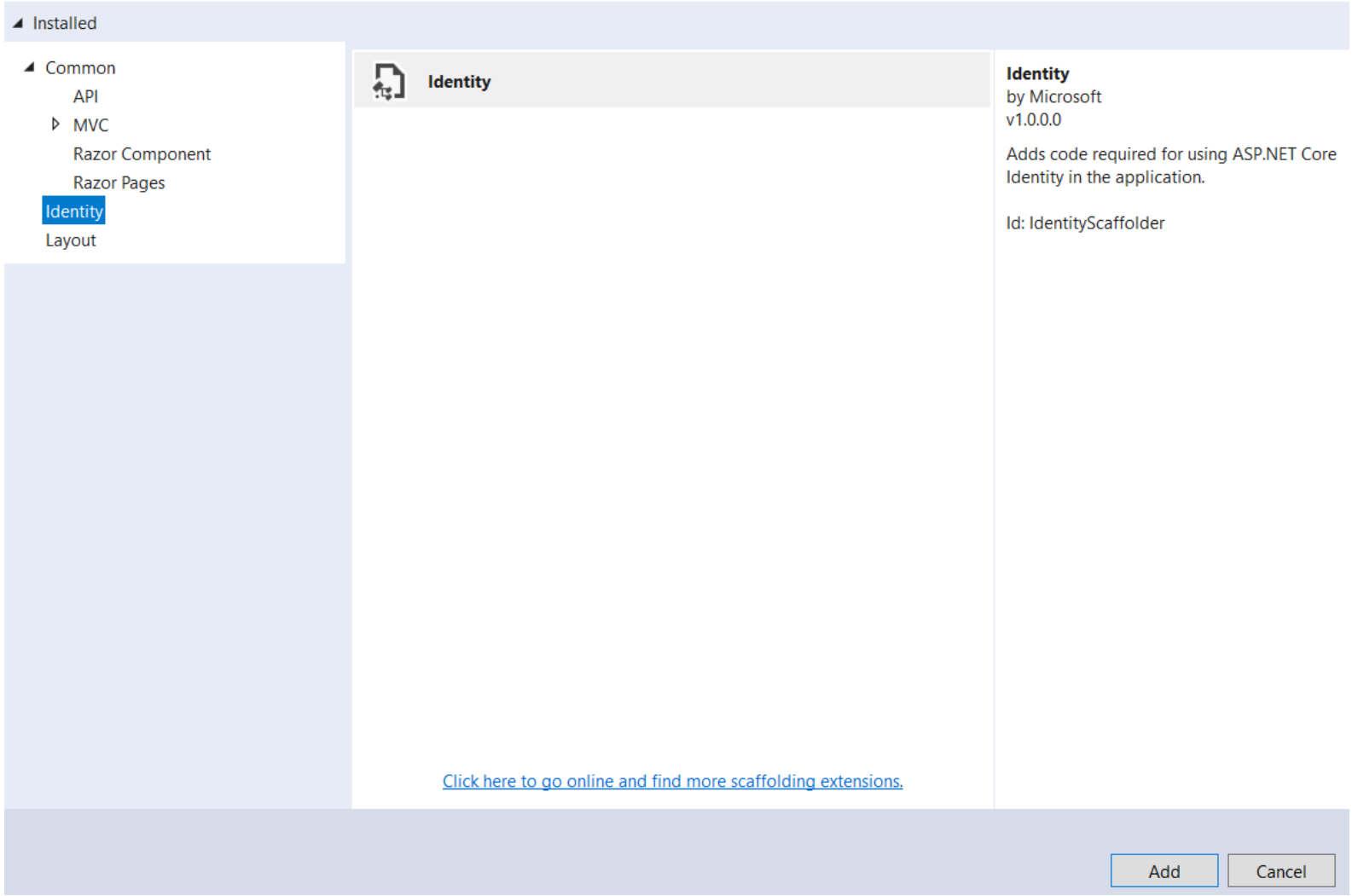
Развој на серверски WEB апликации

Practical: ASP.NET Core Identity Authentication and Authorization in the MVCMovie project

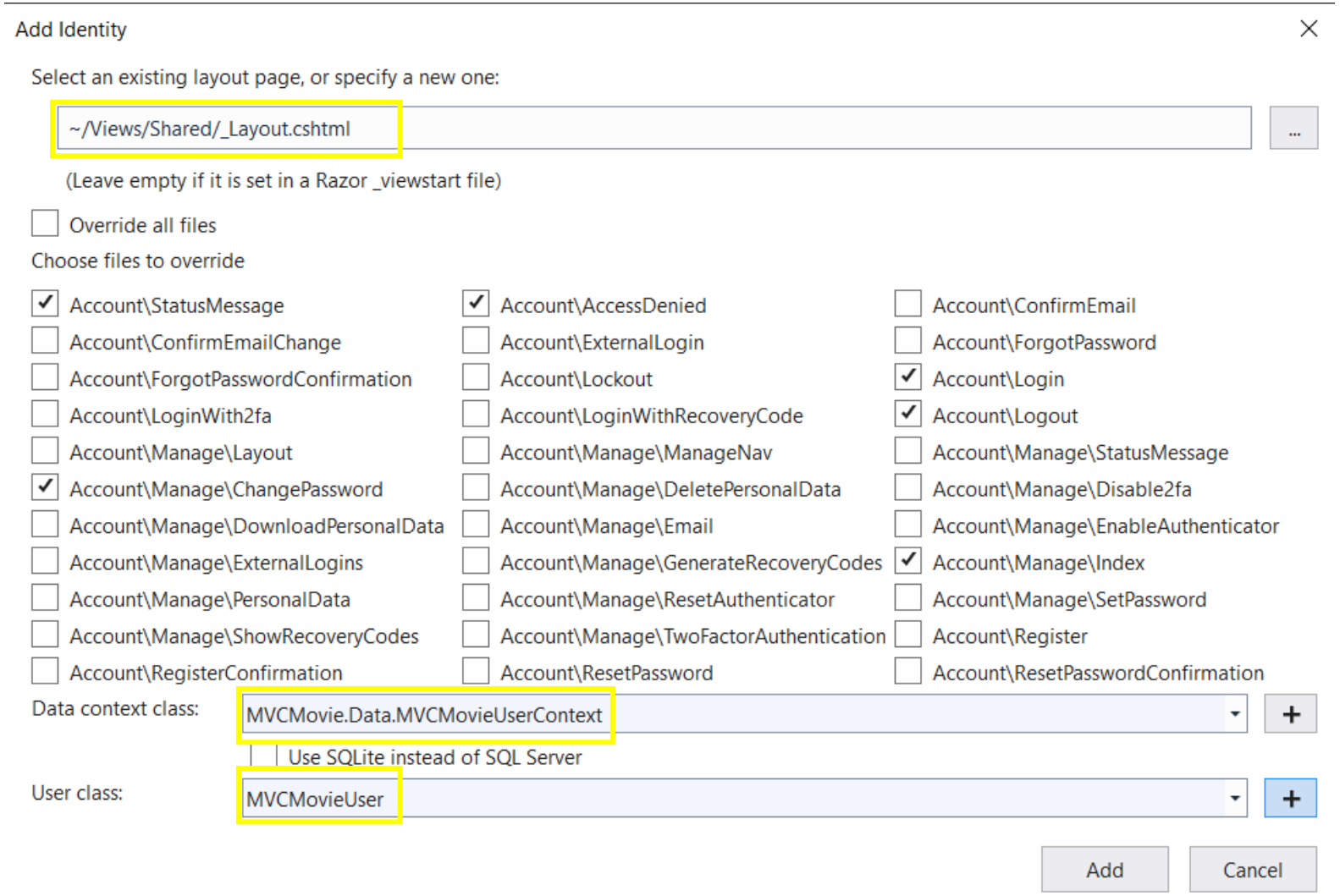
ASP.NET Core Identity

- ASP.NET Core Identity is a **membership system** which allows you to add login functionality to your application
- Users can **create an account** with the login information stored in Identity **or** they can **use an external login provider**
- Supported external login providers include Facebook, Google, Microsoft Account, and Twitter
- You can configure ASP.NET Core Identity to use a **SQL Server database** to store usernames, passwords, and profile data
- Alternatively, you can use your own persistent store to store data in another persistent storage, such as Azure Table Storage

From Solution Explorer, right-click on the project > Add > New Scaffolded Item > Add

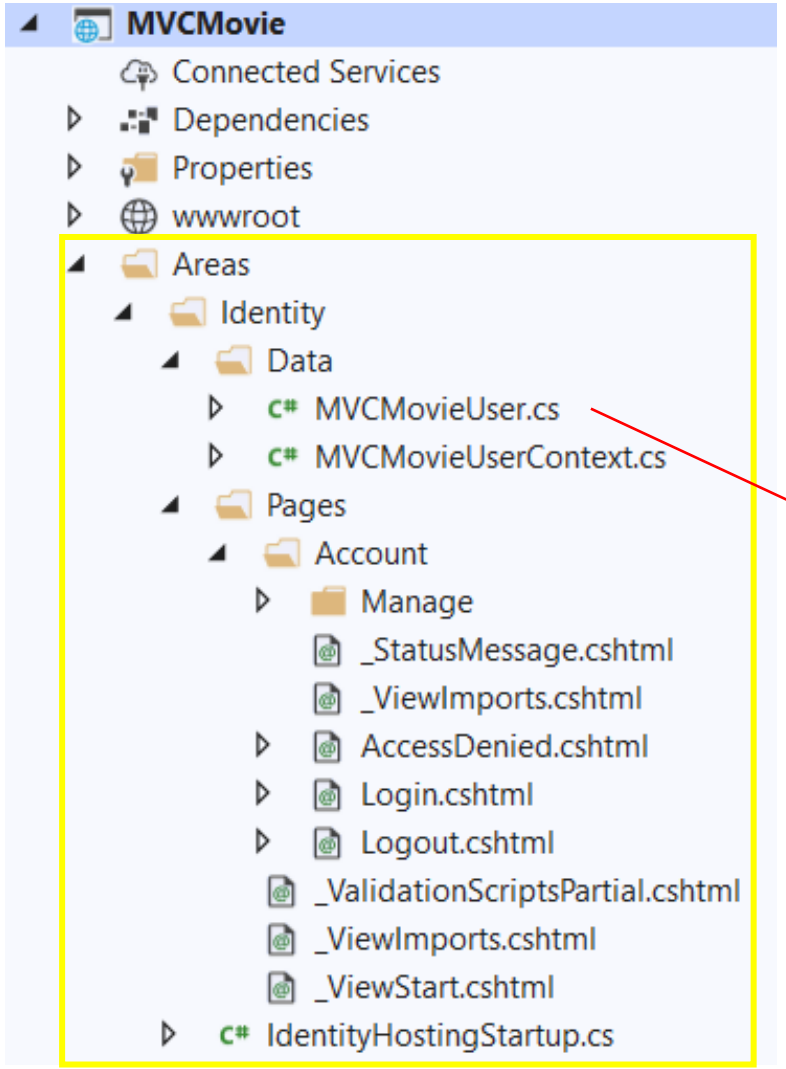


Scaffolding Identity



- Configure Identity to use the same layout as the MVC project
- Select the respective Identity files (functionalities) that should be overridden
- Add (+) the Identity Data Context Class (MVCMovieUserContext)
- Add (+) the Identity User class (MVCMovieUser)

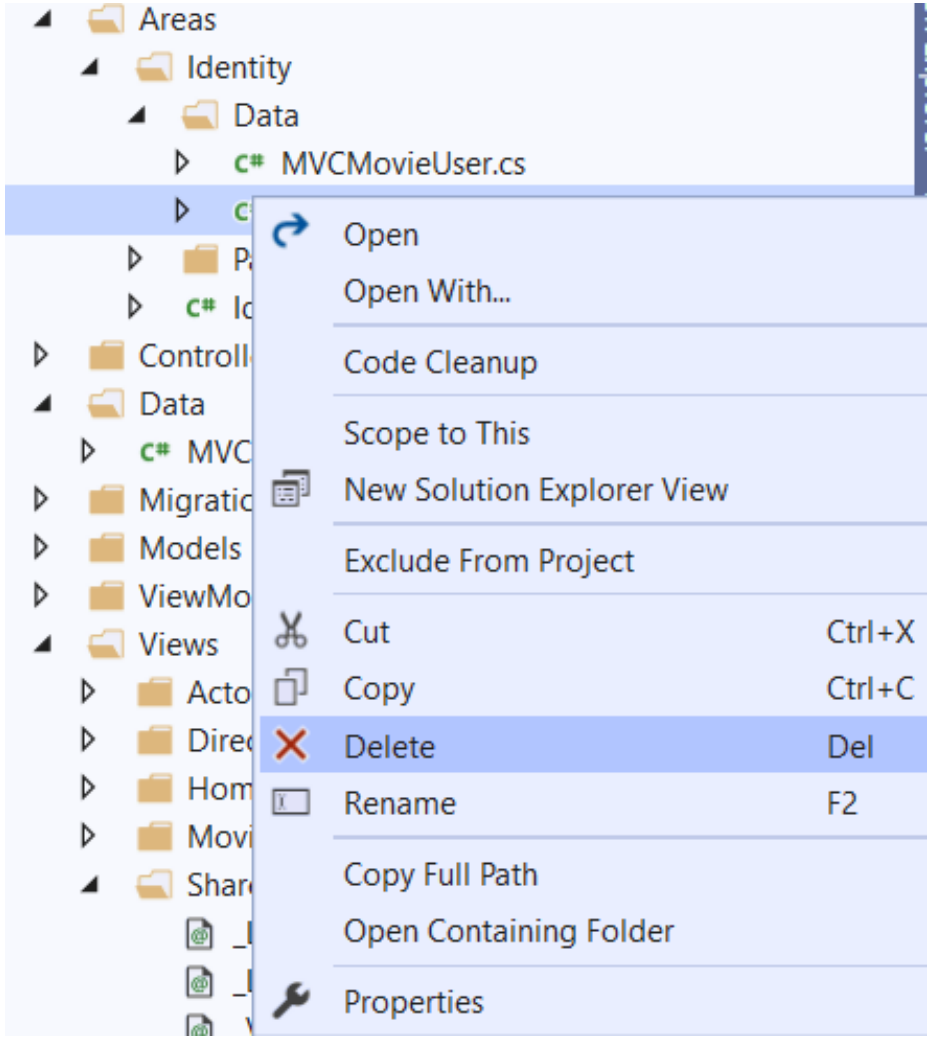
Identity is scaffolded using Razor Pages




```
using Microsoft.AspNetCore.Identity;

namespace MVCMovie.Areas.Identity.Data
{
    // Add profile data for application users
    24 references
    public class MVCMovieUser : IdentityUser
    {
    }
}
```

```
namespace Microsoft.AspNetCore.Identity
{
    public class IdentityUser<TKey> where TKey : IEquatable<TKey>
    {
        public IdentityUser();
        public IdentityUser(string userName);

        public virtual DateTimeOffset? LockoutEnd { get; set; }
        public virtual bool TwoFactorEnabled { get; set; }
        public virtual bool PhoneNumberConfirmed { get; set; }
        public virtual string PhoneNumber { get; set; }
        public virtual string ConcurrencyStamp { get; set; }
        public virtual string SecurityStamp { get; set; }
        public virtual string PasswordHash { get; set; }
        public virtual bool EmailConfirmed { get; set; }
        public virtual string NormalizedEmail { get; set; }
        public virtual string Email { get; set; }
        public virtual string NormalizedUserName { get; set; }
        public virtual string UserName { get; set; }
        public virtual TKey Id { get; set; }
        public virtual bool LockoutEnabled { get; set; }
        public virtual int AccessFailedCount { get; set; }

        public override string ToString();
    }
}
```

Delete Areas/Data/MVCMovieUserContext class
Delete the connection string in appsettings.json



appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "MVCMovieContext": "Server=(localdb)\\mssqllocaldb;Database=MVCMovieContext-2251ab04-26d3-42b8-b9a3-ebaca61ca968;Trusted_Connection=True;MultipleActiveResultSets=true",
    "MVCMovieUserContextConnection": "Server=(localdb)\\mssqllocaldb;Database=MVCMovie;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```

Remove the mssql2 from Properties/serviceDependencies.json and Properties/serviceDependencies.local.json

```
{
  "dependencies": {
    "mssql1": {
      "type": "mssql",
      "connectionId": "MVCMovieContext"
    },
    "mssql2": {
      "type": "mssql",
      "connectionId": "MVCMovieUserContextConnection"
    }
  }
}
```

```
{
  "dependencies": {
    "mssql1": {
      "type": "mssql.local",
      "connectionId": "MVCMovieContext"
    },
    "mssql2": {
      "type": "mssql.local",
      "connectionId": "MVCMovieUserContextConnection"
    }
  }
}
```

Change the existing db context (Data/MVCMovieContext) to incorporate Identity

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using MVCMovie.Areas.Identity.Data;
using MVCMovie.Models;

namespace MVCMovie.Data
{
    public class MVCMovieContext : IdentityDbContext<MVCMovieUser>
    {
        public MVCMovieContext (DbContextOptions<MVCMovieContext> options)
            : base(options) { }

        public DbSet<MVCMovie.Models.Movie> Movie { get; set; }
        public DbSet<MVCMovie.Models.Director> Director { get; set; }
        public DbSet<MVCMovie.Models.Actor> Actor { get; set; }
        public DbSet<MVCMovie.Models.ActorMovie> ActorMovie { get; set; }

        protected override void OnModelCreating(ModelBuilder builder) {
            base.OnModelCreating(builder);
        }
    }
}
```

Change Program.cs to include and configure Identity services and features (1)

```
builder.Services.AddDbContext<MVCMovieContext>(options =>
    options.UseSqlServer(builder.Configuration.GetConnectionString("MVCMovieContext")));
builder.Services.AddDefaultIdentity<MVCMovieUser>(options => options.SignIn.RequireConfirmedAccount = true)
.AddEntityFrameworkStores<MVCMovieUserContext>();builder.Services.AddDbContext<MVCMovieUserContext>(options =>
options.UseSqlServer(connectionString));

builder.Services.Configure<CookiePolicyOptions>(options =>
{
    // This lambda determines whether user consent for non-essential cookies is needed for a given request.
    options.CheckConsentNeeded = context => true;
    options.MinimumSameSitePolicy = SameSiteMode.None;
});
builder.Services.AddRazorPages();
builder.Services.AddIdentity<MVCMovieUser,IdentityRole>().AddEntityFrameworkStores<MVCMovieContext>().AddDefaultTokenProviders();

...

// Add services to the container.
builder.Services.AddControllersWithViews()
    .AddNewtonsoftJson(x =>
    {
        x.SerializerSettings.Formatting = Newtonsoft.Json.Formatting.Indented;
        x.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore;
        x.SerializerSettings.PreserveReferencesHandling = Newtonsoft.Json.PreserveReferencesHandling.None;
    })
    .AddRazorPagesOptions(options =>
    {
        options.Conventions.AuthorizeAreaFolder("Identity", "/Account/Manage");
        options.Conventions.AuthorizeAreaPage("Identity", "/Account/Logout");
    });
```

Change Program.cs to include and configure Identity services and features (2)

```
//Password Strength Setting
builder.Services.Configure<IdentityOptions>(options =>
{
    // Password settings
    options.Password.RequireDigit = true;
    options.Password.RequiredLength = 8;
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequireUppercase = true;
    options.Password.RequireLowercase = false;
    options.Password.RequiredUniqueChars = 6;
    // Lockout settings
    options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(30);
    options.Lockout.MaxFailedAccessAttempts = 10;
    options.Lockout.AllowedForNewUsers = true;
    // User settings
    options.User.RequireUniqueEmail = true;
});

//Setting the Account Login page
builder.Services.ConfigureApplicationCookie(options =>
{
    // Cookie settings
    options.Cookie.HttpOnly = true;
    options.ExpireTimeSpan = TimeSpan.FromMinutes(30);
    options.LoginPath = $"/Identity/Account/Login";
    options.LogoutPath = $"/Identity/Account/Logout";
    options.AccessDeniedPath = $"/Identity/Account/AccessDenied";
    options.SlidingExpiration = true;
});
```

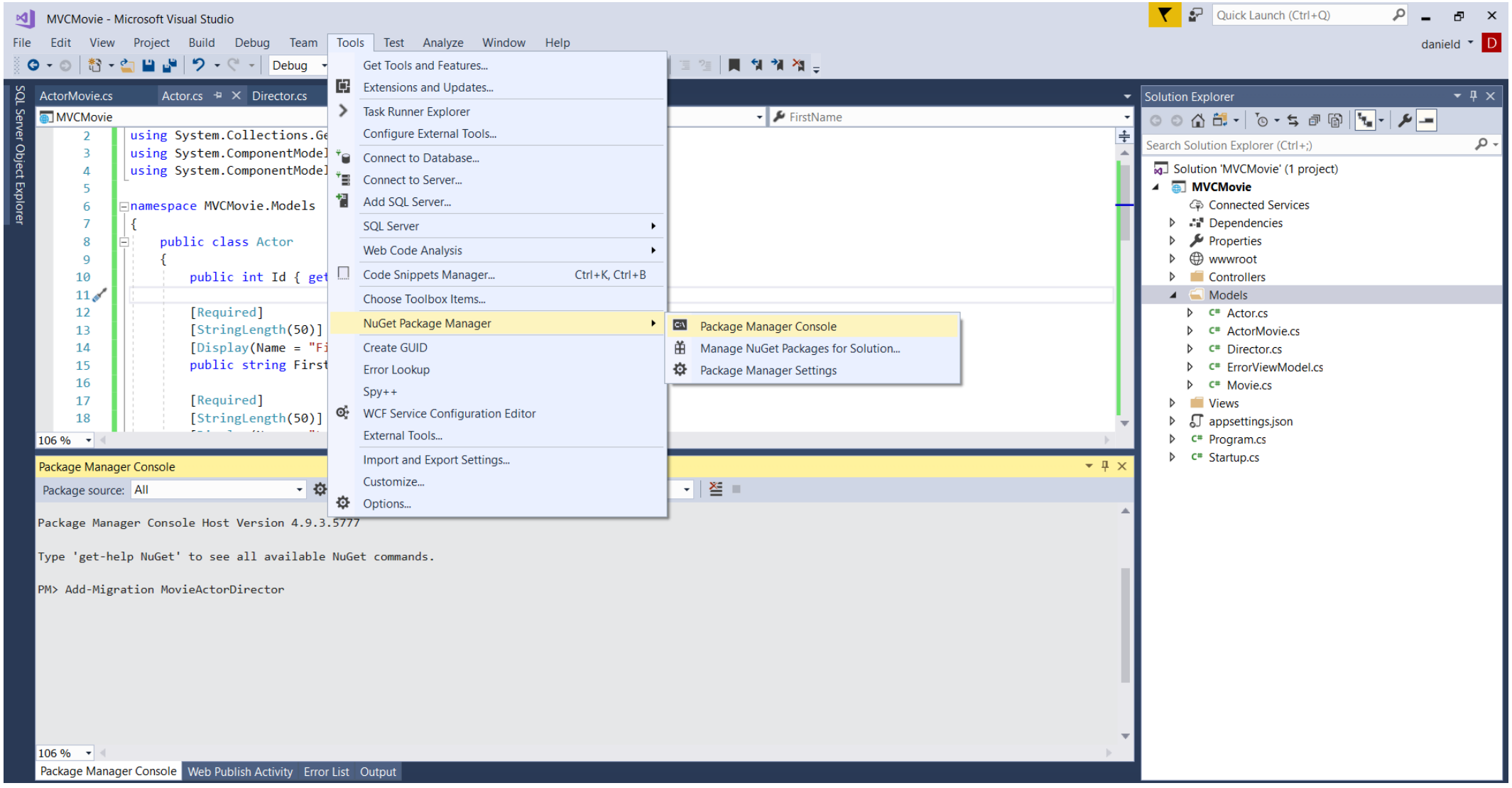
Change Program.cs to include and configure Identity services and features (3)

```
app.UseRouting();
app.UseCors();

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

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Movies}/{action=Index}/{id?}");
app.MapRazorPages();

app.Run();
```

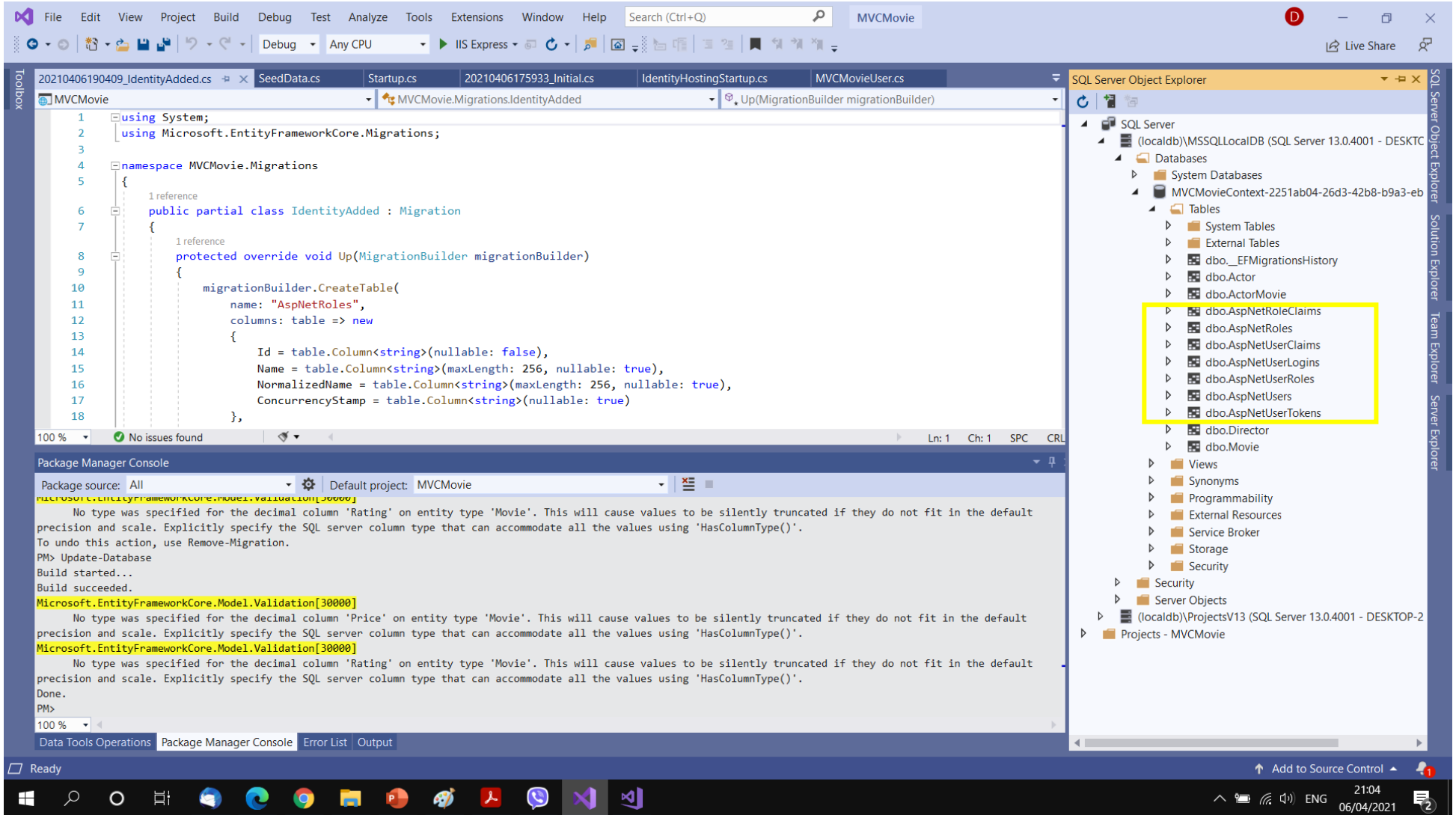
Upgrade the Models/SeedData class to seed admin role and user into the database

```
public class SeedData
{
    public static async Task CreateUserRoles(IServiceProvider serviceProvider)
    {
        var RoleManager = serviceProvider.GetRequiredService<RoleManager<IdentityRole>>();
        var UserManager = serviceProvider.GetRequiredService<UserManager<MVCMovieUser>>();
        IdentityResult roleResult;
        //Add Admin Role
        var roleCheck = await RoleManager.RoleExistsAsync("Admin");
        if (!roleCheck) { roleResult = await RoleManager.CreateAsync(new IdentityRole("Admin")); }

        MVCMovieUser user = await UserManager.FindByEmailAsync("admin@mvcmovie.com");
        if (user == null) {
            var User = new MVCMovieUser();
            User.Email = "admin@mvcmovie.com";
            User.UserName = "admin@mvcmovie.com";
            string userPWD = "Admin123";
            IdentityResult chkUser = await UserManager.CreateAsync(User, userPWD);
            //Add default User to Role Admin
            if (chkUser.Succeeded) { var result1 = await UserManager.AddToRoleAsync(User, "Admin"); }
        }
    }

    public static void Initialize(IServiceProvider serviceProvider)
    {
        using (var context = new MVCMovieContext(
            serviceProvider.GetRequiredService<
                DbContextOptions<MVCMovieContext>>()))
        {
            CreateUserRoles(serviceProvider).Wait();
...
```

1. Start the Package Manager Console (PMC)
2. Type “Add-Migration IdentityAdded”



1. Type “Update-Database” in PMC to update the DB
2. Check if DB contains the Identity tables



Change the Views/Shared/_Layout to show the login and register buttons

```
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
            <div class="container">
                <a class="navbar-brand" asp-area="" asp-controller="Movies" asp-action="Index">Movie App</a>
                <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">

                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Movies" asp-action="Index">Movies</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Directors" asp-action="Index">Directors</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Actors" asp-action="Index">Actors</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
                        </li>
                    </ul>
                    <partial name="_LoginPartial" />
                        ...
```

Modify the Views/Shared/_LoginPartial -> remove register button

```
@using Microsoft.AspNetCore.Identity
@using MVCMovie.Areas.Identity.Data

@inject SignInManager<MVCMovieUser> SignInManager
@inject UserManager<MVCMovieUser> UserManager

<ul class="navbar-nav">
@if (SignInManager.IsSignedIn(User))
{
    <li class="nav-item">
        <a id="manage" class="nav-link text-dark" asp-area="Identity" asp-page="/Account/Manage/Index" title="Manage">Hello
@UserManager.GetUserName(User)!</a>
    </li>
    <li class="nav-item">
        <form id="logoutForm" class="form-inline" asp-area="Identity" asp-page="/Account/Logout" asp-route-returnUrl="@Url.Action("Index",
"Home", new { area = "" })">
            <button id="logout" type="submit" class="nav-link btn btn-link text-dark">Logout</button>
        </form>
    </li>
}
else
{
    <li class="nav-item">
        <a class="nav-link text-dark" id="register" asp-area="Identity" asp-page="/Account/Register">Register</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" id="login" asp-area="Identity" asp-page="/Account/Login">Login</a>
    </li>
}
</ul>
```

Modify MoviesController.cs to allow only Admin users to Create, Edit, Delete movies

```
[Authorize(Roles = "Admin")]
public IActionResult Create()
...

[Authorize(Roles = "Admin")]
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("Id,Title,ReleaseDate,Genre,Price,Rating,DirectorId")] Movie movie)
...

[Authorize(Roles = "Admin")]
public async Task<IActionResult> Edit(int? id)
...

[Authorize(Roles = "Admin")]
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, MovieActorsEditViewModel viewmodel)
...

[Authorize(Roles = "Admin")]
public async Task<IActionResult> Delete(int? id)
...

[Authorize(Roles = "Admin")]
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DeleteConfirmed(int id)
```

Show Create New, Edit and Delete options only to logged in Admin users (Views/Movies/Index.cshtml and Details.cshmtl)

```
@model MVCMovie.ViewModels.MovieGenreViewModel

@{
    ViewData["Title"] = "Index";
}

<h1>Index</h1>

@if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))
{
    <p>
        <a asp-action="Create">Create New</a>
    </p>
}
...
                <td>
                    @if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))
                    { <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> } |
                    <a asp-action="Details" asp-route-id="@item.Id">Details</a> |
                    @if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))
                    { <a asp-action="Delete" asp-route-id="@item.Id">Delete</a> }
                </td>
            </tr>
        }
    </tbody>
</table>
```

```
@model MVCMovie.Models.Movie

@{
    ViewData["Title"] = "Details";
}

<h1>Details</h1>

<div>
    @if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))
    { <a asp-action="Edit" asp-route-id="@Model.Id">Edit</a> } |
    <a asp-action="Index">Back to List</a>
</div>
```

Do the same for the Directors and Actors!
(only Admins can create, edit, delete)