UNITED STATES ▾

MICROSOFT ARCHITECT

By Joydip Kanjilal, Columnist, InfoWorld

JUL 9, 2018 3:00 AM PDT

How to use policy-based authorization in ASP.Net Core

Take advantage of policy-based authorization in ASP.Net Core to implement a flexible, extensible, custom security model

If you have experience building ASP.Net applications, you are undoubtedly familiar with role-based authorization. In ASP.Net Core – Microsoft's lean and modular framework that can be used to build modern-day web applications on Windows, Linux, or MacOS – we have an additional option.

Policy-based authorization is a new feature introduced in ASP.Net Core that allows you to implement a loosely coupled security model. In this article I will explain what policy-based authorization is all about and how we can implement it in ASP.Net Core.

[Get started with Visual Studio Code, Microsoft's lightweight editor for Windows, MacOS, and Linux. • Learn what's new in the latest version of Visual Studio Code. | Keep up with hot topics in programming with InfoWorld's App Dev Report newsletter.]

Assuming that you have .Net Core installed in your system, follow the steps below to create a new ASP.Net Core project in Visual Studio 2017.

1. Open Visual Studio
2. Click File -> New -> Project
3. In the New Project Dialog window, select the "ASP.NET Core Web Application" project template
4. Specify the name and location for your project and click OK to save
5. Select "Web API" from the list of templates displayed, make sure Authentication is set to "No Authentication and the Docker support box is unchecked, and click OK

And that's all you need to do to create an ASP.Net Core Web application that leverages Web API. Let's now explore how we can build a custom policy based security model.

There are three key concepts to understand in a policy-based security model: policy, requirement, and handler. Note that a policy is comprised of requirements, and a requirement is comprised of a collection of parameters. These parameters are used to identify the credentials of the user. A handler is used to evaluate the user's authorization, i.e., to determine which resources the user may access.

Register a policy in ASP.Net Core

Let's begin by understanding how we can register a policy. Here is how a policy is registered in the ConfigureServices method of the Startup.cs file.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddAuthorization(options =>
    {
        options.AddPolicy("IDGCustomPolicy", policy =>
            policy.Requirements.Add(new MinimumExperience(10)));
    });
}
```

Implement a policy requirement class in ASP.Net Core

The following code listing provides an example of a custom policy requirement class. Note that the custom requirement class implements the IAuthorizationRequirement interface. The argument constructor of the class accepts an integer as an argument and then assigns the value to an integer property. In this case, the integer indicates the minimum number of years of experience our authorization policy will require.

```
public class MinimumExperienceRequirement : IAuthorizationRequirement
{
    public MinimumExperienceRequirement(int years)
    {
        ExpInYears = years;
    }
    protected int ExpInYears { get; set; }
}
```

Implement a policy handler class in ASP.Net Core

Now we need to implement the handler. The handler is the class that will evaluate the property or properties of the requirement we created above. (In our case we have only one property, the minimum years of experience.) The following code snippet illustrates a custom handler class.

```
public class MinimumExperienceHandler : AuthorizationHandler<MinimumExperience>
{
    protected override Task HandleRequirementAsync(AuthorizationContext context, MinimumExpe
    {
         if (!context.User.HasClaim(c => c.Type ==  ClaimTypes.YearsOfExp))
        {
            return Task.CompletedTask;
        }
        var expInYears = Convert.ToDateTime(context.User.FindFirst(c => c.Type == ClaimTypes.
        if (expInYears >= 3)
        {
            context.Succeed(requirement);
        }
        return Task.CompletedTask;
    }
}
```

Note how our requirement handler class authorizes the user based on whether or not the value of the YearsOfExp claim meets the minimum requirement.

Apply authorization policies in ASP.Net Core

Lastly, to apply the policy we have created, we need to leverage the Authorize attribute. The following code listing illustrates how this can be achieved.

```
[Authorize(Policy="IDGCustomPolicy")]
public class SampleController : Controller
{
    //Write your code here...
}
```

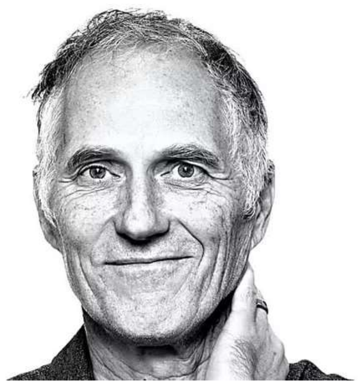
This was just a minimalistic implementation of policy-based authorization in ASP.Net Core. Using this as a template, feel free to implement your own custom authorization policy in ASP.Net Core.

Joydip Kanjilal is a Microsoft MVP in ASP.Net, as well as a speaker and author of several books and articles. He has more than 20 years of experience in IT including more than 16 years in Microsoft .Net and related technologies.

Follow

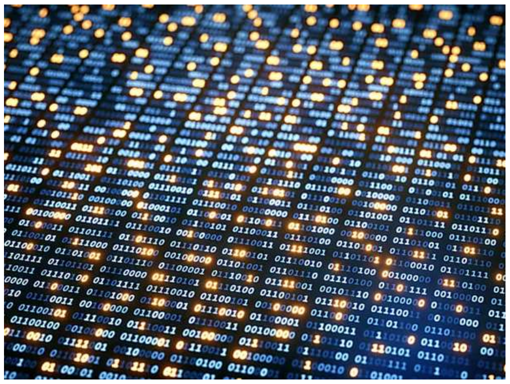
YOU MAY ALSO LIKE

Recommended by


Tim O'Reilly: the golden age of the programmer is over

JDK 17: The new features in Java 17

Edge computing archetypes are emerging, and they are not pretty

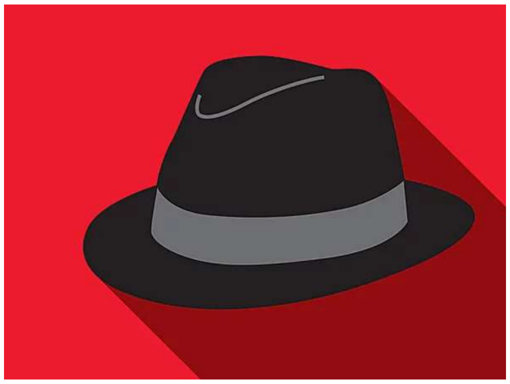

What is a computational storage drive? Much-needed help for CPUs

Microsoft previews Azure service for building WebSocket applications

8 databases supporting in-database machine learning

3 enterprise AI success stories

The decline of Heroku


Red Hat OpenShift ramps up security and manageability

How IT priorities are shifting during the COVID-19 crisis

The shifting market for PostgreSQL