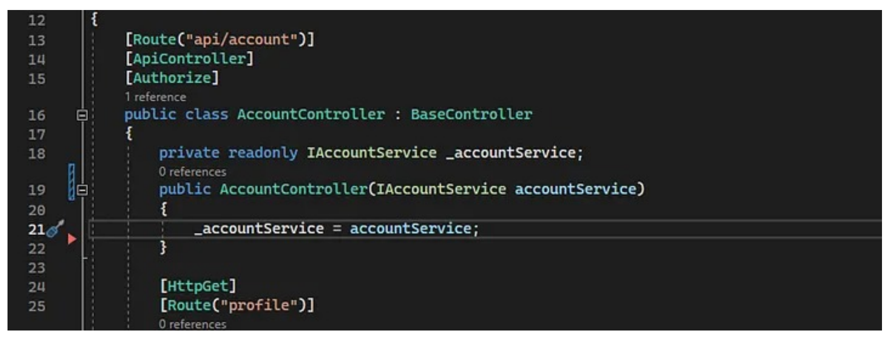
Custom attributes and custom Action filters in .Net Core controllers

Julie Gvozdikova · Follow
3 min read · Jun 9, 2023

I think all of us are familiar with attributes we usually use in controllers in order to meet some predefined requirements like “user should be authorized”.



```
{
    [Route("api/account")]
    [ApiController]
    [Authorize]
    1 reference
    public class AccountController : BaseController
    {
        private readonly IAccountService _accountService;
        0 references
        public AccountController(IAccountService accountService)
        {
            _accountService = accountService;
        }

        [HttpGet]
        [Route("profile")]
        0 references
```

More specific variation of **[Authorize]** is something like this **[Authorize(Roles = “Admin”)]** when we are able to set the Role claim we want user to have in order to use current endpoint.

When it comes to Roles model you may want to restrict the access to certain objects for users who do not really have access to these objects.
And sometimes user’s access is based not on Role.

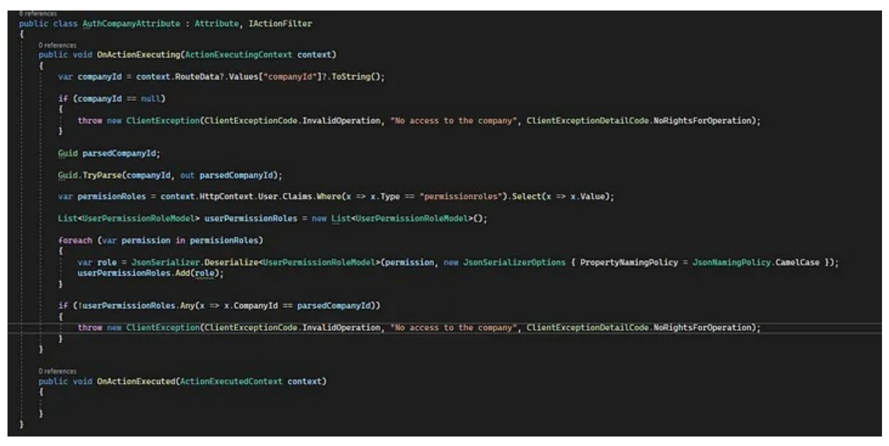
The same situation has been on my way recently.

A little background — you have Company entity. Each Company has 0-endless Branch entities. You have Employee entity which has access to specific Branch in specific Company. All this access information is added to user’s JWT-token.

So, your task is to create a nice way of checking whether current Employee has access to the Company they ask for or not.

In my opinion, custom attributes and custom action filters are the most sophisticated way of doing that.

I’ll show you the whole peace of code and then I’ll explain it part by part.

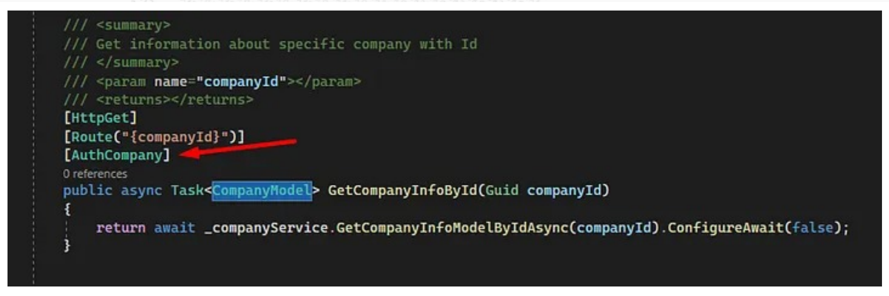


```
0 references
public class AuthCompanyAttribute : Attribute, IActionFilter
{
    0 references
    public void OnActionExecuting(ActionExecutingContext context)
    {
        var companyId = context.RouteData?.Values["companyId"]?.ToString();

        if (companyId == null)
        {
            throw new ClientException(ClientExceptionCode.InvalidOperation, "No access to the company", ClientExceptionDetailCode.NoRightsForOperation);
        }

        Guid parsedCompanyId;

        Guid.TryParse(companyId, out parsedCompanyId);

        var permisionRoles = context.HttpContext.User.Claims.Where(x => x.Type == "permissionroles").Select(x => x.Value);

        List<UserPermissionRoleModel> userPermissionRoles = new List<UserPermissionRoleModel>();

        foreach (var permission in permisionRoles)
        {
            var role = JsonSerializer.Deserialize<UserPermissionRoleModel>(permission, new JsonSerializerOptions { PropertyNamingPolicy = JsonNamingPolicy.CamelCase });
            userPermissionRoles.Add(role);
        }

        if (!userPermissionRoles.Any(x => x.CompanyId == parsedCompanyId))
        {
            throw new ClientException(ClientExceptionCode.InvalidOperation, "No access to the company", ClientExceptionDetailCode.NoRightsForOperation);
        }
    }

    0 references
    public void OnActionExecuted(ActionExecutedContext context)
    {

    }
}
```

First of all, we want our class to inherit functionality from **Attribute** class(in order to use it as an attribute) and **IActionFilter** interface(in order to intercept method execution and check what we need).

The example of token section with permissions is below.

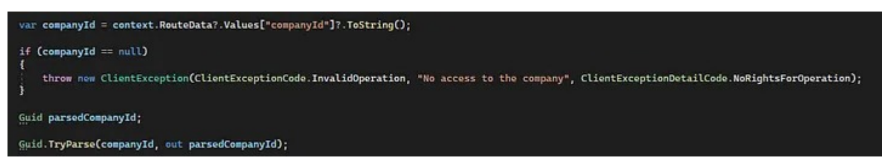
```
"permissionroles": [
```



```
/// <summary>
/// Get information about specific company with Id
/// </summary>
/// <param name="companyId"></param>
/// <returns></returns>
[HttpGet]
[Route("{companyId}")]
[AuthCompany]
0 references
public async Task<CompanyModel> GetCompanyInfoById(Guid companyId)
{
    return await _companyService.GetCompanyInfoModelByIdAsync(companyId).ConfigureAwait(false);
}
```

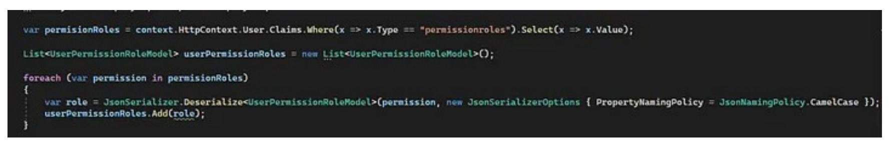
We are focusing on **OnActionExecuting**() method from **IActionFilter** because it fires before method execution.
Our plan is:

1. Get “**companyId**” from route and parse it to have Guid value.



```
var companyId = context.RouteData?.Values["companyId"]?.ToString();

if (companyId == null)
{
    throw new ClientException(ClientExceptionCode.InvalidOperation, "No access to the company", ClientExceptionDetailCode.NoRightsForOperation);
}

Guid parsedCompanyId;

Guid.TryParse(companyId, out parsedCompanyId);
```

2. Get “**permissionroles**” from JWT-token and deserialize it.


```
var permisionRoles = context.HttpContext.User.Claims.Where(x => x.Type == "permissionroles").Select(x => x.Value);

List<UserPermissionRoleModel> userPermissionRoles = new List<UserPermissionRoleModel>();

foreach (var permission in permisionRoles)
{
    var role = JsonSerializer.Deserialize<UserPermissionRoleModel>(permission, new JsonSerializerOptions { PropertyNamingPolicy = JsonNamingPolicy.CamelCase });
    userPermissionRoles.Add(role);
}
```

3. Check if requested **companyId** is equal to **companyId** from user’s “**permissionroles**” token section. If it is not — throw an exception.

So this is how you can decrease the amount of code for checking user’s permissions.

The more you check from the start, the more clean code you are going to have. Separation of concerns is a great pattern. Use it!

Net Core Attributes Action Filter

56

Written by Julie Gvozdikova

10 Followers · 4 Following

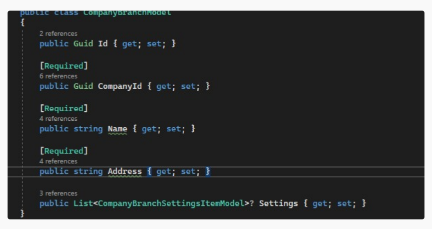
Follow

Full stack developer(Angular/.Net)

No responses yet

What are your thoughts? Respond

More from Julie Gvozdikova


```
public class CompanyBranchModel
{
    2 references
    public Guid Id { get; set; }

    [Required]
    6 references
    public Guid CompanyId { get; set; }

    [Required]
    4 references
    public string Name { get; set; }

    [Required]
    4 references
    public string Address { get; set; }

    3 references
    public List<CompanyBranchSettingsItemModel>? Settings { get; set; }
}
```

Julie Gvozdikova

Custom JsonConverters in .NET. How and Why?

Sometimes an input data for a backend endpoint might be not suitable for your…

Jun 23, 2023 · 12

```
    logging:
      driver: loki
      options:
        loki-url: "http://localhost:3100/loki/api/v1/push"
  loki:
    image: grafana/loki:2.7.0
    restart: unless-stopped
    ports:
      - "3100:3100"
    volumes:
      - ./:/etc/loki
    command: -config.file=/etc/loki/loki-config.yaml

  promtail:
    image: grafana/promtail:2.7.0
    restart: unless-stopped
    volumes:
      - ./:/etc/promtail
```

Julie Gvozdikova

How to handle Docker logging using Grafana’s Loki

Working with logs inside Docker containers can be tiring. Firstly, you can’t see them…

Mar 14, 2023 · 2 · 2

```
k UpdateDictionaryByAdministrator(List<DictionaryItem> dictionaryItems)

 item in dictionaryItems)

em = await _dbContext.DictionaryMKH.AsQueryable().FirstOrDefaultAsync(x => x.Code == item.Code).ConfigureAwai

em != null)

em.Name = item.Name;


newDbItem = new DbDictionaryMKHItem

Code = item.Code,
Name = item.Name


ontext.DictionaryMKH.Add(newDbItem);


text.SaveChangesAsync().ConfigureAwait(false);
```

Julie Gvozdikova

Dictionary is your best friend for updating big amounts of records i…

Project performance and large data sets are not friends at all. We all know that. And for…

May 17, 2023 · 1

See all from Julie Gvozdikova

Recommended from Medium

yusuf sarıkaya

Mastering Entity Framework Core Self-Referencing One-To-Many…

In this article, we’ll explore how to design a self-referencing database table using Entity…

Aug 11, 2024 · 11

Engr. Md. Hasan Monsur

Building a Full-Featured API Monitoring Service with C#,…

Discover how to build a robust API monitoring service using C#, ASP.NET MVC,…

Oct 4, 2024 · 65

Lists

Staff picks
806 stories · 1603 saves

Stories to Help You Level-Up at Work
19 stories · 930 saves

Self-Improvement 101
20 stories · 3259 saves

Productivity 101
20 stories · 2755 saves

Justin Muench

Hangfire: Background Jobs Made Easy for .NET 🛠️

Every developer has faced those tasks that need to run in the background — think of...

Nov 5, 2024 19

Randika Hasheen

Implementing MongoDB in a .NET Core Web API Using Clean...

Introduction

Nov 12, 2024 22

Kamlesh Singh

Enhancing File Upload Security in .NET Core with File Signature...

File upload security is crucial in web applications to prevent malicious files from...

2d ago 23

Ravi Patel

Dependency Injection and Services in ASP.NET Core: A Comprehensi...

Dependency Injection (DI) is a cornerstone of modern software development, enabling...

Sep 30, 2024 44

See more recommendations