Secure Software Development

Diploma in ISF/IT/FI
AY 2018/19 - Semester 3

Week **4**

Razor Pages Security I

This practical will explore the following:

- Part 1: steps in adding authentication to a new project and
- Part 2: steps in adding authentication to the Movies Razor Pages Application

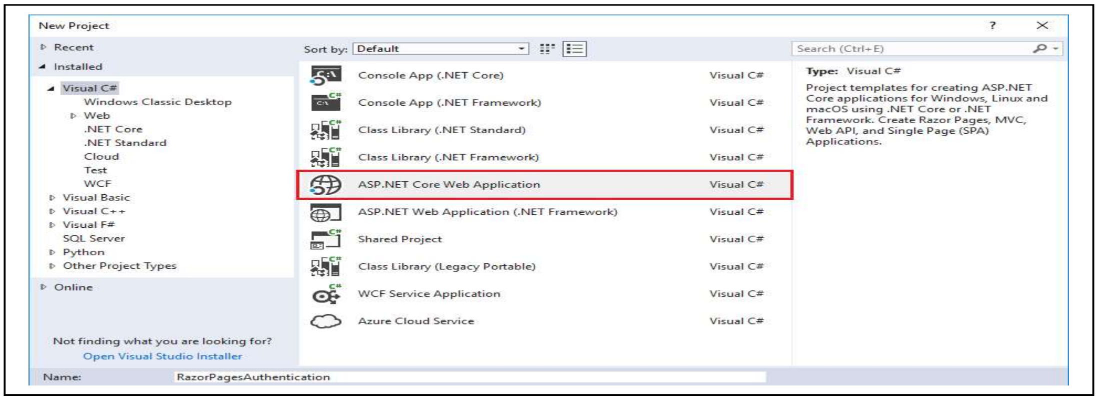
About ASP.NET Core Identity

Razor Pages uses ASP.NET Core Identity as its default membership and authentication system. It is shipped with the ASP.NET Core web development stack, for building web applications and is used to implement forms authentication. It includes membership, login, and management of user data. It allows you to add form authentication and authorization features and customize data about the logged in user in your application. It also provides the features to authenticate a user which is based on his/her login providers such as Facebook, Google or Twitter. It can also provide applications with role-based authorization in which a user is assigned a specific role and level of access to the resources are based on roles.ASP.NET Identity uses Entity Framework Core for data access. EF Core uses migrations to keep the database in sync with the model.

Part 1: Add Authentication to a new project

If you are creating a new project, you can add authentication system into the project with the web pages automatically created. Let looks at the steps to do that.

- Click on **File – New Project**.
- Under **Visual C# – .NET Core**, pick the **ASP.NET Core Web Application (.NET Core)** template. Name the project as **RazorPagesAuthentication**.



New Project

Sort by: Default

Search (Ctrl+E)

Console App (.NET Core)	Visual C#
Console App (.NET Framework)	Visual C#
Class Library (.NET Standard)	Visual C#
Class Library (.NET Framework)	Visual C#
ASP.NET Core Web Application	Visual C#
ASP.NET Web Application (.NET Framework)	Visual C#
Shared Project	Visual C#
Class Library (Legacy Portable)	Visual C#
WCF Service Application	Visual C#
Azure Cloud Service	Visual C#

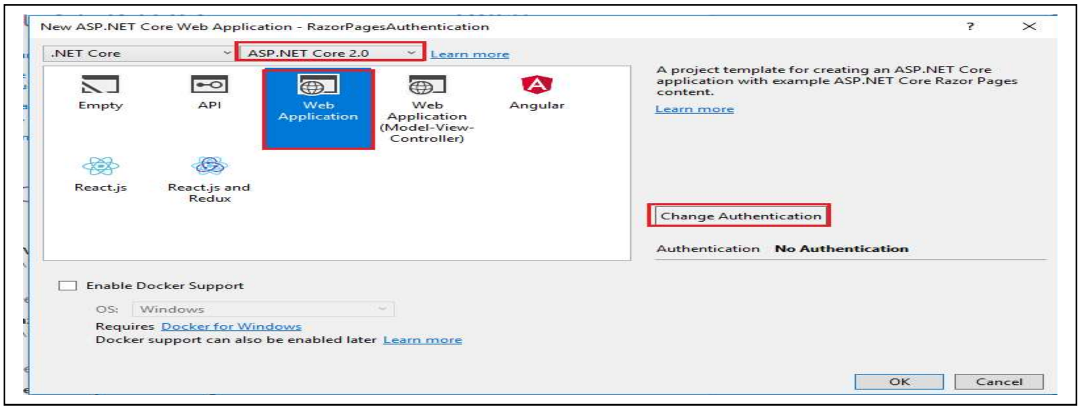
Type: Visual C#

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.

Not finding what you are looking for?
Open Visual Studio Installer

Name: RazorPagesAuthentication

- On the New ASP.NET Core Project dialog, pick the **Web Application** template. Ensure ASP.Net Core 2.0 is selected as well.



New ASP.NET Core Web Application - RazorPagesAuthentication

.NET Core ▾ ASP.NET Core 2.0 ▾ Learn more

A project template for creating a ASP.NET Core application with example ASP.NET Core Razor Pages content.
Learn more

Change Authentication

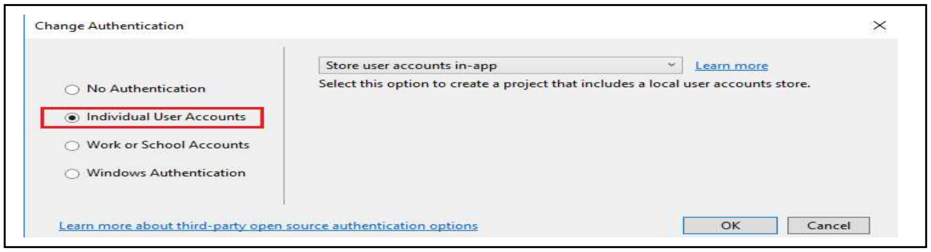
Authentication No Authentication

□ Enable Docker Support

OS: Windows

Requires Docker for Windows
Docker support can also be enabled later Learn more

- Click the **Change Authentication** button and pick **Individual User Accounts**.



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

- Click **OK** to create the project. Visual Studio to create Models, ViewModels, Views, Controllers, and other assets required for authentication as part of the project template.

An explanation of the files and folders created are as follows:

Account folder within the *Pages* folder

- contains a number of Razor Page files designed for managing the most common authentication-related tasks and serve as a reasonable starting point.

Data folder

- contains the files required by Entity Framework Core, including the migrations and DbContext class.

Services folder

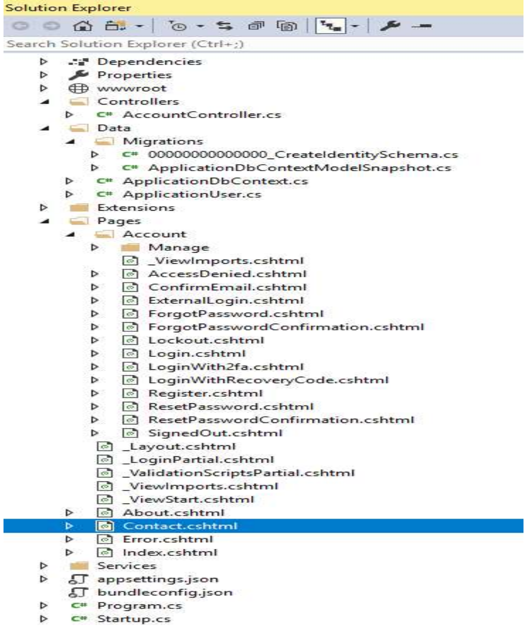
- contains and interface for an EmailSender class, and an implementation that has one non-operational method, SendEmailAsync. You need to provide your own implementation if you plan to use this class.

Extensions folder

- contains a couple of useful extension methods that are used for creating confirmation links in emails and for determining the correct URL to redirect to where necessary.

Controllers folder

- contains code for an MVC controller - AccountController, which has been included to cater for logging out. It has one action method - **Logout**, which signs the user out, and then redirects to the home page. Use of an MVC controller for this process is deliberate - since the logging out process has no associated UI, it was considered unnecessary to use a Razor Page, whose purpose afterall is to generate a UI.



Solution Explorer
Search Solution Explorer (Ctrl+;)

```
Dependencies
Properties
wwwroot
Controllers
  AccountController.cs
Data
  Migrations
    00000000000000_CreateIdentitySchema.cs
    ApplicationDbContextModelSnapshot.cs
  ApplicationDbContext.cs
  ApplicationUser.cs
Extensions
Pages
  Account
    Manage
    _ViewImports.cshtml
    AccessDenied.cshtml
    ConfirmEmail.cshtml
    ExternalLogin.cshtml
    ForgotPassword.cshtml
    ForgotPasswordConfirmation.cshtml
    Lockout.cshtml
    Login.cshtml
    LoginWith2fa.cshtml
    LoginWithRecoveryCode.cshtml
    Register.cshtml
    ResetPassword.cshtml
    ResetPasswordConfirmation.cshtml
    SignedOut.cshtml
  _Layout.cshtml
  _LoginPartial.cshtml
  _ValidationScriptsPartial.cshtml
  _ViewImports.cshtml
  _ViewStart.cshtml
  About.cshtml
  Contact.cshtml
  Error.cshtml
  Index.cshtml
Services
appsettings.json
bundleconfig.json
Program.cs
Startup.cs
```

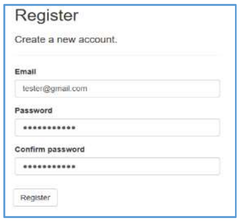
View appsettings.json file

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=RazorPagesAuthentication-
XXXXXXXXX;Trusted_Connection=True;MultipleActiveResultSets=true"},
  "Logging": {
    "IncludeScopes": false,"LogLevel": {"Default": "Warning"}
          }
}
```

- User accounts will be stored in a database called **RazorPagesAuthentication-XXXXXXXXX**.
- May need to edit “Server=(localdb)\\ XXXXXXX” if you have problem accessing later.

Create a user.

- Launch the application (Ctrl+F5) and then click on the **Register** link.
- Register a new user (i.e. tester@gmail.com with password as P@ssword123)

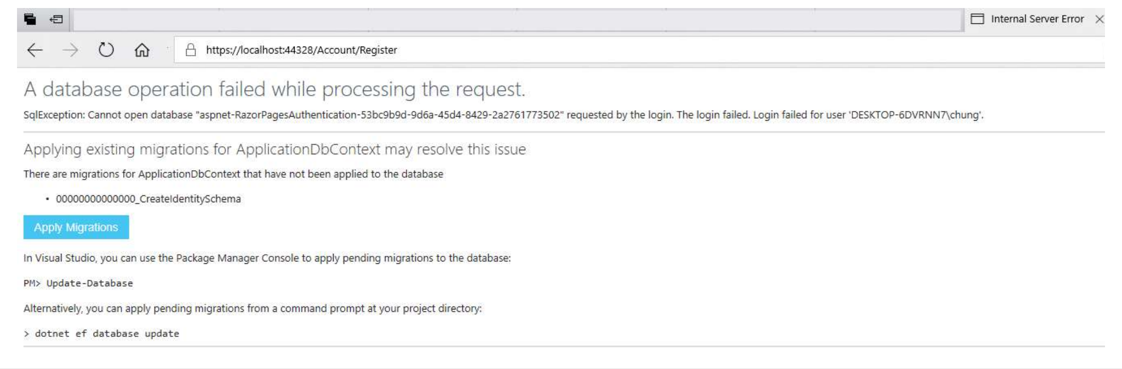


Register
Create a new account.

Email
tester@gmail.com
Password
•••••••••••
Confirm password
•••••••••••

Register

- If this is the first time you're performing this action, you may be required to run migrations. The application prompts you to **Apply Migrations**. Refresh the page if needed.



A database operation failed while processing the request.

SqlException: Cannot open database "aspnet-RazorPagesAuthentication-53bc9b9d-9d6a-45d4-8429-2a2761773502" requested by the login. The login failed. Login failed for user 'DESKTOP-6DVRNN7\chung'.

Applying existing migrations for ApplicationDbContext may resolve this issue

There are migrations for ApplicationDbContext that have not been applied to the database

- 00000000000000_CreateIdentitySchema

In Visual Studio, you can use the Package Manager Console to apply pending migrations to the database:

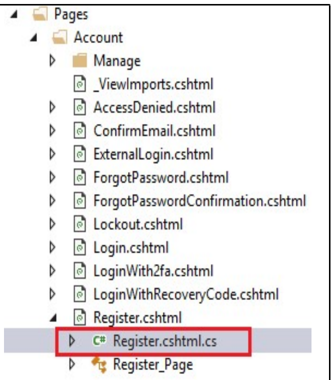
```
PM> Update-Database
```

Alternatively, you can apply pending migrations from a command prompt at your project directory:

```
> dotnet ef database update
```

- When you applied Migration, Visual Studio will **<u>create the database</u>** to store your user accounts.
- Click on Logout.
- Examine the created code in Pages/Account/Register.cshtml.cs - onPostAsync() .

```
….. ……. ……….
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
    {
        ReturnUrl = returnUrl;
        if (ModelState.IsValid)
        {
            var user = new ApplicationUser
            {
                UserName = Input.UserName,
                Email = Input.Email,
                FirstName = Input.FirstName,
                LastName = Input.LastName,
                BirthDate = Input.BirthDate
            };

            var result = await _userManager.CreateAsync(user, Input.Password);

            if (result.Succeeded)
            {
                _logger.LogInformation("User created a new account with password.");

                var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
                var callbackUrl = Url.EmailConfirmationLink(user.Id, code, Request.Scheme);
                await _emailSender.SendEmailConfirmationAsync(Input.Email, callbackUrl);

                await _signInManager.SignInAsync(user, isPersistent: false);
                return LocalRedirect(Url.GetLocalUrl(returnUrl));
            }
            foreach (var error in result.Errors)
            {
                ModelState.AddModelError(string.Empty, error.Description);
            }
        }

        // If we got this far, something failed, redisplay form
        return Page();
    }
```



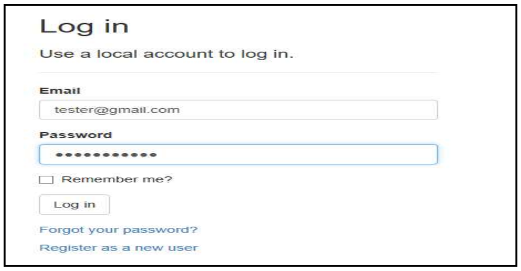
```
Pages
  Account
    Manage
    _ViewImports.cshtml
    AccessDenied.cshtml
    ConfirmEmail.cshtml
    ExternalLogin.cshtml
    ForgotPassword.cshtml
    ForgotPasswordConfirmation.cshtml
    Lockout.cshtml
    Login.cshtml
    LoginWith2fa.cshtml
    LoginWithRecoveryCode.cshtml
    Register.cshtml
      Register.cshtml.cs
      Register_Page
```

Explanation

- When the user clicked on the Register button, the data from Form is posted and the code in the OnPostAsync method is executed. If the model is valid, an application user object created with the posted values.
- CreateAsync() method of UserManager class is used to register the user and user data is inserted into the dbo.AspNetUsers table in Movie-1 database.
- If the user was created successfully, the user is logged in by the call to `_signInManager.SignInAsync`.

Login with the registered user

- Click on Log in.
- Users can sign in by clicking the **Log in** link at the top of the site, or they may be navigated to the Login page if they attempt to access a part of the site that requires authorization (will see that later).



Log in

Use a local account to log in.

Email

tester@gmail.com

Password

●●●●●●●●●●●●

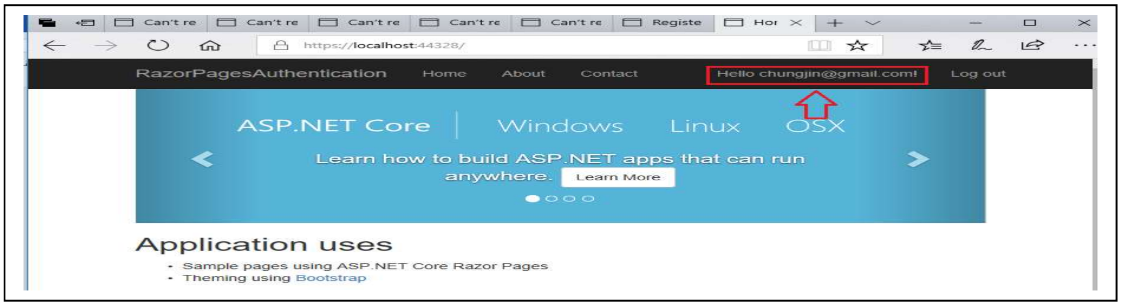
☐ Remember me?

Log in

Forgot your password?

Register as a new user

- After logging in, you should be able to see a similar screen as shown below:



RazorPagesAuthentication Home About Contact Hello chungjin@gmail.com! Log out

ASP.NET Core | Windows Linux OSX

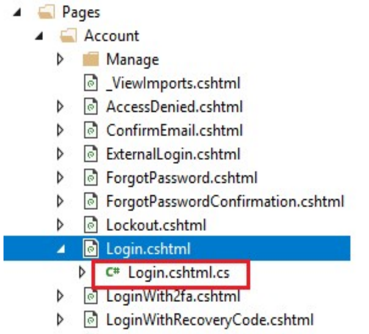
Learn how to build ASP.NET apps that can run anywhere. Learn More

Application uses

- Sample pages using ASP.NET Core Razor Pages
- Theming using Bootstrap

- Examine the created code in **Pages/Account/Login.cshtml.cs** – OnPostAsync().

```
………………..
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
      {
          ReturnUrl = returnUrl;

          if (ModelState.IsValid)
          {
            var result = await _signInManager.PasswordSignInAsync(Input.UserName,
Input.Password, Input.RememberMe, lockoutOnFailure: true);

              if (result.Succeeded)
              {
                  _logger.LogInformation("User logged in.");
                  return LocalRedirect(Url.GetLocalUrl(returnUrl));
              }
              if (result.RequiresTwoFactor)
              {
                  return RedirectToPage("./LoginWith2fa", new { ReturnUrl = returnUrl, RememberMe
= Input.RememberMe });
              }
              if (result.IsLockedOut)
              {
                  _logger.LogWarning("User account locked out.");
                  return RedirectToPage("./Lockout");
              }
              else
              {
                  ModelState.AddModelError(string.Empty, "Invalid login attempt.");
                  return Page();
              }
          }
          // If we got this far, something failed, redisplay form
          return Page();
      }
……………………
```



```
Pages
  Account
    Manage
    _ViewImports.cshtml
    AccessDenied.cshtml
    ConfirmEmail.cshtml
    ExternalLogin.cshtml
    ForgotPassword.cshtml
    ForgotPasswordConfirmation.cshtml
    Lockout.cshtml
    Login.cshtml
      Login.cshtml.cs
    LoginWith2fa.cshtml
    LoginWithRecoveryCode.cshtml
```

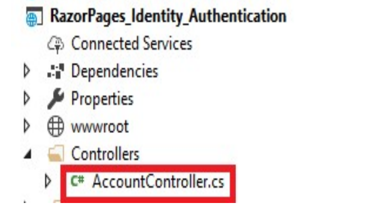
Explanation

- Login action calls PasswordSignInAsync on the _signInManager object .
- PasswordSignInAsync() method of signInManager class is used to verify and login the user. If result is successful, it will be redirected to the returnURL.

Logout a user

- Click the **Log out** link. It will calls the logout action in Controllers/AccountController.cs.
- Examine the code in Controllers/AccountController.cs.

```
. . . . . . . . . . . . . .
[HttpPost]
 [ValidateAntiForgeryToken]
 public async Task<IActionResult> Logout()
 {
     await _signInManager.SignOutAsync();
     _logger.LogInformation("User logged out.");
     return RedirectToPage("/Index");
 }
```

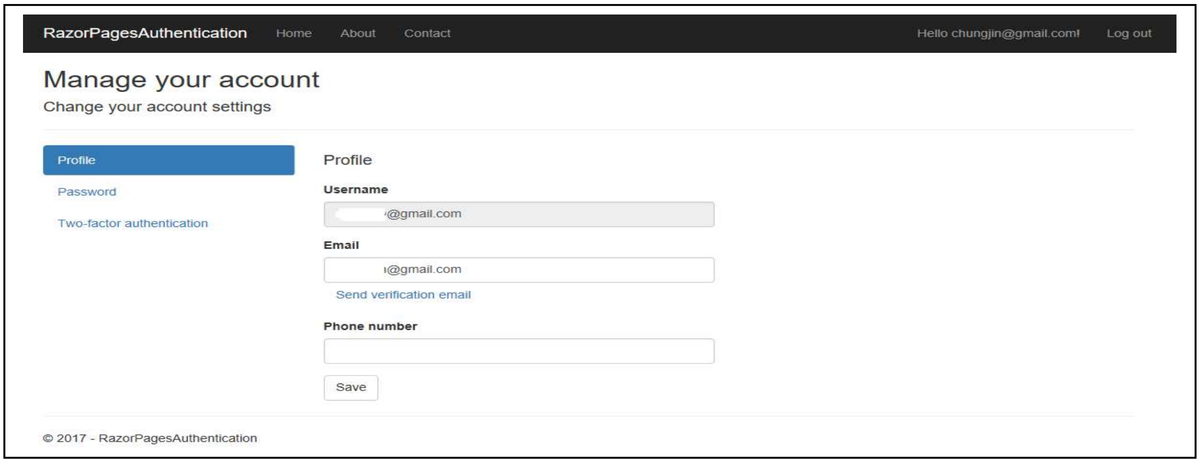


Explanation

- The SignOutAsync method clears the user's claims stored in a cookie and redirect back to the index page. SignOutAsync() method of signInManager class is used to logout the user. If result is successful, it will be redirected to the index page.

Manage Account

- Log in and in the URL type in https://localhost:XXXXX/Account/Manage. This will redirect you to the following page:



Manage your account

Change your account settings

Profile

Username

@gmail.com

Email

@gmail.com

Send verification email

Phone number

Save

© 2017 - RazorPagesAuthentication

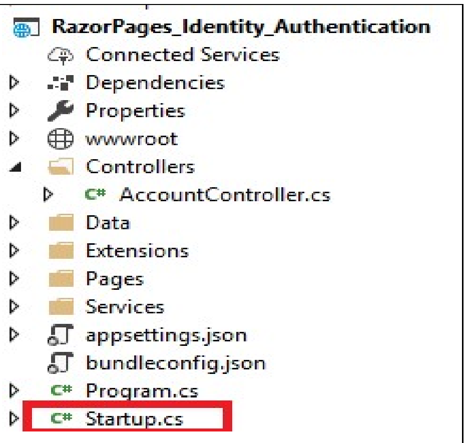
- Here, you can see your profile page, **reset your password (under Password)** and configure two factor authentication.

Password /Lockout and Authentication Cookie Settings Configuration.

Identity has some default behaviors that can be overridden in the app's startup class. IdentityOptions do not need to be configured when using the default behaviors. However, you can choose to customise the password strength options, lockout settings and cookie settings.

Modify the **ConfigureServices() method** of startup.cs to as follows: (add in the highlighted code – copy here n paste correctly):

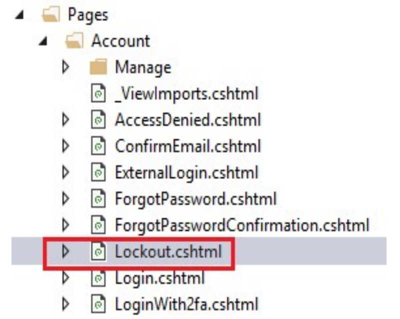
```
……………..

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddDbContext<ApplicationDbContext>(options =>
            options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

        services.AddIdentity<ApplicationUser, IdentityRole>()
            .AddEntityFrameworkStores<ApplicationDbContext>()
            .AddDefaultTokenProviders();


       services.Configure<IdentityOptions>(options =>
       {
            // Password settings
            options.Password.RequireDigit = false;
            options.Password.RequiredLength = 5;
            options.Password.RequireNonAlphanumeric = false;
            options.Password.RequireUppercase = false;
            options.Password.RequireLowercase = false;
            options.Password.RequiredUniqueChars = 1;

            // Lockout settings
            options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);
            options.Lockout.MaxFailedAccessAttempts = 10;
            options.Lockout.AllowedForNewUsers = true;

            // User settings
            options.User.RequireUniqueEmail = true;
       });

            services.ConfigureApplicationCookie(options =>
            {
                // Cookie settings
                options.Cookie.HttpOnly = true;
                options.ExpireTimeSpan = TimeSpan.FromMinutes(30);
                // If the LoginPath isn't set, ASP.NET Core defaults
                // the path to /Account/Login.
                options.LoginPath = "/Account/Login";
                // If the AccessDeniedPath isn't set, ASP.NET Core defaults
                // the path to /Account/AccessDenied.
                options.AccessDeniedPath = "/Account/AccessDenied";
                options.SlidingExpiration = true;
            });


        services.AddMvc()
            .AddRazorPagesOptions(options =>
            {
                options.Conventions.AuthorizeFolder("/Account/Manage");
                options.Conventions.AuthorizePage("/Account/Logout");
            });

        services.AddSingleton<IEmailSender, EmailSender>();
    }
```



- Note that password settings has been set to allow less complex requirements (all set to false)
- Run the application and **register a new user with a simple password (i.e password).** Previously you would not be able to because of the password complexity.
- Other useful settings include login path,lockout time and access denied path. Default login page is /Account/Login. which can be changed. For more information about how to configure Identity, see Configure Identity and check out various options available.

- Edit the **Lockout.chtml** razor page in Pages/Account to as follows. Just a slight modification (highlighted as shown).

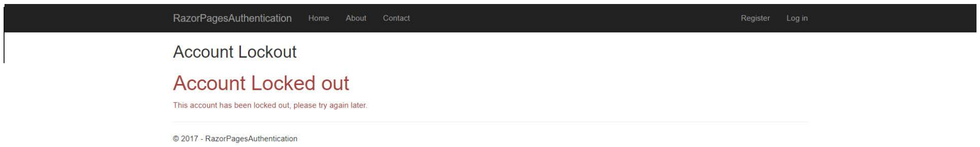
```
@page
@model LockoutModel
@{
    ViewData["Title"] = "Account Locked out";
}

<h2> Account Lockout</h2>
<header>
    <h1 class="text-danger">@ViewData["Title"]</h1>
    <p class="text-danger">This account has been locked out, please try again later.</p>
</header>
```



- Go back to edit the **ConfigureServices() method of startup.cs**, set the option: **Lockout.MaxFailedAccessAttempts = 1**;

- Run the application to simulate a failed login by entering a wrong password for an existing user (i.e. tester@gmail.com). Able to see the account locked out page?

Record your observation.



Account Lockout

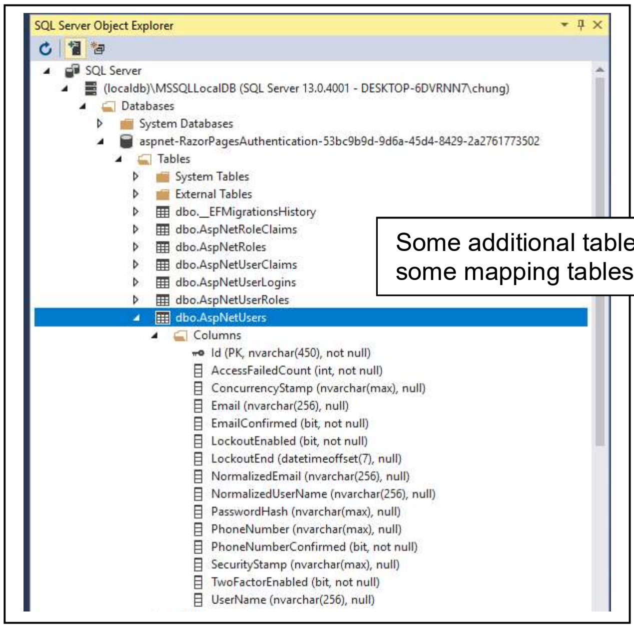
Account Locked out

This account has been locked out, please try again later.

© 2017 - RazorPagesAuthentication

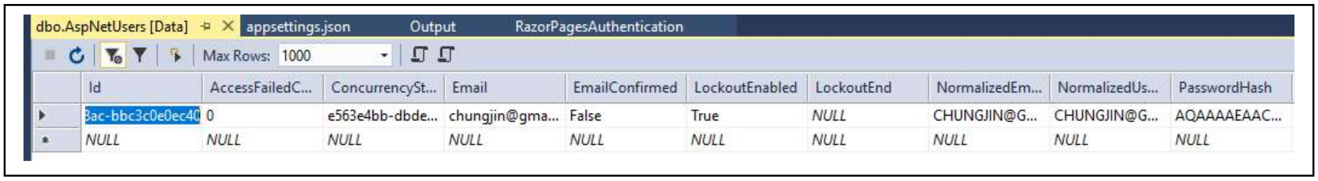
View the database.

- From Visual Studio, select **View** > **SQL Server Object Explorer**. Connect to **(localdb)\MSSQLLocalDB**
- Database with a name matching **aspnet-<*name of your project*>-<*date string*>** is displayed. Our example should be **aspnet-RazorPagesAuthentication-<*date string*>.**
- Expand the database and its **Tables**, then right-click the **dbo.AspNetUsers** table and select **View Data**.



- SQL Server
  - (localdb)\MSSQLLocalDB (SQL Server 13.0.4001 - DESKTOP-6DVRNN7\chung)
    - Databases
      - System Databases
      - aspnet-RazorPagesAuthentication-53bc9b9d-9d6a-45d4-8429-2a2761773502
        - Tables
          - System Tables
          - External Tables
          - dbo.__EFMigrationsHistory
          - dbo.AspNetRoleClaims
          - dbo.AspNetRoles
          - dbo.AspNetUserClaims
          - dbo.AspNetUserLogins
          - dbo.AspNetUserRoles
          - dbo.AspNetUsers
            - Columns
              - Id (PK, nvarchar(450), not null)
              - AccessFailedCount (int, not null)
              - ConcurrencyStamp (nvarchar(max), null)
              - Email (nvarchar(256), null)
              - EmailConfirmed (bit, not null)
              - LockoutEnabled (bit, not null)
              - LockoutEnd (datetimeoffset(7), null)
              - NormalizedEmail (nvarchar(256), null)
              - NormalizedUserName (nvarchar(256), null)
              - PasswordHash (nvarchar(max), null)
              - PhoneNumber (nvarchar(max), null)
              - PhoneNumberConfirmed (bit, not null)
              - SecurityStamp (nvarchar(max), null)
              - TwoFactorEnabled (bit, not null)
              - UserName (nvarchar(256), null)

Some additional tables to store users, claims, roles, and some mapping tables that map users to specific roles.

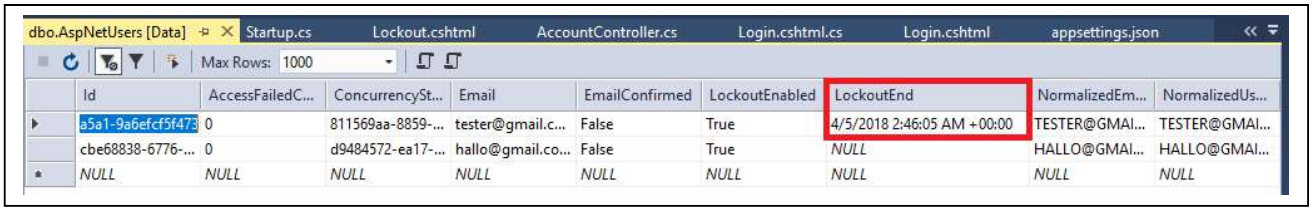


Id	AccessFailedC...	ConcurrencySt...	Email	EmailConfirmed	LockoutEnabled	LockoutEnd	NormalizedEm...	NormalizedUs...	PasswordHash
3ac-bbc3c0e0ec40	0	e563e4bb-dbde...	chungjin@gma...	False	True	NULL	CHUNGJIN@G...	CHUNGJIN@G...	AQAAAAEAAC...
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Is the password or password hash stored on the table? Yes.

This password hash (instead of password) is used for user authentication. The password is passed directly to the Identity framework, which will hash the password.

Are you able to see the **field LockoutEnd** for the user whose account has been just locked out? Yes.

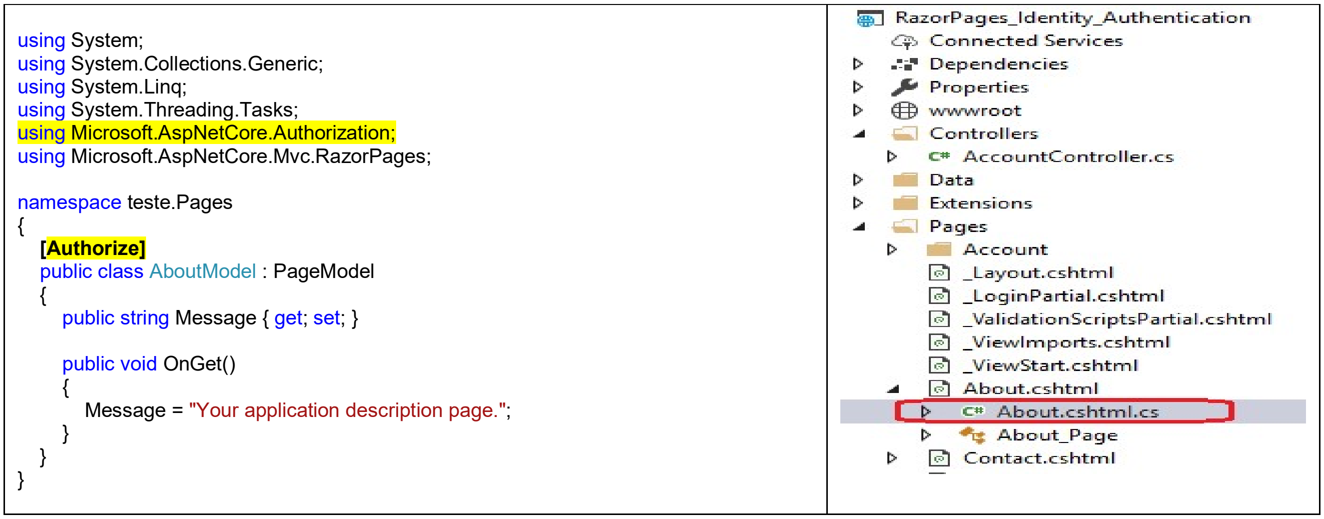


Id	AccessFailedC...	ConcurrencySt...	Email	EmailConfirmed	LockoutEnabled	LockoutEnd	NormalizedEm...	NormalizedUs...
a5a1-9a6efcf5f473	0	811569aa-8859-...	tester@gmail.c...	False	True	4/5/2018 2:46:05 AM +00:00	TESTER@GMAI...	TESTER@GMAI...
cbe68838-6776-...	0	d9484572-ea17-...	hallo@gmail.co...	False	True	NULL	HALLO@GMAI...	HALLO@GMAI...
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Edit data in the LockoutEnd field directly to NULL to re-enable the locked-out user account. Just type NULL.

Verify Identity works

- The default *ASP.NET Core Web Application* project template allows users to access any action in the application without having to login.
- To verify that ASP.NET Identity works, add an **[Authorize]** attribute and [using Microsof.AspNetCore.Authorization] highlighted code to the **About.cshtml.cs** file as shown below.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace teste.Pages
{
    [Authorize]
    public class AboutModel : PageModel
    {
        public string Message { get; set; }

        public void OnGet()
        {
            Message = "Your application description page.";
        }
    }
}
```

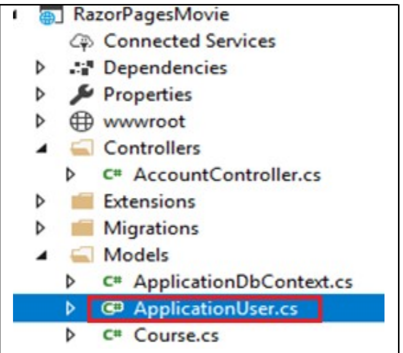
- Run the project using **Ctrl** + **F5** . If you are logged in, log out first.
- Navigate to the **About** page. What do you observe?It redirects me to the login page.
- Only authenticated users may access the **About** page now, so ASP.NET redirects you to the login page to login or register.

Part 2: Steps to add authentication & authorization into Movies Razor Pages App using Identity

1. Open the existing RazorPagesMovie application in Visual Studio.

2. Add a new class called **ApplicationUser.cs** in **Models** folder. Right click on Models folder, select class. name it as ApplicationUser.cs. Add the following code into the class. **(Copy and paste correctly – ensure the namespace is correct).**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;

namespace RazorPagesMovie.Models
{
    // Add profile data for application users by adding properties to the          ApplicationUser
class
    public class ApplicationUser : IdentityUser
    {
        public string FullName { get; set; }
        public DateTime BirthDate { get; set; }
        public int Age { get; set; }
    }
}
```



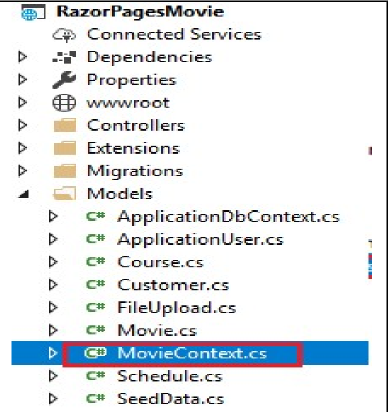
RazorPagesMovie
- Connected Services
- Dependencies
- Properties
- wwwroot
- Controllers
  - AccountController.cs
- Extensions
- Migrations
- Models
  - ApplicationDbContext.cs
  - ApplicationUser.cs
  - Course.cs

Note:

- A user in ASP.NET Identity is represented by the ApplicationUser class which inherits the IdentityUser base class.You want to capture more information from the user at the point of registration than just their email address and username. Can do this by adding properties to the ApplicationUser class for storing the additional values. Add in to capture FullName and BirthDate of a user - adding additional properties in MyIdentityUser class.
- IdentityUser base class contains basic user details(Microsoft.AspNetCore.Identity) such as UserName, Password and Email.

3. Edit the **MovieContext.cs** in Models folder. Add the following code into the class. (Copy and paste correctly – ensure the namespace is correct).

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace RazorPagesMovie.Models
{
    public class MovieContext : IdentityDbContext<ApplicationUser>
    {
        //DBset property for the entity set
        //Entity set corresponds to a row in the table

         public MovieContext(DbContextOptions<MovieContext> options)
            : base(options)
        {
        }

protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);
            // Customize the ASP.NET Identity model and override the defaults if needed.
            // For example, you can rename the ASP.NET Identity table names and more.
            // Add your customizations after calling base.OnModelCreating(builder);
        }

        public DbSet<Movie> Movie { get; set; }
        public DbSet<Schedule> Schedule { get; set; }
        public DbSet<Course> Courses { get; set; }
    }
}
```

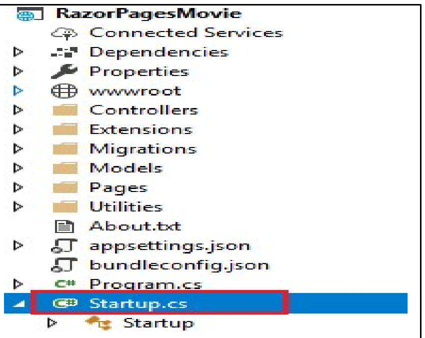


RazorPagesMovie
- Connected Services
- Dependencies
- Properties
- wwwroot
- Controllers
- Extensions
- Migrations
- Models
  - ApplicationDbContext.cs
  - ApplicationUser.cs
  - Course.cs
  - Customer.cs
  - FileUpload.cs
  - Movie.cs
  - MovieContext.cs
  - Schedule.cs
  - SeedData.cs

- MovieContext class inherits from IdentityDbContext base class.
- User names , passwords + other data will be stored in the Movie-1 database.

4. Open **Startup.cs** file and modify the **ConfigureServices()** method as shown.

```
Using Microsoft.AspNetCore.Identity

…………………………………….

public void ConfigureServices(IServiceCollection services)
      {

         services.AddDbContext<MovieContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("MovieContext")));

            services.AddIdentity<ApplicationUser, IdentityRole>()
             .AddEntityFrameworkStores<MovieContext>()
             .AddDefaultTokenProviders();

         services.AddMvc()
      }
```

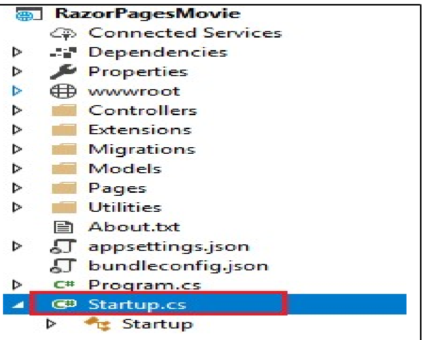


Note:

- ConfigureServices() method calls AddIdentity() method to add ASP.NET Core Identity services to the container. This is where ApplicationUser and IdentityRole classes are also mentioned.

5. Open **Startup.cs** file and modify the **Configure()** method as shown.

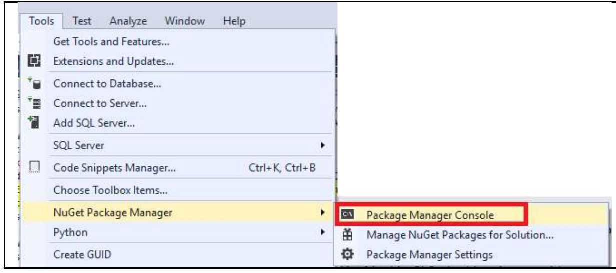
```
…………………………………….
          // This method gets called by the runtime.
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
      {
         if (env.IsDevelopment())
         {
            app.UseBrowserLink();
            app.UseDeveloperExceptionPage();
         }
         else
         {
            app.UseExceptionHandler("/Error");
         }
         app.UseStaticFiles();
         app.UseAuthentication();
         app.UseMvc();
      }
```



Note:

- Configure() method calls UserAuthentication() method to add ASP.NET Core Identity to the request pipeline.

6. Launch the **Package Manager Console** under **Tools => NuGet Package Manager.**



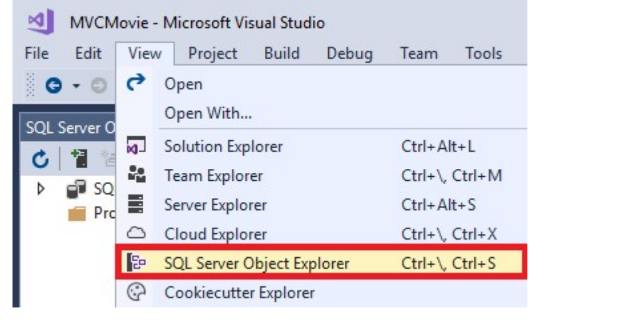
Type the following commands in the console:

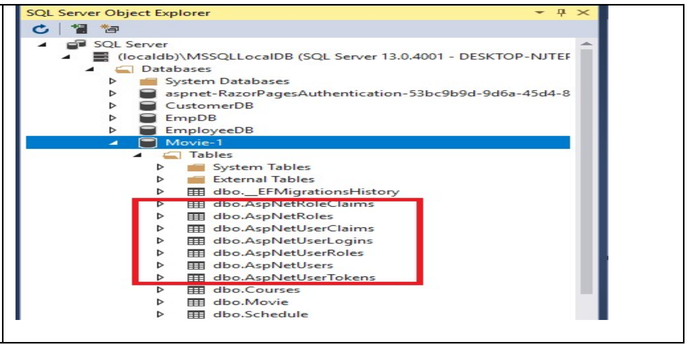
```
PM> Add-Migration AddIdentity
PM> update-database
```

- First command will create a migration that when applied will create and modify the schema of the tables in database. Second command will apply the migration and update the database.

7. Verfity the user tables in Movie-1 database are created.

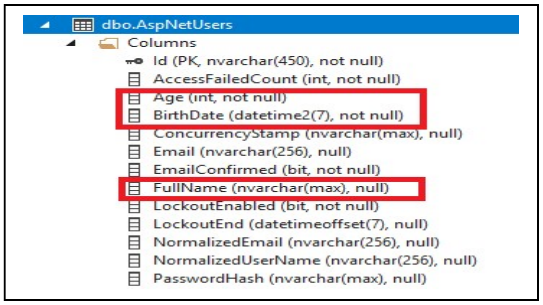
   Click on **View and SQL Server Object Explorer.** Look for **(localdb)\MSSQLLocalDB => Databases => Movie-1.**
   [Some students may not using MSSQLLocalDB – check appsettings.json file for the connection string].


MVCMovie - Microsoft Visual Studio
File
Edit
View
Project
Build
Debug
Team
Tools
SQL Server O
SQ
Pro
Open
Open With...
Solution Explorer Ctrl+Alt+L
Team Explorer Ctrl+\, Ctrl+M
Server Explorer Ctrl+Alt+S
Cloud Explorer Ctrl+\, Ctrl+X
SQL Server Object Explorer Ctrl+\, Ctrl+S
Cookiecutter Explorer


SQL Server Object Explorer

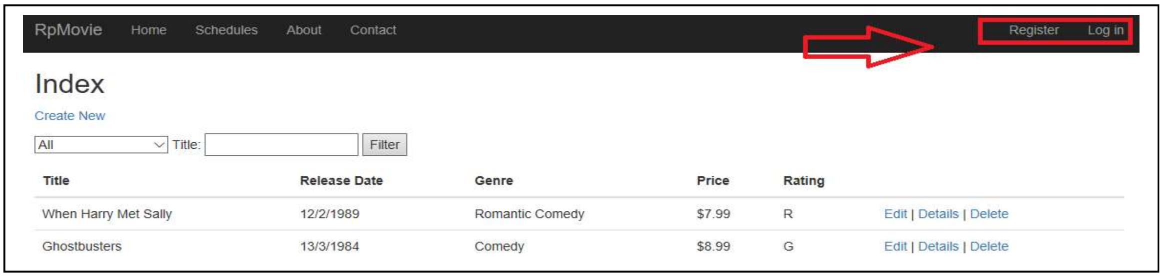
```
SQL Server
  (localdb)\MSSQLLocalDB (SQL Server 13.0.4001 - DESKTOP-NJTEF
    Databases
      System Databases
      aspnet-RazorPagesAuthentication-53bc9b9d-9d6a-45d4-8
      CustomerDB
      EmpDB
      EmployeeDB
      Movie-1
        Tables
          System Tables
          External Tables
          dbo.__EFMigrationsHistory
          dbo.AspNetRoleClaims
          dbo.AspNetRoles
          dbo.AspNetUserClaims
          dbo.AspNetUserLogins
          dbo.AspNetUserRoles
          dbo.AspNetUsers
          dbo.AspNetUserTokens
          dbo.Courses
          dbo.Movie
          dbo.Schedule
```

- dbo.Asp.NetUsers table will contain the user data and fields.
- Check out the table – click on Columns under the table. Did the properties specified in the ApplicationUser Class (Age, BirthDate, FullName) created as additional fields in the table ? Yes.



```
dbo.AspNetUsers
  Columns
    Id (PK, nvarchar(450), not null)
    AccessFailedCount (int, not null)
    Age (int, not null)
    BirthDate (datetime2(7), not null)
    ConcurrencyStamp (nvarchar(max), null)
    Email (nvarchar(256), null)
    EmailConfirmed (bit, not null)
    FullName (nvarchar(max), null)
    LockoutEnabled (bit, not null)
    LockoutEnd (datetimeoffset(7), null)
    NormalizedEmail (nvarchar(256), null)
    NormalizedUserName (nvarchar(256), null)
    PasswordHash (nvarchar(max), null)
```

8. Next we create the Register and Log in/out bar on website application as shown below.



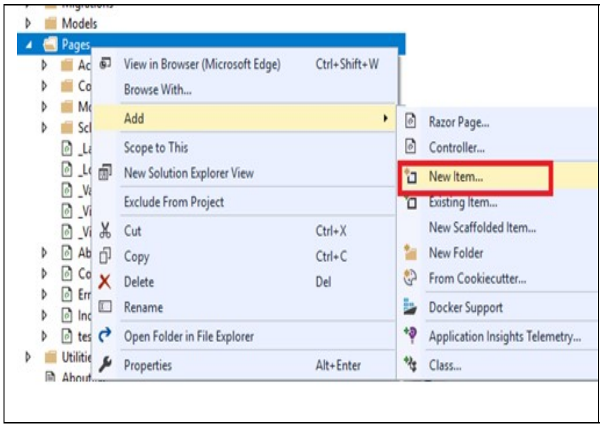
Index

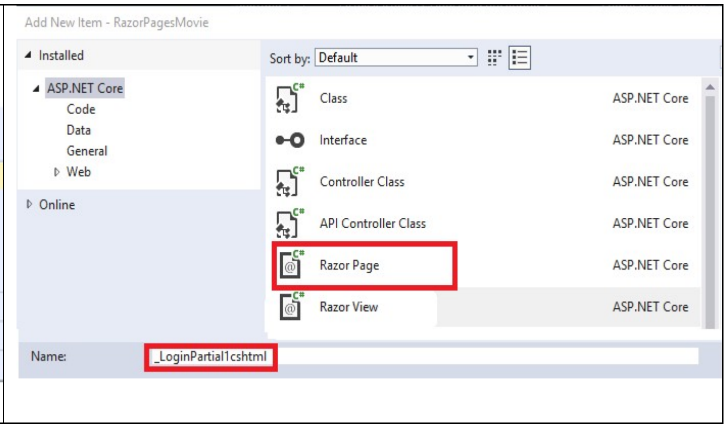
Create New

All Title: Filter

Title	Release Date	Genre	Price	Rating	
When Harry Met Sally	12/2/1989	Romantic Comedy	$7.99	R	Edit | Details | Delete
Ghostbusters	13/3/1984	Comedy	$8.99	G	Edit | Details | Delete

- Right-click on Pages Folder. Click **Add** and **New Item**. Choose **Razor Page under Web**. Specify File name as **_LoginPartial.chtml**.

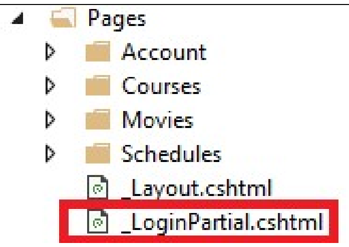




Name: _LoginPartial1cshtml

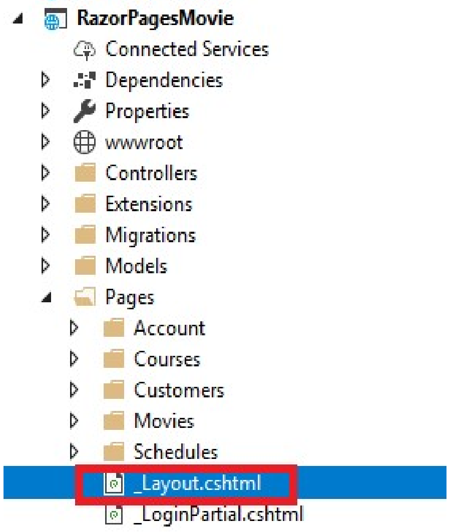
- Add the following code in _LoginPartial.chtml. [Copy from here and paste the code properly]. [Alternatively, you can copy the _LoginPartial.chtml file from the earlier new project in Part 1, and paste it here.]

```
@using Microsoft.AspNetCore.Identity
@using RazorPagesMovie.Models
@inject SignInManager<ApplicationUser> SignInManager
@inject UserManager<ApplicationUser> UserManager

@if (SignInManager.IsSignedIn(User))
{
    <form asp-controller="Account" asp-action="Logout" method="post"
id="logoutForm" class="navbar-right">
        <ul class="nav navbar-nav navbar-right">
            <li>
                <a asp-page="/Account/Manage/Index" title="Manage">Hello
@UserManager.GetUserName(User)!</a>
            </li>
            <li>
                <button type="submit" class="btn btn-link navbar-btn navbar-link">Log
out</button>
            </li>
        </ul>
    </form>
}
else
{
    <ul class="nav navbar-nav navbar-right">
        <li><a asp-page="/Account/Register">Register</a></li>
        <li><a asp-page="/Account/Login">Log in</a></li>
    </ul>
}
```



- Edit **_Layout.chtml** and add in the highlighted code as shown below.

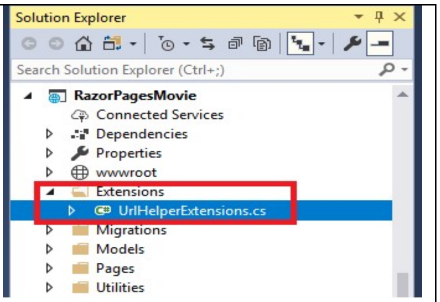
```
 ........................................................

<nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-
target=".navbar-collapse">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a asp-page="/Movies/Index" class="navbar-brand">RpMovie</a>
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li><a asp-page="/Index">Home</a></li>
                    <li><a asp-page="/About">About</a></li>
                    <li><a asp-page="/Contact">Contact</a></li>
                </ul>
                @await Html.PartialAsync("_LoginPartial")
            </div>
        </div>
    </nav>
 ..............................................................
```



- Press Ctrl F5 to run the application. Check that the Register and Login Bar on the right side of the Movies application can be seen.

9. Create a folder called **Extensions** under RazorPagesMovie. Add in a class called **UrlHelperExtensions.cs**. Paste the following code into this class.

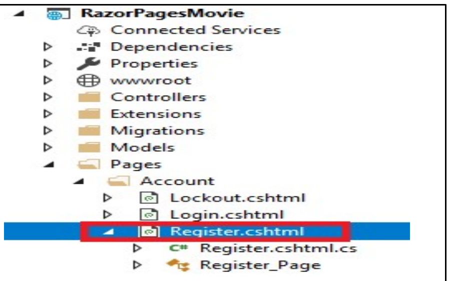
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Microsoft.AspNetCore.Mvc
{
    public static class UrlHelperExtensions
    {
        public static string GetLocalUrl(this IUrlHelper urlHelper, string localUrl)
        {
            if (!urlHelper.IsLocalUrl(localUrl))
            {
               return urlHelper.Page("/Index");
             }
               return localUrl;
        }

        public static string EmailConfirmationLink(this IUrlHelper urlHelper, string userId, string
code, string scheme)
        {
            return urlHelper.Page(
                "/Account/ConfirmEmail",
                pageHandler: null,
                values: new { userId, code },
                protocol: scheme);                    }

        public static string ResetPasswordCallbackLink(this IUrlHelper urlHelper, string userId,
string code, string scheme)
        {
            return urlHelper.Page(
                "/Account/ResetPassword",
                pageHandler: null,
                values: new { userId, code },
                protocol: scheme);
        }
    }
}
```

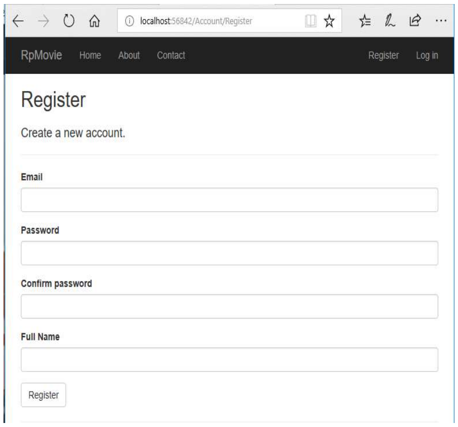


10. Create Register functionality.
    Under **Pages** Folder , create a folder called **Account.**
    Right click on **Account folder** , add a razor page , name it **Register.**

    Add the following code in **Register.cshtml** (razor page).

```
@page
@model RegisterModel
@{
   ViewData["Title"] = "Register";
}
<h2>@ViewData["Title"]</h2>

<div class="row">
   <div class="col-md-4">
      <form asp-route-returnUrl="@Model.ReturnUrl" method="post">
         <h4>Create a new account.</h4>
         <hr />
         <div asp-validation-summary="All" class="text-danger"></div>
         <div class="form-group">
            <label asp-for="Input.Email"></label>
            <input asp-for="Input.Email" class="form-control" />
            <span asp-validation-for="Input.Email" class="text-danger"></span>
         </div>
         <div class="form-group">
            <label asp-for="Input.Password"></label>
            <input asp-for="Input.Password" class="form-control" />
            <span asp-validation-for="Input.Password" class="text-danger"></span>
         </div>
         <div class="form-group">
            <label asp-for="Input.ConfirmPassword"></label>
            <input asp-for="Input.ConfirmPassword" class="form-control" />
         <span asp-validation-for="Input.ConfirmPassword" class="text-danger"></span>
         </div>
          <div class="form-group">
            <label asp-for="Input.FullName"></label>
            <input asp-for="Input.FullName" class="form-control" />
            <span asp-validation-for="Input.FullName" class="text-danger"></span>
         </div>
         <button type="submit" class="btn btn-default">Register</button>
      </form>
   </div>
</div>
@section Scripts {
   @await Html.PartialAsync("_ValidationScriptsPartial")
}
```

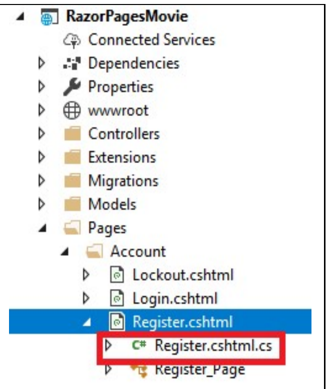




- We only add in the Full Name as part of the registration. Customize with more fields like Age and BirthDate (which were added in the ApplicationUser class) is possible.

Add the following code in **Register.cshtmls.cs** (code-behind page). Ensure namespace is correct.

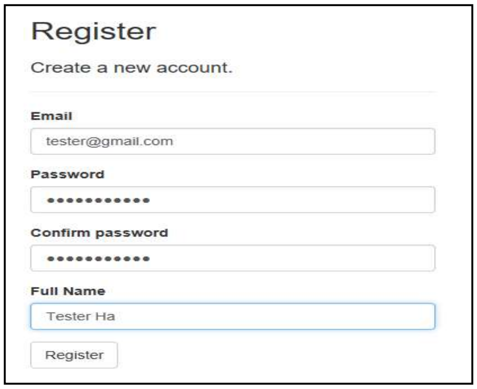
```
using System.ComponentModel.DataAnnotations;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using RazorPagesMovie.Models;

namespace RazorPagesMovie.Pages.Account
{
  public class RegisterModel : PageModel
  {
    private readonly SignInManager<ApplicationUser> _signInManager;
    private readonly UserManager<ApplicationUser> _userManager;

    public RegisterModel(
      UserManager<ApplicationUser> userManager,
      SignInManager<ApplicationUser> signInManager)
    {
      _userManager = userManager;
      _signInManager = signInManager;
     }

    [BindProperty]
    public InputModel Input { get; set; }
    public string ReturnUrl { get; set; }

   public class InputModel
    {
      [Required]
      [EmailAddress]
      [Display(Name = "Email")]
      public string Email { get; set; }

      [Required]
      [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at max {1} characters long.",
MinimumLength = 6)]
      [DataType(DataType.Password)]
      [Display(Name = "Password")]
      public string Password { get; set; }

      [DataType(DataType.Password)]
      [Display(Name = "Confirm password")]
      [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
      public string ConfirmPassword { get; set; }

     [Display(Name = "Full Name")]
      public string FullName { get; set; }
     }

    public void OnGet(string returnUrl = null)
    {   ReturnUrl = returnUrl;     }

    public async Task<IActionResult> OnPostAsync(string returnUrl = null)
    {
      ReturnUrl = returnUrl;
      if (ModelState.IsValid)
      {
        var user = new ApplicationUser { UserName = Input.Email, Email = Input.Email,FullName
=Input.FullName };
        var result = await _userManager.CreateAsync(user, Input.Password);

        if (result.Succeeded)
        {
          await _signInManager.SignInAsync(user, isPersistent: false);
           return LocalRedirect(Url.GetLocalUrl(returnUrl));
        }

        foreach (var error in result.Errors)
        {   ModelState.AddModelError(string.Empty, error.Description);    }
      }
       return Page();
    }
  }
}
```



- UserManager class is creating and managing users. InputModel is view model that hold the data entered on the register view.

11. Run the application - Ctrl F5.

    Register a new user with some data.



Register

Create a new account.

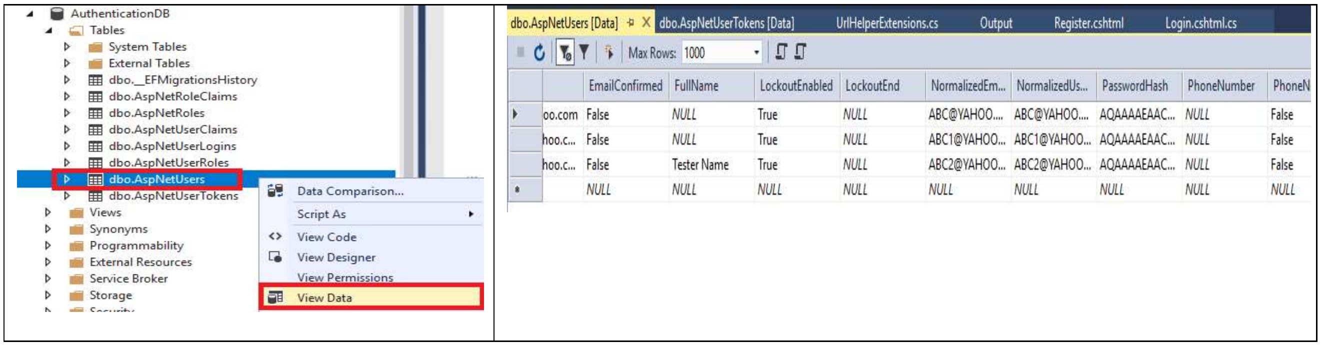
Email: tester@gmail.com

Password: ••••••••••••

Confirm password: ••••••••••••

Full Name: Tester Ha

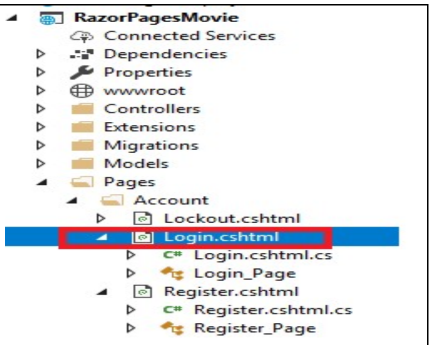
Use SQL Server Object Explorer to check if the record is created. Might have to refresh a few times.



	EmailConfirmed	FullName	LockoutEnabled	LockoutEnd	NormalizedEm...	NormalizedUs...	PasswordHash	PhoneNumber	PhoneN
oo.com	False	NULL	True	NULL	ABC@YAHOO....	ABC@YAHOO....	AQAAAAEAAC...	NULL	False
hoo.c...	False	NULL	True	NULL	ABC1@YAHOO...	ABC1@YAHOO...	AQAAAAEAAC...	NULL	False
hoo.c...	False	Tester Name	True	NULL	ABC2@YAHOO...	ABC2@YAHOO...	AQAAAAEAAC...	NULL	False
	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

12. Create Login functionality.
    Right click on **Account folder** , add a razor page , name it **Login.**
    Add the following code in **Login.cshtml** (razor page).

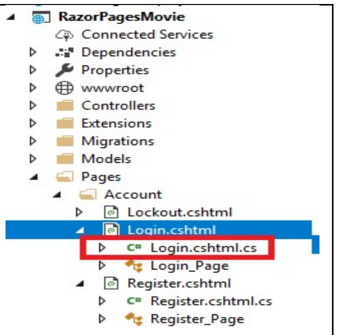
```
@page
@model LoginModel
@{
    ViewData["Title"] = "Log in";
}
<h2>@ViewData["Title"]</h2>
<div class="row">
    <div class="col-md-4">
        <section>
            <form method="post">
                <h4>Use a local account to log in.</h4>
                <hr />
                <div asp-validation-summary="All" class="text-danger"></div>
                <div class="form-group">
                    <label asp-for="Input.Email"></label>
                    <input asp-for="Input.Email" class="form-control" />
                    <span asp-validation-for="Input.Email" class="text-danger"></span>
                </div>
                <div class="form-group">
                    <label asp-for="Input.Password"></label>
                    <input asp-for="Input.Password" class="form-control" />
                    <span asp-validation-for="Input.Password" class="text-danger"></span>
                </div>
                <div class="form-group">
                    <div class="checkbox">
                        <label asp-for="Input.RememberMe">
                            <input asp-for="Input.RememberMe" />
                            @Html.DisplayNameFor(m => m.Input.RememberMe)
                        </label>
                    </div>
                </div>
                <div class="form-group">
                    <button type="submit" class="btn btn-default">Log in</button>
                </div>
                <div class="form-group">
                    <p>
                        <a asp-page="./ForgotPassword">Forgot your password?</a>
                    </p>
                    <p>
                        <a asp-page="./Register" asp-route-
returnUrl="@Model.ReturnUrl">Register as a new user</a>
                    </p>
                </div>
            </form>
        </section>
    </div>
</div>

@section Scripts {
    @await  Html.PartialAsync("_ValidationScriptsPartial")
}
```



```
RazorPagesMovie
    Connected Services
    Dependencies
    Properties
    wwwroot
    Controllers
    Extensions
    Migrations
    Models
    Pages
        Account
            Lockout.cshtml
            Login.cshtml
                Login.cshtml.cs
                Login_Page
            Register.cshtml
                Register.cshtml.cs
                Register_Page
```

13. Add the following code in **Login.cshtml.cs** (code-behind page).

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;
using RazorPagesMovie.Models;

namespace RazorPagesMovie.Pages.Account
{
    public class LoginModel : PageModel
    {
        private readonly SignInManager<ApplicationUser> _signInManager;
        private readonly ILogger<LoginModel> _logger;

public LoginModel(SignInManager<ApplicationUser> signInManager, ILogger<LoginModel> logger)
        {
            _signInManager = signInManager;
            _logger = logger;
        }

        [BindProperty]
        public InputModel Input { get; set; }

        public IList<AuthenticationScheme> ExternalLogins { get; set; }

        public string ReturnUrl { get; set; }

        [TempData]
        public string ErrorMessage { get; set; }

        public class InputModel
        {
            [Required]
            [EmailAddress]
            public string Email { get; set; }

            [Required]
            [DataType(DataType.Password)]
            public string Password { get; set; }

            [Display(Name = "Remember me?")]
            public bool RememberMe { get; set; }
        }

        public async Task OnGetAsync(string returnUrl = null)
        {
            if (!string.IsNullOrEmpty(ErrorMessage))
            {
                ModelState.AddModelError(string.Empty, ErrorMessage);
            }
            // Clear the existing external cookie to ensure a clean login process
            await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);
            ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
            ReturnUrl = returnUrl;
        }

        public async Task<IActionResult> OnPostAsync(string returnUrl = null)
        {
            ReturnUrl = returnUrl;

            if (ModelState.IsValid)
            {
                var result = await _signInManager.PasswordSignInAsync(Input.Email, Input.Password,
Input.RememberMe, lockoutOnFailure: true);
                if (result.Succeeded)
                {
                    _logger.LogInformation("User logged in.");
                    return LocalRedirect(Url.GetLocalUrl(returnUrl));
                }
```



- RazorPagesMovie
  - Connected Services
  - Dependencies
  - Properties
  - wwwroot
  - Controllers
  - Extensions
  - Migrations
  - Models
  - Pages
    - Account
      - Lockout.cshtml
      - Login.cshtml
        - Login.cshtml.cs
        - Login_Page
      - Register.cshtml
        - Register.cshtml.cs
        - Register_Page

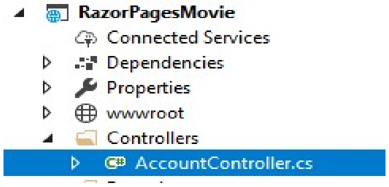
```
                if (result.IsLockedOut)
                {
                    _logger.LogWarning("User account locked out.");
                    return RedirectToPage("./Lockout");
                }
                else
                {
                    ModelState.AddModelError(string.Empty, "Invalid login attempt.");
                    return Page();
                }
            }
            return Page();
        }
    }
}
```

14. Run the application with Ctrl F5.
    Click on Login to login with a registered user account.

15. Create Logout functionality.
    Create a folder called Controllers under RazorPagesMovie.
    Right click on Controllers **folder** , add a **Controller** , choose **MVC Controller -Empty** and name it **AccountController.**

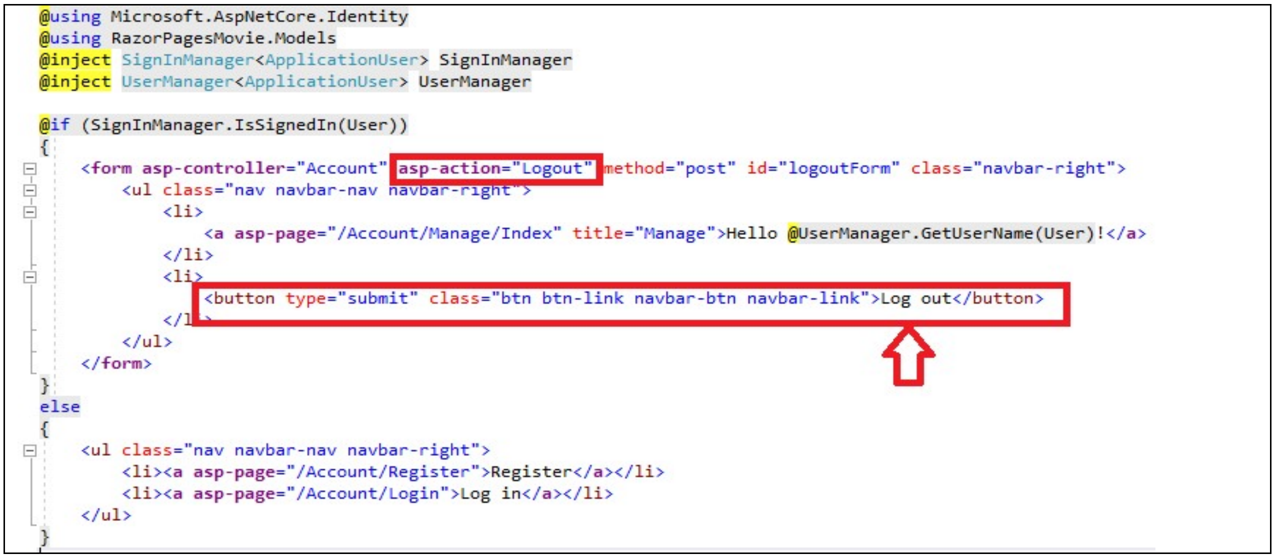
    Add the following code in **AccountController.cshtml** (razor page).

```
 using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using RazorPagesMovie.Models;

namespace RazorPagesMovie.Controllers
{

    [Route("[controller]/[action]")]
    public class AccountController : Controller
    {
        private readonly SignInManager<ApplicationUser> _signInManager;
        private readonly ILogger _logger;

        public AccountController(SignInManager<ApplicationUser> signInManager,
ILogger<AccountController> logger)
        {
            _signInManager = signInManager;
            _logger = logger;
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Logout()
        {
            await _signInManager.SignOutAsync();
            _logger.LogInformation("User logged out.");
            return RedirectToPage("/Index");
        }
    }
}
```



```
◢ RazorPagesMovie
    Connected Services
  ▷ Dependencies
  ▷ Properties
  ▷ wwwroot
  ◢ Controllers
    ▷ AccountController.cs
```

- Logout navbar button is shown in _LoginPartial.chtml – as created earlier.



```
@using Microsoft.AspNetCore.Identity
@using RazorPagesMovie.Models
@inject SignInManager<ApplicationUser> SignInManager
@inject UserManager<ApplicationUser> UserManager

@if (SignInManager.IsSignedIn(User))
{
    <form asp-controller="Account" asp-action="Logout" method="post" id="logoutForm" class="navbar-right">
        <ul class="nav navbar-nav navbar-right">
            <li>
                <a asp-page="/Account/Manage/Index" title="Manage">Hello @UserManager.GetUserName(User)!</a>
            </li>
            <li>
                <button type="submit" class="btn btn-link navbar-btn navbar-link">Log out</button>
            </li>
        </ul>
    </form>
}
else
{
    <ul class="nav navbar-nav navbar-right">
        <li><a asp-page="/Account/Register">Register</a></li>
        <li><a asp-page="/Account/Login">Log in</a></li>
    </ul>
}
```

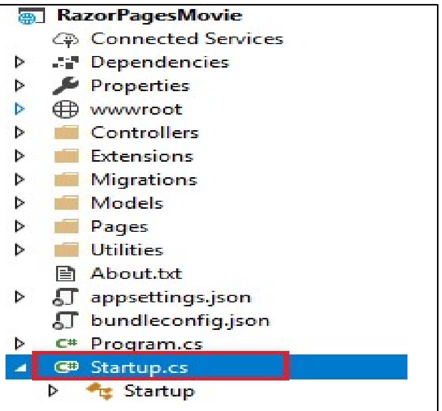
16. Run the application. Login first and logout to test out the Log out functionality.

Razor Pages Authorization

- Authorization refers to the process that determines what a user is able to do. Authorization is orthogonal and independent from authentication. However, authorization requires an authentication mechanism.
- This part will explore methods to allow authorised and anonymous access to certain web pages in a web application.

1. Modify the **ConfigureServices()** method of **startup.cs** to as follows : [Copy and paste highlighted code correctly]

```
. . . . . . . . . . . . . . . . . . . .

public void ConfigureServices(IServiceCollection services)
     {
          services.AddDbContext<MovieContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("MovieContext")));

          services.AddIdentity<ApplicationUser, IdentityRole>()
             .AddEntityFrameworkStores<ApplicationDbContext>()
             .AddDefaultTokenProviders();

           services.AddMvc()
           .AddRazorPagesOptions(options =>
           {
              // options.Conventions.AllowAnonymousToFolder("/Movies");
               options.Conventions.AuthorizePage("/Movies/Create");
               options.Conventions.AuthorizePage("/Account/Create");
               });
     }
...........
```



- Use of authorization filters - AuthorizePage(), AuthorizeFolder(), AllowAnonymousToPage() and AllowAnonymousToFolder() methods at startup to control access.
- AuthorizeFolder method to restrict access to a folder and all of its contents.
- AuthorizePage method to restrict access on a page-by-page basis.

2. Run the Movies application. If logged in, log out of the application first.

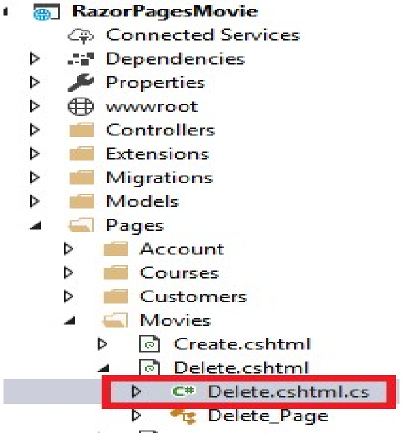
3. Go to url http://localhost:XXXXX/Movies/Create to create a new record.

   Compare it with going to the index page - http://localhost:XXXXX/Movies.

   Record your observations I was prompted to log in in order to access the ./Movies/Create page. This is not seen when going to the ./Movies page.

4. Add the **[Authorize]** attribute and highlighted code as shown below in **Delete.chtml.cs** file

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;
using RazorPagesMovie.Models;

namespace RazorPagesMovie.Pages.Movies
{
    [Authorize]
    public class DeleteModel : PageModel
    {
        private readonly RazorPagesMovie.Models.MovieContext _context;

        public DeleteModel(RazorPagesMovie.Models.MovieContext context)
        {
            _context = context;
        }

        [BindProperty]
        public Movie Movie { get; set; }

        ……………………………………………….
```



5. Run the Movies application and attempt to delete a record.

   Record your observation: When I clicked on the delete button of a particular record, they prompted me to log in.

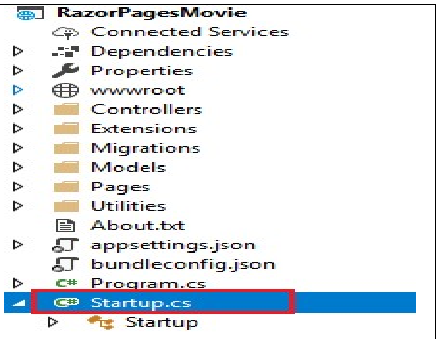
6. How to allow only authorized users to access to the **entire Movies application?** Try out the step below:

- AuthorizeFolder method to restrict access to a folder and all of its contents.
- Set it in **ConfigureServices()** method of **startup.cs:**

```
services.AddMvc()
.AddRazorPagesOptions(options =>
{
    options.Conventions.AuthorizeFolder("/Movies");
 });
```

7. Modify the **ConfigureServices()** method of **startup.cs** to as follows. Copy and paste the highlighted code.

```
……………………………….…
public void ConfigureServices(IServiceCollection services)
    {
        services.AddDbContext<MovieContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("MovieContext")));

        services.AddIdentity<ApplicationUser, IdentityRole>()
            .AddEntityFrameworkStores<MovieContext>()
            .AddDefaultTokenProviders();

        services.Configure<IdentityOptions>(options =>
        {
            // Password settings
            options.Password.RequireDigit = false;
            options.Password.RequiredLength = 5;
            options.Password.RequireNonAlphanumeric = false;
            options.Password.RequireUppercase = false;
            options.Password.RequireLowercase = false;
            options.Password.RequiredUniqueChars = 1;

            // Lockout settings
            options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);
            options.Lockout.MaxFailedAccessAttempts = 10;
            options.Lockout.AllowedForNewUsers = true;

            // User settings
            options.User.RequireUniqueEmail = true;
        });

        services.AddMvc()
         .AddRazorPagesOptions(options =>
         {
             options.Conventions.AuthorizeFolder("/Movies");
         });
   }
```

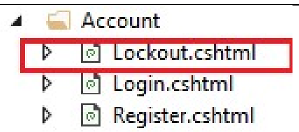


.

8. Password, lockout and user settings can be configured. Try out the password settings by configuring it to allow less complex requirements (all set to false) and run the application to register another user – one with a simple password.

9. Create a razor page called **Lockout** in Account folder. Add the following code in lockout.chtml.cs.

```
@page
@model RazorPagesMovie.Pages.Account.LockoutModel
@{
    ViewData["Title"] = "Lockout";
}

<h2>Lockout</h2>
<header>
    <h1 class="text-danger">@ViewData["Title"]</h1>
    <p class="text-danger">This account has been locked out, please try again later.</p>
</header>
```



10. Set the option **Lockout.MaxFailedAccessAttempts = 1** in **ConfigureServices()** method of **startup.cs.**

Run the application to simulate a failed login by entering wrong password for a registered user.

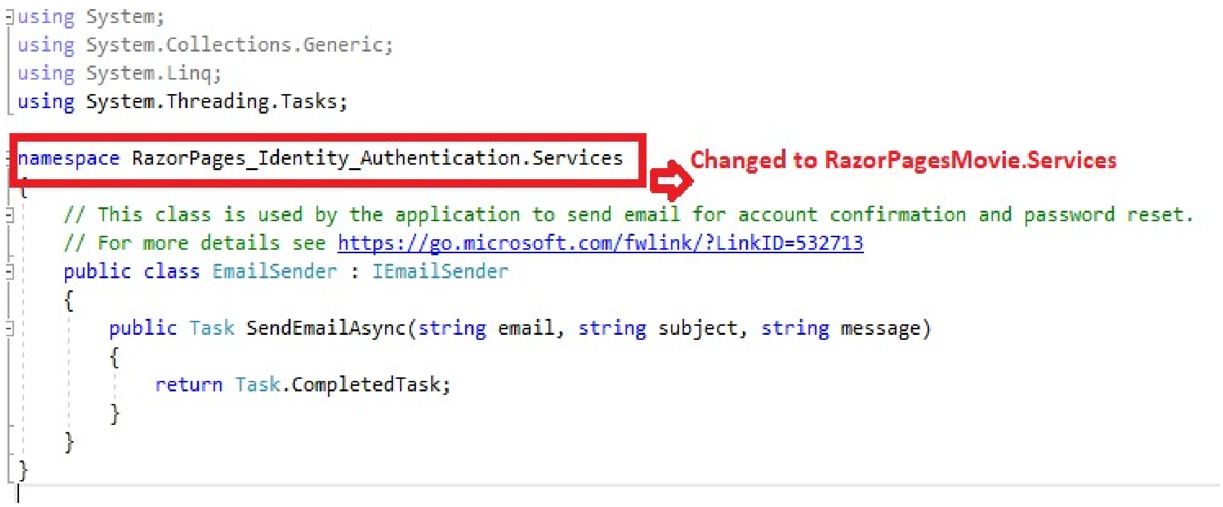
Optional Activity – This part of the practical is optional

11. We have only add in the register and login functionality to the movie application.

    To add other functions we can use the project we created in part 1 as a guide.

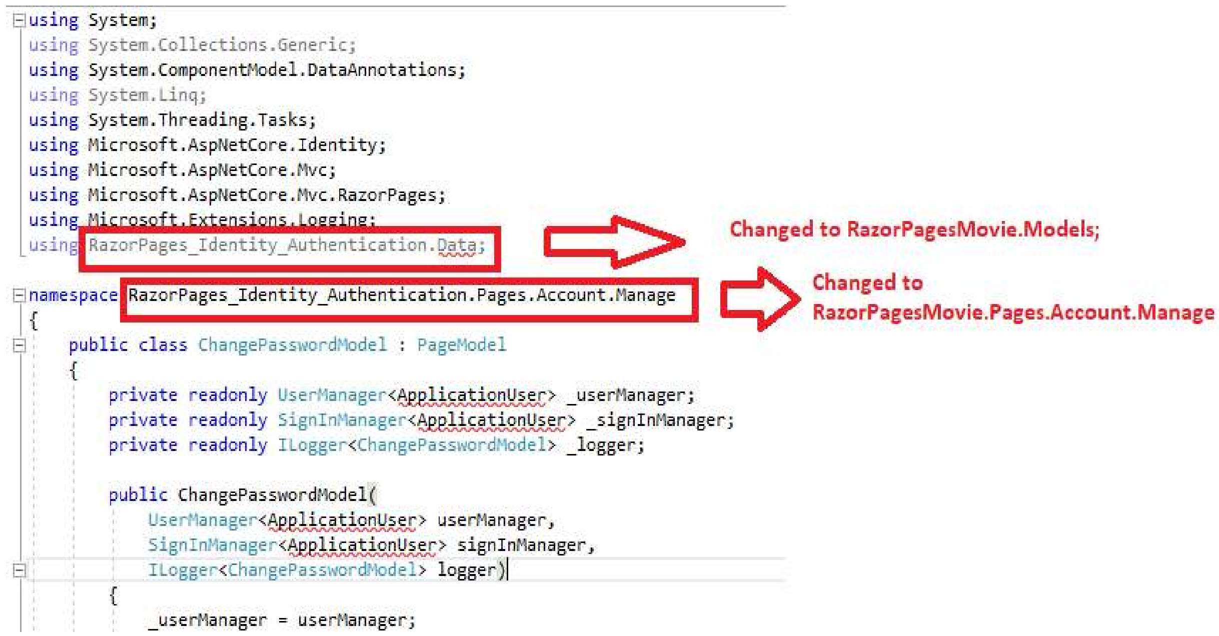
    Folders such as **Extensions (EmailSenderExtensions.cs and UrlHelperExtensions.cs), Services and Pages/Account/Manage folders** need to be copied from the project in Part 1 to the movie application in Part 2 . Can use copy and paste.

    Namespace for the copied files need to be changed to point to the correct paths. For example:



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace RazorPages_Identity_Authentication.Services
{
    // This class is used by the application to send email for account confirmation and password reset.
    // For more details see https://go.microsoft.com/fwlink/?LinkID=532713
    public class EmailSender : IEmailSender
    {
        public Task SendEmailAsync(string email, string subject, string message)
        {
            return Task.CompletedTask;
        }
    }
}
```

Changed to RazorPagesMovie.Services

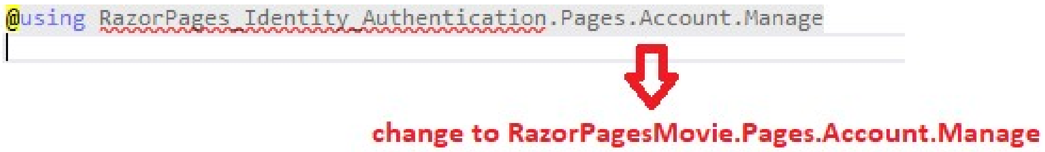


```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Logging;
using RazorPages_Identity_Authentication.Data;

namespace RazorPages_Identity_Authentication.Pages.Account.Manage
{
    public class ChangePasswordModel : PageModel
    {
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly SignInManager<ApplicationUser> _signInManager;
        private readonly ILogger<ChangePasswordModel> _logger;

        public ChangePasswordModel(
            UserManager<ApplicationUser> userManager,
            SignInManager<ApplicationUser> signInManager,
            ILogger<ChangePasswordModel> logger)
        {
            _userManager = userManager;
```

Changed to RazorPagesMovie.Models;

Changed to RazorPagesMovie.Pages.Account.Manage

_ViewImports.cshtml in the Account/Manage folder need to be changed :



```
@using RazorPages_Identity_Authentication.Pages.Account.Manage
```

change to RazorPagesMovie.Pages.Account.Manage

Edit **startup.cs file** – copy and paste the highlighted code.

```
using RazorPagesMovie.Services;

// This method gets called by the runtime. Use this method to add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddDbContext<ApplicationDbContext>(options =>
            options.UseSqlServer(Configuration.GetConnectionString("MovieContext")));


        services.Configure<IdentityOptions>(options =>
        {
            // Password settings
            options.Password.RequireDigit = false;
            options.Password.RequiredLength = 5;
            options.Password.RequireNonAlphanumeric = false;
            options.Password.RequireUppercase = false;
            options.Password.RequireLowercase = false;
            options.Password.RequiredUniqueChars = 1;

            // Lockout settings
            options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);
            options.Lockout.MaxFailedAccessAttempts = 10;
            options.Lockout.AllowedForNewUsers = true;

            // User settings
            options.User.RequireUniqueEmail = true;
        });

        services.AddMvc()
            .AddRazorPagesOptions(options =>
            {
                options.Conventions.AuthorizeFolder("/Account/Manage");
                options.Conventions.AuthorizePage("/Account/Logout");
                options.Conventions.AuthorizePage("/About");
            });

        // Register no-op EmailSender used by account confirmation and password reset during development
        // For more information on how to enable account confirmation and password reset please visit
https://go.microsoft.com/fwlink/?LinkID=532713
        services.AddSingleton<IEmailSender, EmailSender>();
    }
```

Edit **Account/Manage/_ManageNav.cs** file – add in the highlighted code.

```
@using Microsoft.AspNetCore.Identity
@using RazorPagesMovie.Models
@inject SignInManager<ApplicationUser> SignInManager
@{
    var hasExternalLogins = (await SignInManager.GetExternalAuthenticationSchemesAsync()).Any();
}

<ul class="nav nav-pills nav-stacked">
    <li class="@ManageNavPages.IndexNavClass(ViewContext)"><a asp-page="./Index">Profile</a></li>
    <li class="@ManageNavPages.ChangePasswordNavClass(ViewContext)"><a asp-page="./ChangePassword">Password</a></li>
    @if (hasExternalLogins)
    {
        <li class="@ManageNavPages.ExternalLoginsNavClass(ViewContext)"><a asp-page="./ExternalLogins">External logins</a></li>
    }
    <li class="@ManageNavPages.TwoFactorAuthenticationNavClass(ViewContext)"><a asp-page="./TwoFactorAuthentication">Two-factor
authentication</a></li>
</ul>
```