100 XP

Exercise - Use claims with policy-based authorization

15 minutes

This module requires a sandbox to complete. You have used 4 of 10 sandboxes for today. More sandboxes will be available tomorrow.

Activate sandbox

Choose the ASP.NET Core Identity data store

PostgreSQL SQL Server

In this unit, you'll create a new user with administrative privileges. A demonstration of creating and storing user claims is provided. An authorization policy is also defined to determine whether an authenticated user has elevated privileges in the UI.

Secure the products catalog

The products catalog page should be visible only to authenticated users. However, only administrators are allowed to edit, create, and delete products.

1. In *Pages/Products/Index.cshtml.cs*, apply the following changes:

   a. Replace the `// Add [Authorize] attribute` comment with the following attribute:

   C# Copy

   ```
   [Authorize]
   ```

   The preceding attribute describes user authentication requirements for the page. In this case, there are no requirements beyond the user being authenticated. Anonymous users aren't allowed to view the page and are redirected to the login page.

   b. Uncomment the `//using Microsoft.AspNetCore.Authorization;` line at the top of the file.

The preceding change resolves the `[Authorize]` attribute in the previous step.

c. Replace the `// Add IsAdmin property` comment with the following property:

```
C#
```

```
public bool IsAdmin =>
    HttpContext.User.HasClaim("IsAdmin", bool.TrueString);
```

The preceding code determines whether the authenticated user has an `IsAdmin` claim with a value of `True`. The result of this evaluation is accessed via a read-only property named `IsAdmin`.

d. Replace the `// Add IsAdmin check` comment in the `OnDelete` method with the following code:

```
C#
```

```
if (!IsAdmin)
{
    return Forbid();
}
```

When an authenticated employee attempts to delete a product via the UI or by manually sending an HTTP DELETE request to this page, an HTTP 403 status code is returned.

2. In *Pages/Products/Index.cshtml*, update the **Edit**, **Delete**, and **Add Product** links with the highlighted code:

Edit & Delete links:

```
CSHTML
```

```
<td>
    @if (Model.IsAdmin)
    {
    <a asp-page="Edit" asp-route-id="@product.Id">Edit</a> <span>|</span>
    <a href="#" onclick="deleteProduct('@product.Id',
antiForgeryToken())">Delete</a>
    }
</td>
```

Add Product link:

```
CSHTML
```

```
@if (Model.IsAdmin)
{
<a asp-page="./Create">Add Product</a>
}
```

The preceding changes cause the links to be rendered only when the authenticated user is an administrator.

Register and apply the authorization policy

The **Create Product** and **Edit Product** pages should be accessible only to administrators. To encapsulate the authorization criteria for such pages, an `Admin` policy will be created.

1. In the `ConfigureServices` method of *Startup.cs*, make the following changes:

   a. Replace the `// Add call to AddAuthorization` comment with the following code:

   C# Copy

   ```
   services.AddAuthorization(options =>
       options.AddPolicy("Admin", policy =>
           policy.RequireAuthenticatedUser()
               .RequireClaim("IsAdmin", bool.TrueString)));
   ```

   The preceding code defines an authorization policy named `Admin`. The policy requires that the user is authenticated and has an `IsAdmin` claim set to `True`.

   b. Incorporate the following highlighted code:

   C# Copy

   ```
   services.AddAntiforgery(options => options.HeaderName = "X-CSRF-TOKEN");
   services.AddRazorPages(options =>
       options.Conventions.AuthorizePage("/Products/Edit", "Admin"));
   services.AddControllers();
   ```

   The `AuthorizePage` method call secures the */Products/Edit* Razor Page route by applying the `Admin` policy. An advantage to this approach is that the Razor Page being secured requires no modifications. The authorization aspect is instead managed in *Startup.cs*. Anonymous users will be redirected to the login page. Authenticated users who don't satisfy the policy requirements are presented an **Access denied** message.

2. In *Pages/Products/Create.cshtml.cs*, apply the following changes:

a. Replace the `// Add [Authorize(Policy = "Admin")] attribute` comment with the following attribute:

```
C#
[Authorize(Policy = "Admin")]
```

The preceding code represents an alternative to the `AuthorizePage` method call in *Startup.cs*. The `[Authorize]` attribute enforces that the `Admin` policy requirements are satisfied. Anonymous users will be redirected to the login page. Authenticated users who don't satisfy the policy requirements are presented an **Access denied** message.

b. Uncomment the `//using Microsoft.AspNetCore.Authorization;` line at the top of the file.

The preceding change resolves the `[Authorize(Policy = "Admin")]` attribute in the previous step.

Modify the registration page

Modify the registration page to allow administrators to register using the following steps.

1. In *Areas/Identity/Pages/Account/Register.cshtml.cs*, make the following changes:

a. Add the following property to the `InputModel` nested class:

```
C#
public class InputModel
{
    [DataType(DataType.Password)]
    [Display(Name = "Admin enrollment key")]
    public ulong? AdminEnrollmentKey { get; set; }

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

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

    [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; }
}
```

b. Apply the highlighted changes to the `OnPostAsync` method:

C# Copy

```
public async Task<IActionResult> OnPostAsync(
    [FromServices] AdminRegistrationTokenService tokenService,
    string returnUrl = null)
{
    returnUrl = returnUrl ?? Url.Content("~/");
    ExternalLogins = (await
_signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
    if (ModelState.IsValid)
    {
        var user = new ContosoPetsUser
        {
            FirstName = Input.FirstName,
            LastName = Input.LastName,
            UserName = Input.Email,
            Email = Input.Email,
        };
        var result = await _userManager.CreateAsync(user, Input.Password);
        if (result.Succeeded)
        {
            _logger.LogInformation("User created a new account with
password.");

            await _userManager.AddClaimAsync(user,
                new Claim("IsAdmin",
                    (Input.AdminEnrollmentKey ==
tokenService.CreationKey).ToString()));

            var code = await
_userManager.GenerateEmailConfirmationTokenAsync(user);
            code =
WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));
            var callbackUrl = Url.Page(
```

```
                "/Account/ConfirmEmail",
                pageHandler: null,
                values: new { area = "Identity", userId = user.Id, code =
code },
                protocol: Request.Scheme);

            await _emailSender.SendEmailAsync(Input.Email, "Confirm your
email",
                $"Please confirm your account by <a
href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking here</a>.");

            if (_userManager.Options.SignIn.RequireConfirmedAccount)
            {
                return RedirectToPage("RegisterConfirmation", new { email
= Input.Email });
            }
            else
            {
                await _signInManager.SignInAsync(user, isPersistent:
false);
                return LocalRedirect(returnUrl);
            }
        }
        foreach (var error in result.Errors)
        {
            ModelState.AddModelError(string.Empty, error.Description);
        }
    }

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

In the preceding code:

- The `[FromServices]` attribute provides an instance of `AdminRegistrationTokenService` from the IoC container.
- The `UserManager` class's `AddClaimAsync` method is invoked to save an `IsAdmin` claim in the `AspNetUserClaims` table.

c. Add the following code to the top of the file. It resolves the `AdminRegistrationTokenService` and `Claim` class references in the `OnPostAsync` method:

C# Copy

```
using ContosoPets.Ui.Services;
using System.Security.Claims;
```

2. In *Areas/Identity/Pages/Account/Register.cshtml*, add the following markup:

CSHTML Copy

```
<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.AdminEnrollmentKey"></label>
    <input asp-for="Input.AdminEnrollmentKey" class="form-control" />
    <span asp-validation-for="Input.AdminEnrollmentKey" class="text-danger">
</span>
</div>
<button type="submit" class="btn btn-primary">Register</button>
```

Test admin claim

1. Run the following command to build the app:

   .NET Core CLI Copy

   ```
   dotnet build --no-restore
   ```

   The `--no-restore` option is included because no NuGet packages were added since the last build. The build process bypasses restoration of NuGet packages and succeeds with no warnings. If the build fails, check the output for troubleshooting information.

2. Deploy the app to Azure App Service by running the following command:

   Azure CLI Copy

   ```
   az webapp up
   ```

3. Navigate to your app and log in with an existing user, if not already logged in. Select **Products** from the header. Notice the user isn't presented links to edit, delete, or create products.

4. In the browser's address bar, navigate directly to the **Create Product** page. That page's URL can be obtained by running the following command:

   Bash Copy

   ```
   echo "$webAppUrl/Products/Create"
   ```

The user is forbidden from navigating to the page. An **Access denied** message is displayed. Similarly, the user will be forbidden from navigating to a route such as */Products/Edit/1* route.

5. Select **Logout**.

6. Obtain an administrator self-enrollment token using the following command:

```
Bash
echo $(wget -q -O - $webAppUrl/admintoken)
```

> ⚠ **Warning**
>
> The administrator self-enrollment mechanism is for illustrative purposes only. The */api/Admin* endpoint for obtaining a token should be secured before using in a production environment.

7. In the web app, register a new user. The token from the previous step should be provided in the **Admin enrollment key** text box.

8. Once logged in with the new administrative user, click the **Products** link in the header.

   The administrative user can view, edit, and create products.

Examine the AspNetUserClaims table

Run the following command:

```
Bash
db -Q "SELECT u.Email, c.ClaimType, c.ClaimValue FROM dbo.AspNetUserClaims AS c
INNER JOIN dbo.AspNetUsers AS u ON c.UserId = u.Id" -Y25 -y10
```

A variation of the following output appears:

```
Console
Email                     ClaimType  ClaimValue
------------------------- ---------- ----------
scott@contoso.com         IsAdmin    True
```

The `IsAdmin` claim is stored as a key-value pair in the `AspNetUserClaims` table. The `AspNetUserClaims` record is associated with the user record in the `AspNetUsers` table.

Next unit: Knowledge check

Continue >


```
Azure Cloud Shell
This module requires a sandbox to
complete. A sandbox gives you access to
Azure resources. Your Azure subscription
will not be charged. The sandbox may only
be used to complete training on Microsoft
Learn. Use for any other reason is
prohibited, and may result in permanent
loss of access to the sandbox.
```