Introduction to Identity on ASP.NET Core

ASP.NET Core Identity is a membership system which allows you to add login functionality to your application. Users can create an account and login with a user name and password or they can use an external login provider such as Facebook, Google, Microsoft Account, Twitter or others.

You can configure ASP.NET Core Identity to use a SQL Server database to store user names, passwords, and profile data. Alternatively, you can use your own persistent store, for example, an Azure Table Storage. This document contains instructions for Visual Studio and for using the CLI.

<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-2.1&tabs=visual-studio%2Caspnetcore2x>

1. Configure Identity services and add middleware in Startup.

The Identity services are added to the application in the ConfigureServices  method in the Startup class:

* + [ASP.NET Core 2.x](https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-2.1&tabs=visual-studio%2Caspnetcore2x#tabpanel_CeZOj-G++Q-1_aspnetcore2x)

C#Copy

// 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("DefaultConnection")));

services.AddIdentity<ApplicationUser, IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders();

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;

});

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;

});

// Add application services.

services.AddTransient<IEmailSender, EmailSender>();

services.AddMvc();

}

These services are made available to the application through [dependency injection](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-2.1).

Identity is enabled for the application by calling UseAuthentication in the Configure method. UseAuthentication adds authentication [middleware](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/index?view=aspnetcore-2.1) to the request pipeline.

C#Copy

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

app.UseBrowserLink();

app.UseDatabaseErrorPage();

}

else

{

app.UseExceptionHandler("/Home/Error");

}

app.UseStaticFiles();

app.UseAuthentication();

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

template: "{controller=Home}/{action=Index}/{id?}");

});

}

For more information about the application start up process, see [Application Startup](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/startup?view=aspnetcore-2.1).

1. Create a user.

Launch the application and then click on the **Register** link.

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.

Alternately, you can test using ASP.NET Core Identity with your app without a persistent database by using an in-memory database. To use an in-memory database, add the Microsoft.EntityFrameworkCore.InMemory package to your app and modify your app's call to AddDbContext in ConfigureServices as follows:

C#Copy

services.AddDbContext<ApplicationDbContext>(options =>

options.UseInMemoryDatabase(Guid.NewGuid().ToString()));

When the user clicks the **Register** link, the Register action is invoked on AccountController. The Register action creates the user by calling CreateAsync on the \_userManager object (provided to AccountController by dependency injection):

C#Copy

//

// POST: /Account/Register

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

var user = new ApplicationUser { UserName = model.Email, Email = model.Email };

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

if (result.Succeeded)

{

// For more information on how to enable account confirmation and password reset please visit http://go.microsoft.com/fwlink/?LinkID=532713

// Send an email with this link

//var code = await \_userManager.GenerateEmailConfirmationTokenAsync(user);

//var callbackUrl = Url.Action("ConfirmEmail", "Account", new { userId = user.Id, code = code }, protocol: HttpContext.Request.Scheme);

//await \_emailSender.SendEmailAsync(model.Email, "Confirm your account",

// "Please confirm your account by clicking this link: <a href=\"" + callbackUrl + "\">link</a>");

await \_signInManager.SignInAsync(user, isPersistent: false);

\_logger.LogInformation(3, "User created a new account with password.");

return RedirectToAction(nameof(HomeController.Index), "Home");

}

AddErrors(result);

}

// If we got this far, something failed, redisplay form

return View(model);

}

If the user was created successfully, the user is logged in by the call to \_signInManager.SignInAsync.

**Note:** See [account confirmation](https://docs.microsoft.com/en-us/aspnet/core/security/authentication/accconfirm?view=aspnetcore-2.1#prevent-login-at-registration) for steps to prevent immediate login at registration.

1. 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. When the user submits the form on the Login page, the AccountController Login action is called.

The Login action calls PasswordSignInAsync on the \_signInManager object (provided to AccountController by dependency injection).

C#Copy

//

// POST: /Account/Login

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null)

{

ViewData["ReturnUrl"] = returnUrl;

if (ModelState.IsValid)

{

// This doesn't count login failures towards account lockout

// To enable password failures to trigger account lockout, set lockoutOnFailure: true

var result = await \_signInManager.PasswordSignInAsync(model.Email,

model.Password, model.RememberMe, lockoutOnFailure: false);

if (result.Succeeded)

{

\_logger.LogInformation(1, "User logged in.");

return RedirectToLocal(returnUrl);

}

if (result.RequiresTwoFactor)

{

return RedirectToAction(nameof(SendCode), new { ReturnUrl = returnUrl, RememberMe = model.RememberMe });

}

if (result.IsLockedOut)

{

\_logger.LogWarning(2, "User account locked out.");

return View("Lockout");

}

else

{

ModelState.AddModelError(string.Empty, "Invalid login attempt.");

return View(model);

}

}

// If we got this far, something failed, redisplay form

return View(model);

}

The base Controller class exposes a User property that you can access from controller methods. For instance, you can enumerate User.Claims and make authorization decisions. For more information, see [Authorization](https://docs.microsoft.com/en-us/aspnet/core/security/authorization/index?view=aspnetcore-2.1).

1. Log out.

Clicking the **Log out** link calls the LogOut action.

C#Copy

//

// POST: /Account/LogOut

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> LogOut()

{

await \_signInManager.SignOutAsync();

\_logger.LogInformation(4, "User logged out.");

return RedirectToAction(nameof(HomeController.Index), "Home");

}

The preceding code above calls the \_signInManager.SignOutAsync method. The SignOutAsync method clears the user's claims stored in a cookie.

6. Configuration.

Identity has some default behaviors that can be overridden in the app's startup class. IdentityOptions don't need to be configured when using the default behaviors. The following code sets several password strength options:

* [ASP.NET Core 2.x](https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-2.1&tabs=visual-studio%2Caspnetcore2x#tabpanel_CeZOj-G++Q-2_aspnetcore2x)

C#Copy

// 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("DefaultConnection")));

services.AddIdentity<ApplicationUser, IdentityRole>()

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders();

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;

});

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;

});

// Add application services.

services.AddTransient<IEmailSender, EmailSender>();

services.AddMvc();

}

For more information about how to configure Identity, see [Configure Identity](https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity-configuration?view=aspnetcore-2.1).

You also can configure the data type of the primary key, see [Configure Identity primary keys data type](https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity-primary-key-configuration?view=aspnetcore-2.1).

1. View the database.

If your app is using a SQL Server database (the default on Windows and for Visual Studio users), you can view the database the app created. You can use **SQL Server Management Studio**. Alternatively, from Visual Studio, select **View** > **SQL Server Object Explorer**. Connect to **(localdb)\MSSQLLocalDB**. The database with a name matching **aspnet-<name of your project>-<date string>** is displayed.

Expand the database and its **Tables**, then right-click the **dbo.AspNetUsers** table and select **View Data**.

1. 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 to the About action of the Home Controller.

C#Copy

[Authorize]

public IActionResult About()

{

ViewData["Message"] = "Your application description page.";

return View();

}

* + [Visual Studio](https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-2.1&tabs=visual-studio%2Caspnetcore2x#tabpanel_CeZOj-G++Q-3_visual-studio)
  + [.NET Core CLI](https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-2.1&tabs=visual-studio%2Caspnetcore2x#tabpanel_CeZOj-G++Q-3_netcore-cli)

Run the project using **Ctrl** + **F5** and navigate to the **About** page. Only authenticated users may access the **About** page now, so ASP.NET redirects you to the login page to login or register.

## Identity Components

The primary reference assembly for the Identity system is Microsoft.AspNetCore.Identity. This package contains the core set of interfaces for ASP.NET Core Identity, and is included by Microsoft.AspNetCore.Identity.EntityFrameworkCore.

These dependencies are needed to use the Identity system in ASP.NET Core applications:

* Microsoft.AspNetCore.Identity.EntityFrameworkCore - Contains the required types to use Identity with Entity Framework Core.
* Microsoft.EntityFrameworkCore.SqlServer - Entity Framework Core is Microsoft's recommended data access technology for relational databases like SQL Server. For testing, you can use Microsoft.EntityFrameworkCore.InMemory.
* Microsoft.AspNetCore.Authentication.Cookies - Middleware that enables an app to use cookie-based authentication.