Gateway Group

ASP.NET CORE

THE GATEWAY CORP.

INTRODUCTION & PURPOSE

- Binkal Patel
- Technical Leader
- The Gateway Corp.

Gateway Group

WHAT IS ASP.NET CORE?

- Web framework from the Microsoft
- Redesigned asp.net to be faster, flexible and work for any platform
- ASP.NET Core is the framework that can be used for web development with .NET.
- ASP.NET Core is an open source and cloud-optimized web framework.
- Used for developing modern web applications that can be developed and run on Windows, Linux and the Mac.
- Includes MVC Framework
- Combines feature of MVC and Web Api in single web application programming framework

WHAT IS ASP.NET CORE?

- ASP.NET Core apps can run on .NET Core or on the full .NET Framework.
- architected to provide an optimized development framework for apps that are deployed to the cloud or run on-premises.
- It consists of modular components with minimal overhead,
- flexible while constructing solutions.
- develop and run your ASP.NET Core apps cross-platform on Windows, Mac and Linux.

ADVANTAGES OF ASP.NET CORE

- ASP.NET Core has a number of architectural changes that result in a much leaner and modular framework.
- ASP.NET Core is no longer based on System.Web.dll.
- It is based on a set of granular and well factored NuGet packages.
- allows to optimize app to include just the NuGet packages needed.
- The benefits of a smaller app surface area include tighter security, reduced servicing, improved performance, and decreased costs
- Cloud-ready environment-based configuration.
- Built-in support for dependency injection.
- Tag Helpers which makes Razor markup more natural with HTML.
- Ability to host on IIS or self-host in your own process.

EVOLUTION

- Asp(classic ASP) – 1996
- Asp.Net web forms – 2002 (statemanagement, viewstates)
  - First time introducted code behind
- ASP.Net MVC - 2009
  - Overcome issues of ASP.Net
  - Unit test
  - Separation of concern
  - Created on top of web platform component
  - Created before cloud platform
- Asp.net core – 2016
  - New .net core framework
  - First version of .net
  - 2017 – core 2
  - Sep. 2019 – core 3

TOOLS

- Visual Studio 2019
  - https://visualstudio.microsoft.com/vs/
- Sql Server 2017 or higher
  - https://www.microsoft.com/en-au/sql-server/sql-server-downloads
- Asp.net Core
  - https://dotnet.microsoft.com/download/dotnet-core/3.0
- Visual studio code
  - https://code.visualstudio.com/docs/?dv=win
- Dotnet Cli
- Node js
- Npm

FUNDAMENTALS & PROJECT STRUCTURE WALKTHROUGH

- Create first .net core application using vs 2019
- Create first .net core application using dotnet cli and VS code

- .csproj file
- Launchsetting.json
- Wwwroot
- Razor pages
- Page folder

.CSPROJ

- csproj file
  - Containing framework information
  - Installed nugget packages
  - Prior to .net core 3.0 meta packages were installed automatically as a part of project thru Nuget package
  - In .net core 3 it is a part of .net core installation

LAUNCHSETTING.JSON

- Launchsettings.json
  - What to do when execute application
  - Default profiles created
    - Iisexpress
    - Project profile – cli
  - Navigate and validate the project properties

TAG HELPERS

- New in asp.net core
- Server side component
- Similar to html helpers
- In order to use it, required to import it first (_ViewImports)
  - @addTagHelper *,Microsoft.AspNetCore.Mvc.TagHelpers

WWWROOT

- Containing css, js and lib
- Used to be store css,js and other libraries
- No other files
- Same as assets in angular
- To separate static files from the application pages

RAZOR AGES

- Introduced first time in .net core 2.0
- New feature of .net core mvc for coding page
- Simple way to do the same as we are doing with mvc
- Razor pages have 2 parts
  - Razor page - UI
  - Page model – contain handler – controller
- Contain all the methods like get , put post with prefix "On"

PAGES

- Shared folder
  - Partial views
  - Partial pages
  - Shared layout
  - Validation scripts partial
    - Containg the script only
  - ViewImport.cshtml (Global declaration)
    - Contains name space
    - Taghelpers
  - Viewstart.html
    - Which master page is to be used by default
  - Erros page
    - Redirection page in case any error occurred

ROUTING

- What is routing ?
  - Used to matches url – file paths
- Rules
  - Required root folder (pages)
  - We Can add areas
  - Index.cshtml will be default route
- How URL and page mapped in routing
- No need to write explicit routing rule
- Following the path location strategy

ACTION RESULTS

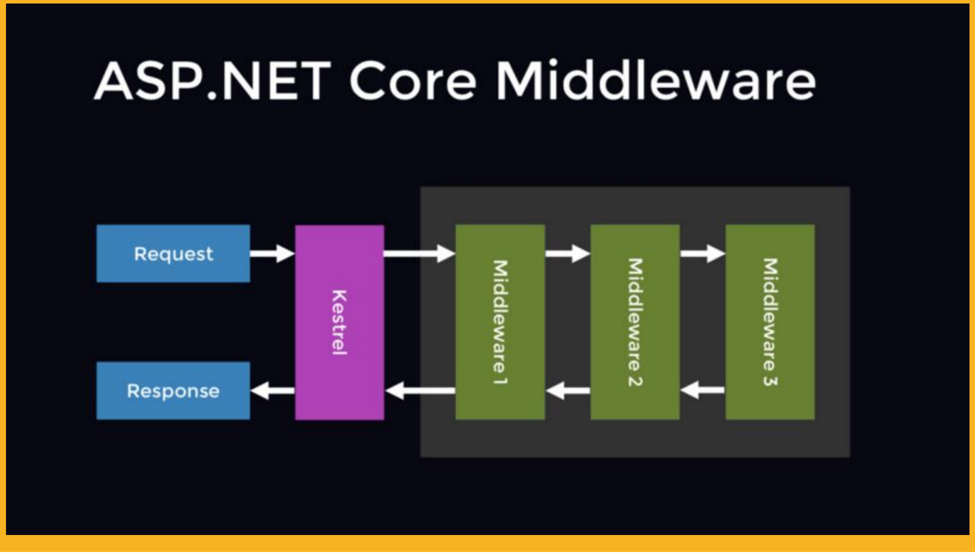
- Action result is result of all action methods
- Is a parent class of many derived classes
- iAction result return type is to be used when may types can be returned
- Action results in Razor pages
  - Content result – to return text/html, application/json
  - File content result – file from byte array , virtual path
  - Notfountresult – 404 not found
  - Pageresult – process and return result of page
  - Partialresult – return partial page
  - Redirecttopageresult – redirects user to specified page
  - View component result – result of executing viewcomponent

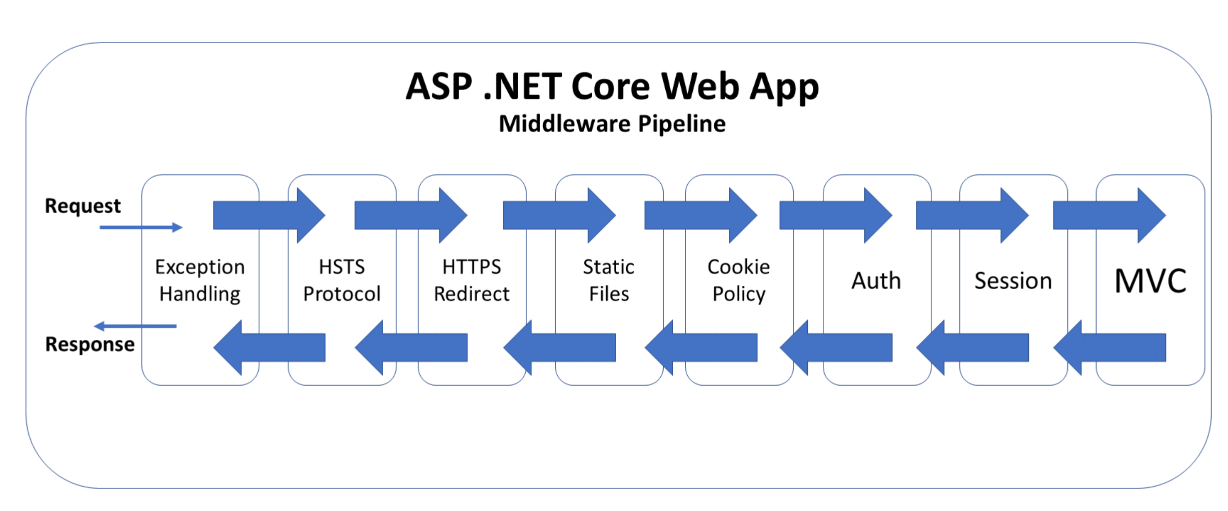
HOW BOOTSTRAPPING ASP.NET CORE APPLICATION

- Main method
- Startup
- Middleware
- App settings

MIDDLE WARE

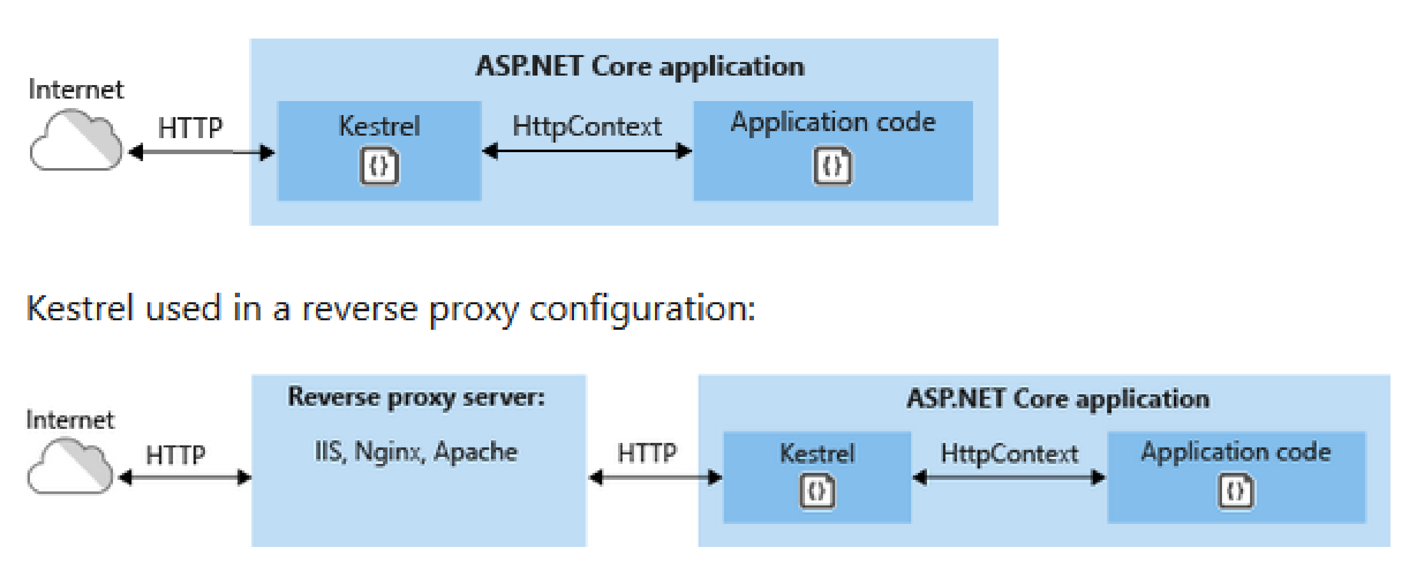
- What is Middle ware?
- Middleware in ASP.NET Core are software components that are specifically orchestrated in the application pipeline to handle requests and responses.
- Each component can choose to potentially modify and let the request thru to the next component or immediately return a response based on specific implementation logic.





Gateway Group

HTTPCONTEXT & MIDDLEWARE PIPE LINE



Kestrel used in a reverse proxy configuration:

APPSETTINGS

- All application specific settings written in this file
- File name will be appsettings. Json
- Any changes to this file required restart app on iis to take effect

DEPENDENCY INJECTION

- What is dependency injection?
- .netcore is designed from scratch to support dependency injection
- It will be used to inject dependency class through constructor or method by using in built IOC container
- DI is a pattern that can help developer to decouple the difference piece of code – module
- Framework service and application service can be injected in class in .net core
- Do not need to write and use an individual dependency containers and libraries
- Container is responsible for create and manage instance of the class as and when required

API CONTROLLERS

- Api contollers can be added to the same MVC application like we had in previous version of MVC.
- Api controllers need to be registerd in the starup orchestration
- In configure service method of startup class
  - Services,AddControllersWithViews()
  - services.AddControllers();
- Required to add the same in middleware section of the application

```
App.UseEndpoints(endpoints=>
{
       endpoints.MapRazorPages();
       endpoints.MapControllers();
});
```

Gateway Group

API CONTROLLERS

- Routing can be defined same as web api routing using [Route(“api/controller”)]
- To make any controller as API Controller need to add [ApiController] attribute on top of controller

MVC APPLICATION

- MVC
  - Model ?
  - View ?
  - Controller ?
- Controller > model > view
- Controller > view > model

MVC ROUTING

```
App.UseEndpoints(endpoints=>
{
    endpoints.MapControllersRoute(
            name:“defalult”,
            Pattern:“{controller=Home}/{action=Index}/{id?}”
            );
    endpoints.MapRazorPages
});
```

URL PATTERNS

- DomainName/Controller/Action/Parameters
- DomainName/AreaName/Controller/Action/Parameters

ROUTING WITH AREA

```
App.UseEndpoints(endpoints=>
{
        endpoints.MapControllersRoute(
                        name:“defalult”,
                        Pattern:“{area=Users}/{controller=Home}/{action=Index}/{id?}”
                        );
        endpoints.MapRazorPages
});
```

- Areaname has to be added on top of controller name
  - [Area(“Users”)]
- Copy _viewimports.cshtml and _viewStart.cshtml to area’s view folder

.NET CORE SECURITY – USING IDENTITY

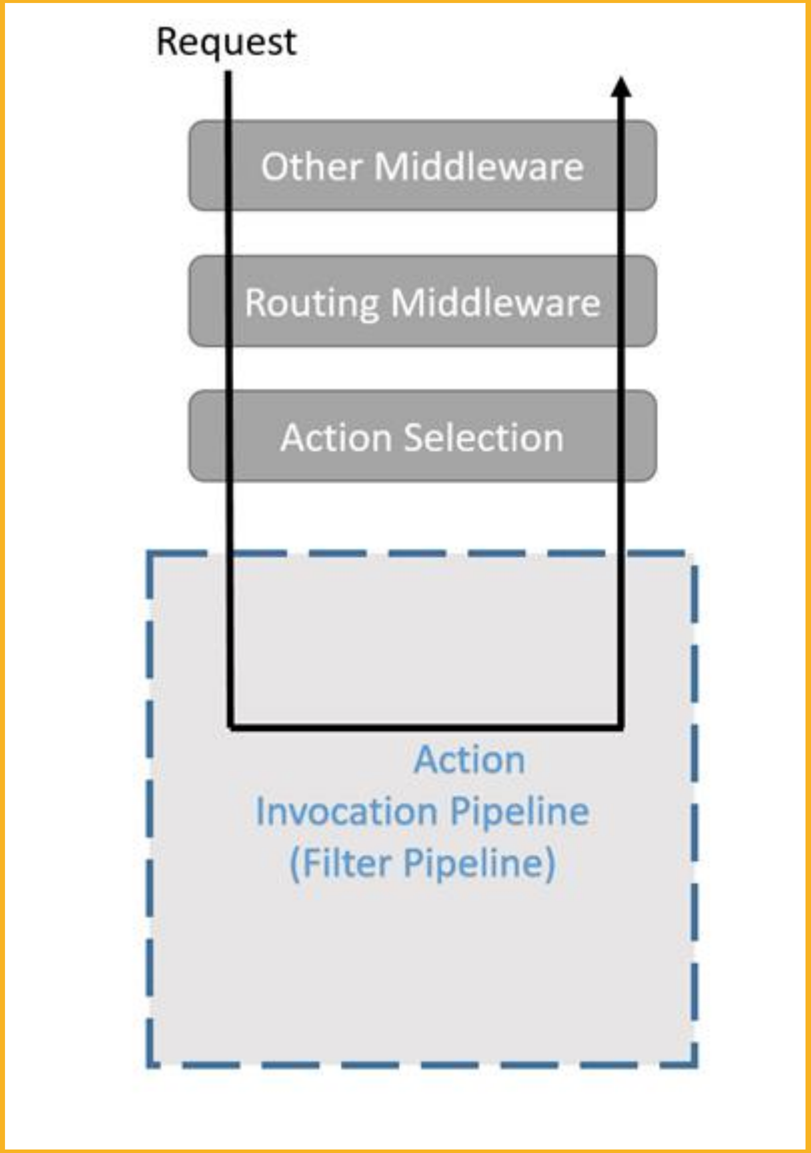
- .net Core Identity implementation

FILTERS

- It will allow ASP .NET Core to be run after or before specific stage in the request processing pipe line
- There are 2 built-in filters available
  - Authorization
  - Response Caching
- Custom filters can be created to handle cross cutting concerns
  - Error Handling
  - Caching
  - Configuration
  - Authorization
  - Logging

HOW FILTER WORKS?
TYPES OF FILTERS

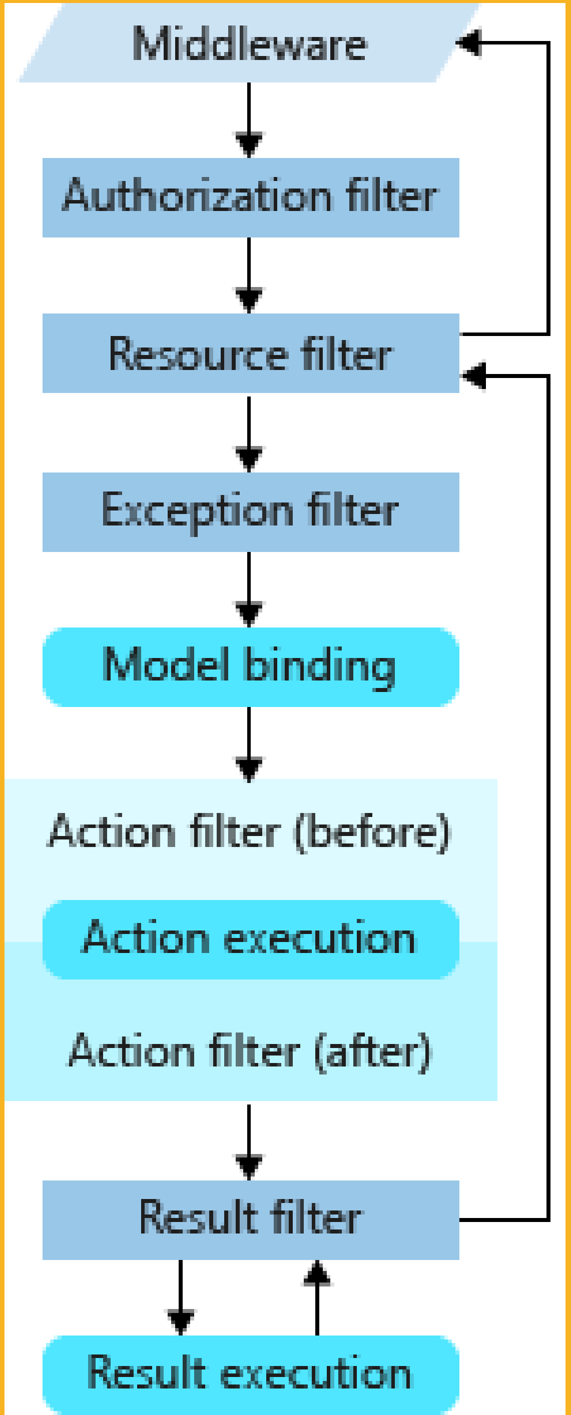
- Authorization
- Resource
- Action
- Exception
- Result



Gateway Group

FILTER SEQUENCE

- Filters support both synchronous and asynchronous implementations through different interface definitions.
- Synchronous filters can run code before (On-Stage-Executing) and after (On-Stage-Executed) their pipeline stage.
- OnActionExecuting is called before the action method is called. OnActionExecuted is called after the action method returns.



Gateway Group

SYNC AND ASYNC FILTER

synchronous filters define an On-Stage-Executionmethod:

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

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

Asynchronous filters define an On-Stage-ExecutionAsync method:

```
public class SampleAsyncActionFilter : IAsyncActionFilter
{
    public async Task OnActionExecutionAsync(
        ActionExecutingContext context,
        ActionExecutionDelegate next)
    {
        // Do something before the action executes.

        // next() calls the action method.
        var resultContext = await next();
        // resultContext.Result is set.
        // Do something after the action executes.
    }
}
```

FILTER SCOPES AND ORDER OF EXECUTION

- A filter can be added to the pipeline at one of three *scopes*:
  - Using an attribute on an action.
  - Using an attribute on a controller.
  - Globally for all controllers and actions as shown in the following code:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(options =>
    {
        options.Filters.Add(new AddHeaderAttribute("GlobalAddHeader",
            "Result filter added to MvcOptions.Filters"));         // An instance
        options.Filters.Add(typeof(MySampleActionFilter));         // By type
        options.Filters.Add(new SampleGlobalActionFilter());       // An instance
    }).SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
}
```

DEFAULT ORDER OF EXECUTION

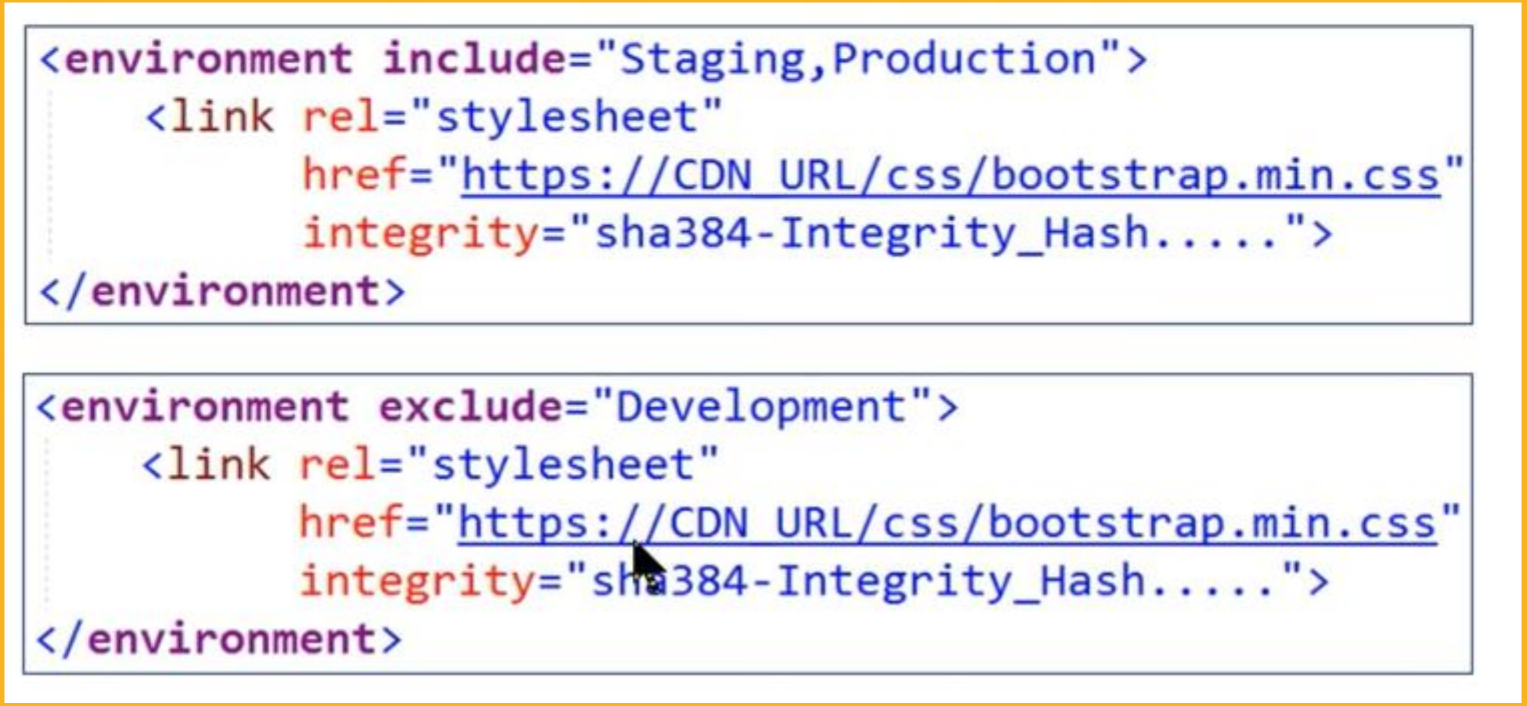
Sequence	Filter Scope	Filter Method
1	Global	OnActionExecuting
2	Controller	OnActionExecuting
3	Method	OnActionExecuting
4	Method	OnActionExecuted
5	Controller	OnActionExecuted
6	Global	OnActionExecuted

https://docs.microsoft.com/en-us/aspnet/core/razor-pages/filter?view=aspnetcore-3.0#implement-razor-page-filters-by-overriding-filter-methods

https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/filters?view=aspnetcore-3.0#filter-scopes-and-order-of-execution

TAG HELPERS

- Similar to angular directives
- Introduced in only asp.net core
- Enables server side coding and rendering html
- Focused around HTMl elements
  - i.e. asp-area, asp-page
- Similar to html helper
  - Syntax is different
- Examples
  - Anchor
  - Image
  - Environment
  - form
- Why Tag Helpers
- Custom Tag Helper



```
<environment include="Staging,Production">
    <link rel="stylesheet"
          href="https://CDN_URL/css/bootstrap.min.css"
          integrity="sha384-Integrity_Hash.....">
</environment>
```

```
<environment exclude="Development">
    <link rel="stylesheet"
          href="https://CDN_URL/css/bootstrap.min.css"
          integrity="sha384-Integrity_Hash.....">
</environment>
```