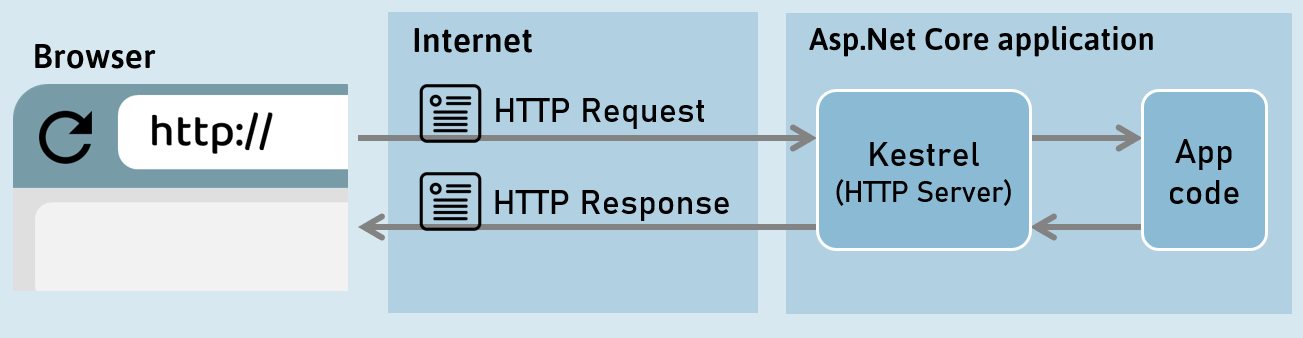
**Section Cheat Sheet (PPT)**

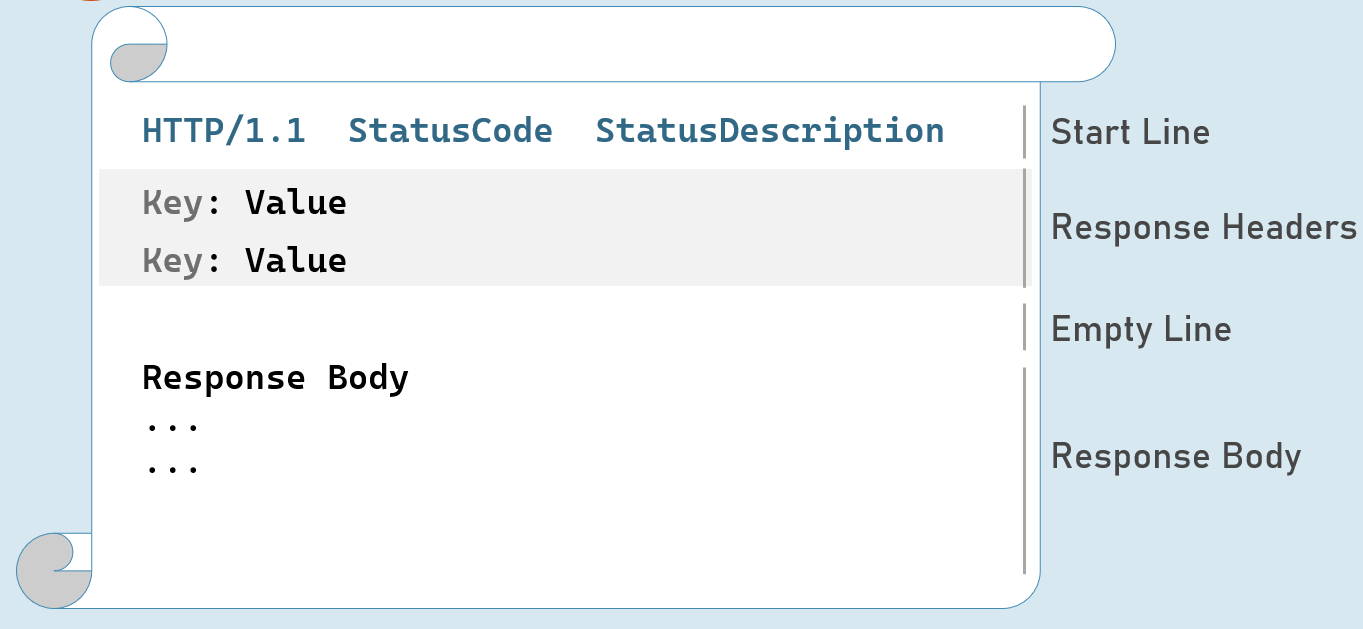
Introduction to HTTP

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



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

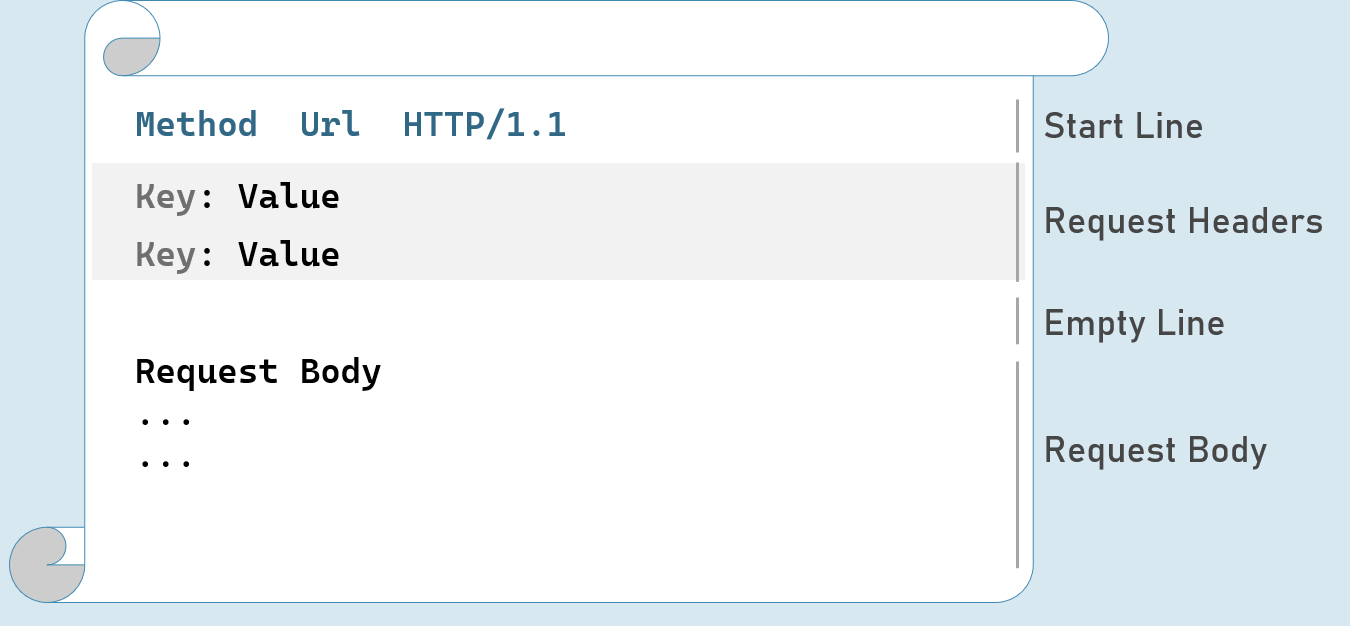
**Location**

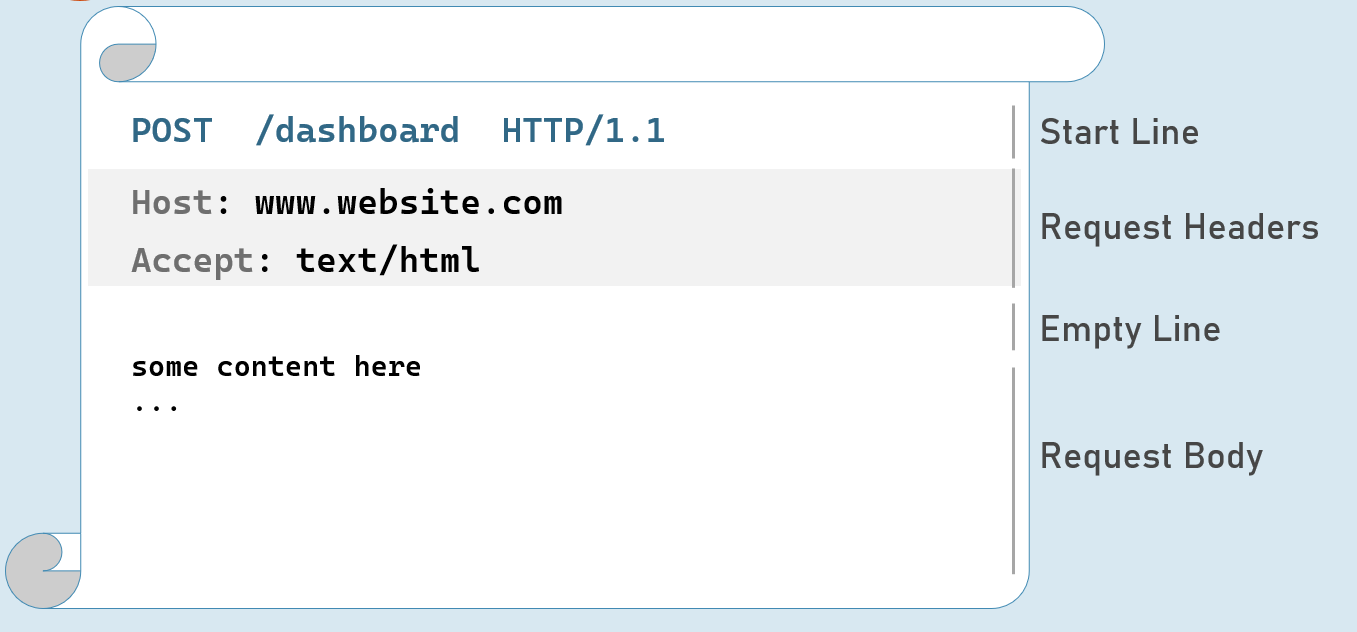
Contains url to redirect.

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

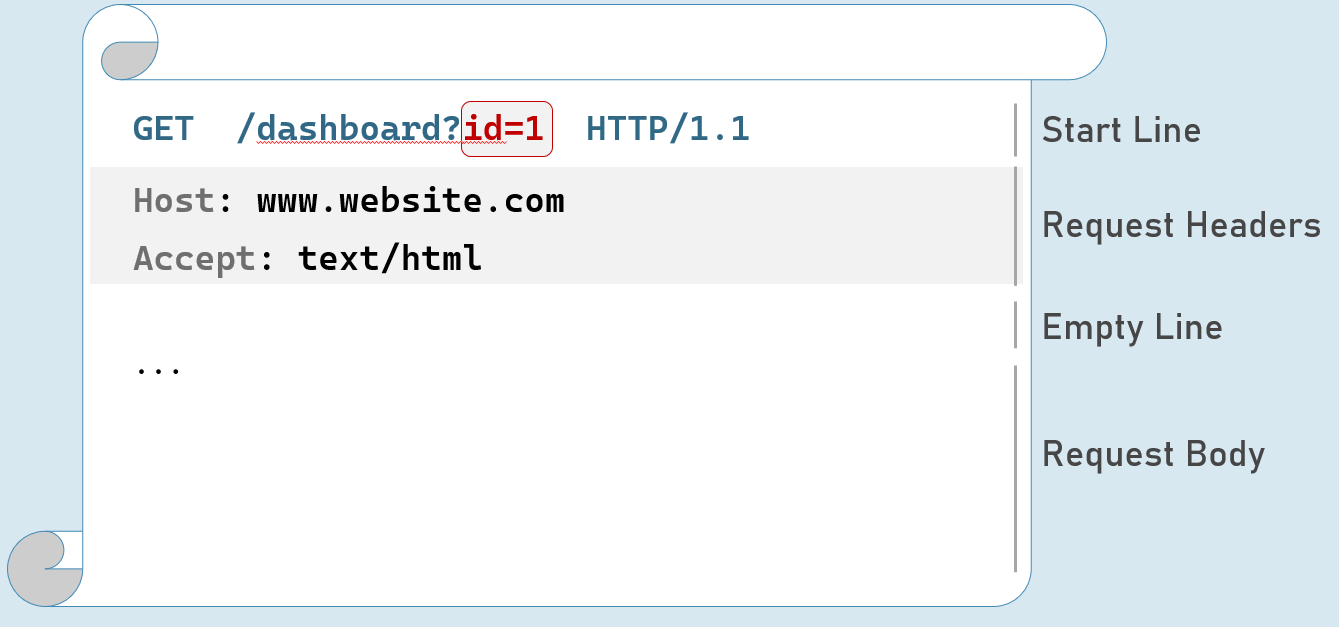
Further reading: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>

HTTP Request





**HTTP Request - with Query String**



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

Further reading: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>

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:**

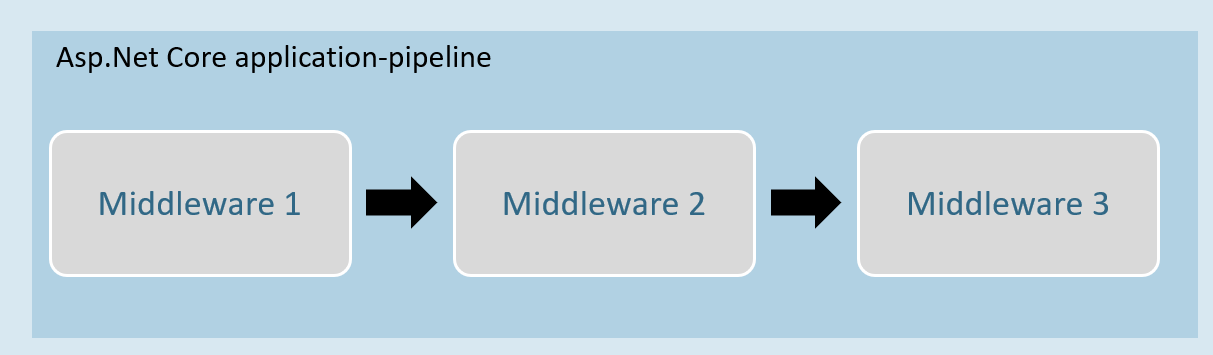
* Used to insert data into server
* Parameters will be in the request body (as query string, json, xml or form-data).
* Can send unlimited data to server.
* Mostly used for form submission / XHR calls
* Can't be cached by browsers / search engines.

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.

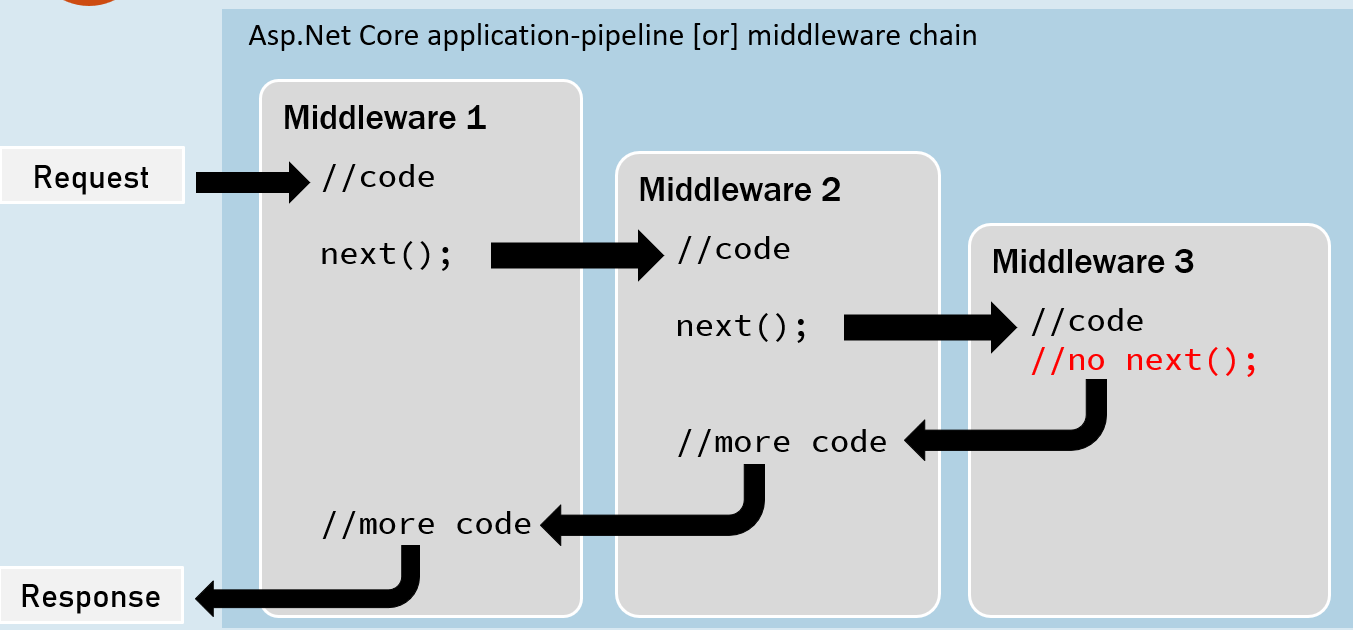
Middleware - Run

**app.Run( )**

1. app.Run(async (HttpContext context) =>
2. {
3. //code
4. });

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( )**

1. app.Use(async (HttpContext context, RequestDelegate next) =>
2. {
3. //before logic
4. await next(context);
5. //after logic
6. });

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.

1. class MiddlewareClassName : IMiddleware
2. {
3. public async Task InvokeAsync(HttpContext context, RequestDelegate next)
4. {
5. //before logic
6. await next(context);
7. //after logic
8. }
9. }

app.UseMiddleware<MiddlewareClassName>();

Middleware Extensions

1. class MiddlewareClassName : IMiddleware
2. {
3. public async Task InvokeAsync(HttpContext context,RequestDelegate next)
4. {
5. //before logic
6. await next(context);
7. //after logic
8. }
9. });

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

1. static class ClassName
2. {
3. public static IApplicationBuilder ExtensionMethodName(this IApplicationBuilder app)
4. {
5. return app.UseMiddleware<MiddlewareClassName>();
6. }
7. }

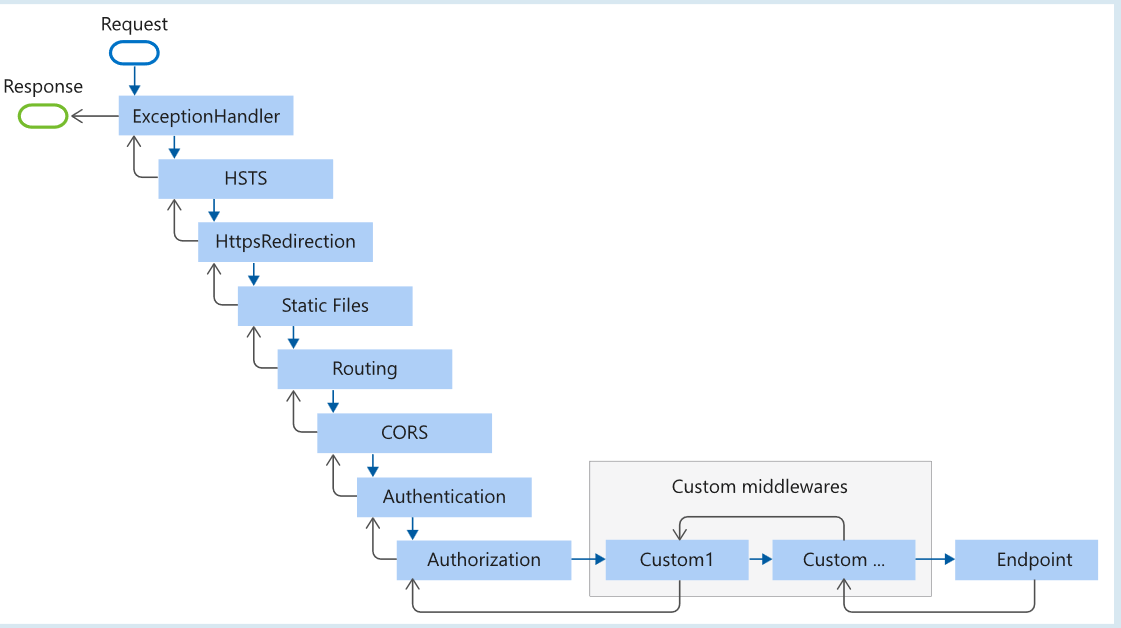
app.ExtensionMethodName();

Conventional Middleware

1. class MiddlewareClassName
2. {
3. private readonly RequestDelegate \_next;
5. public MiddlewareClassName(RequestDelegate next)
6. {
7. \_next = next;
8. }
10. public async Task InvokeAsync(HttpContext context)
11. {
12. //before logic
13. await \_next(context);
14. //after logic
15. }
16. });
17. static class ClassName
18. {
19. public static IApplicationBuilder ExtensionMethodName(this IApplicationBuilder app)
20. {
21. return app.UseMiddleware<MiddlewareClassName>();
22. }
23. }

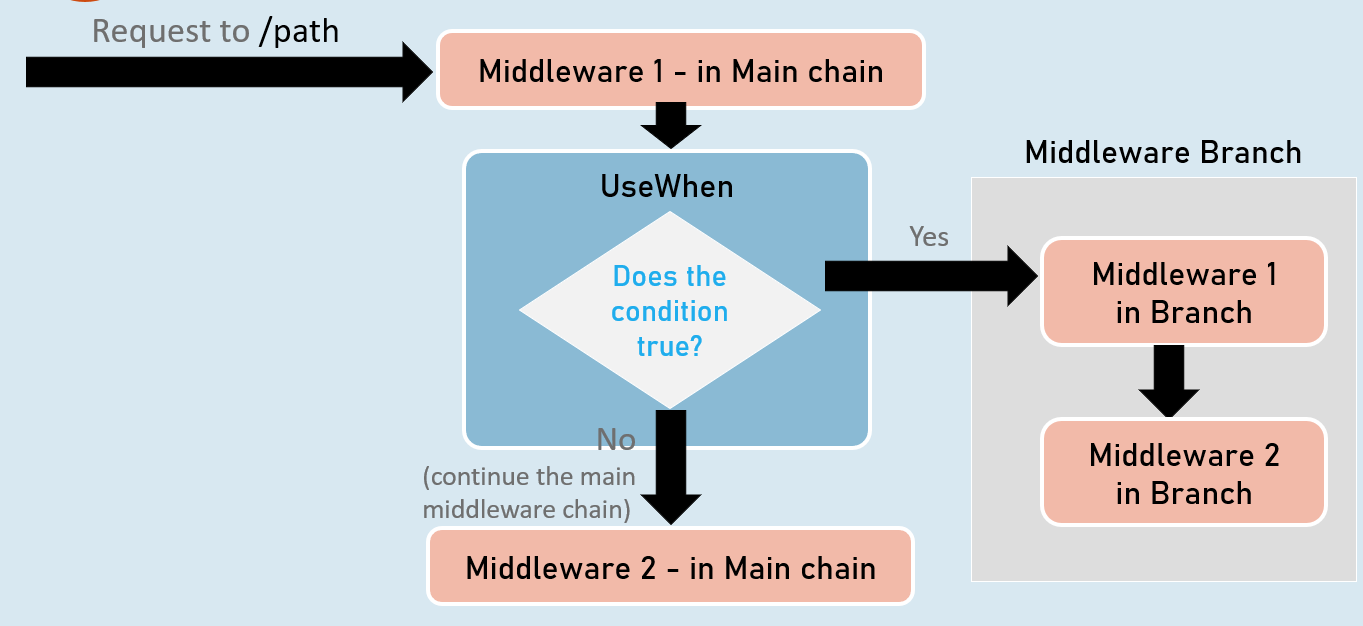
app.ExtensionMethodName();

The Right Order of Middleware



1. app.UseExceptionHandler("/Error");
2. app.UseHsts();
3. app.UseHttpsRedirection();
4. app.UseStaticFiles();
5. app.UseRouting();
6. app.UseCors();
7. app.UseAuthentication();
8. app.UseAuthorization();
9. app.UseSession();
10. app.MapControllers();
11. //add your custom middlewares
12. app.Run();

Middleware - UseWhen



**app.UseWhen( )**

1. app.UseWhen(
2. context => { return boolean; },
3. app =>
4. {
5. //add your middlewares
6. }
7. );

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