**Authentication & Authorization in ASP .NET Core 3.1**

By Shahed C on January 6, 2020

[13 Replies](https://wakeupandcode.com/authentication-authorization-in-asp-net-core-3-1/#comments)

[](https://wakeupandcode.com/aspnetcore/#aspnetcore2020)

This is the first 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.

If you need some guidance before you get started with this series, check out my [late 2019 posts](https://wakeupandcode.com/aspnetcore/#aspnetcore2020prelude), which serve as a prelude to the 2020 series:

* [ASP .NET Core code sharing between Blazor, MVC and Razor Pages](https://wakeupandcode.com/asp-net-core-code-sharing-between-blazor-mvc-and-razor-pages/)
* [Hello ASP .NET Core v3.1!](https://wakeupandcode.com/hello-asp-net-core-v3-1/)
* [NetLearner on ASP .NET Core 3.1](https://wakeupandcode.com/netlearner-on-asp-net-core-3-1/)

**NetLearner on GitHub**:

* Repository: <https://github.com/shahedc/NetLearnerApp>
* v0.1-alpha release: <https://github.com/shahedc/NetLearnerApp/releases/tag/v0.1-alpha>

**In this Article:**

* A is for Authentication & Authorization
* Authentication in ASP .NET Core
* Authentication in NetLearner
* Authorization in ASP.NET Core (MVC)
* Authorization in ASP.NET Core (Razor Pages)
* Authorization in ASP.NET Core (Blazor)
* Testing Authorization in NetLearner
* Other Authorization Options
* References

**A is for Authentication & Authorization**

*Authentication*and *Authorization*are two different things, but they also go hand in hand. Think of Authentication as letting someone into your home and Authorization as allowing your guests to do specific things once they’re inside (e.g. wear their shoes indoors, eat your food, etc). In other words, Authentication lets your web app’s users identify themselves to get access to your app and Authorization allows them to get access to specific features and functionality.

In this article, we will take a look at the [NetLearner](https://wakeupandcode.com/netlearner-on-asp-net-core-3-1/) app, on how specific pages can be restricted to users who are logged in to the application. Throughout the series, I will try to focus on new code added to NetLearner or build a smaller sample app if necessary.

**Authentication in ASP .NET Core**

The quickest way to add authentication to your ASP .NET Core app is to use one of the pre-built templates with one of the Authentication options. The examples below demonstrate both the CLI commands and Visual Studio UI.

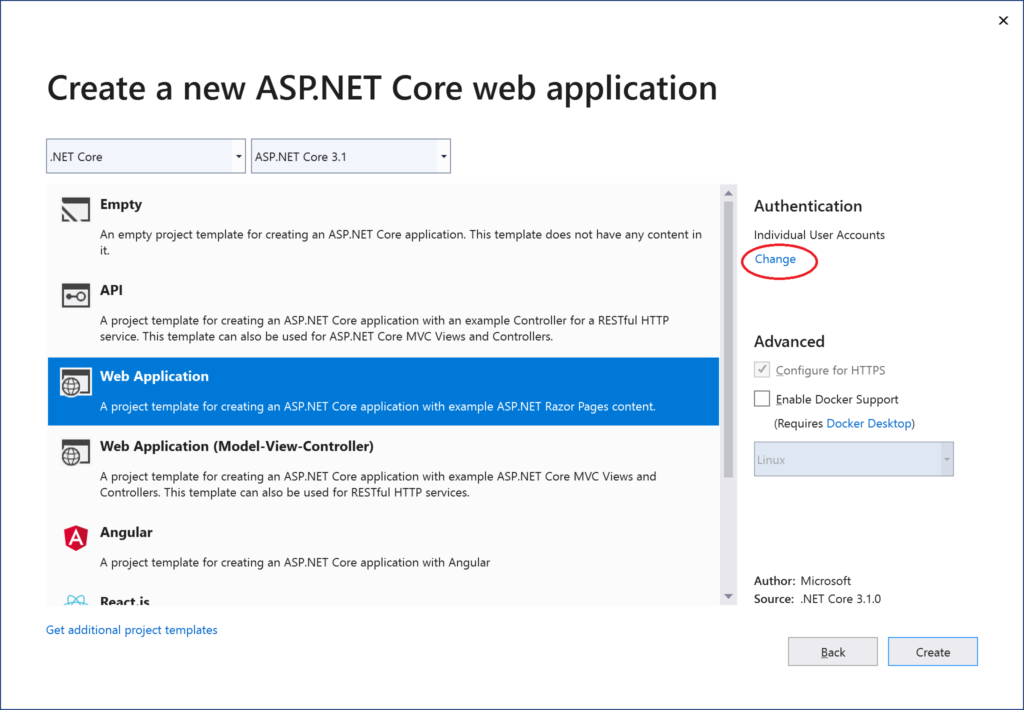
Here are the CLI Commands for MVC, Razor Pages and Blazor (Server), respectively:

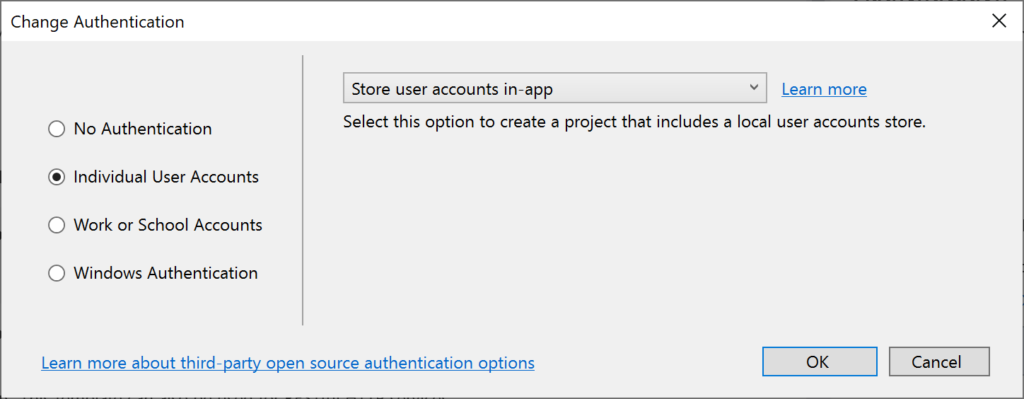
> dotnet new mvc --auth Individual -o mvcsample  
> dotnet new webapp --auth Individual -o pagessample  
> dotnet new blazorserver --auth Individual -o blazorsample

Things to note:

* The **dotnet new** command is followed by the template name (mvc, webapp, blazorserver).
* The –auth option allows you to specify the authentication type, e.g. Individual
* The -o option is an optional parameter that provides the output folder name for the project to be created in.

Here is the opening dialog in Visual Studio 2019, for creating a new project with Authentication:

 Change Authentication upon creating a new project

 Select Authentication Type

The above example uses “Individual” authentication, which offers a couple of options:

* **Store user accounts in-app**: includes a local user accounts store
* **Connect to an existing user store in the cloud**: connect to an existing Azure AD B2C application

Even if I choose to start with a local database, I can update the connection string to point to a SQL Server instance on my network or in the cloud, depending on which configuration is being loaded. If you’re wondering where your Identity code lives, check out my 2019 post on [Razor UI Libraries](https://wakeupandcode.com/your-first-razor-ui-library-with-asp-net-core/), and jump to the last bonus section where Identity is mentioned.

From the documentation, the types of authentication are listed below.

* **None**: No authentication
* **Individual**: Individual authentication
* **IndividualB2C**: Individual authentication with Azure AD B2C
* **SingleOrg**: Organizational authentication for a single tenant
* **MultiOrg**: Organizational authentication for multiple tenants
* **Windows**: Windows authentication

To get help information about Authentication types, simply type ––help after the ––auth flag, e.g.

> dotnet new webapp --auth --help

**Authentication in NetLearner**

Within my NetLearner MVC app, the following snippets of code are added to the [Startup.cs configuration](https://github.com/shahedc/NetLearnerApp/blob/main/src/NetLearner.Mvc/Startup.cs):

public void ConfigureServices(IServiceCollection services)  
{  
...  
 services.AddDbContext<LibDbContext>(options =>  
 {  
 options  
 .UseSqlServer(Configuration.GetConnectionString("DefaultConnection"),  
 assembly =>  
 assembly.MigrationsAssembly  
 (typeof(LibDbContext).Assembly.FullName));  
 });  
  
 services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = true)  
 .AddEntityFrameworkStores<LibDbContext>();  
...  
} public void Configure(IApplicationBuilder app, IHostingEnvironment env)  
{  
...  
 app.UseStaticFiles();  
...  
 app.UseAuthentication();  
 app.UseAuthorization();   
...  
 app.Endpoints(...);  
}

In the above, note that:

* The **ConfigureServices**() method has calls to services.**AddDbContext** and server.**AddDefaultIdentity**. The call to add a DB Context will vary depending on which data store you choose for authentication. The call to [AddDefaultIdentity](https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity) ensures that your app calls AddIdentity, AddDefaultUI and AddDefaultTokenProviders to add common identity features and user Register/Login functionality.
* The **Configure**() method has calls to app.UseAuthentication and app.UseAuthorization to ensure that authentication and authorization are used by your web app. Note that this appears *after* app.UseStaticFiles() but *before* app.UseEndpoints() to ensure that static files (html, css, js, etc) can be served without any authentication but MVC application-controlled routes and views/pages will follow authentication rules.
* Similar to the MVC web project, you can also browse the Startup.cs file for the equivalent [Razor Pages](https://github.com/shahedc/NetLearnerApp/blob/main/src/NetLearner.Pages/Startup.cs) and [Blazor](https://github.com/shahedc/NetLearnerApp/blob/main/src/NetLearner.Blazor/Startup.cs) web app projects.

**Authorization in ASP.NET Core (MVC)**

Even after adding authentication to a web app using the project template options, we can still access many parts of the application without having to log in. In order to restrict specific parts of the application, we will implement Authorization in our app.

If you’ve already worked with ASP .NET Core MVC apps before, you may be familiar with the **[Authorize]** attribute. This attribute can be added to a controller at the class level or even to specific action methods within a class.

[Authorize]  
public class SomeController1: Controller  
{  
 // this controller class requires authentication  
 // for all action methods  
 public ActionResult SomeMethod()  
 {  
 //  
 }   
}  
  
public class SomeController2: Controller  
{  
 public ActionResult SomeOpenMethod()  
 {  
 }  
  
 [Authorize]  
 public ActionResult SomeSecureMethod()  
 {  
 // this action method requires authentication  
 }  
}

*Well, what about Razor Pages in ASP .NET Core? If there are no controller classes, where would you add the****[Authorize]****attribute?*

**Authorization in ASP.NET Core (Razor Pages)**

For Razor Pages, the quickest way to add Authorization for your pages (or entire folders of pages) is to update your **ConfigureServices()** method in your Startup.cs class, by calling **AddRazorPagesOptions()** after **AddMvc()**. The [NetLearner configuration](https://github.com/shahedc/NetLearnerApp/blob/main/src/NetLearner.Pages/Startup.cs) includes the following code:

services.AddRazorPages()  
 .AddRazorPagesOptions(options =>  
 {  
 options.Conventions.AuthorizePage("/LearningResources/Create");  
 options.Conventions.AuthorizePage("/LearningResources/Edit");  
 options.Conventions.AuthorizePage("/LearningResources/Delete");  
 options.Conventions.AuthorizePage("/ResourceLists/Create");  
 options.Conventions.AuthorizePage("/ResourceLists/Edit");  
 options.Conventions.AuthorizePage("/ResourceLists/Delete");  
 options.Conventions.AllowAnonymousToPage("/Index");  
 });

The above code ensures that the CRUD pages for Creating, Editing and Deleting any of the LearningResources are only accessible to someone who is currently logged in. Each call to **AuthorizePage**() includes a specific page name identified by a known route. In this case, the [LearningResources folder](https://github.com/shahedc/NetLearnerApp/tree/main/src/NetLearner.Pages/Pages/LearningResources) exists within the Pages folder of the application.

Finally, the call to **AllowAnonymousPage()** ensures that the app’s [index](https://github.com/shahedc/NetLearnerApp/blob/main/src/NetLearner.Pages/Pages/LearningResources/Index.cshtml) page (at the root of the Pages folder) is accessible to any user without requiring any login.

If you still wish to use the **[Authorize]** attribute for Razor Pages, you may apply this attribute in your PageModel classes for each Razor Page, as needed. If I were to add it to one of my Razor Pages in the LearningResources folder, it could look like this:

[Authorize]  
public class CreateModel : PageModel  
{  
    ...  
}

**Authorization in Blazor**

In a Blazor project, you can wrap elements in an <AuthorizeView> component, which may contain nested elements named <Authorized>, <NotAuthorized> and <Authorizing>.

<AuthorizeView>  
 <Authorized Context="Auth">  
 authorized elements go here  
 </Authorized>  
 <NotAuthorized>  
 anonymous-accessed elements  
 </NotAuthorized>  
 <Authorizing>  
 waiting message...  
 </Authorizing>  
</AuthorizeView>

The root element here has the following characteristics:

* <**AuthorizeView**> element used as a container for nested elements
* Optional role attribute, e.g. <**AuthorizeView** Roles=”Admin,RoleA”> used to set a comma-separated list roles that will be used to determine who can view the authorized nested elements
* Optional context attribute, e.g. <**AuthorizeView** Context=”Auth”> to define a custom context, to avoid any [conflicts with nested comments](https://github.com/chanan/BlazorStrap/issues/149#issuecomment-519691692)

The nested elements represent the following:

* **Authorized**: shown to users who have been authorized to view and interact with these elements
* **NotAuthorized**: shown to users who have *not* been authorized
* **Authorizing**: temporary message shown while authorization is being processed

**Testing Authorization in NetLearner**

When I run my application, I can register and log in as a user to create new Learning Resources. On first launch, I have to apply migrations to create the database from scratch. Please refer to my 2018 post on [EF Core Migrations](https://wakeupandcode.com/ef-core-migrations-in-asp-net-core/) to learn how you can do the same in your environment.

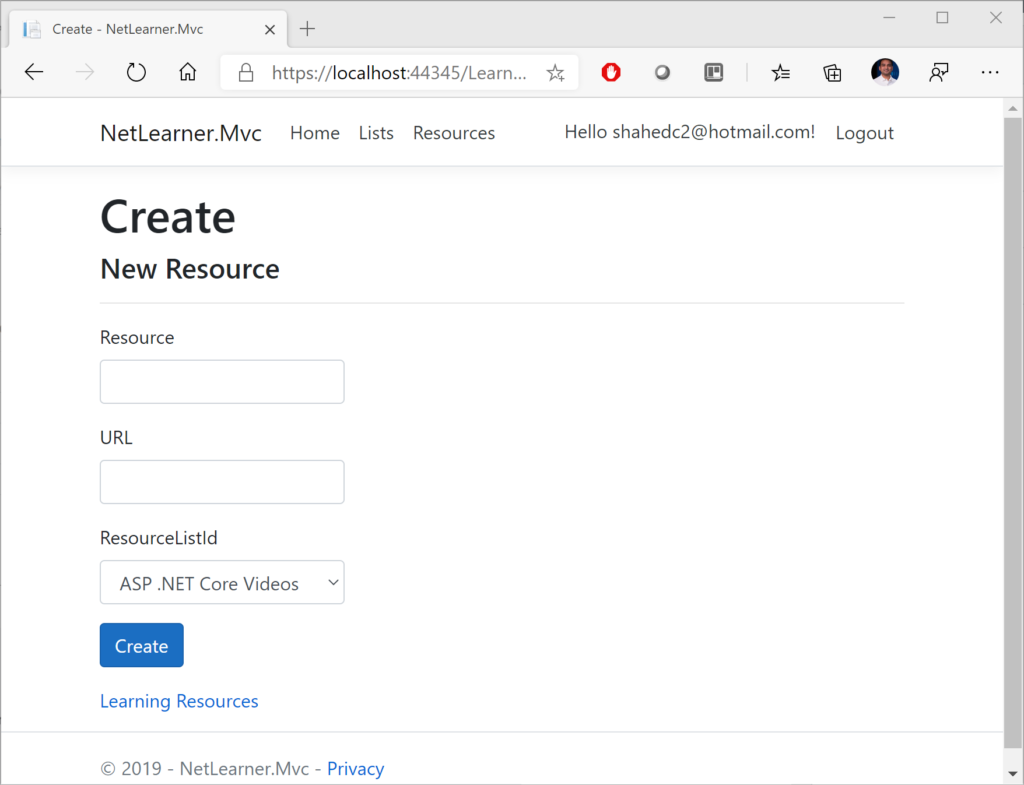
**NOTE:** the registration feature for each web app has been disabled by default. To enable registration, please do the following:

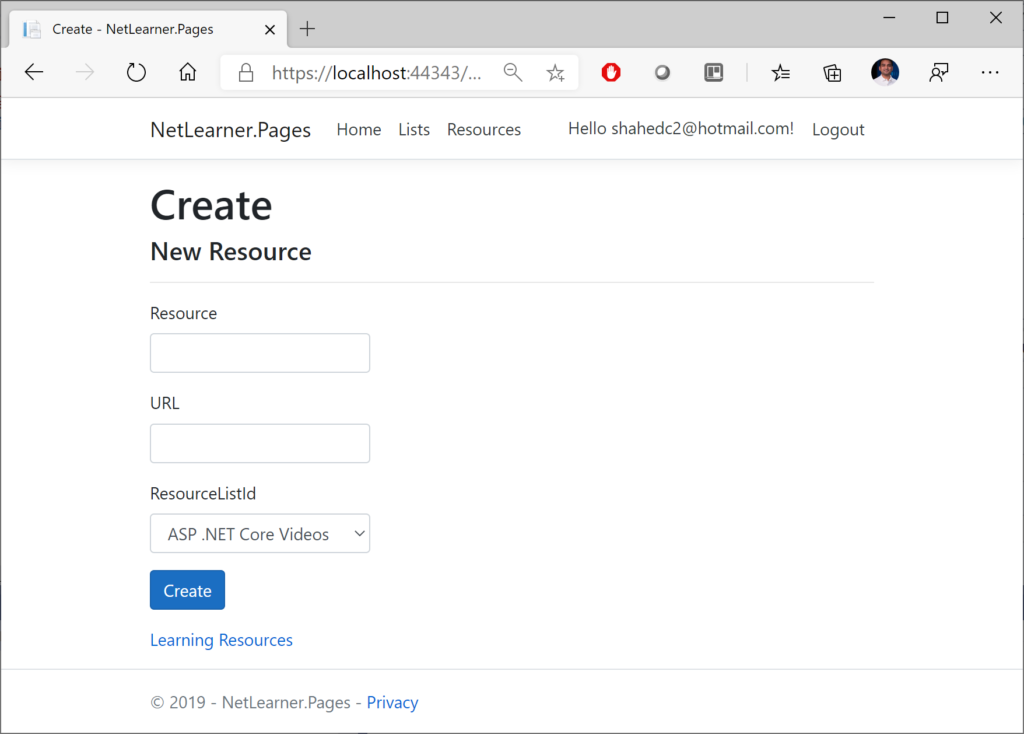
1. Locate scaffolded Identity pages under /Areas/Identity/Pages/Account/

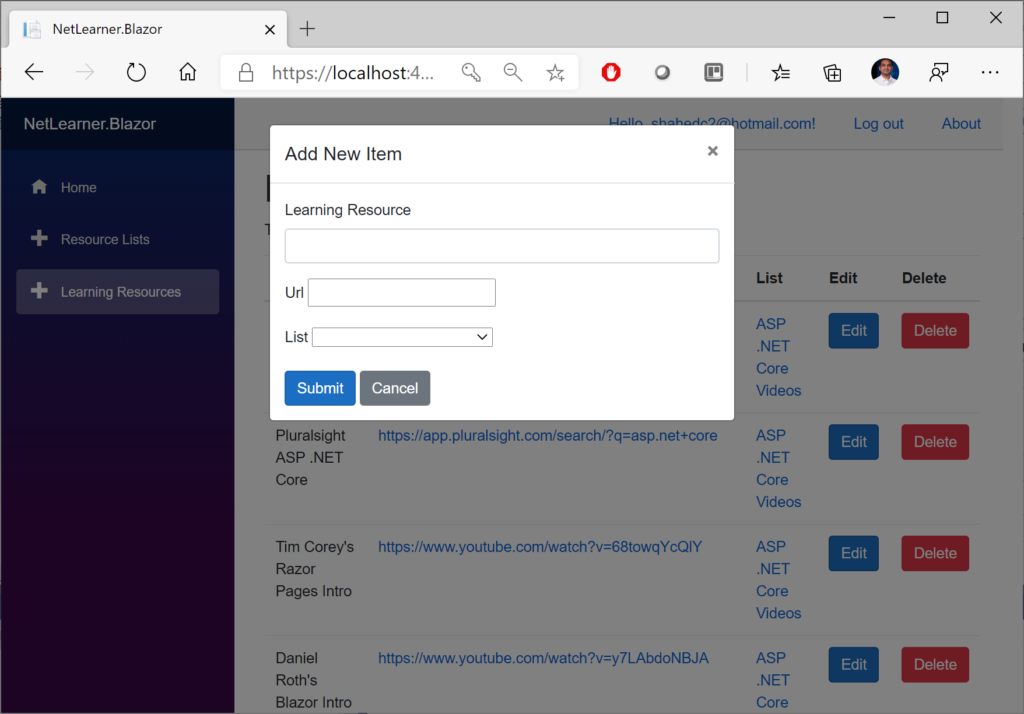
* [MVC](https://github.com/shahedc/NetLearnerApp/tree/main/src/NetLearner.Mvc/Areas/Identity/Pages/Account)
* [Razor Pages](https://github.com/shahedc/NetLearnerApp/tree/main/src/NetLearner.Pages/Areas/Identity/Pages/Account)
* [Blazor](https://github.com/shahedc/NetLearnerApp/tree/main/src/NetLearner.Blazor/Areas/Identity/Pages/Account)

1. In Register.cshtml, update the page to include environments in addition to Development, if desired.
2. In Register.cshtml.cs, replace [**Authorize**] with [**AllowAnonymous**] to allow access to registration

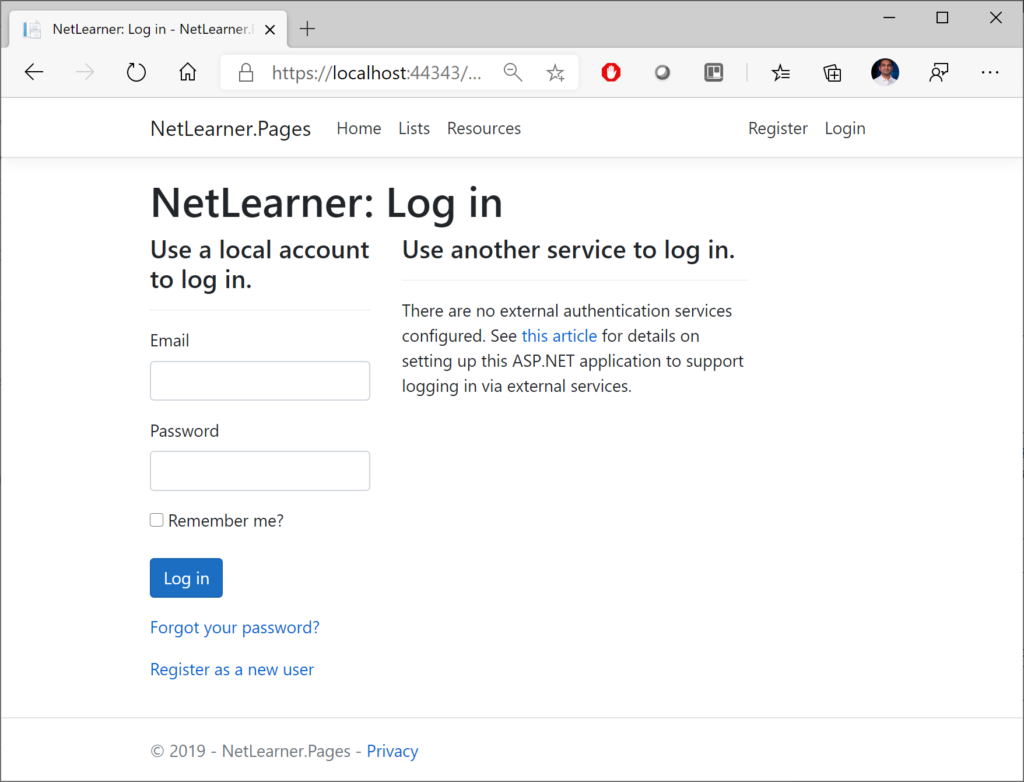
Here are the screenshots of the Create page for a user who is logged in:

NetLearner MVC: Create New Resource

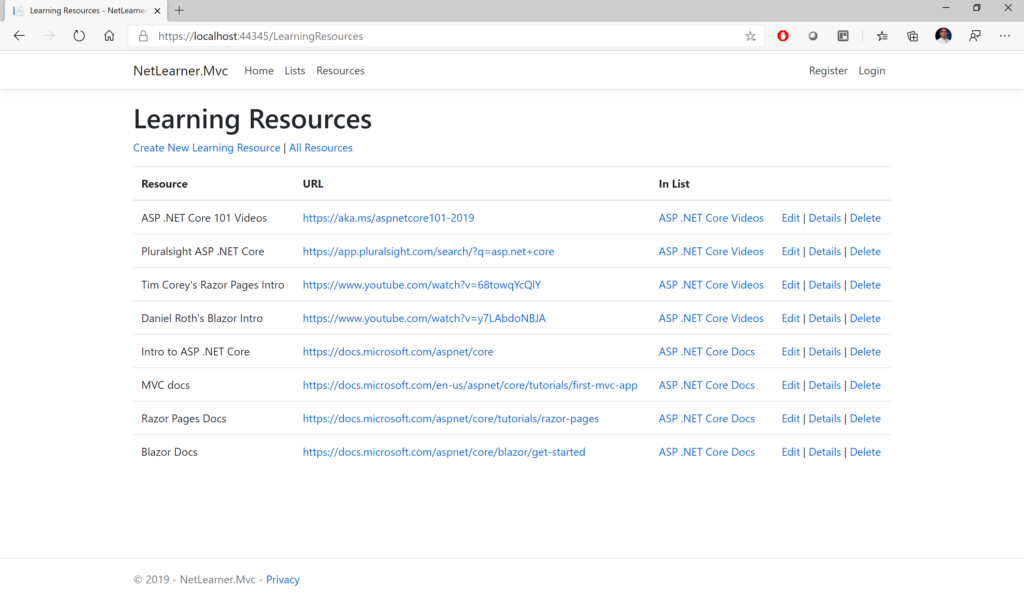
 NetLearner Razor Pages: Create New Resource

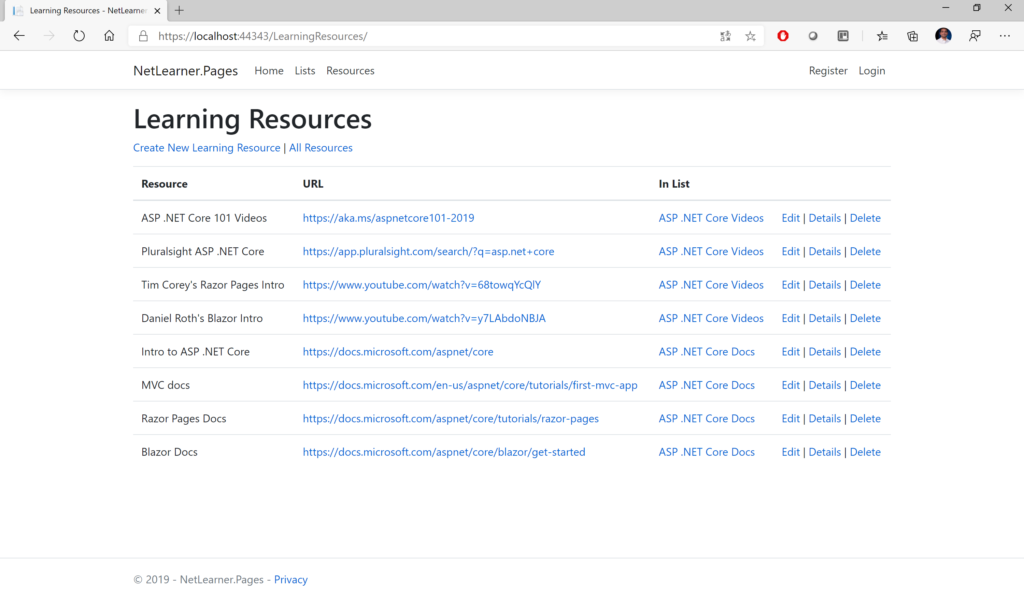
 NetLearner Blazor: Create New Resource

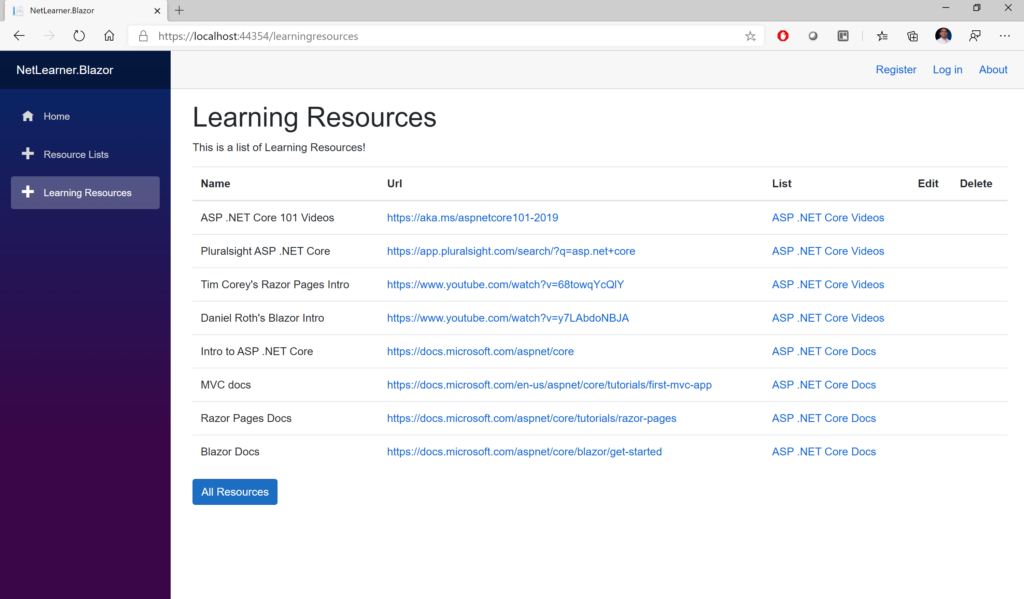
Here’s a screenshot of the page that an “anonymous” user sees when no one is logged in, indicating that the user has been redirected to the Login page. Note that all 3 web apps (MVC, Razor Pages and Blazor) have similar Identity pages. To allow manual customization, they were also auto-generated via scaffolding and included in all 3 projects.

NetLearner: Log in

Here are the screenshots showing the list of Learning Resources, visible to anyone whether they’re logged in or not:

 NetLearner MVC: List of Learning Resources

 NetLearner Razor Pages: List of Learning Resources

 NetLearner Blazor: List of Learning Resources

**Other Authorization Options**

Razor Pages have multiple ways of restricting access to pages and folders, including the following methods (*as described in the official docs*):

* **AuthorizePage**: Require authorization to access a page
* **AuthorizeFolder**: Require authorization to access a folder of pages
* **AuthorizeAreaPage**: Require authorization to access an area page
* **AuthorizeAreaFolder**: Require authorization to access a folder of areas
* **AllowAnonymousToPage**: Allow anonymous access to a page
* **AllowAnonymousToFolder**: Allow anonymous access to a folder of pages

You can get more information on all of the above methods at the following URL:

* Razor Pages authorization conventions in ASP.NET Core: <https://docs.microsoft.com/en-us/aspnet/core/security/authorization/razor-pages-authorization>
* Detailed Tutorial: <https://docs.microsoft.com/en-us/aspnet/core/security/authorization/secure-data>

**References**

To learn more about Authentication, Authorization and other related topics (e.g. Roles and Claims), check out the official docs:

* Razor Pages authorization conventions in ASP.NET Core: <https://docs.microsoft.com/en-us/aspnet/core/security/authorization/razor-pages-authorization>
* Is the Authorize attribute needed in Razor Pages? What about Roles, Claims and Policies?: <https://github.com/aspnet/Docs/issues/6301>
* [Authorize] Filter methods for Razor Pages in ASP.NET Core: <https://docs.microsoft.com/en-us/aspnet/core/razor-pages/filter/#authorize-filter-attribute>
* Simple authorization in ASP.NET Core: <https://docs.microsoft.com/en-us/aspnet/core/security/authorization/simple>
* Role-based authorization in ASP.NET Core: <https://docs.microsoft.com/en-us/aspnet/core/security/authorization/roles>
* Claims-based authorization in ASP.NET Core: <https://docs.microsoft.com/en-us/aspnet/core/security/authorization/claims>
* Policy-based authorization in ASP.NET Core: <https://docs.microsoft.com/en-us/aspnet/core/security/authorization/policies>
* Blazor Authentication & Authorization: <https://docs.microsoft.com/en-us/aspnet/core/security/blazor/?view=aspnetcore-3.1>