**WEBAPI 03**

**Aim:**

Create a Custom class ‘Employee’ of the below defined structure

public class Employee

{

public int Id { get; set; }

public string Name { get; set; }

public int Salary { get; set; }

public bool Permanent { get; set; }

public Department Department { get; set; }

public List<Skill> Skills { get; set; }

public DateTime DateOfBirth { get; set; }

}

Create a new controller - EmployeeController with Read Write actions

Constructor: Create few records, HTTPGet, HTTPPost/HTTPPut

Create a Private method GetStandardEmployeeList that returns a List of Employee class. Invoke this method in the Get action method of the EmployeeController that was created in the previous step.

Public ActionResult<Employee> GetStandrad()

Modify the return type of the Get action method(without parameter) to return List of Employee class object

Add ProducesResponseType to the GET action method for Status code 200

Check the Swagger description for the GET method for success status code

1. **Create a Custom action filter for Authorization.**

The requirement is to intercept incoming requests and check if there is a key ‘Authorization’ in the request header or not. If it is there, then to check if it contains a value ‘Bearer’ or not.

Create a folder ‘Filters’ in the application solution. Create a class ‘**CustomAuthFilter**’ to filter requests. Inherit ActionFilterAttribute. Override OnActionExecuting method to check if the request object has Header ‘Authorization’ or not. If not, throw BadRequestResult with the message

Invalid request - No Auth token

If the header is present, then check if the value contains the word ‘Bearer’. If not, throw BadRequestResult with the message

Invalid request - Token present but Bearer unavailable

Add an attribute **CustomAuthFilter** to the Employee controller to filter any request to check for the Authorization token in the request header.

1. **Custom Exception filter**

Create a class ‘CustomExceptionFilter’ to catch the exceptions occuring the application. Implement IExceptionFilter thru the OnException method  
  
Use the exception context to fetch the exception detail. Capture that and write it to a File in the system.  
  
Set the Result property of the exception context to ExceptionResult.  
  
Throw an exception in GET action method.  
Ensure that the GET action method has ProducesResponseType for 500 - Internal server error  
  
Use Swagger to test the exception and message being thrown.  
  
Note: This needs WebApiCompatShim NuGet package installation

**EmployeeController.cs:**

using Microsoft.AspNetCore.Mvc;

using EmployeeApi.Models;

using EmployeeApi.Filters;

namespace EmployeeApi.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    [ServiceFilter(typeof(CustomAuthFilter))]

    [ServiceFilter(typeof(CustomExceptionFilter))]

    public class EmployeeController : ControllerBase

    {

        private List<Employee> GetStandardEmployeeList()

        {

            return new List<Employee>

            {

                new Employee

                {

                    Id = 1,

                    Name = "John",

                    Salary = 50000,

                    Permanent = true,

                    Department = new Department { Id = 1, Name = "HR" },

                    Skills = new List<Skill>

                    {

                        new Skill { Id = 1, Name = "C#" },

                        new Skill { Id = 2, Name = "SQL" }

                    },

                    DateOfBirth = new DateTime(1990, 1, 1)

                }

            };

        }

        [HttpGet("standard")]

        [ProducesResponseType(StatusCodes.Status200OK)]

        [ProducesResponseType(StatusCodes.Status500InternalServerError)]

        public ActionResult<List<Employee>> GetStandard()

        {

            throw new Exception("Test exception for filter logging");

            // return Ok(GetStandardEmployeeList()); // Use this if not testing exception

        }

        [HttpPost]

        public IActionResult CreateEmployee([FromBody] Employee employee)

        {

            return Ok($"Employee '{employee.Name}' added.");

        }

        [HttpPut("{id}")]

        public IActionResult UpdateEmployee(int id, [FromBody] Employee employee)

        {

            return Ok($"Employee with ID {id} updated.");

        }

    }

}

**Department.cs:**

namespace EmployeeApi.Models

{

    public class Department

    {

        public int Id { get; set; }

        public string Name { get; set; } = "";

    }

}

**Skill.cs:**

namespace EmployeeApi.Models

{

    public class Skill

    {

        public int Id { get; set; }

        public string Name { get; set; } = "";

    }

}

**Employee.cs:**

using System;

using System.Collections.Generic;

namespace EmployeeApi.Models

{

    public class Employee

    {

        public int Id { get; set; }

        public string Name { get; set; } = "";

        public int Salary { get; set; }

        public bool Permanent { get; set; }

        public Department Department { get; set; } = new Department();

        public List<Skill> Skills { get; set; } = new List<Skill>();

        public DateTime DateOfBirth { get; set; }

    }

}

**CustomAuthFilter.cs:**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

namespace EmployeeApi.Filters

{

    public class CustomAuthFilter : ActionFilterAttribute

    {

        public override void OnActionExecuting(ActionExecutingContext context)

        {

            var headers = context.HttpContext.Request.Headers;

            if (!headers.ContainsKey("Authorization"))

            {

                context.Result = new BadRequestObjectResult("Invalid request - No Auth token");

                return;

            }

            var authValue = headers["Authorization"].ToString();

            if (!authValue.Contains("Bearer"))

            {

                context.Result = new BadRequestObjectResult("Invalid request - Token present but Bearer unavailable");

            }

        }

    }

}

**CustomExceptionFilter.cs:**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

namespace EmployeeApi.Filters

{

    public class CustomExceptionFilter : IExceptionFilter

    {

        public void OnException(ExceptionContext context)

        {

            string logPath = "ExceptionLog.txt";

            File.AppendAllText(logPath,

                $"[{DateTime.Now}] {context.Exception.Message}\n{context.Exception.StackTrace}\n\n");

            context.Result = new ObjectResult("An error occurred")

            {

                StatusCode = 500

            };

        }

    }

}

**Program.cs:**

using EmployeeApi.Filters;

using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddScoped<CustomAuthFilter>();

builder.Services.AddScoped<CustomExceptionFilter>();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>

{

    c.SwaggerDoc("v1", new OpenApiInfo

    {

        Title = "Employee API with Filters",

        Version = "v1",

        Description = "API with Custom Authorization and Exception Filters"

    });

});

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

    app.UseSwagger();

    app.UseSwaggerUI();

}

app.UseAuthorization();

app.MapControllers();

app.Run();

**Output :**





