FILTER

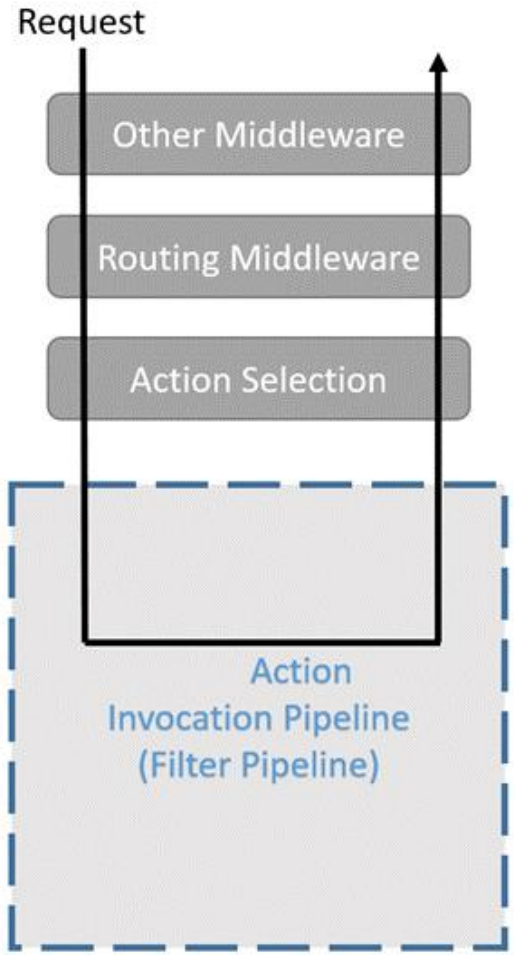
Vincent Uhlmann
IT-Akademie Dr. Heuer GmbH

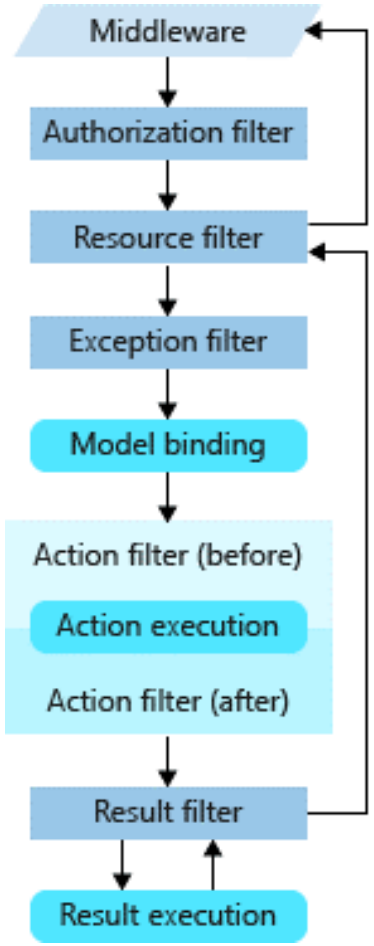
FILTER IN ASP.NET CORE

- Filter sind Komponenten in ASP.NET Core, die es ermöglichen, vor oder nach der Ausführung von Controller-Aktionen zusätzliche Logik einzufügen
- Sie bieten eine zentrale Möglichkeit, sich wiederholende Aufgaben zu handhaben, wie z.B. Authentifizierung, Logging, Fehlerbehandlung und mehr

- **Vorteile**
- Reduzierung von Boilerplate-Code
- Verbesserung der Wartbarkeit
- Erleichterung der Einhaltung von Cross-Cutting Concerns

FILTER IN ASP.NET CORE





FILTER IN ASP.NET CORE

- **Authentifizierung und Autorisierung**
- Sicherstellen, dass nur berechtigte Benutzer auf bestimmte Ressourcen zugreifen können
- Beispiel: [Authorize]-Attribut, um unautorisierte Zugriffe zu verhindern

- **Logging und Monitoring**
- Erfassen von Informationen über Anfragen und Antworten zur Analyse und Debugging
- Beispiel: Ein Action-Filter, der Anfragen und Antworten protokolliert

FILTER IN ASP.NET CORE

- **Fehlerbehandlung**
- Einheitliche Behandlung von Ausnahmen und Fehlern in der Anwendung
- Beispiel: Ein Exception-Filter, der Fehler abfängt und benutzerdefinierte Fehlerantworten zurückgibt

- **Datenvalidierung**
- Überprüfen von Eingabedaten vor der Weiterverarbeitung, um sicherzustellen, dass sie den erwarteten Standards entsprechen
- Beispiel: Ein Action-Filter, der Eingaben validiert und bei ungültigen Daten Fehler zurückgibt

AUTHORIZATION FILTERS

- Prüfen, ob der Benutzer berechtigt ist, die Aktion auszuführen
- [Authorize]-Attribut

```
[ApiController]
[Route("api/[controller]")]
[Authorize(Roles = "Admin")]
public class AdminController : ControllerBase
{
    [HttpGet]
    public IActionResult Get()
    {
        return Ok("This is an admin-only endpoint.");
    }
}
```

RESOURCE FILTERS

- Werden vor der Auswahl des Controllers und der Aktion ausgeführt, oft für Caching oder Initialisierungen
- Implementieren von IResourceFilter

```
public class CacheResourceFilter : IResourceFilter
{
    private readonly IMemoryCache _cache;

    public CacheResourceFilter(IMemoryCache cache)
    {
        _cache = cache;
    }

    public void OnResourceExecuting(ResourceExecutingContext context)
    {
        var cacheKey = context.HttpContext.Request.Path.ToString();
        if (_cache.TryGetValue(cacheKey, out var cachedResponse))
        {
            context.Result = (IActionResult)cachedResponse;
        }
    }

    public void OnResourceExecuted(ResourceExecutedContext context)
    {
        var cacheKey = context.HttpContext.Request.Path.ToString();
        _cache.Set(cacheKey, context.Result);
    }
}
```

ACTION FILTERS

- Vor und nach der Ausführung der Aktion
- Implementieren von IActionFilter

```
public class LoggingActionFilter : IActionFilter
{
    private readonly ILogger _logger;

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

    public void OnActionExecuting(ActionExecutingContext context)
    {
        _logger.LogInformation("Action {ActionName} is executing.",
context.ActionDescriptor.DisplayName);
    }

    public void OnActionExecuted(ActionExecutedContext context)
    {
        _logger.LogInformation("Action {ActionName} executed.",
context.ActionDescriptor.DisplayName);
    }
}
```

EXCEPTION FILTERS

- Handhaben von Ausnahmen, die während der Aktion auftreten Implementieren von IResourceFilter
- Implementieren von IExceptionFilter

```
public class ExceptionFilter : IExceptionFilter
{
    public void OnException(ExceptionContext context)
    {
        context.Result = new ObjectResult(new
        {
            error = context.Exception.Message
        })
        {
            StatusCode = 500
        };
    }
}
```

RESULT FILTERS

- Vor und nach der Ausführung des Aktionsresultats
- Implementieren von IResultFilter

```
public class SampleResultFilter : IResultFilter
{
    public void OnResultExecuting(ResultExecutingContext context)
    {
        // Logik vor der Ergebnis-Ausführung
    }

    public void OnResultExecuted(ResultExecutedContext context)
    {
        // Logik nach der Ergebnis-Ausführung
    }
}
```

REGISTRIEREN VON FILTERN

- Zum benutzen des Filters muss dieser im DI-Container registriert werden

```
builder.Services.AddScoped<LoggingActionFilter>();
```

- Anschließend kann dieser über Action Methoden innerhalb von Controllern genutzt werden

```
[HttpGet]
[ServiceFilter(typeof(LoggingActionFilter))]
public IActionResult Get()
{
    return Ok("Hello from UsersController!");
}
```

VERWENDUNG VON FILTERN ALS ATTRIBUTE

- Filter können auch als Attribut erstellt werden
- Sorgt für eine einfache und deklarative Anwendung direkt auf Aktionen oder Controller
- Vermeidung der Notwendigkeit, Filter explizit im DI-Container zu registrieren

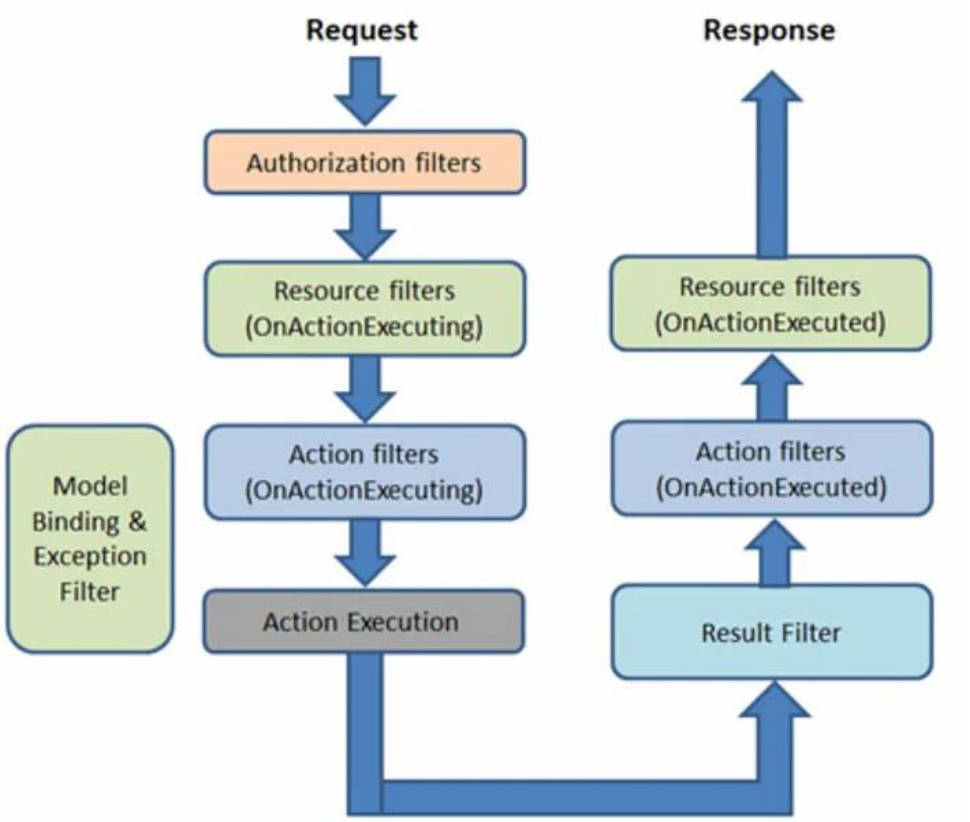
```
public class LoggingActionFilterAttribute : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext context)
    {
        // Do something before the action executes.
    }

    public override void OnActionExecuted(ActionExecutedContext context)
    {
        // Do something after the action executes.
    }
}
```

VERWENDUNG VON FILTERN ALS ATTRIBUTE

```
[HttpGet("test")]
[LoggingActionFilter]
public IActionResult GetTest()
{
    return Ok("Hello from UsersController!");
}
```

STANDARDREIHENFOLGE DER AUSFÜHRUNG



ZUGRIFF AUF MODELS IN ACTION FILTERN

- Da Action Filter nach dem Model Binding ausgeführt werden, ist der Zugriff auf die Daten bzw. Parameter der Action Methode möglich

```
public class LoggingActionFilterAttribute : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext context)
    {
        if(context.ActionArguments.TryGetValue("user", out var user)) {
            if(user is User u) {
                // do something
            }
        }
    }
}
```

UNTERBRECHEN DER FILTERAUSFÜHRUNG

- Um die Ausführung eines Filters vorzeitig abzubrechen und ein Ergebnis zurückzugeben muss das Ergebnis des Contextes gesetzt werden
- Setzen Sie z.B. context.Result auf BadRequestObjectResult, NotFoundResult oder OkObjectResult

```
public class LoggingActionFilterAttribute : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext context)
    {
        // Abbrechen der Ausführung und Rückgabe eines Ergebnisses
        context.Result = new BadRequestObjectResult("Bedingung nicht erfüllt.");
        return;
    }
}
```