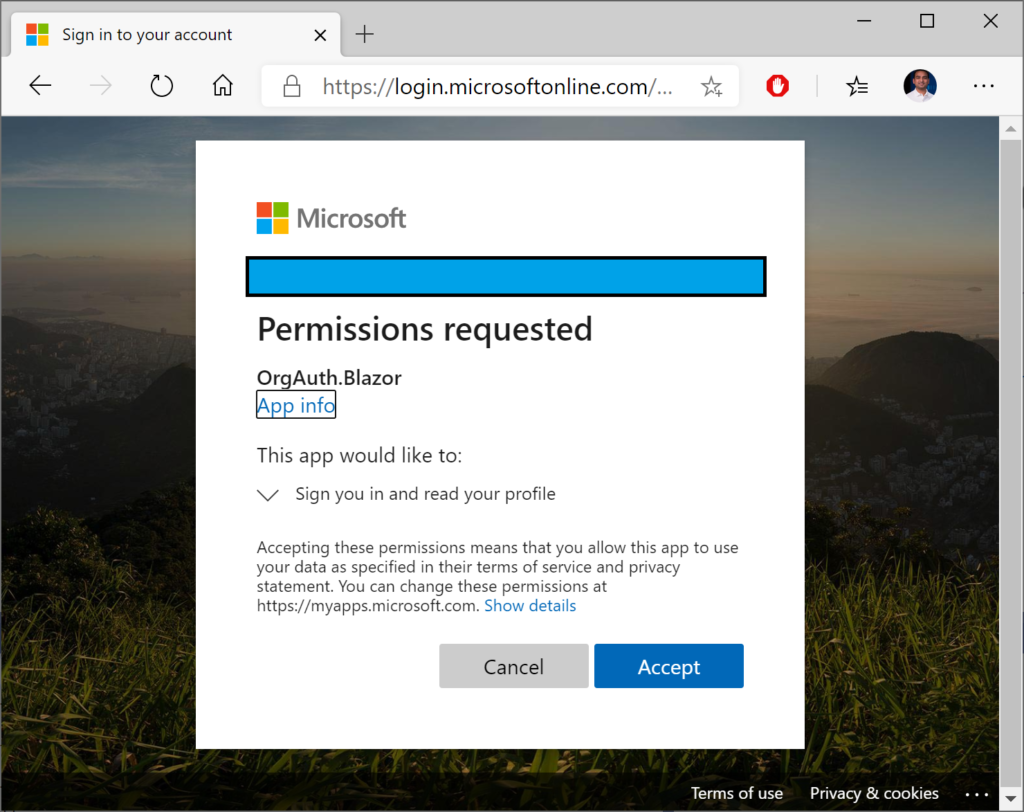
 User.Identity.Name with Claims in debugger

**Running the Samples**

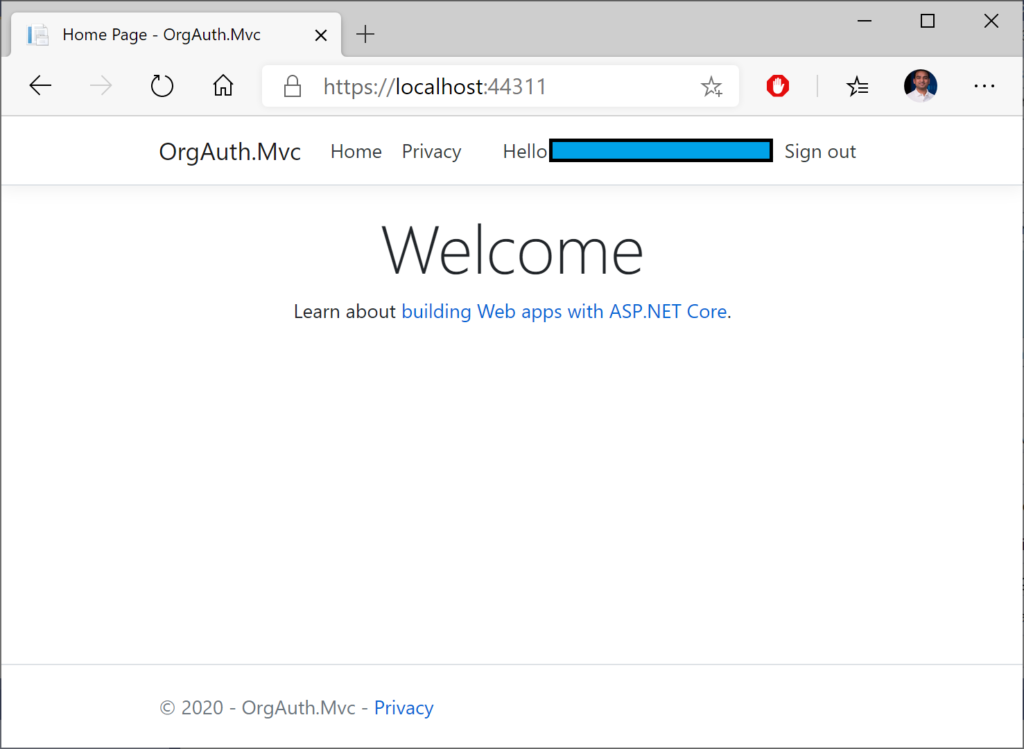
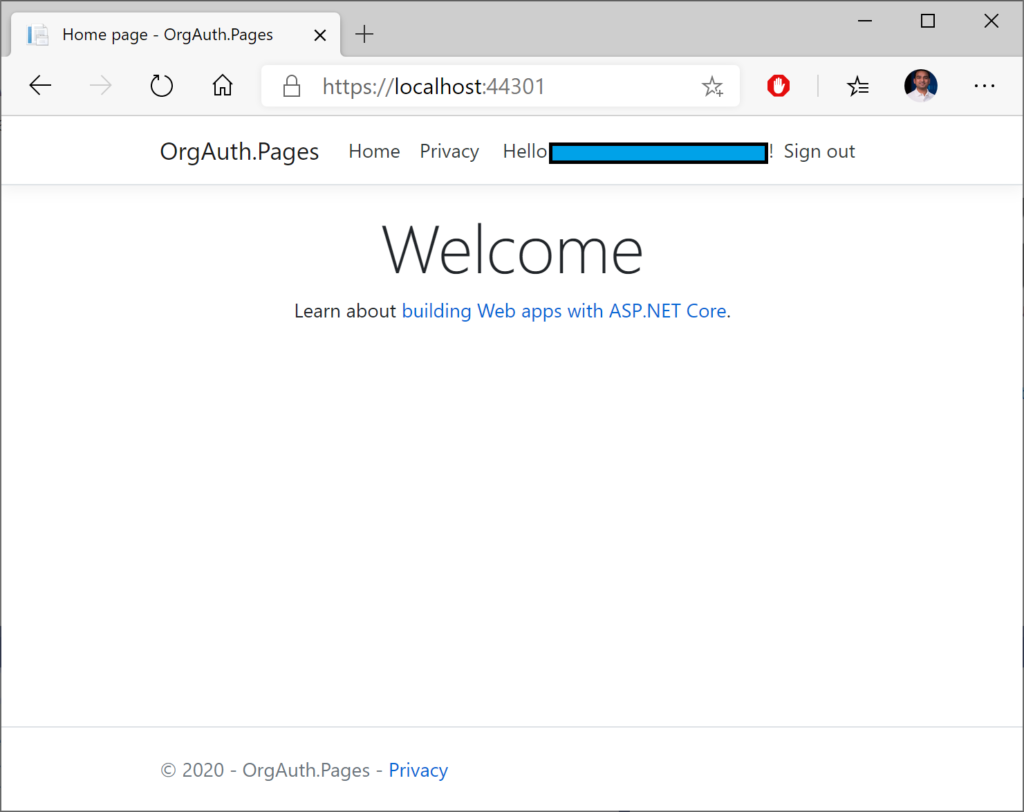
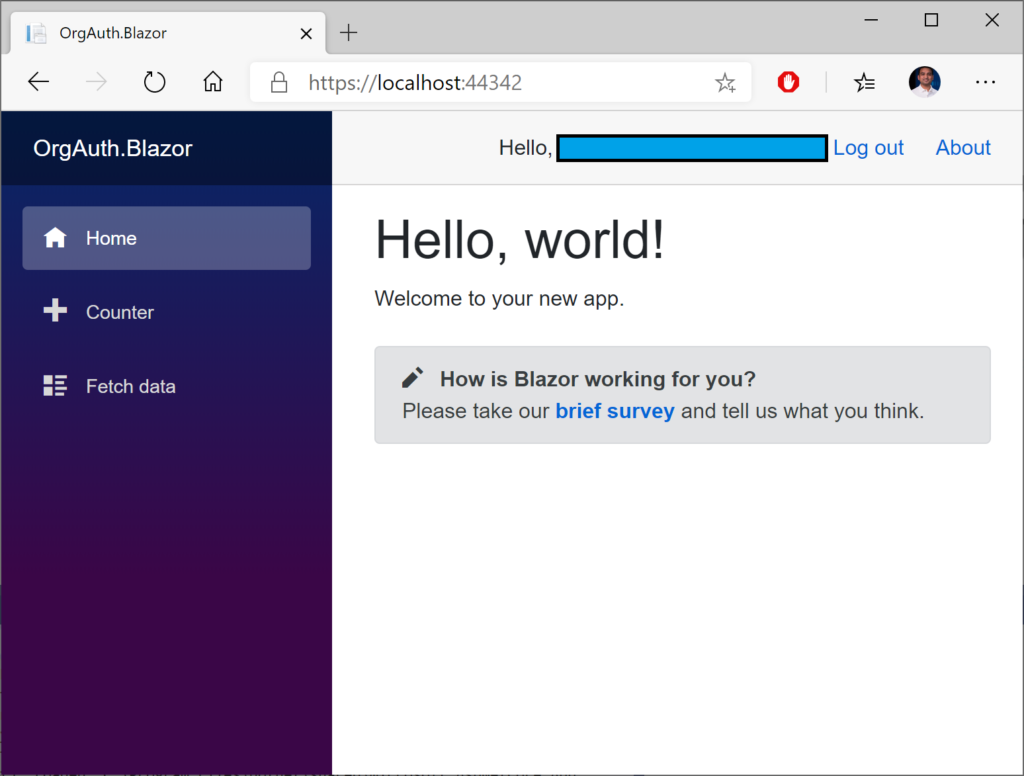
Running the sample web projects should redirect you to the Azure AD login page for your tenant. The login page should look identical for all 3 project types: [MVC](https://github.com/shahedc/NetLearnerApp/tree/main/experimental/NetLearner.OrgAuth/OrgAuth.Mvc), [Razor Pages](https://github.com/shahedc/NetLearnerApp/tree/main/experimental/NetLearner.OrgAuth/OrgAuth.Pages) or [Blazor](https://github.com/shahedc/NetLearnerApp/tree/main/experimental/NetLearner.OrgAuth/OrgAuth.Blazor). If you’ve already logged in to one or more accounts in your browser, you should have one or more identities to select from.



Once you’ve selected an account, you should see a confirmation screen (at least the first time) to Accept the authentication and login. This screen will look similar for all your projects, and the project name should be displayed in the middle.

 MVC app authentication confirmation Razor Pages app authentication confirmation Blazor app authentication confirmation

Finally, you should see each web application with some indication that you are currently logged in. In the template-generated sample apps, this is demonstrated by the “Hello” message in the header area.

MVC web app, logged in Razor Pages web app, logged in  Blazor web app, logged in

**References**

* Role-based authorization in ASP.NET Core | Microsoft Docs: <https://docs.microsoft.com/en-us/aspnet/core/security/authorization/roles>
* Configure an application to access web APIs (Preview) | Microsoft Docs: <https://docs.microsoft.com/en-us/azure/active-directory/develop/quickstart-configure-app-access-web-apis>
* Understanding the Azure Active Directory app manifest | Microsoft Docs: <https://docs.microsoft.com/en-us/azure/active-directory/develop/reference-app-manifest>