**Authentication vs. Authorization**

Authentication is a process in which a user provides credentials that are then compared to those stored in an operating system, database, app or resource. If they match, users authenticate successfully, and can then perform actions that they're authorized for, during an authorization process. The authorization refers to the process that determines what a user is allowed to do.

**ASP.NET Core Identity Framework**

ASP.NET Core Identity is a membership system that adds login functionality to ASP.NET Core apps. 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.

**Identity Components**

All the Identity dependent NuGet packages are included in the **Microsoft.AspNetCore.App** metapackage.
The primary package for Identity is **Microsoft.AspNetCore.Identity**.

**The Identity model**

**Entity types**

The Identity model consists of the following entity types.

| Entity type | Description |
|---|---|
| IdentityUser | Represents the user. |
| IdentityRole | Represents a role. |
| IdentityUserClaim | Represents a claim that a user possesses. |
| IdentityUserToken | Represents an authentication token for a user. |
| IdentityUserLogin | Associates a user with a login. |
| IdentityRoleClaim | Represents a claim that's granted to all users within a role. |
| IdentityUserRole | A join entity that associates users and roles. |

**Entity type relationships**

The entity types are related to each other in the following ways:

- Each User can have many UserClaims.
- Each User can have many UserLogins.
- Each User can have many UserTokens.
- Each Role can have many associated RoleClaims.
- Each User can have many associated Roles, and each Role can be associated with many Users.

**Microsoft.AspNetCore.Identity.UserManager<TUser>**

- This class basically provides a façade for signing users in and out. Exposes user related APIs which will automatically save changes to the UserStore.
- The UserManager knows when to hash a password, when to validate a user, and how to manage claims.

**Microsoft.AspNetCore.Identity.SignInManager<Tuser>**

*SignInManager* makes it easier to add **Two-Factor authentication**, account lockout and other features when you login.

It internally uses ApplicationUserManager and IAuthenticationManager for implementing its functionalities.

## Walkthrough

1. Create a New ASP.NET Core project with **Individual User Accounts**
2. Create a migration and update the database

```
Update-Database
```

3. Look at the Autogenerated code in Startup.cs and Data\ApplicationDbContext.cs an Areas\Identity

**AddDefaultIdentity and AddIdentity**

AddDefaultIdentity was introduced in ASP.NET Core 2.1. Calling `AddDefaultIdentity` is similar to calling the following:

- AddIdentity
- AddDefaultUI
- AddDefaultTokenProviders

**UseAuthentication** adds authentication middleware to the request pipeline

4. Add the following to Configure method in Startup Class.

```
app.UseAuthentication();
```

5. Add **[Authorize]** to the Privacy page.
6. Run the application and Register a User.
7. Find the **database name** from appsettings.json and open the same using **SQL Server Object Explorer** → Look at the **AspNetUsers** table.

8. If required edit the ConfigureService Method.

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


  // services.AddDefaultIdentity<IdentityUser>()
```

```
services.AddIdentity<IdentityUser>()
    .AddEntityFrameworkStores<ApplicationDbContext>();

services.ConfigureApplicationCookie(options =>
{
    options.LoginPath = $"/Identity/Account/Login";
    options.LogoutPath = $"/Identity/Account/Logout";
    options.AccessDeniedPath = $"/Identity/Account/AccessDenied";
});

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
```

## Scaffold Identity

ASP.NET Core provides **ASP.NET Core Identity** as a **Razor Class Library (RCL)**.

You might want to generate **source code** so you can modify the code and change the behavior.

For example, you could instruct the scaffolder to generate the code used in registration. Generated code takes precedence over the same code in the Identity RCL.

1. **Create a New Project (don't use any Authentication Option)**
2. Right click on Project → Add → New Scaffold Item → Select Identity in left tab → Identity → Add
3. In the **ADD Identity** dialog , select /Check the options you want.
4. Select the **+** button to create a new **User class**. Accept the type (**WebApp1User** if the project is named **WebApp1**) > **Add**.
5. Create a migration and update the database.
6. Add app.**UseAuthentication** to Startup.Configure method.
7. Add `<partial name="_LoginPartial" />` to the layout file.
8. **Examine Register:**
   - **a.** When a user clicks the **Register** link, the **RegisterModel.OnPostAsync** action is invoked. The user is created by **CreateAsync** on the **_userManager** object. _userManager is provided by dependency injection):
   - **b.** When the form on the Login page is submitted, the **OnPostAsync** action is called. **PasswordSignInAsync** is called on the **_signInManager**
   - **c.** The **Log out** link invokes the **LogoutModel.OnPost** action.

## Add custom user data to Identity in an ASP.NET Core project

9. **Add custom user data to the Identity DB**

```
public class DemoUser : IdentityUser
{
   [PersonalData]
   public string Name { get; set; }
   [PersonalData]
   public DateTime DOB { get; set; }
}
```

10. Properties decorated with the PersonalData attribute are:
    1. Deleted when the *Areas/Identity/Pages/Account/Manage/DeletePersonalData.cshtml* Razor Page calls `UserManager.Delete`.
    2. Included in the downloaded data by the *Areas/Identity/Pages/Account/Manage/DownloadPersonalData.cshtml* Razor Page.

11. Look at the DemoUser and DemoContext classes under the folder "Areas.Identity.Data"
12. Update **Startup.ConfigureServices** to use the new **DemoUser** class:

```
services.AddDefaultIdentity<DemoUser>()
      .AddEntityFrameworkStores<DemoContext>()
             .AddDefaultUI();
```

13. Update the **Account/Manage/Index.cshtml.cs** page

```
public partial class IndexModel : PageModel
{
   public InputModel Input { get; set; }

   public class InputModel
   {
      [Required]
      [DataType(DataType.Text)]
      [Display(Name = "Full name")]
      public string Name { get; set; }

      [Required]
      [Display(Name = "Birth Date")]
      [DataType(DataType.Date)]
      public DateTime DOB { get; set; }
```

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

        [Phone]
        [Display(Name = "Phone number")]
        public string PhoneNumber { get; set; }
    }

    public async Task<IActionResult> OnGetAsync()
    {
        Input = new InputModel
        {
            Name = user.Name,
            DOB = user.DOB,
            Email = email,
            PhoneNumber = phoneNumber
        };
    }

    public async Task<IActionResult> OnPostAsync()
    {
        if (Input.Name != user.Name)
        {
            user.Name = Input.Name;
        }

        if (Input.DOB != user.DOB)
        {
            user.DOB = Input.DOB;
        }
```

```
    var email = await _userManager.GetEmailAsync(user);
```

```
       // . . .

        await _userManager.UpdateAsync(user);
        await _signInManager.RefreshSignInAsync(user);
```

```
    StatusMessage = "Your profile has been updated";
    return RedirectToPage();
  }
```

14. Update the *Areas/Identity/Pages/Account/Manage/Index.cshtml* with the following highlighted markup:

```
@page
@model IndexModel
@{
  ViewData["Title"] = "Profile";
}
<h4>@ViewData["Title"]</h4>
@Html.Partial("_StatusMessage", Model.StatusMessage)
<div class="row">
  <div class="col-md-6">
    <form id="profile-form" method="post">
      <div class="form-group">
        <div class="form-group">
          <label asp-for="Input.Name"></label>
          <input asp-for="Input.Name" class="form-control" />
        </div>
        <div class="form-group">
          <label asp-for="Input.DOB"></label>
          <input asp-for="Input.DOB" class="form-control" />
        </div>
        <label asp-for="Input.PhoneNumber"></label>
        //. . .
    </form>
  </div>
</div>
```

15. Update the *Areas/Identity/Pages/Account/Register.cshtml.cs*

```
public class InputModel
{
  [Required]
  [DataType(DataType.Text)]
  [Display(Name = "Full name")]
  public string Name { get; set; }
```

```
    [Required]
    [Display(Name = "Birth Date")]
    [DataType(DataType.Date)]
    public DateTime DOB { get; set; }
    . . .
}

public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl = returnUrl ?? Url.Content("~/");
    if (ModelState.IsValid)
    {
        var user = new WebApp1User { UserName = Input.Email,  Email = Input.Email,
            Name = Input.Name,
            DOB = Input.DOB
        };
        var result = await _userManager.CreateAsync(user, Input.Password);
    }

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

16. Update the *Areas/Identity/Pages/Account/Register.cshtml* with the following highlighted markup

```
<div class="row">
    <div class="col-md-4">
        <form asp-route-returnUrl="@Model.ReturnUrl" method="post">

            <div class="form-group">
                <label asp-for="Input.Name"></label>
                <input asp-for="Input.Name" class="form-control" />
                <span asp-validation-for="Input.Name" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Input.DOB"></label>
                <input asp-for="Input.DOB" class="form-control" />
                <span asp-validation-for="Input.DOB" class="text-danger"></span>
            </div>
        </form>
    </div>
</div>
```

17. Build the project
18. Add a migration for the custom user data

```
Add-Migration CustomUserData
Update-Database
```

19. Test the application.

## Configure ASP.NET Core Identity

```
services.Configure<IdentityOptions>(options =>
{
  // Default Lockout settings.
  options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);
  options.Lockout.MaxFailedAccessAttempts = 5;
  options.Lockout.AllowedForNewUsers = true;

  options.Password.RequireDigit = true;
  options.Password.RequireLowercase = true;
  options.Password.RequireNonAlphanumeric = true;
  options.Password.RequireUppercase = true;
  options.Password.RequiredLength = 6;
  options.Password.RequiredUniqueChars = 1;

  options.SignIn.RequireConfirmedEmail = true;
  options.SignIn.RequireConfirmedPhoneNumber = false;

  options.User.AllowedUserNameCharacters =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-._@+";
  options.User.RequireUniqueEmail = true;

  options.AccessDeniedPath = "/Identity/Account/AccessDenied";
  options.Cookie.Name = "YourAppCookieName";
```

```
    options.Cookie.HttpOnly = true;
    options.ExpireTimeSpan = TimeSpan.FromMinutes(60);
    options.LoginPath = "/Identity/Account/Login";
    // ReturnUrlParameter requires
    //using Microsoft.AspNetCore.Authentication.Cookies;
    options.ReturnUrlParameter = CookieAuthenticationDefaults.ReturnUrlParameter;
    options.SlidingExpiration = true;
});

services.ConfigureApplicationCookie(options =>
{
    // Cookie settings
    options.Cookie.HttpOnly = true;
    options.ExpireTimeSpan = TimeSpan.FromMinutes(5);
    options.LoginPath = "/Identity/Account/Login";
    options.AccessDeniedPath = "/Identity/Account/AccessDenied";
    options.SlidingExpiration = true;
});
```

## Role Based Authorization

1. Add the following to Configure Method of Area/Identity/**IdentityHostingStartup**.cs

```
services.AddDefaultIdentity<ApplicationUser>()
            .AddRoles<IdentityRole>()
            .AddEntityFrameworkStores<ApplicationDbContext>();
```

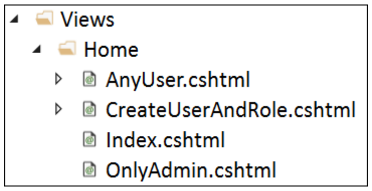
2. **Add the following class to the project**

```
public class User
{
    public string Email { get; set; }
    public string Password { get; set; }
    public string RoleName { get; set; }
}
```

3. Add the following to **HomeController.cs**

```
public class HomeController : Controller
{
    RoleManager<IdentityRole> _roleManager;
    UserManager<ApplicationUser> _userManager;

    public HomeController(UserManager<ApplicationUser> userManager, RoleManager<IdentityRole>
roleManager, SignInManager<ApplicationUser> signInManager)
    {
        _userManager = userManager;
        _roleManager = roleManager;
    }
    public IActionResult Index()
    {
        return View();
    }

    public IActionResult CreateUserAndRole()
    {
        User user = new User();
        return View(user);
    }

    [HttpPost]
    public async Task<IActionResult> CreateUserAndRole(User user1)
    {
        IdentityResult roleResult;
        var roleExist = await _roleManager.RoleExistsAsync(user1.RoleName);
        if (!roleExist)
            roleResult = await _roleManager.CreateAsync(new IdentityRole(user1.RoleName));

        var user = new ApplicationUser
        {
            UserName = user1.Email,
            Email = user1.Email
        };
        var _user = await _userManager.FindByEmailAsync(user1.Email);
        if (_user == null)
        {
            var result = await _userManager.CreateAsync(user, user1.Password);
            if (result.Succeeded)
                await _userManager.AddToRoleAsync(user, user1.RoleName);
        }
        return View();
    }
    [Authorize()]
    public IActionResult AnyUser()
    {
        return View();
    }
    [Authorize(Roles ="Admin")]
    public IActionResult OnlyAdmin()
    {
        return View();
    }
}
```

4. Add the following Views



5. Add the following to CreateUserAndRole.cshtml

```
@model User
<h1>Create User And Role</h1>
<form 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="Email"></label>
    <input asp-for="Email" class="form-control" />
    <span asp-validation-for="Email" class="text-danger"></span>
  </div>
  <div class="form-group">
    <label asp-for="Password"></label>
    <input asp-for="Password" class="form-control" />
    <span asp-validation-for="Password" class="text-danger"></span>
  </div>
  <div class="form-group">
    <label asp-for="RoleName"></label>
    <input asp-for="RoleName" class="form-control" />
    <span asp-validation-for="RoleName" class="text-danger"></span>
  </div>
  <button type="submit" class="btn btn-primary">Register</button>
</form>
```

6. Add the following to AnyUser.cshtml
7. Add the following to OnlyAdmin.cshtml
8. Run the Application. Visit http://localhost:56117/Home/CreateUserAndRole
9. Register new users. One with "**Admin"** role and another with a "**Demo**" role.
10. Visit: http://localhost:56117/Home/AnyUser
11. Visit http://localhost:56117/Home/Onlyadmin

12. Note that with user having "**Admin**" Role we are able to view both the pages and with "**Demo**" Role we are able to visit only AnyUser. We get **Access Denied** for OnlyAdmin

## External Authentication Providers

**Configure Facebook Authentication**

```
services.AddAuthentication().AddFacebook(facebookOptions =>
{
    facebookOptions.AppId = Configuration["Authentication:Facebook:AppId"];
    facebookOptions.AppSecret = Configuration["Authentication:Facebook:AppSecret"];
});
```

https://docs.microsoft.com/en-us/aspnet/core/security/authentication/social/facebook-logins?view=aspnetcore-2.1

**Configure Google Authentication**

```
services.AddAuthentication().AddGoogle(googleOptions =>
{
    googleOptions.ClientId = Configuration["Authentication:Google:ClientId"];
    googleOptions.ClientSecret = Configuration["Authentication:Google:ClientSecret"];
});
```

https://docs.microsoft.com/en-us/aspnet/core/security/authentication/social/google-logins?view=aspnetcore-2.1

**Multiple authentication providers**

When the app requires multiple providers, chain the provider extension methods behind AddAuthentication:

```
services.AddAuthentication()
    .AddMicrosoftAccount(microsoftOptions => { ... })
    .AddGoogle(googleOptions => { ... })
    .AddTwitter(twitterOptions => { ... })
    .AddFacebook(facebookOptions => { ... });
```

## Integrating Azure AD into an ASP.NET Core web app

1. Add the following to appsettings.json

```
"AzureAd": {
    "Instance": "https://login.microsoftonline.com/",
    "Domain": "[Enter the domain of your tenant, e.g. contoso.onmicrosoft.com]",
    "TenantId": "[Enter the Tenant Id (Obtained from the Azure portal. Select 'Endpoints' from the 'App
registrations' blade and use the GUID in any of the URLs), e.g. da41245a5-11b3-996c-00a8-4d99re19f292]",
  "ClientId": "[Enter the Client Id (Application ID obtained from the Azure portal), e.g. ba74781c2-53c2-442a-97c2-
3d60re42f403]",
  "CallbackPath": "/signin-oidc"
},
```

2. Add the following to ConfigureService Method

```
services.AddAuthentication(sharedOptions =>
{
  sharedOptions.DefaultScheme = CookieAuthenticationDefaults.AuthenticationScheme;
  sharedOptions.DefaultChallengeScheme = OpenIdConnectDefaults.AuthenticationScheme;
})
.AddAzureAd(options => Configuration.Bind("AzureAd", options))
.AddCookie();
```

3. **Add the following to Configure method**

```
app.UseAuthentication();
```

4. **Controller/AccountController.cs**

```
[Route("[controller]/[action]")]
public class AccountController : Controller
{
  [HttpGet]
  public IActionResult SignIn()
  {
    var redirectUrl = Url.Action(nameof(HomeController.Index), "Home");
    return Challenge(
      new AuthenticationProperties { RedirectUri = redirectUrl },
      OpenIdConnectDefaults.AuthenticationScheme);
  }

  [HttpGet]
  public IActionResult SignOut()
  {
    var callbackUrl = Url.Action(nameof(SignedOut), "Account", values: null, protocol: Request.Scheme);
    return SignOut(
      new AuthenticationProperties { RedirectUri = callbackUrl },
      CookieAuthenticationDefaults.AuthenticationScheme,
      OpenIdConnectDefaults.AuthenticationScheme);
  }
```

```
    [HttpGet]
    public IActionResult SignedOut()
    {
        if (User.Identity.IsAuthenticated)
        {
            // Redirect to home page if the user is authenticated.
            return RedirectToAction(nameof(HomeController.Index), "Home");
        }

        return View();
    }

    [HttpGet]
    public IActionResult AccessDenied()
    {
        return View();
    }
}
```

**Sample App**: E:\NET Core\ASP.NET Core\ASP NET Core Course Material\015 - Security\AzureActiveDirectory\active-directory-dotnet-webapp-openidconnect-aspnetcore

**Explanation:** https://azure.microsoft.com/en-us/resources/samples/active-directory-dotnet-webapp-openidconnect-aspnetcore/