Section 1 Entry

Section Cheat Sheet

Asp.Net Core

Asp.Net Core is a cross-platform, high-performance, open-source framework for building modern, cloud-enabled web applications and services.

Cross-platform

Asp.Net Core apps can be hosted on Windows, LINUX and Mac.

Can be hosted on different servers

Supports Kestrel, IIS, Nginx, Docker, Apache

Open-source

Contributed by over 1000+ contributors on GitHub

https://github.com/dotnet/aspnetcore

Cloud-enabled

Out-of-box support for Microsoft Azure

Modules

Asp.Net Core Mvc

For creating medium to complex web applications

Asp.Net Core Web API

For creating RESTful services for all types of client applications.

Asp.Net Core Razor Pages

For creating simple & page-focused web applications

Asp.Net Core Blazor

For creating web applications with C# code both on client-side and server-side

Asp.Net Web Forms [vs] Asp.Net Mvc [vs] Asp.Net Core

Asp.Net Web Forms

- 2002
- Performance issues due to server events and view-state.
- Windows-only
- Not cloud-friendly
- Not open-source
- Event-driven development model.

Asp.Net Mvc

- 2009
- Performance issues due to some dependencies with asp.net (.net framework)
- Windows-only
- Slightly cloud-friendly
- Open source
- Model-view-controller (MVC) pattern
-

Asp.Net Core

- 2016
- Faster performance
- Cross-platform
- Cloud-friendly
- Open-source
- Model-view-controller (MVC) pattern

Section 2 Get Started

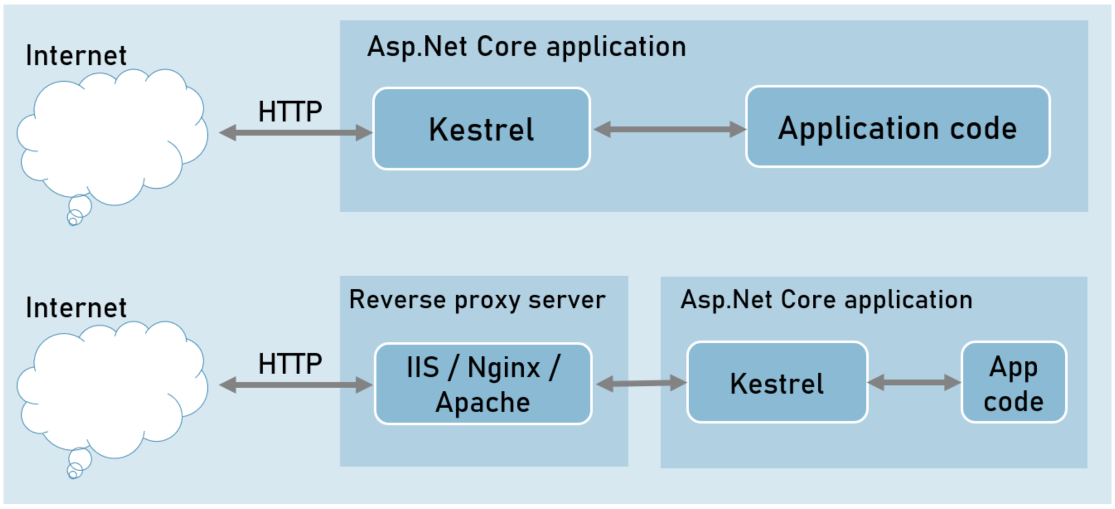
Kestrel and Other Servers

Application Servers

- Kestrel

Reverse Proxy Servers

- IIS
- Nginx
- Apache



Benefits of Reverse Proxy Servers

- Load Balancing
- Caching
- URL Rewriting
- Decompressing the requests
- Authentication
- Decryption of SSL Certificates

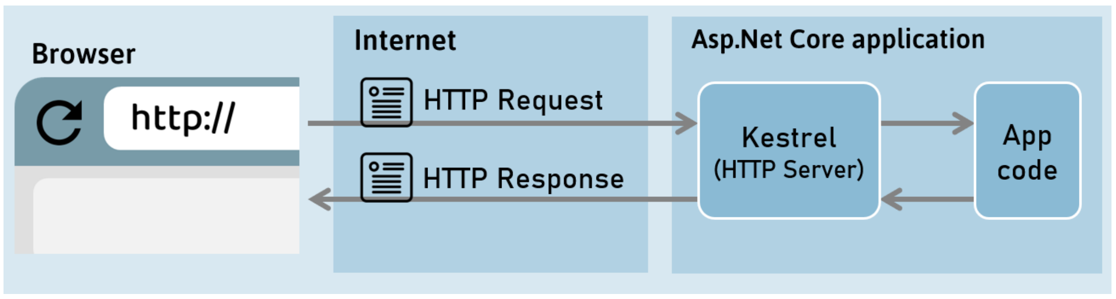
IIS express

- HTTP access logs
- Port sharing
- Windows authentication
- Management console
- Process activation
- Configuration API
- Request filters
- HTTP redirect rules

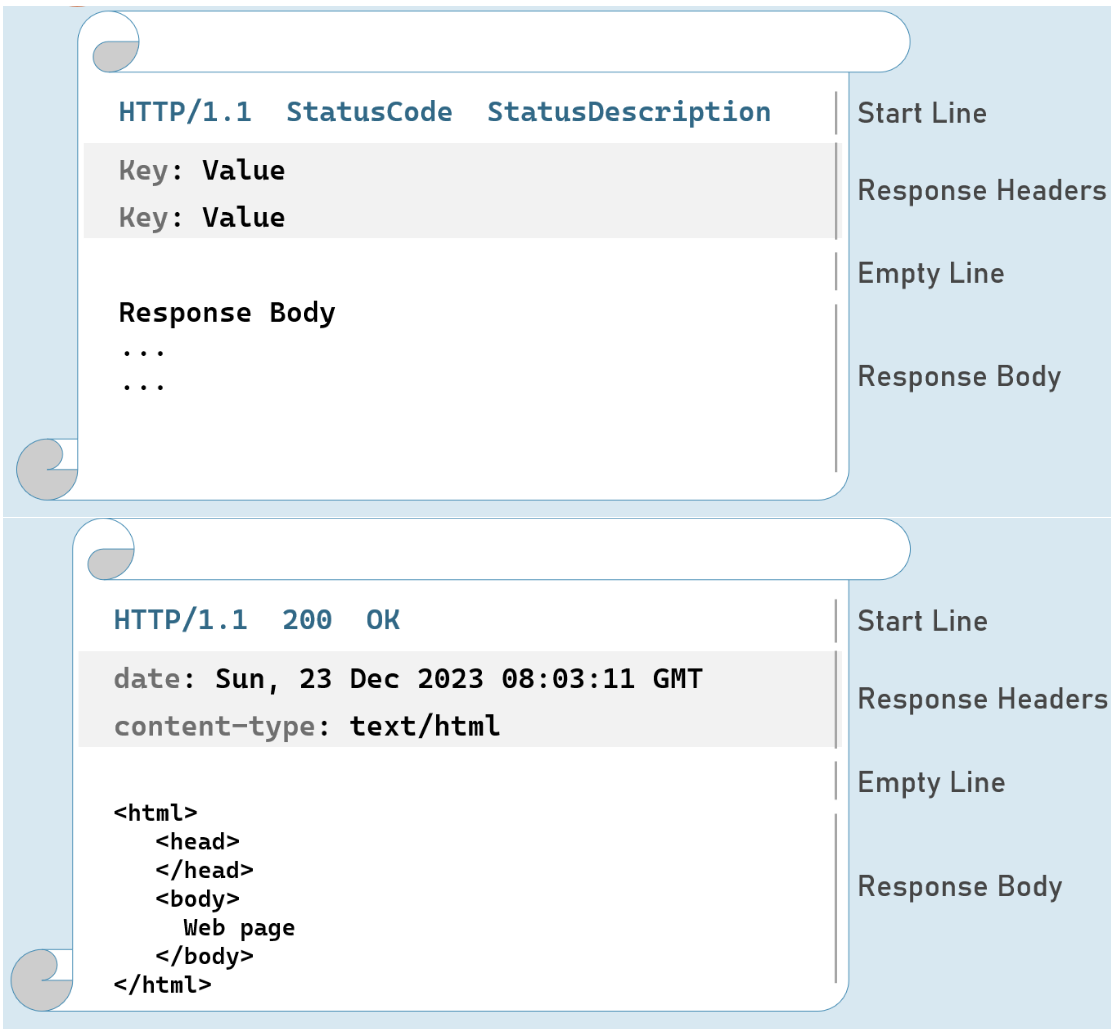
Section 3 Introduction to HTTP

HTTP is an application-protocol that defines set of rules to send request from browser to server and send response from server to browser.

Initially developed by Tim Berners Lee, later standardized by IETF (Internet Engineering Task Force) and W3C (World Wide Web Consortium)



HTTP Response


```
HTTP/1.1  StatusCode  StatusDescription
Key: Value
Key: Value

Response Body
...
...
```

```
HTTP/1.1  200  OK
date: Sun, 23 Dec 2023 08:03:11 GMT
content-type: text/html

<html>
   <head>
   </head>
   <body>
     Web page
   </body>
</html>
```

Response Start Line

Includes HTTP version, status code and status description.

HTTP Version: 1/1 | 2 | 3

Status Code: 101 | 200 | 302 | 400 | 401 | 404 | 500

Status Description: Switching Protocols | OK | Found | Bad Request | Unauthorized | Not Found | Internal Server Error

HTTP Response Status Codes

1xx | Informational

101	Switching Protocols

2xx | Success

200	OK

3xx | Redirection

302	Found
304	Not Modified

4xx | Client error

400	Bad Request
401	Unauthorized
404	Not Found

5xx | Server error

500	Internal Server Error

HTTP Response Headers

Date

Date and time of the response. Ex: Tue, 15 Nov 1994 08:12:31 GMT

Server

Name of the server.

Ex: Server=Kestrel

Content-Type

MIME type of response body.

Ex: text/plain, text/html, application/json, application/xml etc.

Content-Length

Length (bytes) of response body.

Ex: 100

Cache-Control

Indicates number of seconds that the response can be cached at the browser.

Ex: max-age=60

Set-Cookie

Contains cookies to send to browser.

Ex: x=10

Access-Control-Allow-Origin

Used to enable CORS (Cross-Origin-Resource-Sharing)

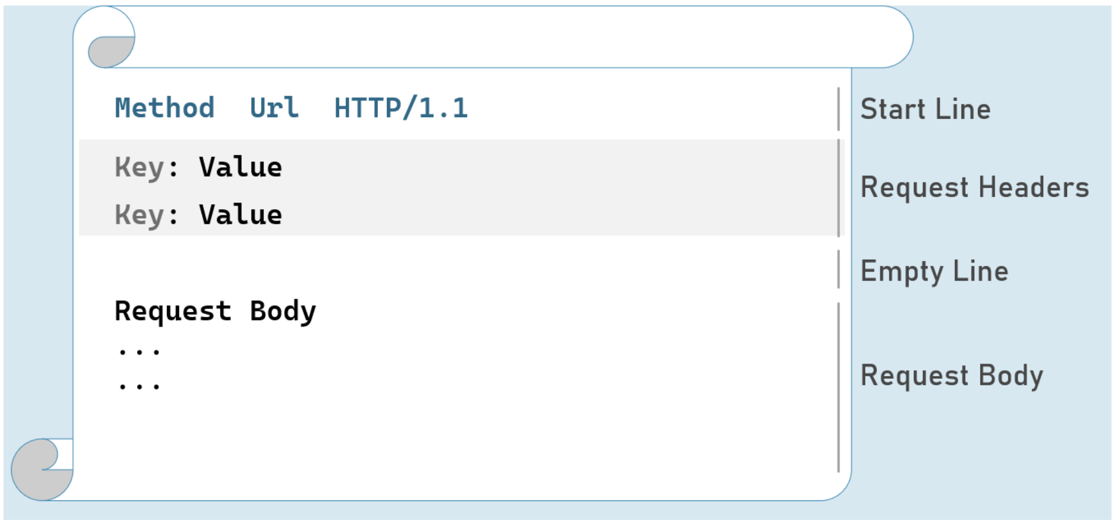
Ex: Access-Control-Allow-Origin: http://www.example.com

Location

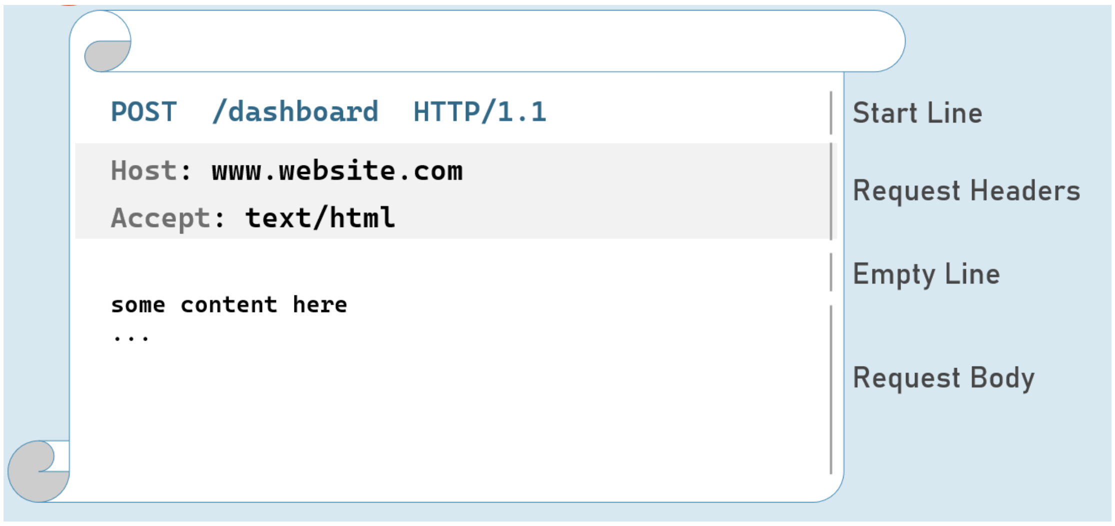
Contains url to redirect.

Ex: http://www.example-redirect.com

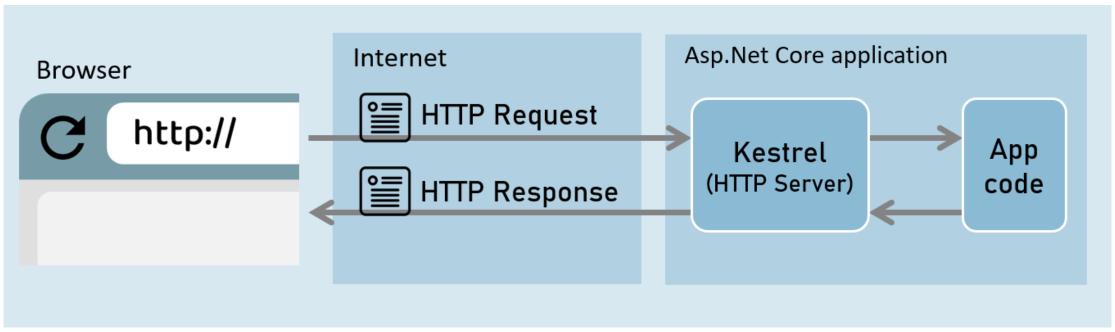
HTTP Request



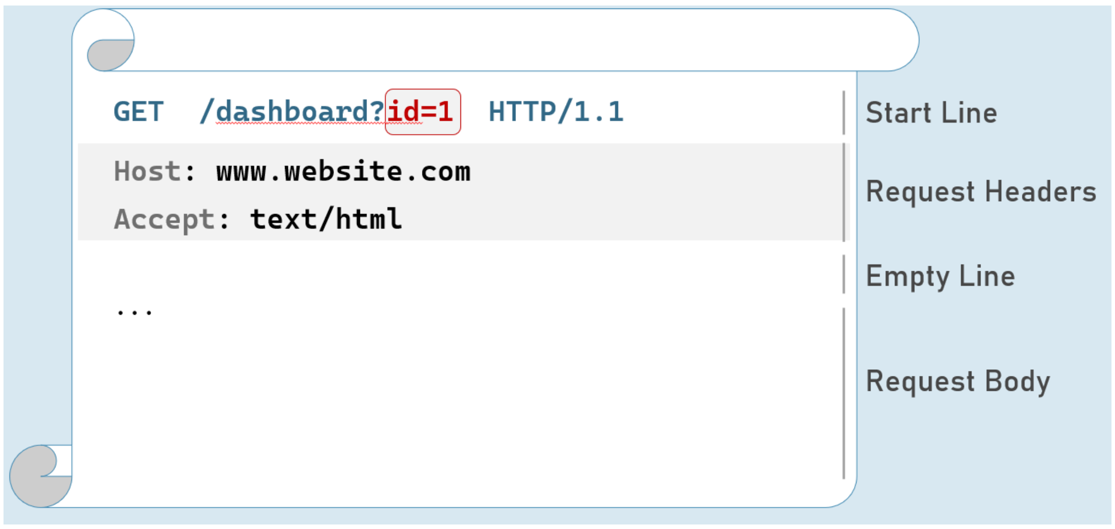
```
Method  Url  HTTP/1.1
Key: Value
Key: Value

Request Body
...
...
```



```
POST  /dashboard  HTTP/1.1
Host: www.website.com
Accept: text/html

some content here
...
```



HTTP Request - with Query String



```
GET  /dashboard?id=1  HTTP/1.1
Host: www.website.com
Accept: text/html

...
```

HTTP Request Headers

Accept

Represents MIME type of response content to be accepted by the client. Ex: text/html

Accept-Language

Represents natural language of response content to be accepted by the client. Ex: en-US

Content-Type

MIME type of request body.

Eg: text/x-www-form-urlencoded, application/json, application/xml, multipart/form-data

Content-Length

Length (bytes) of request body.

Ex: 100

Date

Date and time of request.

Eg: Tue, 15 Nov 1994 08:12:31 GMT

Host

Server domain name.

Eg: www.example.com

User-Agent

Browser (client) details.

Eg: Mozilla/5.0 Firefox/12.0

Cookie

Contains cookies to send to server.

Eg: x=100

HTTP Request Methods

GET

Requests to retrieve information (page, entity object or a static file).

Post

Sends an entity object to server; generally, it will be inserted into the database.

Put

Sends an entity object to server; generally updates all properties (full-update) it in the database.

Patch

Sends an entity object to server; generally updates few properties (partial-update) it in the database.

Delete

Requests to delete an entity in the database.

HTTP Get [vs] Post

Get:

- Used to retrieve data from server.
- Parameters will be in the request url (as query string only).

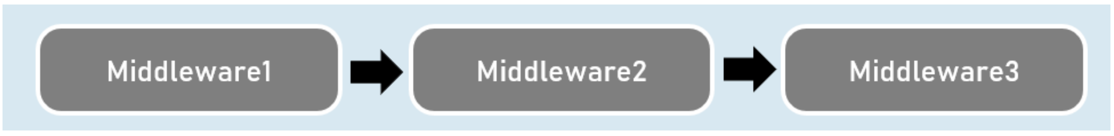
- Can send limited number of characters only to server. Max: 2048 characters
- Used mostly as a default method of request for retrieving page, static files etc.
- Can be cached by browsers / search engines.

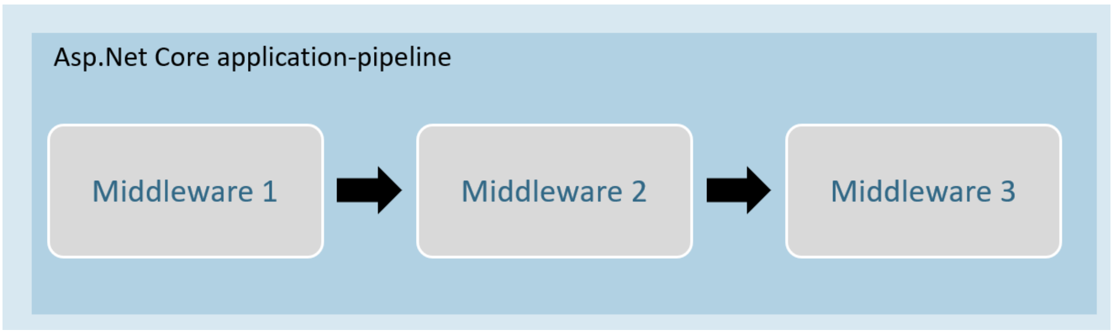
Post:

- Used to insert data into server
- Parameters will be in the request body (as query string, json, xml or form-data).
- Can send unlimited data to server.
- Mostly used for form submission / XHR calls
- Can't be cached by browsers / search engines.

Section 4 Introduction to Middleware

Middleware is a component that is assembled into the application pipeline to handle requests and responses.
Middlewares are chained one-after-other and execute in the same sequence how they're added.





Middleware can be a request delegate (anonymous method or lambda expression) [or] a class.

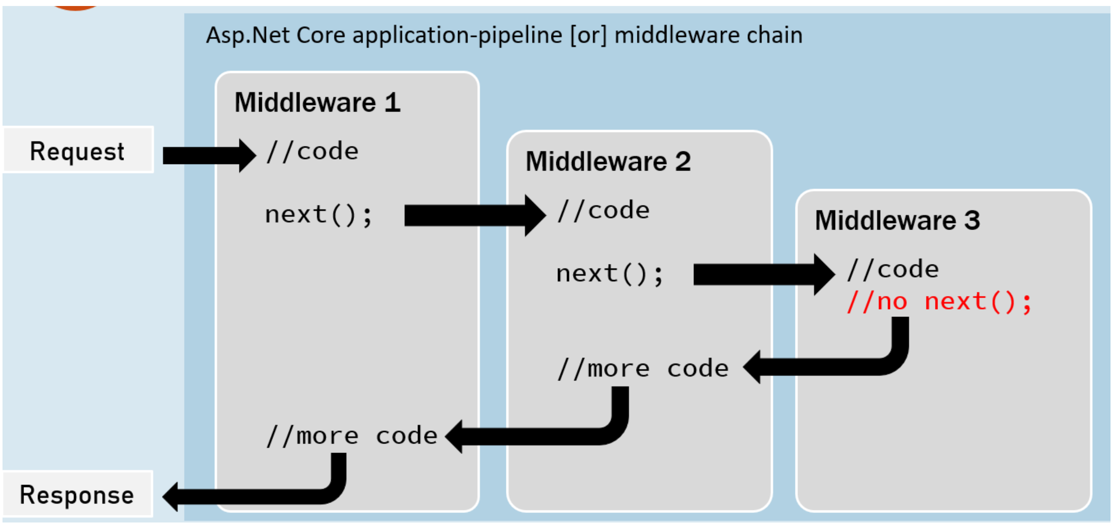
Middleware - Run

app.Run( )

```
app.Run(async (HttpContext context) =>
{
//code
});
```

The extension method called “Run” is used to execute a terminating / short-circuiting middleware that doesn’t forward the request to the next middleware.

Middleware Chain



app.Use( )

```
app.Use(async (HttpContext context, RequestDelegate next) =>
{
  //before logic
  await next(context);
  //after logic
});
```

The extension method called “Use” is used to execute a non-terminating / short-circuiting middleware that may / may not forward the request to the next middleware.

Middleware Class

Middleware class is used to separate the middleware logic from a lambda expression to a separate / reusable class.

```
class MiddlewareClassName : IMiddleware
{
  public async Task InvokeAsync(HttpContext context, RequestDelegate next)
  {
    //before logic
    await next(context);
    //after logic
  }
}
```

```
app.UseMiddleware<MiddlewareClassName>();
```

Middleware Extensions

```
class MiddlewareClassName : IMiddleware
{
  public async Task InvokeAsync(HttpContext context,RequestDelegate next)
  {
    //before logic
    await next(context);
    //after logic
  }
});
```

Middleware extension method is used to invoke the middleware with a single method call.

```
static class ClassName
{
  public static IApplicationBuilder ExtensionMethodName(this IApplicationBuilder app)
  {
    return app.UseMiddleware<MiddlewareClassName>();
  }
}
```

```
app.ExtensionMethodName();
```

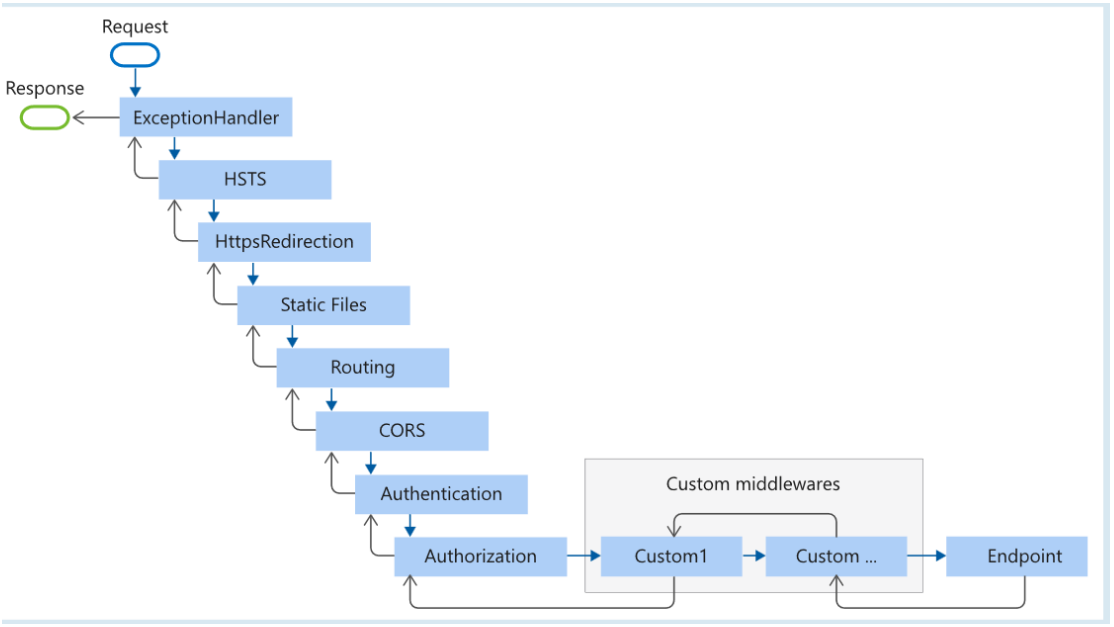
Conventional Middleware

```
class MiddlewareClassName
{
  private readonly RequestDelegate _next;

  public MiddlewareClassName(RequestDelegate next)
  {
    _next = next;
  }

  public async Task InvokeAsync(HttpContext context)
  {
   //before logic
   await _next(context);
   //after logic
  }
});
```

```
static class ClassName
{
  public static IApplicationBuilder ExtensionMethodName(this
IApplicationBuilder app)
  {
   return app.UseMiddleware<MiddlewareClassName>();
  }
}
```

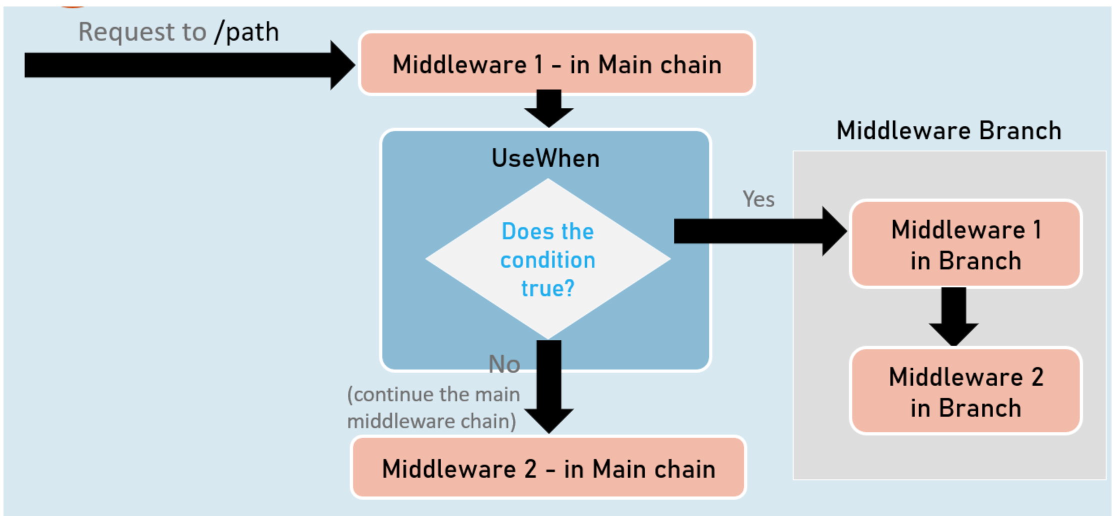
```
app.ExtensionMethodName();
```

The Right Order of Middleware



```
app.UseExceptionHandler("/Error");
app.UseHsts();
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseCors();
app.UseAuthentication();
app.UseAuthorization();
app.UseSession();
app.MapControllers();
//add your custom middlewares
app.Run();
```

Middleware - UseWhen



app.UseWhen( )

```
app.UseWhen(
  context => { return boolean; },
  app =>
  {
    //add your middlewares
  }
);
```

The extension method called “UseWhen” is used to execute a branch of middleware only when the specified condition is true.