Task 3: Custom Model, Authorization Filter, and Exception Handling in Web API

Objectives

1. Create and return a list of custom class entities using Web API.
2. Use `[FromBody]` to extract JSON model data in POST/PUT operations.
3. Implement a custom authorization filter to validate request headers.
4. Implement a global exception filter to capture and log exceptions.

1. Custom Model and Web API Endpoint

Employee Model 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; }
}

public class Department
{
    public int DepartmentId { get; set; }
    public string DepartmentName { get; set; }
}

public class Skill
{
    public int SkillId { get; set; }
    public string SkillName { get; set; }
}
```

EmployeeController Implementation

```
[AllowAnonymous]
[ApiController]
[Route("api/[controller]")]
public class EmployeeController : ControllerBase
{
    private static List<Employee> employeeList = new();

    public EmployeeController()
    {
```

```
        if (!employeeList.Any())
        {
            employeeList = GetStandardEmployeeList();
        }
    }

    private List<Employee> GetStandardEmployeeList()
    {
        return new List<Employee>
        {
            new Employee
            {
                Id = 1,
                Name = "John",
                Salary = 50000,
                Permanent = true,
                Department = new Department { DepartmentId = 1, DepartmentName = "HR" },
                Skills = new List<Skill>
                {
                    new Skill { SkillId = 1, SkillName = "Communication" },
                    new Skill { SkillId = 2, SkillName = "Leadership" }
                },
                DateOfBirth = new DateTime(1990, 01, 01)
            }
        };
    }

    [HttpGet]
    [ProducesResponseType(typeof(List<Employee>), 200)]
    [ProducesResponseType(500)]
    public ActionResult<List<Employee>> GetStandard()
    {
        throw new Exception("Testing custom exception handling");
        return Ok(employeeList);
    }

    [HttpPost]
    public IActionResult AddEmployee([FromBody] Employee employee)
    {
        employeeList.Add(employee);
        return Ok("Employee added.");
    }
}
```

---

2. Using `[FromBody]` Attribute

The `[FromBody]` attribute is used to bind complex objects from the HTTP request body. This is particularly used for POST and PUT methods where data is sent in JSON format.

Example usage:

```
[HttpPost]
public IActionResult AddEmployee([FromBody] Employee employee)
{
    // Add employee logic
}
```

This ensures that the JSON data is automatically deserialized into the `Employee` model object.

3. Custom Authorization Filter

CustomAuthFilter.cs

```
public class CustomAuthFilter : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext context)
    {
        if (!context.HttpContext.Request.Headers.TryGetValue("Authorization", out var
token))
        {
            context.Result = new BadRequestObjectResult("Invalid request - No Auth
token");
        }
        else if (!token.ToString().Contains("Bearer"))
        {
            context.Result = new BadRequestObjectResult("Invalid request - Token present
but Bearer unavailable");
        }
    }
}
```

Applying the Filter to Controller

```
[ServiceFilter(typeof(CustomAuthFilter))]
[Route("api/[controller]")]
public class EmployeeController : ControllerBase
{
    // controller methods
}
```

---

4. Custom Exception Filter

CustomExceptionFilter.cs

```
public class CustomExceptionFilter : IExceptionFilter
{
    public void OnException(ExceptionContext context)
    {
        string message = $"[{DateTime.Now}]
{context.Exception.Message}{Environment.NewLine}";
        File.AppendAllText("error_log.txt", message);

        context.Result = new ObjectResult("An unexpected error occurred.")
        {
            StatusCode = 500
        };
    }
}
```

Registering the Filter Globally in Program.cs

```
builder.Services.AddControllers(options =>
{
```

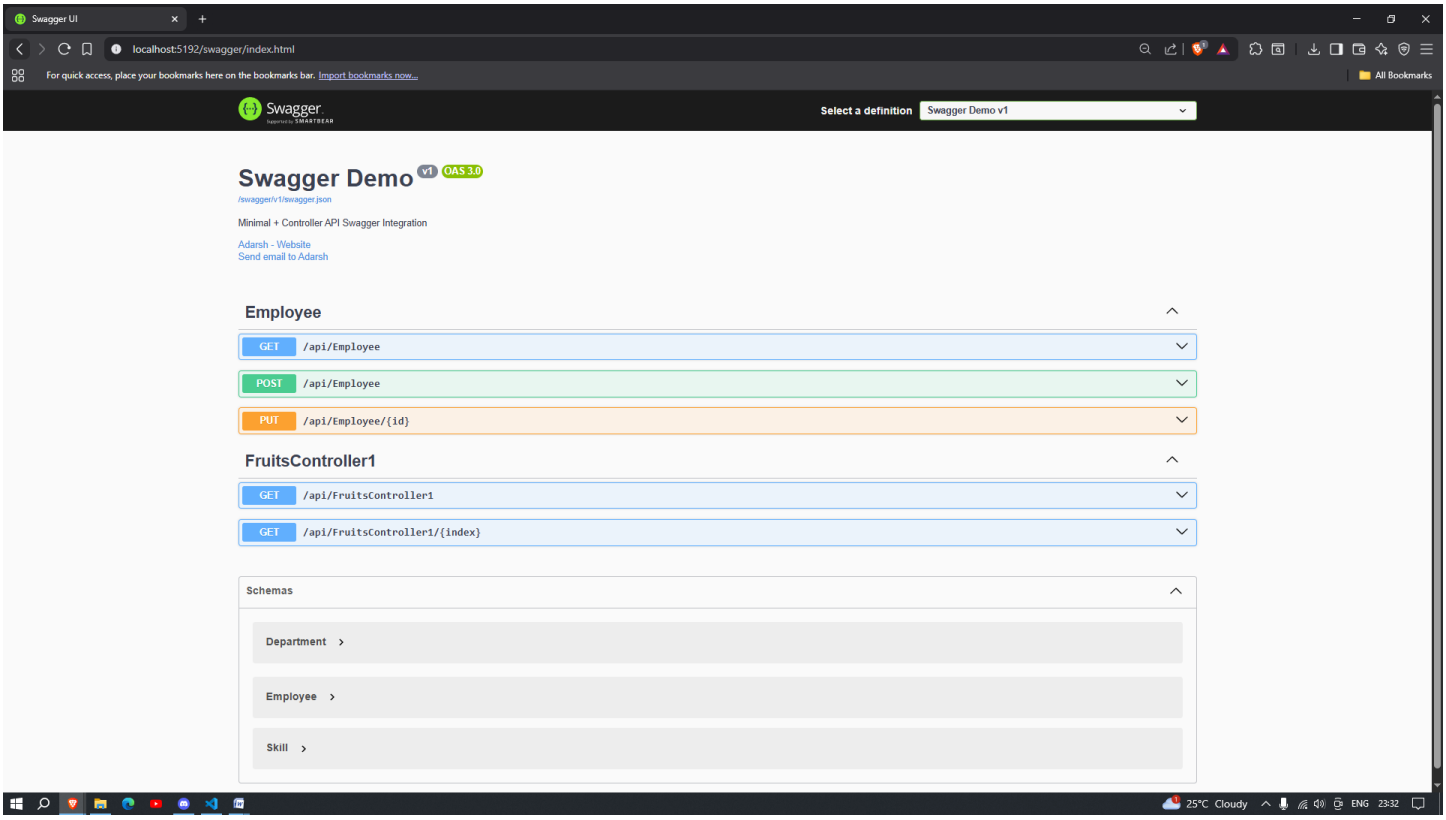
```
    options.Filters.Add<CustomExceptionFilter>();
});
```

Required Package

To use compatibility filters like `WebApiCompatShim`, install the NuGet package:

```
dotnet add package Microsoft.AspNetCore.Mvc.WebApiCompatShim
```

Screenshots



Conclusion

In this task, a complete custom model and controller were created to simulate real-world API data handling. Additionally, custom filters were implemented to enforce authorization policies and to handle unexpected exceptions centrally. These patterns help in building scalable and secure enterprise APIs.