Chapter 18: An introduction to ASP.NET Core Identity

Gill Cleeren

@gillcleeren

Agenda

- Introducing ASP.NET Core Identity
- Configuring the application for Identity
- Adding the login page
- Adding basic authorization

Introducing ASP.NET Core Identity

ASP.NET Core Identity is a membership system which allows us to manage users and create login functionality

ASP.NET Core Identity

- Adds login, authentication & authorization features to your site
- Users can create accounts
  - Identity will persist these in the data store
- Based on “old” membership
- Supports external providers
  - Google, Facebook…
- Works great on top of SQL Server
  - Other data stores are supported as well
- Configuration required in the Startup class

Configuring the application for Identity

Required packages for ASP.NET Identity

- Primary assembly that contains ASP.NET Identity:
  - Microsoft.AspNetCore.Identity.UI

Identity and the database

- Application database context needs to inherit from IdentityContext<IdentityUser> by default
  - We'll change this later

```
public class AppDbContext : IdentityDbContext<IdentityUser>
{
    public AppDbContext(DbContextOptions<AppDbContext> options)
      : base(options)
    {
    }
}
```

IdentityUser

- IdentityUser is built-in class to work with Users

```
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();
    }
}
```

SNOWBALL

Inheriting from IdentityUser

- We can create more properties by creating our own class that inherits from IdentityUser

```
public class PieShopUser : IdentityUser {
        //other properties can be added here
}
```

Application configuration

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<AppDbContext>(options =>
      options.UseSqlServer(

_configurationRoot.GetConnectionString("DefaultConnection")));

    services.AddIdentity<IdentityUser, IdentityRole>()
      .AddEntityFrameworkStores<AppDbContext>();
}
```

Application configuration

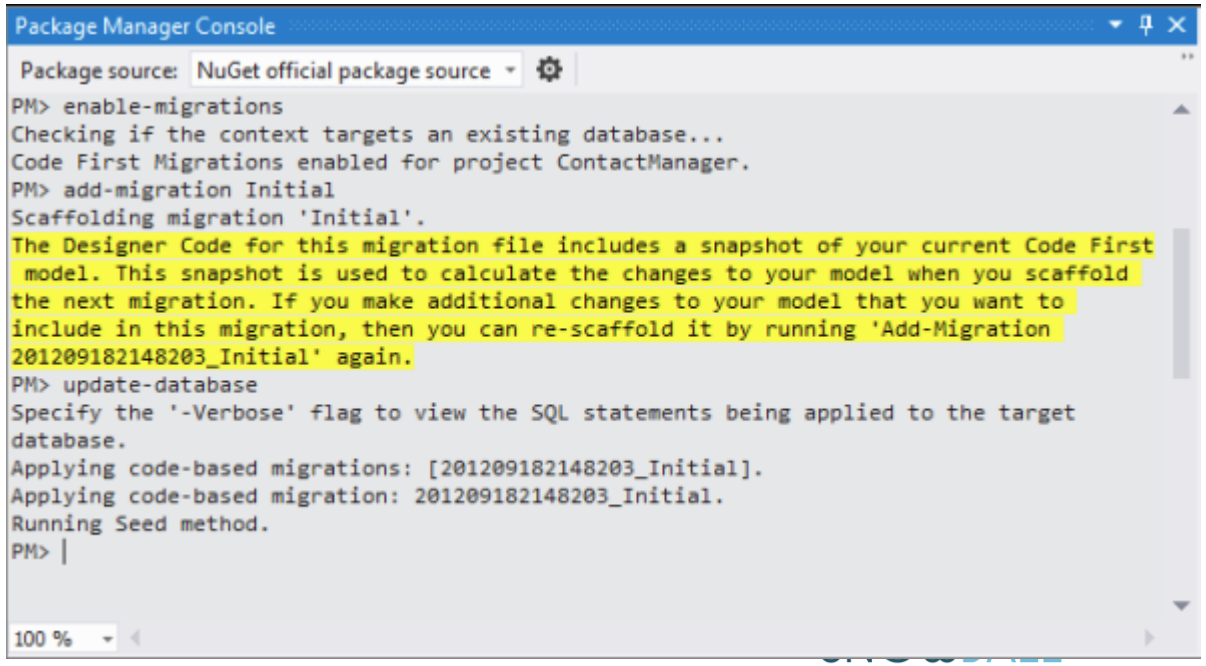
```
public void Configure(IApplicationBuilder app,
    IHostingEnvironment env, ILoggerFactory
loggerFactory)
{
    app.UseDeveloperExceptionPage();
    app.UseStatusCodePages();
    app.UseStaticFiles();

    app.UseSession();
    app.UseAuthentication();
}
```

More configuration options

```
services.Configure<IdentityOptions>(options =>
{
    options.Password.RequireDigit = true;
    options.Password.RequiredLength = 8;
    options.Password.RequireNonAlphanumeric = true;
    options.User.RequireUniqueEmail = true;
});
```

Don’t forget

- A database migration will be needed at this point
  - Add-Migration “IdentityAdded”
  - Update-Database



DEMO

Adding ASP.NET Core Identity

Configuring ASP.NET Core Identity

Adding the login page

Adding the login page

- We'll need
  - A login view
  - The Account Controller
  - Changes to the model
    - LoginViewModel

AccountController

```
public class AccountController : Controller
{
    public IActionResult Login(string returnUrl) {}
    public async Task<IActionResult>
      Login(LoginViewModel loginViewModel) {}

    public IActionResult Register() {}
    public async Task<IActionResult> Logout() {}
}
```

SigninManager and UserManager

- 2 Important classes to manage authentication
- UserManager
  - Creating users
  - Finding users
  - Get Claims for user
  - Get phone number/other properties of the user
  - Supports two-factor authentication
- SignInManager
  - Signing in user
  - Other functionality related to actual logging in of the user

Login method on AccountController

```
[HttpPost]
[AllowAnonymous]
2 references | Gill Cleeren, 109 days ago | 1 author, 1 change | 0 requests | 0 exceptions
public async Task<IActionResult> Login(LoginViewModel loginViewModel)
{
    if (!ModelState.IsValid)
        return View(loginViewModel);

    var user = await _userManager.FindByNameAsync(loginViewModel.UserName);

    if (user != null)
    {
        var result = await _signInManager.PasswordSignInAsync(user, loginViewModel.Password, false, false);
        if (result.Succeeded)
        {
            if (string.IsNullOrEmpty(loginViewModel.ReturnUrl))
                return RedirectToAction("Index", "Home");

            return Redirect(loginViewModel.ReturnUrl);
        }
    }

    ModelState.AddModelError("", "Username/password not found");
    return View(loginViewModel);
}
```

LoginViewModel

```
public class LoginViewModel
{
    [Required]
    [Display(Name="User name")]
    public string UserName { get; set; }

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

SNOWBALL

DEMO

Adding the Login page

Adding authorization

Authorization

- Is the process that checks what the user can and can’t do in the site
  - Typically, this is about restricting access to different parts of the site
- Is not really dependent on authentication
  - Authentication creates an identity for the logged-in user
- Types of authorization in ASP.NET Core
  - Role-based
  - Policy-based
  - Claims-based

The Authorize attribute

- Authorization is controlled using the [Authorize] attribute and its parameters
  - Without parameters, it will simply allow or deny an authenticated user access to the requested resource

```
[Authorize]
   public class AccountController : Controller
   {
       public ActionResult Login()
       {
       }

       public ActionResult Logout()
       {
       }
   }
```

SNOWBALL

The Authorize attribute

- Can also be applied on the action method level

```
public class AccountController : Controller
   {
       public ActionResult Login()
       {
       }

       [Authorize]
       public ActionResult Logout()
       {
       }
   }
```

AllowAnonymous

- Can be overruled

```
[Authorize]
   public class AccountController : Controller
   {
       [AllowAnonymous]
       public ActionResult Login()
       {
       }

       public ActionResult Logout()
       {
       }
   }
```

DEMO

Adding basic authorization

Summary

- ASP.NET Identity replaces Membership
- Covers authentication and authorization
- Configuration-based