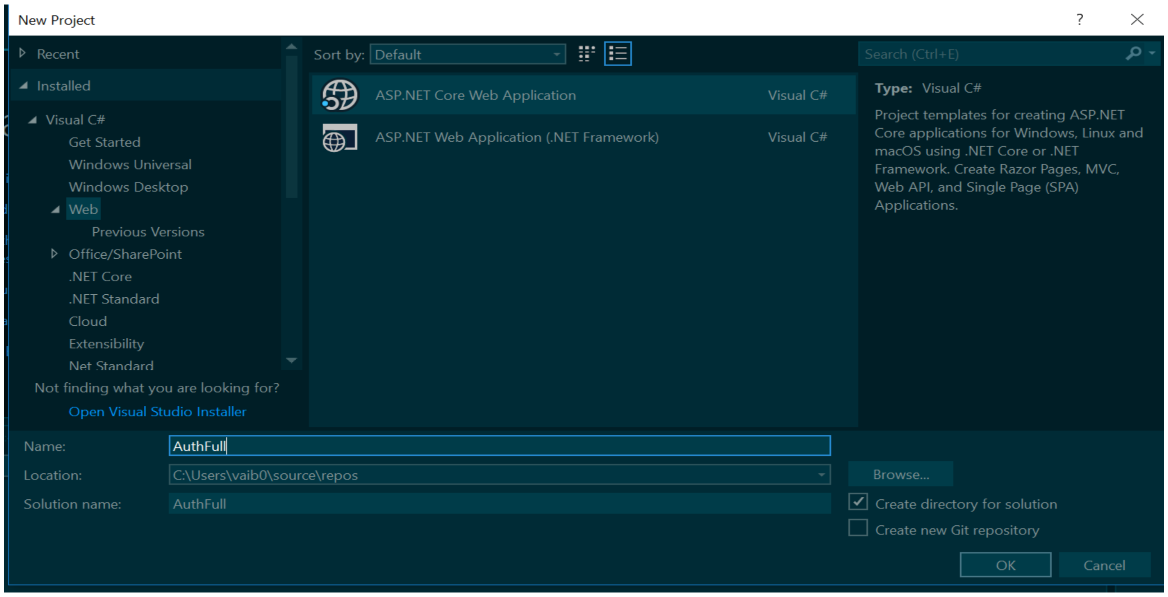
Setting up solution for Authentication and Authorization in ASP.Net Core 2.2

This document gives a step by step approach to set up the solution for implementing Authentication and Authorization in ASP.Net Core 2.2

1. Creating solution with single user authentication.



New Project

Recent
Installed
Visual C#
Get Started
Windows Universal
Windows Desktop
Web
Previous Versions
Office/SharePoint
.NET Core
.NET Standard
Cloud
Extensibility
Net Standard
Not finding what you are looking for?
Open Visual Studio Installer

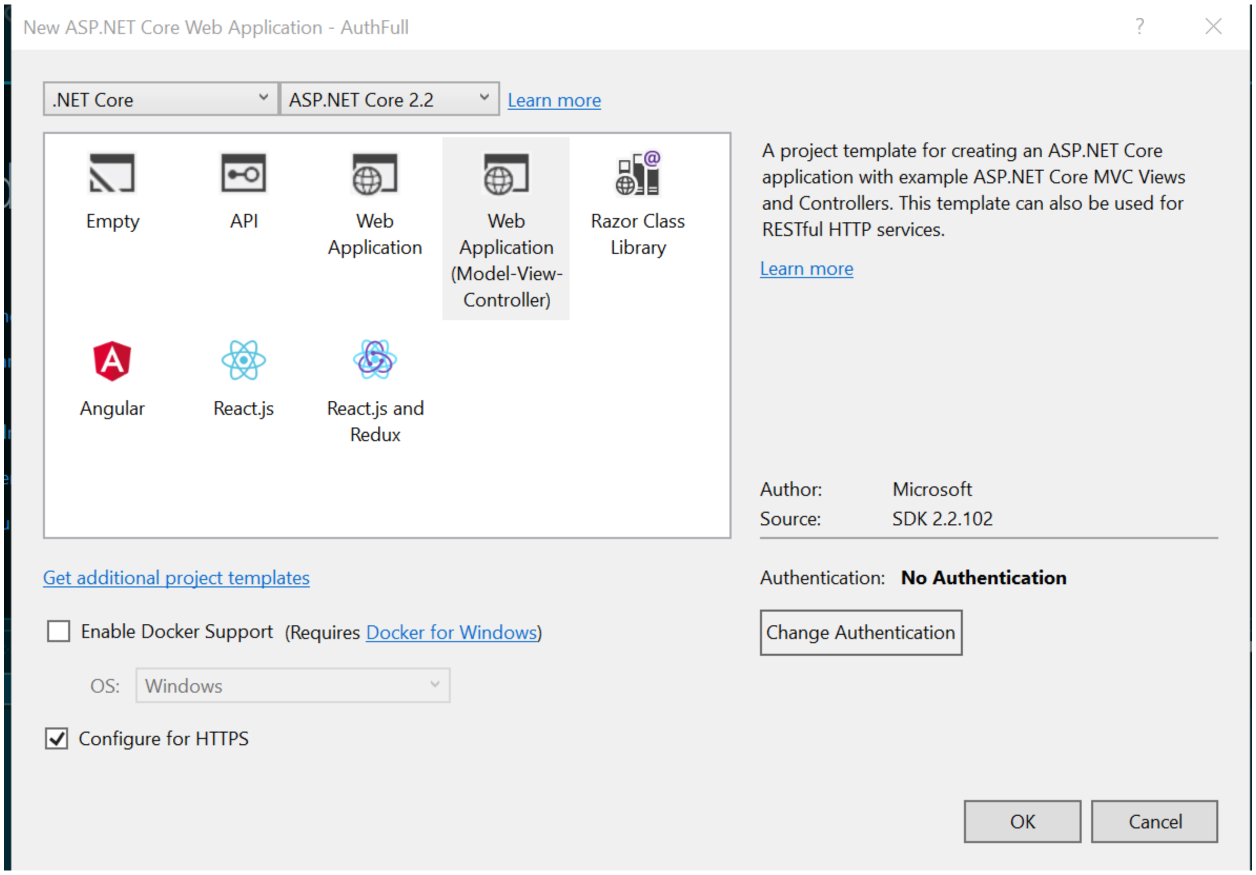
Sort by: Default

ASP.NET Core Web Application — Visual C#
ASP.NET Web Application (.NET Framework) — Visual C#

Type: Visual C#
Project templates for creating ASP.NET Core applications for Windows, Linux and macOS using .NET Core or .NET Framework. Create Razor Pages, MVC, Web API, and Single Page (SPA) Applications.

Name: AuthFull
Location: C:\Users\vaib0\source\repos
Solution name: AuthFull
✓ Create directory for solution
Create new Git repository

Click ok and you will get the following window. Click on change authentication and press OK on the following window.



New ASP.NET Core Web Application - AuthFull

.NET Core | ASP.NET Core 2.2 | Learn more

Empty
API
Web Application
Web Application (Model-View-Controller)
Razor Class Library
Angular
React.js
React.js and Redux

A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

Learn more

Author: Microsoft
Source: SDK 2.2.102

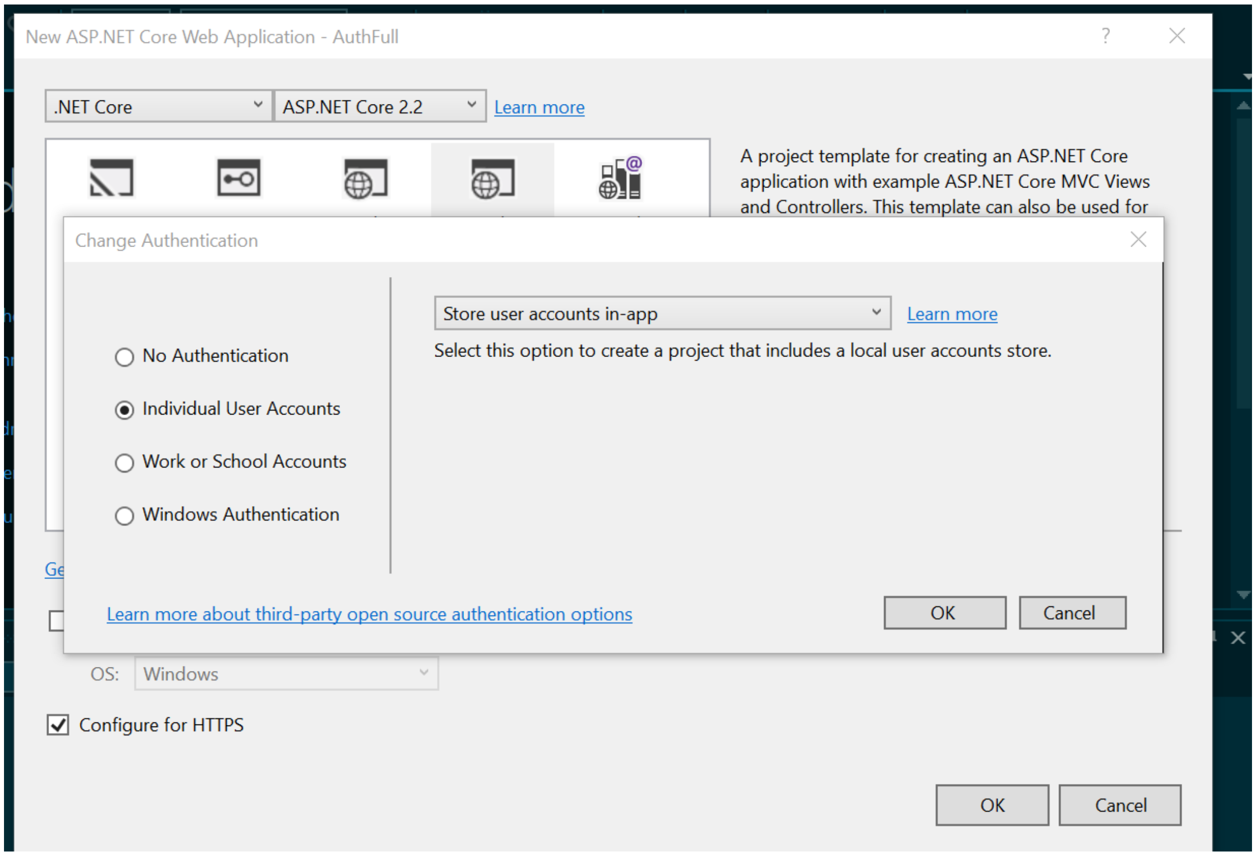
Authentication: **No Authentication**

Get additional project templates

Enable Docker Support (Requires Docker for Windows)

OS: Windows

✓ Configure for HTTPS



New ASP.NET Core Web Application - AuthFull

.NET Core | ASP.NET Core 2.2 | Learn more

A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for

Change Authentication

- No Authentication
- [✓ Individual User Accounts]
- Work or School Accounts
- Windows Authentication

Store user accounts in-app | Learn more

Select this option to create a project that includes a local user accounts store.

Learn more about third-party open source authentication options

OK | Cancel

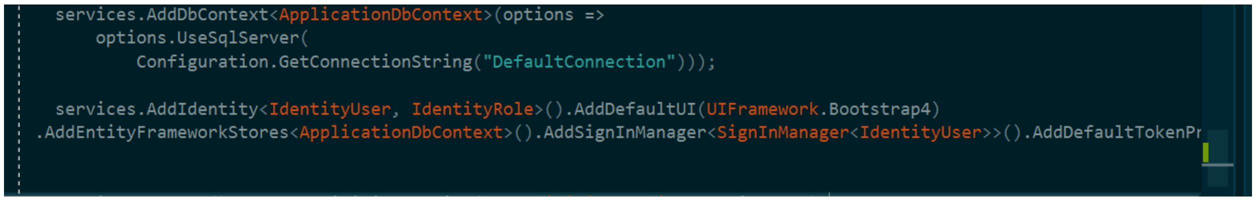
OS: Windows

[✓] Configure for HTTPS

OK | Cancel

Select Individual User Authentication and press OK.

2. Registering Identity service in startup.cs



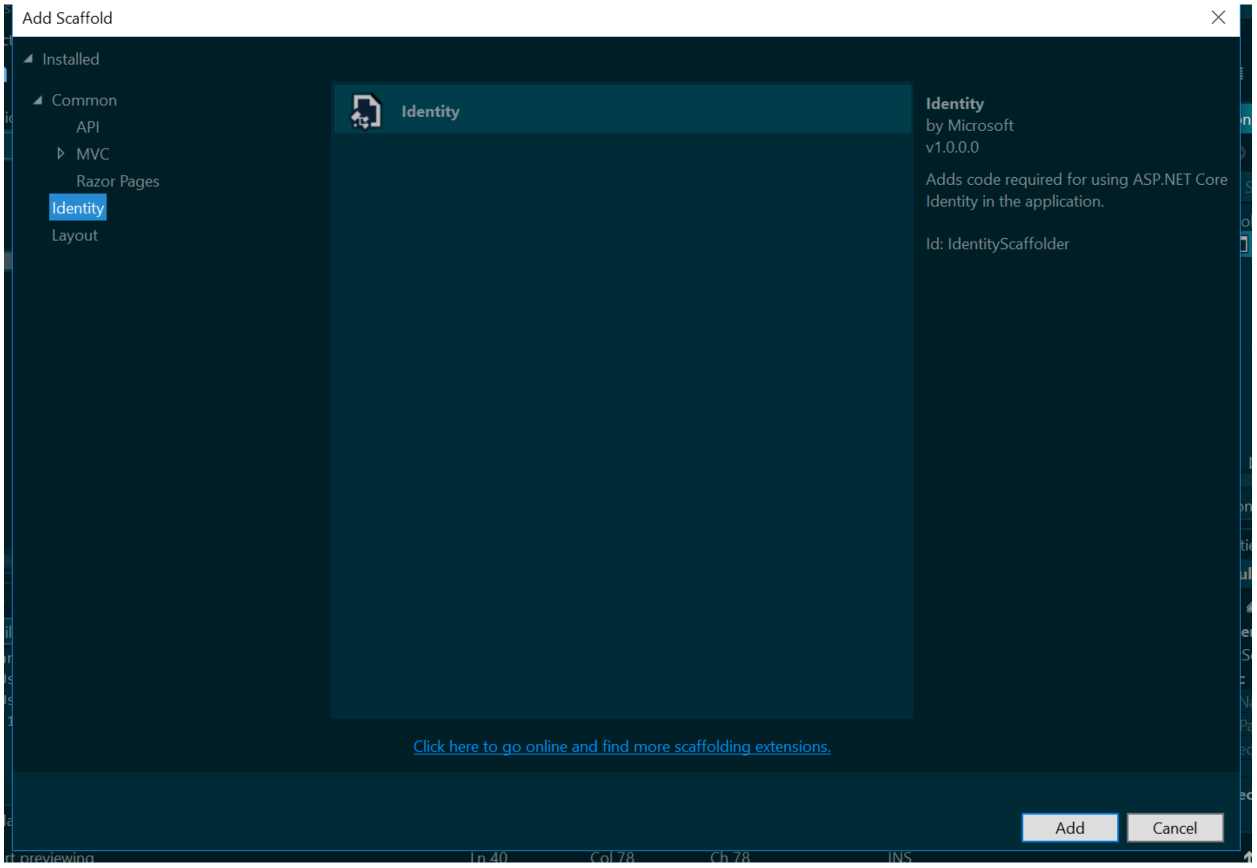
```
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<IdentityUser, IdentityRole>().AddDefaultUI(UIFramework.Bootstrap4)
  .AddEntityFrameworkStores<ApplicationDbContext>().AddSignInManager<SignInManager<IdentityUser>>().AddDefaultTokenPr
```

Add services.AddIdentity() in startup.cs. Important thing to note here is IdentityUser and IdentityRole. Please note we are directly using the IdentityUser classes of the Identity Framework in ASP.Net Core . We can make our own classes (AppUser and AppRole inheriting from IdentityUser and IdentityRole respectively) and use it in place of IdentityUser and IdentityRole.

Remove services.AddDefaultIdentity.

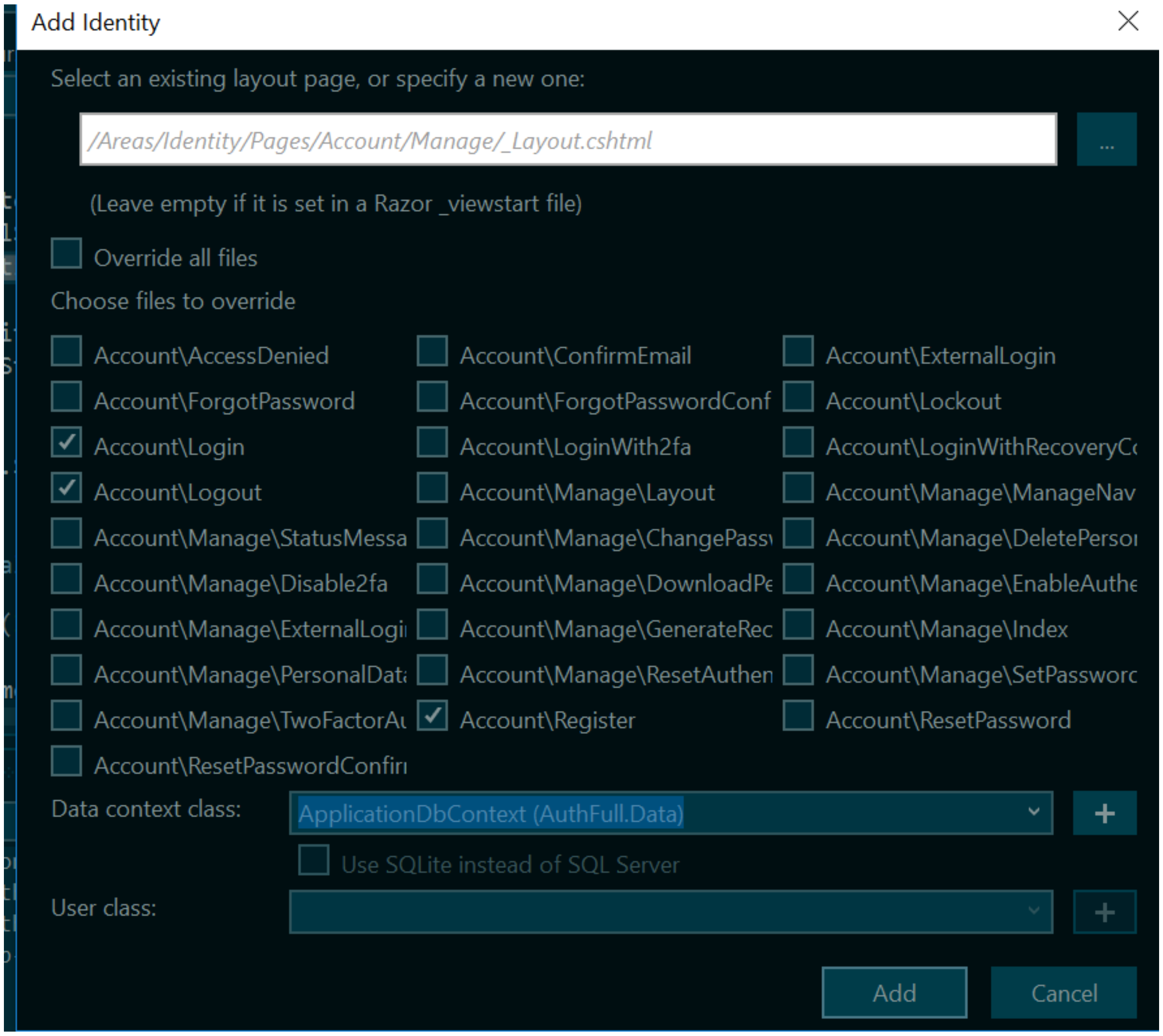
3. Once done with the above step, we will proceed to Scaffolding of the Account related stuff.

Right click on project->click Add-> Add new Scaffold Item we will get the following window



Identity
by Microsoft
v1.0.0.0
Adds code required for using ASP.NET Core Identity in the application.
Id: IdentityScaffolder

Select Identity and press Add.


Add Identity

Select an existing layout page, or specify a new one:

/Areas/Identity/Pages/Account/Manage/_Layout.cshtml

(Leave empty if it is set in a Razor _viewstart file)

☐ Override all files

Choose files to override

☐ Account\AccessDenied	☐ Account\ConfirmEmail	☐ Account\ExternalLogin
☐ Account\ForgotPassword	☐ Account\ForgotPasswordConf	☐ Account\Lockout
✓ Account\Login	☐ Account\LoginWith2fa	☐ Account\LoginWithRecoveryC
✓ Account\Logout	☐ Account\Manage\Layout	☐ Account\Manage\ManageNav
☐ Account\Manage\StatusMessa	☐ Account\Manage\ChangePass	☐ Account\Manage\DeletePerso
☐ Account\Manage\Disable2fa	☐ Account\Manage\DownloadPe	☐ Account\Manage\EnableAuthe
☐ Account\Manage\ExternalLogi	☐ Account\Manage\GenerateRec	☐ Account\Manage\Index
☐ Account\Manage\PersonalDat	☐ Account\Manage\ResetAuthen	☐ Account\Manage\SetPasswor
☐ Account\Manage\TwoFactorA	✓ Account\Register	☐ Account\ResetPassword
☐ Account\ResetPasswordConfir		

Data context class: ApplicationDbContext (AuthFull.Data)

☐ Use SQLite instead of SQL Server

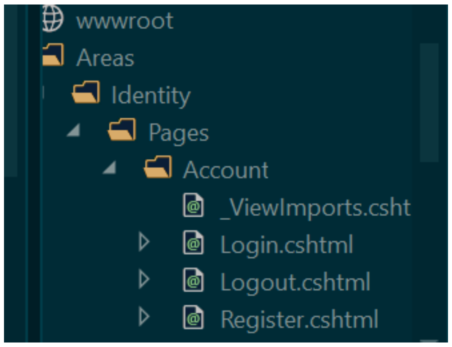
User class:

Add Cancel

In this example I am setting up only Login, Logout and Register views, hence I have selected just these three. You are free to override as many view as you want. Important thing to note here is to select your own DbContext Class (ApplicationDbContext in this case).

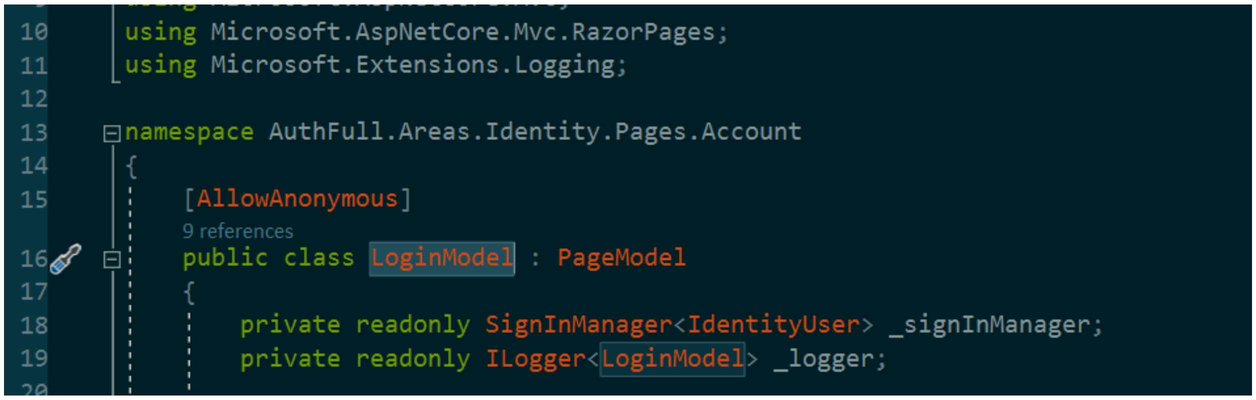
Press Add after selecting the Data context class.

Note the change in Area->Identity->Pages folder in the solution explorer



```
wwwroot
Areas
  Identity
    Pages
      Account
        _ViewImports.csht
        Login.cshtml
        Logout.cshtml
        Register.cshtml
```

A new Account folder is created with login, logout and register view. We do not need an account controller now. Just expand these files individually to see the .cshtml.cs files which has necessary methods and ViewModel.



```
10  using Microsoft.AspNetCore.Mvc.RazorPages;
11  using Microsoft.Extensions.Logging;
12
13  namespace AuthFull.Areas.Identity.Pages.Account
14  {
15      [AllowAnonymous]
        9 references
16      public class LoginModel : PageModel
17      {
18          private readonly SignInManager<IdentityUser> _signInManager;
19          private readonly ILogger<LoginModel> _logger;
```

The above screenshot is of the page login.cshtml.cs. Please note the line 18. Use IdentityUser if you have not created any custom ApplicationUser class inheriting from IdentityUser. Otheriwse leave everything as it is.

These steps along with the default settings in startup.cs sets up the Authentication part of the demo.

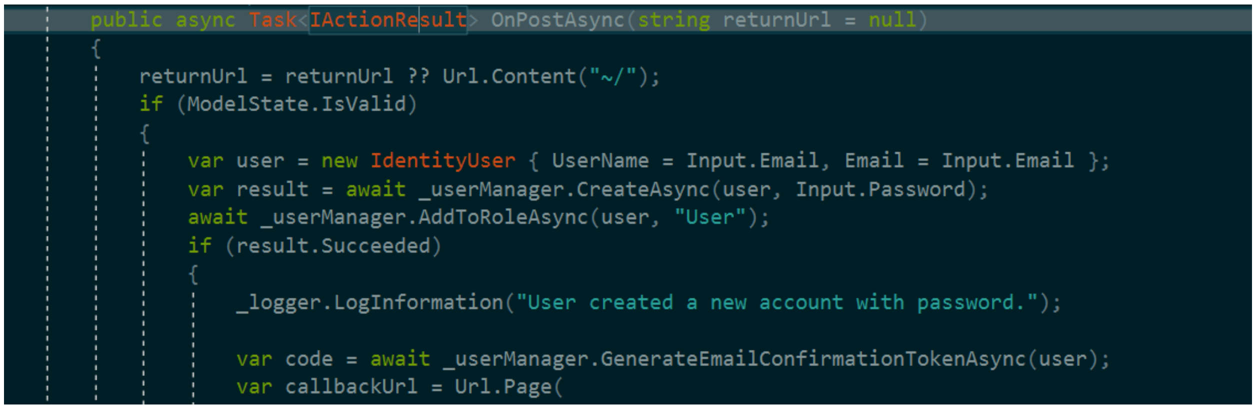
Adding Roles.

Please note the seed.cs class in Data folder. It contains CreateRoles method. The CreateRoles method is called from Program.cs. Please note the Addition of roles viz. Admin and User. Another thing is addition of the Admin. The Admin is added by fetching the detail from appsettings.json.

Once the program is executed and assuming that a brand new Migration is added and update-database is executed already, the solution is set up with Authentication and roles are added. An admin is also created by now.

Addition of “User” is done through the register page.

Now lets move to Areas->Identity->Pages->Account->Register.cshtml->Register.cshtml.cs and notice the following method.



```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl = returnUrl ?? Url.Content("~/");
    if (ModelState.IsValid)
    {
        var user = new IdentityUser { UserName = Input.Email, Email = Input.Email };
        var result = await _userManager.CreateAsync(user, Input.Password);
        await _userManager.AddToRoleAsync(user, "User");
        if (result.Succeeded)
        {
            _logger.LogInformation("User created a new account with password.");

            var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
            var callbackUrl = Url.Page(
```

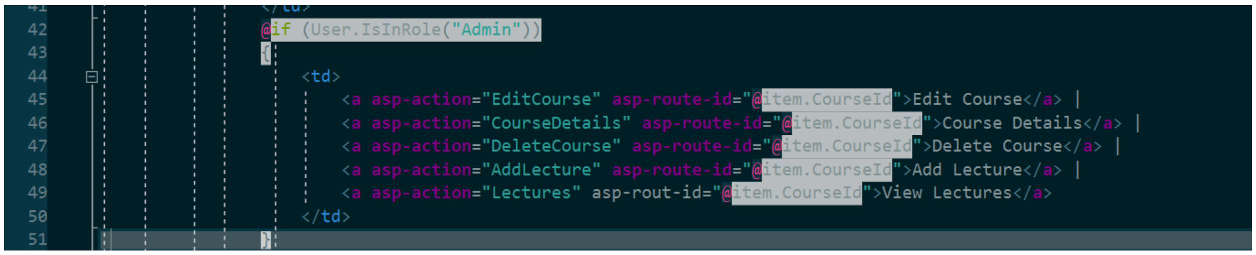
_userManager.AddToRoleAsync(user,”User”) adds the role “User” for the newly created user in the preceding line. This is done to provide the “User” status to the newly added user through the register page.

Authorization

Authorization is done using Authorize attribute. The demo app AuthFull contains the HomeController and Courses class just like the CoursesApp demo. Usage of [Authorize] attribute is evident and used over controller methods in HomeController.cs (Not for All the methods).

Implemeting Authorization in razor pages

There are situations where some functionalities should only be provided to a person with specific role (Create, Update, Delete etc functionalities to Admin only in our example). This can be achieved by showing certain controls to logged in user with a specific role. This can be done by adding a check as shown in following snapshot.



```
@if (User.IsInRole("Admin"))
{
    <td>
        <a asp-action="EditCourse" asp-route-id="@item.CourseId">Edit Course</a> |
        <a asp-action="CourseDetails" asp-route-id="@item.CourseId">Course Details</a> |
        <a asp-action="DeleteCourse" asp-route-id="@item.CourseId">Delete Course</a> |
        <a asp-action="AddLecture" asp-route-id="@item.CourseId">Add Lecture</a> |
        <a asp-action="Lectures" asp-rout-id="@item.CourseId">View Lectures</a>
    </td>
}
```

In the above code sample, the links inside the if block are only visible to user with role="Admin". We can apply policy-based checks similar to the role based checks shown in above snapshot.

--END--