Filters In Asp.Net

إيه هي الـ Filters في ASP.NET؟

الـ Filters في ASP.NET هي أداة قوية بتتيح لك إضافة منطق مخصص (Custom Logic) يتنفذ في مراحل معينة من دورة حياة معالجة الطلب (Request Pipeline) في تطبيقات ASP.NET. بمعنى آخر، الـ Filters بتخليك تتحكم في إزاي التطبيق بيستجيب للطلبات قبل أو بعد تنفيذ أجزاء معينة من الكود، زي الـ Actions في MVC أو الـ Razor Pages في ASP.NET Core.

الفلاتر بتساعدك تضيف سلوكيات زي التحقق من صلاحيات المستخدم (Authorization)، تسجيل الطلبات (Logging)، معالجة الأخطاء (Error Handling)، أو حتى تعديل الردود (Responses) من غير ما تغير الكود الأساسي للـ Action أو الـ Controller.

1. أنواع الفلاتر في ASP.NET

في ASP.NET (سواء MVC أو Core)، فيه خمسة أنواع رئيسية للفلاتر، وكل نوع بيشتغل في مرحلة معينة من الـ Request Pipeline:

- **Authorization Filters**

- بتتنفذ أول حاجة قبل أي حاجة تانية في الـ Pipeline.

- بتتحكم في إذا كان الطلب (Request) ليه الحق يوصل للـ Action أو لأ.

- **مثال:** التحقق من إن المستخدم عنده صلاحية (مثل `[Authorize]` في ASP.NET Core).

- لو التحقق فشل، بيرجع Response زي 401 (Unauthorized) أو 403 (Forbidden).

🔹 Resource Filters

- بتتنفذ بعد الـ Authorization وقبل الـ Model Binding.

- بتُستخدم عشان تدير الموارد (Resources) زي الـ Caching أو تحميل بيانات معينة قبل تنفيذ الـ Action.

- **مثال:** لو عايز تتحقق إذا كانت البيانات موجودة في الـ Cache قبل ما تعمل Query على قاعدة البيانات.

🔹 Action Filters

- بتتنفذ قبل وبعد تنفيذ الـ Action Method في الـ Controller.

- بتُستخدم عشان تضيف منطق زي تعديل الـ Input Parameters أو الـ Action Result.

- **مثال:** إضافة Header معين للـ Response أو تسجيل بيانات الطلب.

🔹 Exception Filters

- بتتنفذ لو حصل خطأ (Exception) أثناء معالجة الطلب.

- بتُستخدم عشان تسيطر على طريقة معالجة الأخطاء، زي إرجاع صفحة خطأ مخصصة أو تسجيل الخطأ في Log.

- **مثال:** إرجاع JSON Response بتفاصيل الخطأ بدل صفحة خطأ افتراضية.

🔹 Result Filters

- بتتنفذ قبل وبعد تنفيذ الـ Action Result (زي إرجاع View أو JSON).

- بتُستخدم عشان تعديل النتيجة، زي ضغط الـ Response أو إضافة Headers.

- **مثال:** تحويل كل الـ Responses لـ JSON إلى صيغة معينة.

2. إزاي الـ Filters بتشتغل؟

الـ Filters بتشتغل داخل الـ ASP.NET Pipeline، وهي عبارة عن سلسلة من الخطوات اللي بيمر بيها الطلب (Request) من أول ما يوصل للسيرفر لحد ما يترجع الرد (Response). ترتيب تنفيذ الـ Filters بيكون كالتالي:

1. Authorization Filters (بتتحقق من الصلاحيات).

2. Resource Filters (بتدير الموارد).

3. Model Binding (تحويل الـ Request Data إلى Parameters).

4. Action Filters (قبل و/أو بعد الـ Action).

5. Action Execution (تنفيذ الـ Action Method).

6. Result Filters (قبل و/أو بعد الـ Result).

7. Exception Filters (لو حصل خطأ في أي مرحلة).

لو أي فلتر (زي Authorization) رفض الطلب، الـ Pipeline بتتوقف ومابتكملش.

3. إزاي نكتب فلتر مخصص (Custom Filter)؟

في ASP.NET Core، تقدر تكتب فلتر مخصص عن طريق تنفيذ واحد من الـ Interfaces الخاصة بالفلاتر (زي IActionFilter أو IExceptionFilter). خلينا نشوف مثال لـ Action Filter مخصص:

```
using Microsoft.AspNetCore.Mvc.Filters;
using System;

public class LogActionFilter : IActionFilter
{
    public void OnActionExecuting(ActionExecutingContext context)
    {
        // الكود اللي هيتنفذ قبل الـ Action
        Console.WriteLine($"Action
{context.ActionDescriptor.DisplayName} is executing at
{DateTime.Now}");
    }

    public void OnActionExecuted(ActionExecutedContext context)
    {
        // الكود اللي هيتنفذ بعد الـ Action
        Console.WriteLine($"Action
{context.ActionDescriptor.DisplayName} executed at
{DateTime.Now}");
    }
}
```

إزاي نستخدم الفلتر ده؟

تسجيل الفلتر عالميًا (Global Filter):

في Startup.cs أو Program.cs (في ASP.NET Core):

```
csharpbuilder.Services.AddControllers(options =>
```

```
{
    options.Filters.Add<LogActionFilter>();
});
```

تطبيق الفلتر على Controller أو Action معين:

```
[LogActionFilter]
public class HomeController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```

4. أمثلة عملية لاستخدام الفلاتر

📌 **مثال 1:** تسجيل الطلبات (Logging):

استخدام Action Filter عشان تسجل كل طلب يدخل لل Action:

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

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

    public void OnActionExecuting(ActionExecutingContext
```

```
context)
    {
        _logger.LogInformation($"Action
{context.ActionDescriptor.DisplayName} started.");
    }

    public void OnActionExecuted(ActionExecutedContext context)
    {
        _logger.LogInformation($"Action
{context.ActionDescriptor.DisplayName} completed.");
    }
}
```

📌 **مثال 2:** معالجة الأخطاء:

استخدام Exception Filter عشان ترجع رد موحد للأخطاء:

```
public class CustomExceptionFilter : IExceptionFilter
{
    public void OnException(ExceptionContext context)
    {
        context.Result = new JsonResult(new
        {
            Error = "An error occurred",
            Message = context.Exception.Message
        })
        {
            StatusCode = 500
        };
        context.ExceptionHandled = true;
    }
}
```

📌 **مثال 3:** التحقق من الصلاحيات:

استخدام Authorization Filter عشان تمنع الوصول لبعض الـ Actions بناءً على شروط مخصصة:

```
public class CustomAuthorizeFilter : IAuthorizationFilter
{
    public void OnAuthorization(AuthorizationFilterContext
context)
    {
        if (!context.HttpContext.User.Identity.IsAuthenticated)
        {
            context.Result = new UnauthorizedResult();
        }
    }
}
```

5. نصايح للتعامل مع الـ Filters

✔️ **استخدم الـ Filters بحكمة:** الفلاتر قوية، لكن الإفراط في استخدامها ممكن يأثر على الأداء.

✔️ **استخدم Dependency Injection:** لو الفلتر بيحتاج خدمات (زي Logger أو Database Context)، استخدم DI عشان تحافظ على الكود نظيف.

✔️ **اختبر الـ Filters:** تأكد إن الـ Filters بتشتغل زي ما أنت عايز عن طريق Unit Testing.

✔️ **افصل المنطق:** حاول تفصل المنطق بتاع الفلتر عن الـ Controller عشان الكود يفضل قابل للصيانة.

✔️ **استخدم Middleware للمنطق العام:** لو المنطق بتاعك بيطبق على كل الطلبات (مش بس الـ Actions)، يبقى Middleware أفضل من الـ Filters.