Implement Identity On Existing ASP.NET Project

January 25, 2022

ASP.NET Identity is a .NET library that allows you to easily implement a **user management system** for your application. It offers out of the box the following: **registration**, login, **two-factor authentication**, external login providers, email confirmation, security roles, and claims.

If you create an application from start, you have the option to choose to add the Identity.

What if you forgot to check that option and you want to add Identity to an existing project?

Configure Identity in .NET Core 6

Programming in CSharp

1. Install the NuGet Packages:
2. Configure the connection string in appsettings file.
3. Create the DbContext class
4. Register Database Context service in the Startup
5. Register the Identity service
6. Configure the Authentification and Authorization middleware:
7. Add the first migration
8. Update the database
9. Add Scaffolded Item
10. Solving Identity Scaffold Errors
11. Modify the layout and include the Login partial

Install the NuGet Packages:

```
Install-Package Microsoft.AspNetCore.Identity
Install-Package Microsoft.AspNetCore.Identity.EntityFrameworkCore
Install-Package Microsoft.EntityFrameworkCore.SqlServer
Install-Package Microsoft.EntityFrameworkCore.Tools
Install-Package Microsoft.EntityFrameworkCore.Design

-- If you want to use the default UI offered by ASP, install the following libr
Install-Package Microsoft.AspNetCore.Identity.UI

-- For scaffold
Install-Package Microsoft.VisualStudio.Web.CodeGeneration.Design
```

For this tutorial, I assume that you use Entity Framework with the code first migrations approach. I have used the SQL Server library, but you can find the corresponding library for other types. Also, I will scaffold the default views offered by ASP, but you can use your custom views or you can modify the

Configure the connection string in *appsettings file*.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=localhost\\SQLEXPRESS;Initial Catalo
  },
  "AllowedHosts": "*"
}
```

If you don't have a database, install the SQL Express. Follow this tutorial if you don't know how to do it.

Create the DbContext class

The Identity library comes with **IdentityDbContext** class which already extends the DbContext class. So, we have to extend this class. Create the following class only if you don't have a DbContext already:

```
public class AppDbContext : IdentityDbContext<IdentityUser, IdentityRole,
{
    public AppDbContext(DbContextOptions<AppDbContext> options): bas
    {
    }
}
```

If you already have a DbContext, make sure that you inherit the IdentityDbContext class.

Register Database Context service in the Startup

```
builder.Services.AddDbContext<AppDbContext>(options =>
        options.UseSqlServer(
            builder.Configuration.GetConnectionString("DefaultConnection")));
```

If you don't use SQL Server, change the *UseSqlServer* with the corresponding method.

Register the Identity service

```
builder.Services.AddDefaultIdentity<IdentityUser>
    (options =>
{
    options.SignIn.RequireConfirmedAccount = true;
    options.Password.RequireDigit = false;
    options.Password.RequiredLength = 6;
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequireUppercase = false;
    options.Password.RequireLowercase = false;
})
.AddEntityFrameworkStores<AppDbContext>();
```

For this demo, I don't want to confirm the account with a confirmation email. Also, the password can be simple.

Configure the Authentification and Authorization middleware:

```
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();

app.MapRazorPages();
app.Run();
```

Make sure that you add the Authentification middleware after the Routing middleware.

Add the first migration

Open the Package Manager Console and enter:

```
Add-Migration InitialMigration -verbose
```

The verbose option is not mandatory, but you can see step-by-step what was executed. Also, you can see detailed errors. Check the generated class.

Update the database

```
Update-Database -verbose
```

After you execute the above command, the tables to store users/roles were created. You can check them in the database.

Add Scaffolded Item

Right-click on the project and select Add New Scaffold Item.

You must set up the **layout of your application** and the **data context class**. Then you can select the views that you want to customize.

Solving Identity Scaffold Errors

Probably, you have received the following error: *Scaffolding failed. The path is empty. (Parameter 'path')*

To solve this error, remove Microsoft.AspNetCore.Identity package from the csproj file:

```
Uninstall-Package Microsoft.AspNetCore.Identity
```

Try again to add the Scaffold item (step 9). There is a big chance to get another error, telling you the following message: *Missing IdentityHostingStartup.cshtml*.

The solution came from this GitHub issue.

2. Add this file to the location where is missing:

https://github.com/dotnet/Scaffolding/blob/main/src/Scaffoldin

Now the scaffold should work. **The files were created in the Area folder and a LoginPartial was created in the Pages/Controllers folder**.

Modify the layout and include the Login partial

In the *_Layout.cshtml* file, make sure that you include the login partial. This partial include the labels.

```
<nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light
    <div class="container">
        <a class="navbar-brand" asp-area="" asp-page="/Index">NotIdentity
        <button class="navbar-toggler" type="button" data-bs-toggle="colla
                aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="navbar-collapse collapse d-sm-inline-flex justify-content-
            <ul class="navbar-nav flex-grow-1">
                <li class="nav-item">
                    <a class="nav-link text-dark" asp-area="" asp-page="/Index">
                </li>
                <li class="nav-item">
                    <a class="nav-link text-dark" asp-area="" asp-page="/Privacy'
                </li>
            </ul>
            <partial name="_LoginPartial" />
        </div>
    </div>
</nav>
```

In the end, **you have implemented the login system**. When you run the application you must see the login button. You can scaffold more views if you don't want to use the default ones.

Dotnet

< Generate strings from a Regex pattern in C#

> The Best Guide to Enhance Your Career as a C# Developer

3 thoughts on "Implement Identity On Existing ASP.NET Project"

Marco

September 2, 2022 at 7:58 pm

hi, first congratulations for your post, I followed carefully yours but I can not call but in the end I can not call the pages I attach the code :

Register

Login

I see Register and Login link but when I click nothing happens ,have you some suggestion? ,
Thankyou in advance
Regards
Marco

Reply

anghelvalentin

September 7, 2022 at 2:20 pm

It seems that you didn't attach a href attribute to you anchors.
Take a look at this example:

Login

Reply

James

December 19, 2022 at 5:02 pm

Best tutorial online I found for this online...just followed step by step and was good!

Reply

Leave a Comment

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

Post Comment

Categories

Dotnet

SEO In ASP.NET

SQL For Programmers

Tech Tips

Visual Studio

Programming in CSharp

Newsletter

First Name First Name

Email address: Your email address

Sign up