Note :

1.To maintain log information in a separate file , you need to configure in using Serilog.

2.Maintain All log in Console & also in log.txt

Assignment: Implementing Filters in an ASP.NET Core MVC Application

Problem Statement:

You are developing an **Employee Management System** in **ASP.NET Core MVC**. The system requires different **filters** to handle authentication, authorization, exceptions, actions, results, and resource management efficiently. Additionally, you need to implement a **custom logger** that logs the execution order of these filters.

Tasks:

1. Implement Different Types of Filters in MVC

Implement the following filters in the application:

Filter Type	Purpose
Authorization Filter	Restrict access to authorized users only.
Resource Filter	Execute logic before/after request execution (e.g., caching).
Action Filter	Execute logic before/after an action method executes.
Exception Filter	Handle exceptions at the global level.
Result Filter	Execute logic before/after returning a result to the client.

---

2. Create a Controller (`EmployeeController.cs`)

Implement an `EmployeeController` with various actions that trigger different filters.

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;
using Microsoft.Extensions.Logging;
using System;

namespace EmployeeManagement.Controllers
{
    [Authorize] // Authorization Filter (Applied at the Controller
Level)
    public class EmployeeController : Controller
    {
        private readonly ILogger<EmployeeController> _logger;

        public EmployeeController(ILogger<EmployeeController> logger)
        {
            _logger = logger;
        }

        [ServiceFilter(typeof(CustomResourceFilter))]
        [ServiceFilter(typeof(CustomActionFilter))]
        [ServiceFilter(typeof(CustomResultFilter))]
        [ServiceFilter(typeof(CustomExceptionFilter))]
        public IActionResult Index()
        {
            _logger.LogInformation("Executing EmployeeController ->
Index Action.");
            return View();
        }

        public IActionResult Error()
        {
            throw new Exception("This is a test exception.");
        }
    }
}
```

---

3. Implement Custom Filters

Each filter should log when it is executed.

a) Custom Authorization Filter (`CustomAuthorizationFilter.cs`)

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;
using Microsoft.Extensions.Logging;

public class CustomAuthorizationFilter : IAuthorizationFilter
{
    private readonly ILogger<CustomAuthorizationFilter> _logger;

    public
CustomAuthorizationFilter(ILogger<CustomAuthorizationFilter> logger)
    {
        _logger = logger;
    }

    public void OnAuthorization(AuthorizationFilterContext context)
    {
        _logger.LogInformation("Executing Authorization Filter.");
        // Simulate an unauthorized request
        if (!context.HttpContext.User.Identity.IsAuthenticated)
        {
            context.Result = new UnauthorizedResult();
        }
    }
}
```

---

b) Custom Resource Filter (`CustomResourceFilter.cs`)

```
using Microsoft.AspNetCore.Mvc.Filters;
using Microsoft.Extensions.Logging;

public class CustomResourceFilter : IResourceFilter
{
    private readonly ILogger<CustomResourceFilter> _logger;

    public CustomResourceFilter(ILogger<CustomResourceFilter> logger)
    {
        _logger = logger;
    }

    public void OnResourceExecuting(ResourceExecutingContext context)
    {
        _logger.LogInformation("Executing Resource Filter - Before
Execution.");
    }

    public void OnResourceExecuted(ResourceExecutedContext context)
    {
        _logger.LogInformation("Executing Resource Filter - After
Execution.");
    }
}
```

---

c) Custom Action Filter (`CustomActionFilter.cs`)

```
using Microsoft.AspNetCore.Mvc.Filters;
using Microsoft.Extensions.Logging;

public class CustomActionFilter : IActionFilter
{
    private readonly ILogger<CustomActionFilter> _logger;

    public CustomActionFilter(ILogger<CustomActionFilter> logger)
    {
        _logger = logger;
    }

    public void OnActionExecuting(ActionExecutingContext context)
    {
```

```
        _logger.LogInformation("Executing Action Filter - Before
Action Method.");
    }

    public void OnActionExecuted(ActionExecutedContext context)
    {
        _logger.LogInformation("Executing Action Filter - After Action
Method.");
    }
}
```

---

d) Custom Exception Filter (`CustomExceptionFilter.cs`)

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Filters;
using Microsoft.Extensions.Logging;

public class CustomExceptionFilter : IExceptionFilter
{
    private readonly ILogger<CustomExceptionFilter> _logger;

    public CustomExceptionFilter(ILogger<CustomExceptionFilter>
logger)
    {
        _logger = logger;
    }

    public void OnException(ExceptionContext context)
    {
        _logger.LogError($"Exception occurred:
{context.Exception.Message}");
        context.Result = new ViewResult { ViewName = "Error" };
    }
}
```

---

e) Custom Result Filter (`CustomResultFilter.cs`)

```
using Microsoft.AspNetCore.Mvc.Filters;
using Microsoft.Extensions.Logging;

public class CustomResultFilter : IResultFilter
{
    private readonly ILogger<CustomResultFilter> _logger;

    public CustomResultFilter(ILogger<CustomResultFilter> logger)
    {
        _logger = logger;
    }

    public void OnResultExecuting(ResultExecutingContext context)
    {
        _logger.LogInformation("Executing Result Filter - Before
Returning Result.");
    }

    public void OnResultExecuted(ResultExecutedContext context)
    {
        _logger.LogInformation("Executing Result Filter - After
Returning Result.");
    }
}
```

---

4. Register Filters in `Program.cs`

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

var builder = WebApplication.CreateBuilder(args);

// Add services
builder.Services.AddControllersWithViews();

// Register filters
builder.Services.AddScoped<CustomAuthorizationFilter>();
builder.Services.AddScoped<CustomResourceFilter>();
builder.Services.AddScoped<CustomActionFilter>();
builder.Services.AddScoped<CustomExceptionFilter>();
builder.Services.AddScoped<CustomResultFilter>();

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Employee}/{action=Index}/{id?}");

app.Run();
```

---

5. Output Logging in Console

After running the application, navigate to **EmployeeController > Index Action**, and the **logger** should output messages in the following order:

```
Executing Resource Filter - Before Execution.
Executing Authorization Filter.
Executing Action Filter - Before Action Method.
Executing EmployeeController -> Index Action.
Executing Action Filter - After Action Method.
Executing Result Filter - Before Returning Result.
Executing Result Filter - After Returning Result.
Executing Resource Filter - After Execution.
```

If you access the `/Employee/Error` action, the **exception filter** should also log:

```
Exception occurred: This is a test exception.
```

Note :

1.To maintain log information in a separate file , you need to configure in using Serilog.

2.Maintain All log in Console & also in log.txt