Role-based authorization in ASP.NET Core

10/14/2016 • 2 minutes to read •      +7

In this article

When an identity is created it may belong to one or more roles. For example, Tracy may belong to the Administrator and User roles whilst Scott may only belong to the User role. How these roles are created and managed depends on the backing store of the authorization process. Roles are exposed to the developer through the IsInRole method on the ClaimsPrincipal class.

Adding role checks

Role-based authorization checks are declarative—the developer embeds them within their code, against a controller or an action within a controller, specifying roles which the current user must be a member of to access the requested resource.

For example, the following code limits access to any actions on the `AdministrationController` to users who are a member of the `Administrator` role:

C# Copy

```
[Authorize(Roles = "Administrator")]
public class AdministrationController : Controller
{
}
```

You can specify multiple roles as a comma separated list:

C# Copy

```
[Authorize(Roles = "HRManager,Finance")]
public class SalaryController : Controller
{
}
```

This controller would be only accessible by users who are members of the `HRManager` role or the `Finance` role.

If you apply multiple attributes then an accessing user must be a member of all the roles specified; the following sample requires that a user must be a member of both the `PowerUser` and `ControlPanelUser` role.

C# Copy

```
[Authorize(Roles = "PowerUser")]
[Authorize(Roles = "ControlPanelUser")]
public class ControlPanelController : Controller
{
}
```

You can further limit access by applying additional role authorization attributes at the action level:

C# Copy

```
[Authorize(Roles = "Administrator, PowerUser")]
public class ControlPanelController : Controller
{
    public ActionResult SetTime()
    {
    }

    [Authorize(Roles = "Administrator")]
    public ActionResult ShutDown()
    {
    }
}
```

In the previous code snippet members of the `Administrator` role or the `PowerUser` role can access the controller and the `SetTime` action, but only members of the `Administrator` role can access the `ShutDown` action.

You can also lock down a controller but allow anonymous, unauthenticated access to individual actions.

C# Copy

```
[Authorize]
public class ControlPanelController : Controller
{
```

```
    public ActionResult SetTime()
    {
    }

    [AllowAnonymous]
    public ActionResult Login()
    {
    }
}
```

For Razor Pages, the `AuthorizeAttribute` can be applied by either:

- Using a convention, or
- Applying the `AuthorizeAttribute` to the `PageModel` instance:

C# Copy

```
[Authorize(Policy = "RequireAdministratorRole")]
public class UpdateModel : PageModel
{
    public ActionResult OnPost()
    {
    }
}
```

Important

Filter attributes, including `AuthorizeAttribute`, can only be applied to PageModel and cannot be applied to specific page handler methods.

Policy based role checks

Role requirements can also be expressed using the new Policy syntax, where a developer registers a policy at startup as part of the Authorization service configuration. This normally occurs in `ConfigureServices()` in your *Startup.cs* file.

C# Copy

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
    services.AddRazorPages();
```

```
    services.AddAuthorization(options =>
    {
        options.AddPolicy("RequireAdministratorRole",
             policy => policy.RequireRole("Administrator"));
    });
}
```

Policies are applied using the `Policy` property on the `AuthorizeAttribute` attribute:

C# Copy

```
[Authorize(Policy = "RequireAdministratorRole")]
public IActionResult Shutdown()
{
    return View();
}
```

If you want to specify multiple allowed roles in a requirement then you can specify them as parameters to the `RequireRole` method:

C# Copy

```
options.AddPolicy("ElevatedRights", policy =>
                  policy.RequireRole("Administrator", "PowerUser",
"BackupAdministrator"));
```

This example authorizes users who belong to the `Administrator`, `PowerUser` or `BackupAdministrator` roles.

Add Role services to Identity

Append AddRoles to add Role services:

C# Copy

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));
    services.AddDefaultIdentity<IdentityUser>()
        .AddRoles<IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>();
```

```
    services.AddControllersWithViews();
    services.AddRazorPages();
}
```