csharp-video-tutorials.blogspot.com

Role based authorization vs claims based authorization in asp.net core

5–6 minutes

Suggested Videos
Part 92 - Model binding not working on submitting razor view with foreach loop | Text | Slides
Part 93 - Manage user claims in asp.net core | Text | Slides
Part 94 - Claims based authorization in asp.net core | Text | Slides

In this video we will discuss the **difference between Role based authorization and claims based authorization.**

User Authorization

Authorization is the process of determining what the logged-in user can and cannot do. Based on our application authorization requirements we could use Roles, Claims or a combination of both.

Role Based Access Control (RBAC)

In your organisation you may have the following roles

- Employee
- Manager
- HR

Depending on the role or roles the logged-in user belong to, you may or may not authorize access to certain resources in the application. Since we are using Roles to make authorization checks, this is commonly called **Role Based Access Control** (RBAC) or **Role Based Authorization**.

In ASP.NET Core to implement role based authorization we use Authorize attribute with Roles parameter.

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

We discussed role based authorization in detail in Part 82 of ASP.NET core tutorial

Claims Based Access Control (CBAC)

Claims Based Authorization is also called Claims Based Access Control. Before we understand Claims Based Authorization, let's understand **what is a Claim**.

A claim is a name-value pair. It's really a piece of information about the user, not what the user can and cannot do. For example username, email, age, gender etc are all claims. How you use these claims for authorization checks in your application is up to your application business and authorization requirements.

For example, if you are building an employee portal you may allow the logged-in user to apply for a maternity leave if the gender claim value is female. Similarly, if you are building an ecommerce application, you may allow the logged-in user to submit an order if the age claim value is greater than or equal to 18.

Claims are policy based. We create a policy and include one or more claims in that policy. The policy is then used along with the policy parameter of the Authorize attribute to implement claims based authorization.

```
[Authorize(Policy = "DeleteRolePolicy")]
public async Task<IActionResult> DeleteRole(string id)
{
}
```

We discussed claims based authorization in detail in Part 94 of ASP.NET core tutorial

In ASP.NET Core, a role is just a claim with type Role.

To confirm this, execute the following expression in the Immediate window in Visual Studio. All the logged-in user roles will be listed. You will also see these roles are just claims of type role.

```
User.Claims.Where(c=> c.Type == ClaimTypes.Role).ToList()
```

Roles Policy

We know claims are policy based. Since, a role is also a claim of type role, we can also use a role with the new policy syntax.

With claims, we create a policy and include one or more claims in that policy. We can do the same thing with roles as well. Create a policy and include one or more roles in that policy.

```
public void ConfigureServices(IServiceCollection services)
{
   services.AddAuthorization(options =>
   {
      options.AddPolicy("AdminRolePolicy", policy =>
policy.RequireRole("Admin"));
   });
}
```

If you want to include multiple roles in the policy simply separate them with a comma

```
options.AddPolicy("SuperAdminPolicy", policy =>
              policy.RequireRole("Admin", "User", "Manager"));
```

The policy can then be used on a controller or a controller action.

```
[HttpPost]
[Authorize(Policy = "SuperAdminPolicy")]
public async Task<IActionResult> DeleteRole(string id)
{
   // Delete Role
}
```

Why do we have both in ASP.NET Core

In previous versions of asp.net we did not have claims based authorization, only role based authorization.

Claims based authorization is relatively new and is the recommended approach. With it we can also use claims from external identity providers like Facebook, Google, Twitter etc. We will discuss using external identity providers and the claims they provide in our upcoming videos.

Role based authorization is still supported in asp.net core for backward compatibility. While Claims based authorization is the recommended approach, depending on your application authorization requirements you may use role based authorization, claims based authorization or a combination of both.