Claims-based authorization in ASP.NET Core

10/14/2016 • 3 minutes to read • +7

In this article

When an identity is created it may be assigned one or more claims issued by a trusted party. A claim is a name value pair that represents what the subject is, not what the subject can do. For example, you may have a driver's license, issued by a local driving license authority. Your driver's license has your date of birth on it. In this case the claim name would be `DateOfBirth`, the claim value would be your date of birth, for example `8th June 1970` and the issuer would be the driving license authority. Claims based authorization, at its simplest, checks the value of a claim and allows access to a resource based upon that value. For example if you want access to a night club the authorization process might be:

The door security officer would evaluate the value of your date of birth claim and whether they trust the issuer (the driving license authority) before granting you access.

An identity can contain multiple claims with multiple values and can contain multiple claims of the same type.

Adding claims checks

Claim based authorization checks are declarative - the developer embeds them within their code, against a controller or an action within a controller, specifying claims which the current user must possess, and optionally the value the claim must hold to access the requested resource. Claims requirements are policy based, the developer must build and register a policy expressing the claims requirements.

The simplest type of claim policy looks for the presence of a claim and doesn't check the value.

First you need to build and register the policy. This takes place as part of the Authorization service configuration, which normally takes part in `ConfigureServices()` in your *Startup.cs* file.

C# Copy

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

    services.AddAuthorization(options =>
    {
        options.AddPolicy("EmployeeOnly", policy =>
policy.RequireClaim("EmployeeNumber"));
    });
}
```

In this case the `EmployeeOnly` policy checks for the presence of an `EmployeeNumber` claim on the current identity.

You then apply the policy using the `Policy` property on the `AuthorizeAttribute` attribute to specify the policy name;

C# Copy

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

The `AuthorizeAttribute` attribute can be applied to an entire controller, in this instance only identities matching the policy will be allowed access to any Action on the controller.

C# Copy

```
[Authorize(Policy = "EmployeeOnly")]
public class VacationController : Controller
{
    public ActionResult VacationBalance()
    {
    }
}
```

If you have a controller that's protected by the `AuthorizeAttribute` attribute, but want to allow anonymous access to particular actions you apply the `AllowAnonymousAttribute` attribute.

C# Copy

```
[Authorize(Policy = "EmployeeOnly")]
public class VacationController : Controller
{
    public ActionResult VacationBalance()
    {
    }

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

Most claims come with a value. You can specify a list of allowed values when creating the policy. The following example would only succeed for employees whose employee number was 1, 2, 3, 4 or 5.

C# Copy

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

    services.AddAuthorization(options =>
    {
        options.AddPolicy("Founders", policy =>
                          policy.RequireClaim("EmployeeNumber", "1",
"2", "3", "4", "5"));
    });
}
```

Add a generic claim check

If the claim value isn't a single value or a transformation is required, use RequireAssertion. For more information, see Use a func to fulfill a policy.

Multiple Policy Evaluation

If you apply multiple policies to a controller or action, then all policies must pass before access is granted. For example:

C# Copy

```
[Authorize(Policy = "EmployeeOnly")]
public class SalaryController : Controller
{
    public ActionResult Payslip()
    {
    }

    [Authorize(Policy = "HumanResources")]
    public ActionResult UpdateSalary()
    {
    }
}
```

In the above example any identity which fulfills the `EmployeeOnly` policy can access the `Payslip` action as that policy is enforced on the controller. However in order to call the `UpdateSalary` action the identity must fulfill *both* the `EmployeeOnly` policy and the `HumanResources` policy.

If you want more complicated policies, such as taking a date of birth claim, calculating an age from it then checking the age is 21 or older then you need to write custom policy handlers.