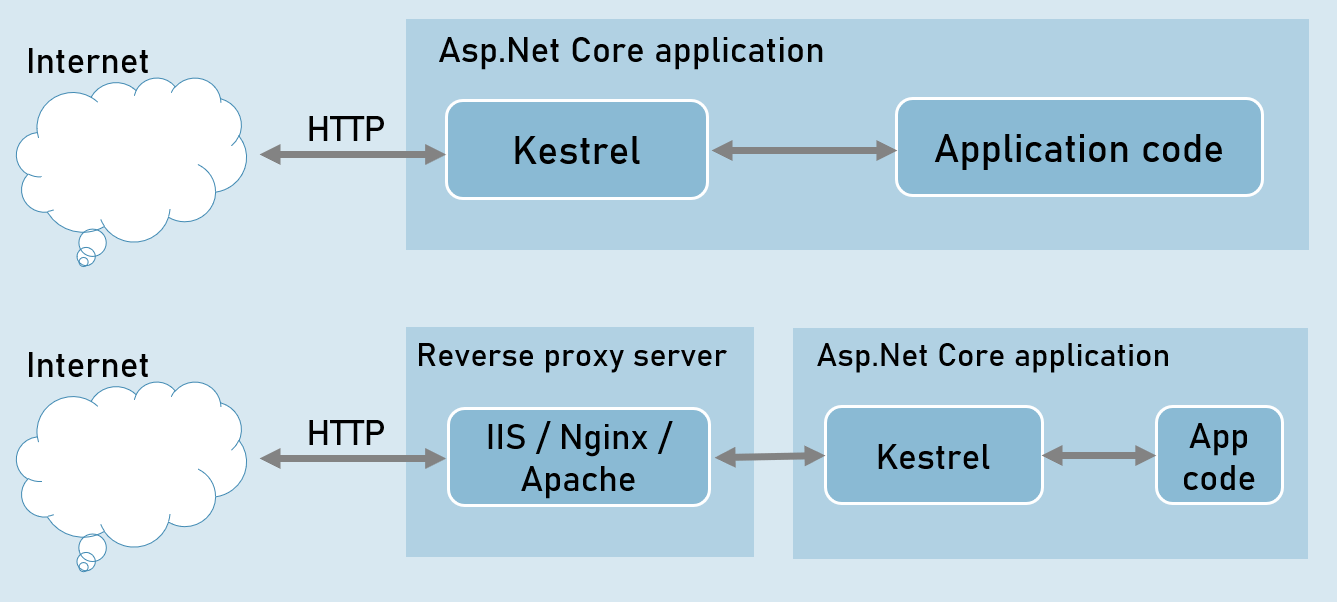
Kestrel and Other Servers

**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

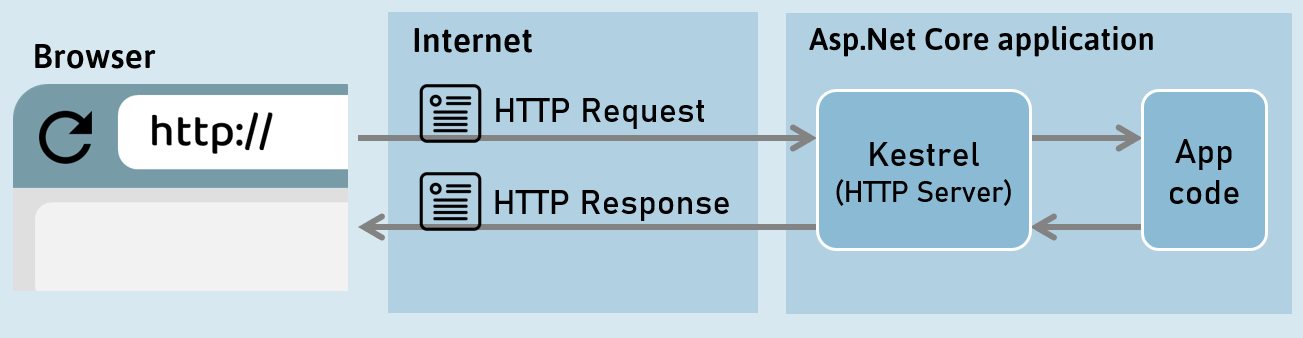
**IIS express**

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

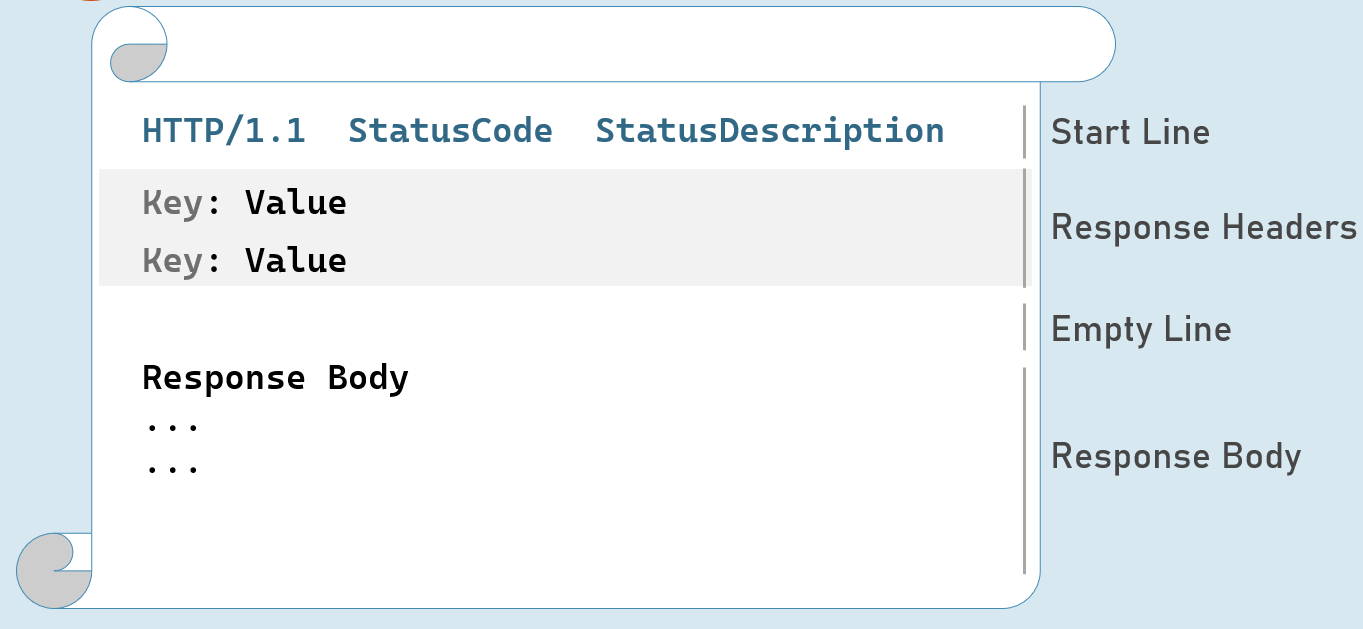
Introduction to HTTP

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 (Http to Https)

**2xx | Success**

200          OK

**3xx | Redirection**

302          Found

304          Not Modified (File not modifies as it in the cache)

**4xx | Client error**

400           Bad Request (Support Course Id not provided, )

401            Unauthorized

404           Not Found (Url 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

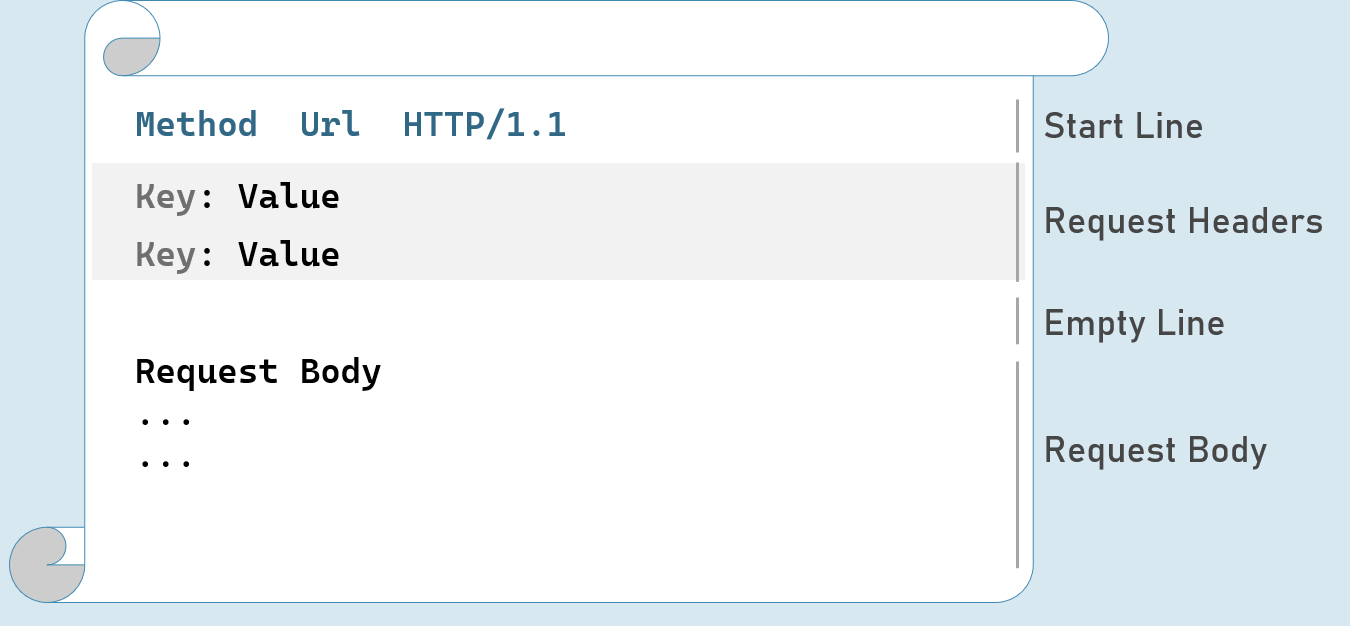
**Location**

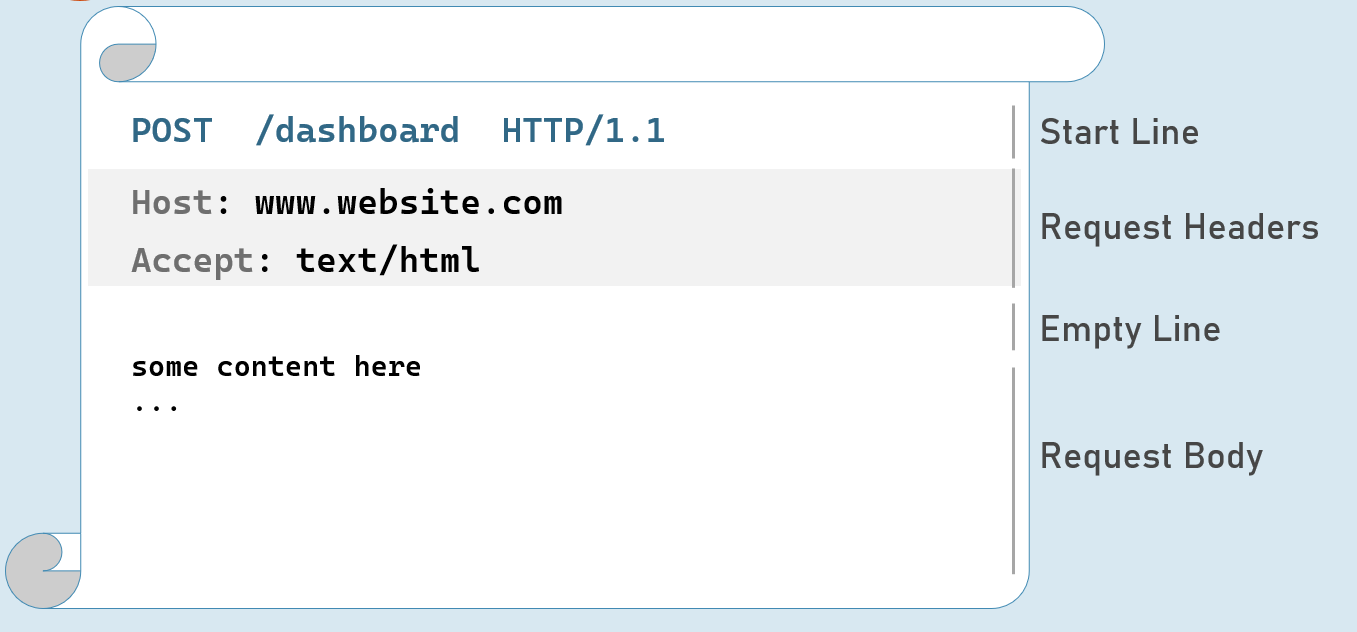
Contains URL to redirect.

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

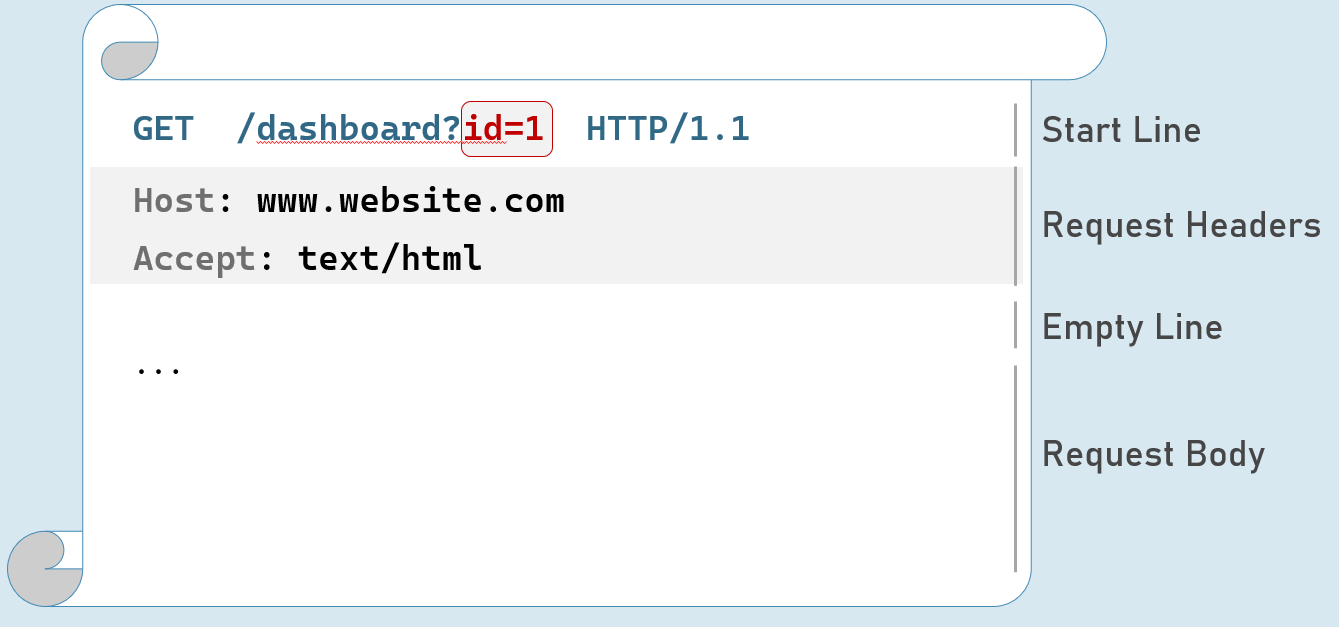
Further reading: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>

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

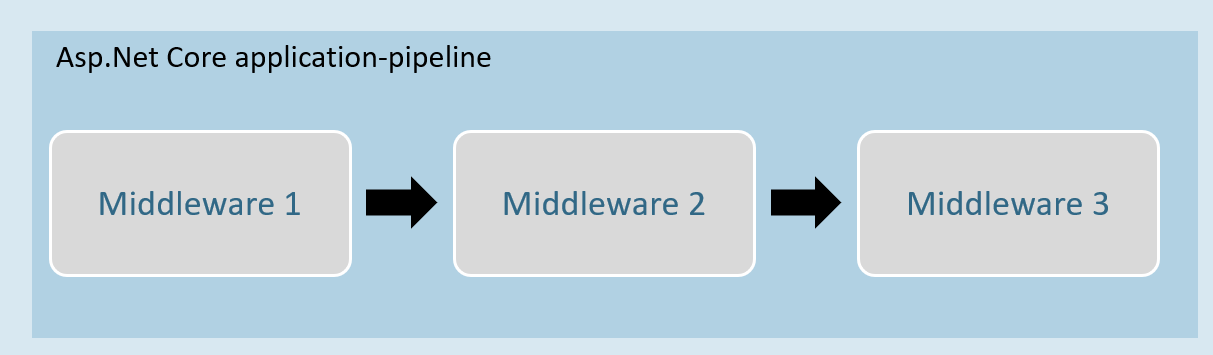
Can't be cached by browsers / search engines.

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( )**

app.Run(async (HttpContext context) =>

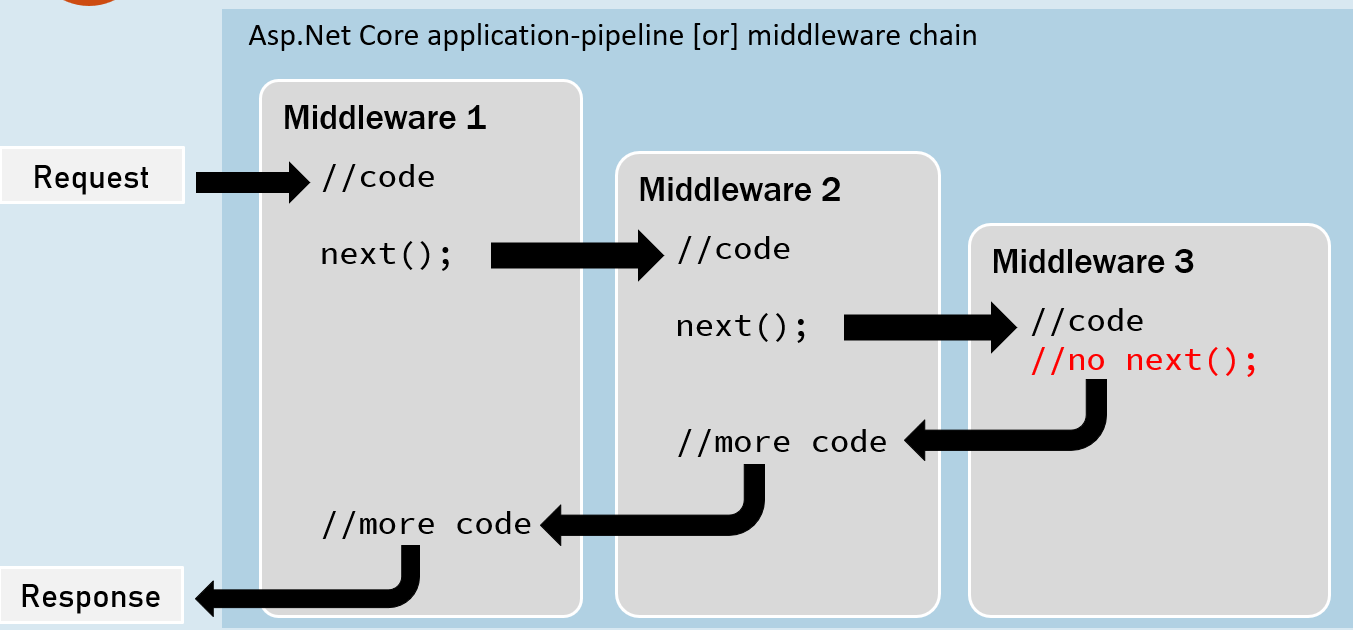
{

//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)

{

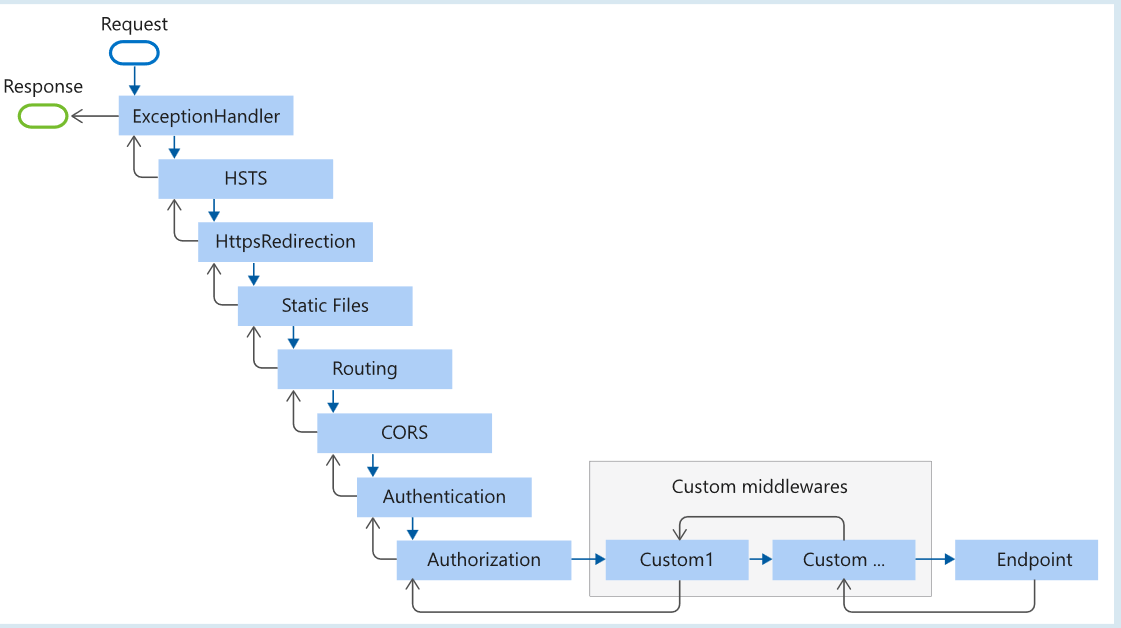
return app.UseMiddleware<MiddlewareClassName>();

}

}

app.ExtensionMethodName();

The Right Order of Middleware



app.UseExceptionHandler("/Error");

app.UseHsts();

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseCors();

app.UseAuthentication();

app.UseAuthorization();

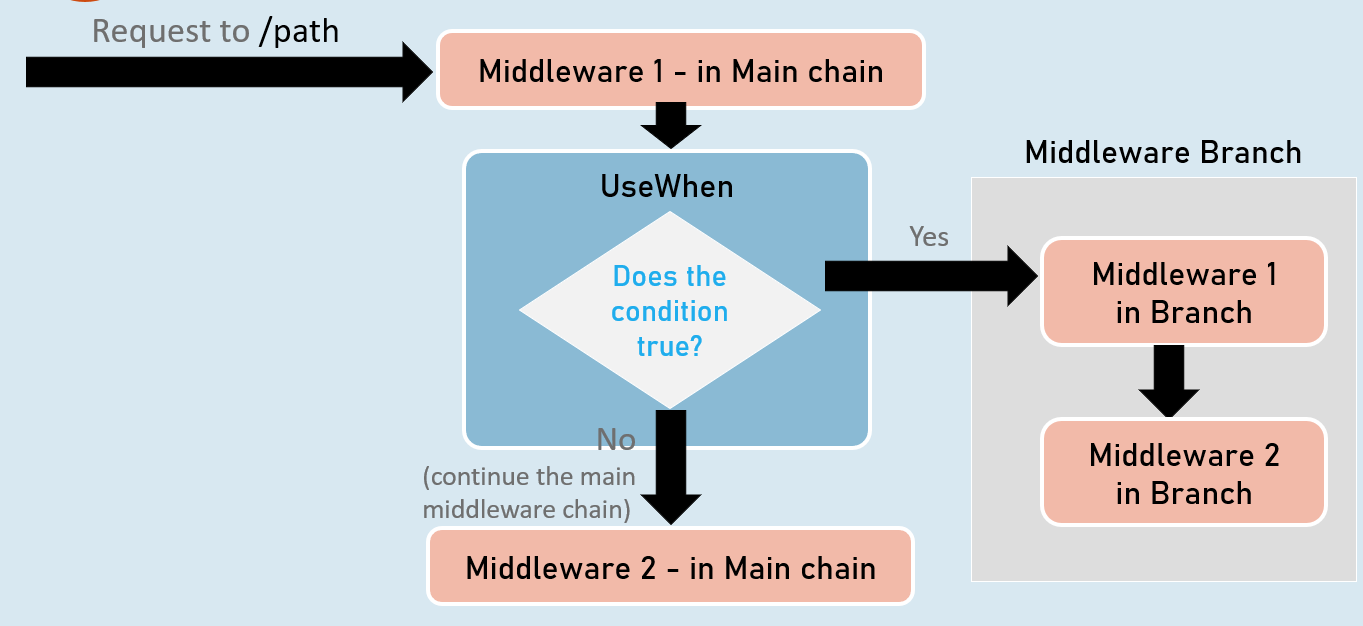
app.UseSession();

app.MapControllers();

//add your custom middlewares

app.Run();

**Middleware - UseWhen**



app.UseWhen( )

app.UseWhen(

context => { return boolean; },

app =>

{

//add your middlewares

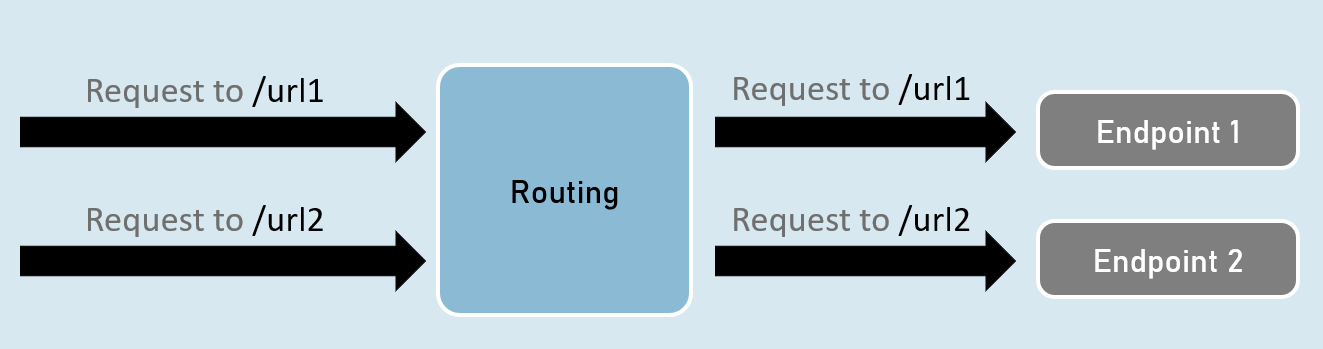
}

);

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

Introduction to Routing

Routing is a process of matching incoming HTTP requests by checking the HTTP method and url; and then invoking corresponding endpoints.



Routing - UseRouting and UseEndPoints

**UseRouting( )**

app.UseRouting();

Enables routing and selects an appropriate end point based on the url path and HTTP method.

**UseEndPoints( )**

app.UseEndPoints(endpoints =>

{

endpoints.Map(…);

endpoints.MapGet(…);

endpoints.MapPost(…);

);

Executes the appropriate endpoint based on the endpoint selected by the above UseRouting() method.

**Map, MapGet, MapPost**

**endpoints.Map( )**

endpoints.Map("path", async (HttpContext context) =>

{

//code

});

Executes the endpoint when a HTTP request's url path begins with the specified path.

**endpoints.MapGet( )**

endpoints.MapGet("path", async (HttpContext context) =>

{

//code

});

Executes the endpoint when a HTTP GET request's url path begins with the specified path.

**endpoints.MapPost( )**

endpoints.MapPost("path", async (HttpContext context) =>

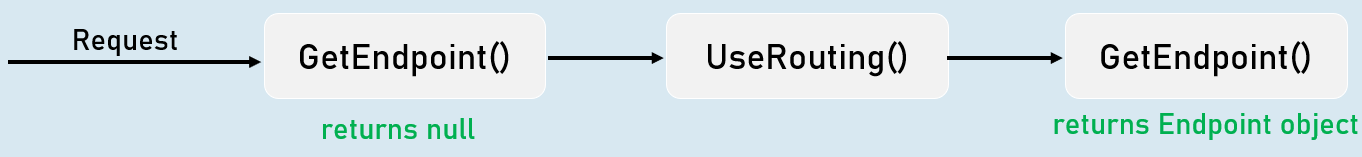
{

//code

});

Executes the endpoint when a HTTP POST request's url path begins with the specified path.

GetEndpoint( )



context.GetEndpoint();

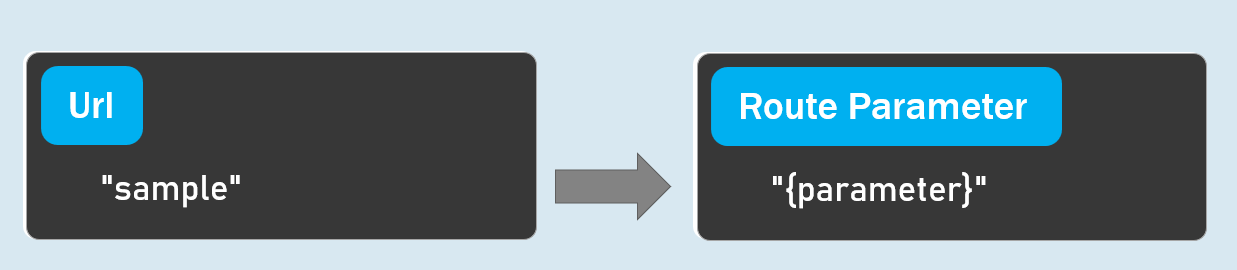
Returns an instance of Microsoft.AspNetCore.Http.Endpoint type, which represents an endpoint.

That instance contains two important properties: DisplayName, RequestDelegate.

Route Parameters

**"{parameter}"**

A route parameter can match with any value.



**Default Route Parameters**

"{parameter=default\_value}"

A route parameter with default value matches with any value.

It also matches with empty value. In this case, the default value will be considered into the parameter.

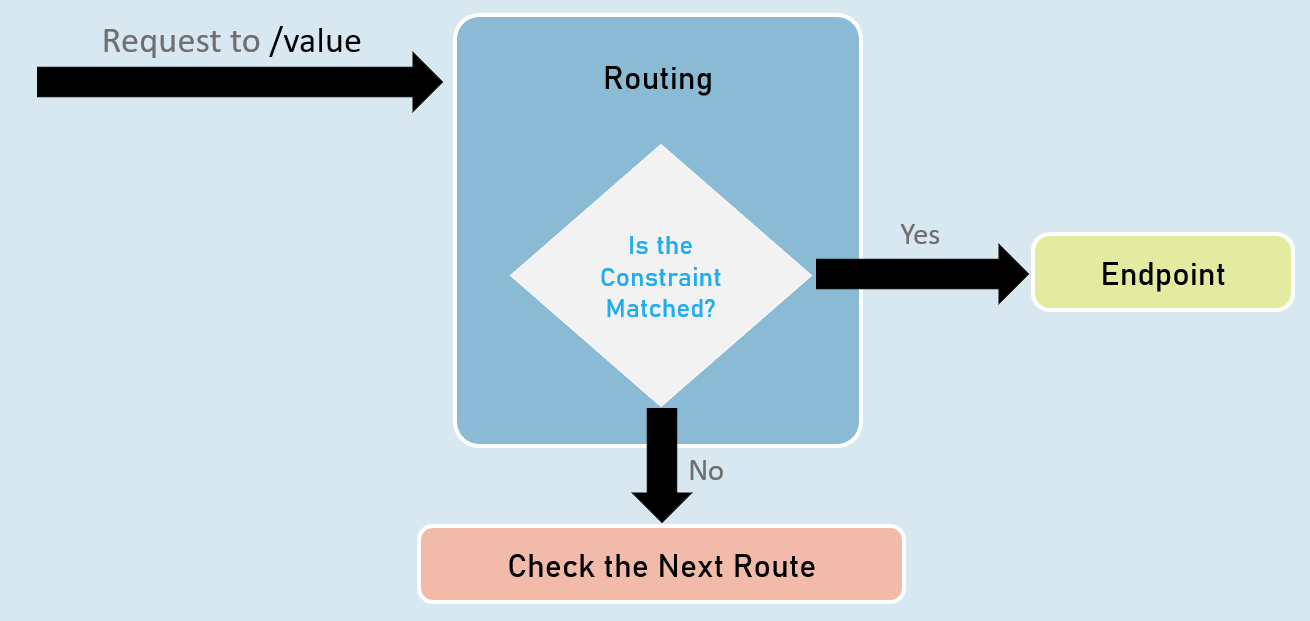
**Optional Route Parameters**

"{parameter?}"

"?" indicates an optional parameter.

That means, it matches with a value or empty value also.

Route Constraints



**Route Parameter with Constraint:**

"{parameter:constraint}"

A route parameter that has a constraint can match with a value that satisfies the given constraint.

**Multiple Constraints**

"{parameter:constraint1:constraint2}"

A route parameter can have more than one constraint, separated with colon ( : ).

**int**

Matches with any integer.

Eg: {id:int} matches with 123456789, -123456789

**bool**

Matches with true or false. Case-insensitive.

Eg: {active:bool} matches with true, false, TRUE, FALSE

**datetime**

Matches a valid DateTime value with formats "yyyy-MM-dd hh:mm:ss tt" and "MM/dd/yyyy hh:mm:ss tt".

Eg: {id:datetime} matches with 2030-01-01%2011:59%20pm

Note: '%20' is equal to space.

**decimal**

Matches with a valid decimal value.

Eg: {price:decimal} matches with 49.99, -1, 0.01

**long**

Matches a valid long value.

Eg: {id:long} matches with 123456789, -123456789

**guid**

Matches with a valid Guid value (Globally Unique Identifier - A hexadecimal number that is universally unique).

Eg: {id:guid} matches with 123E4567-E89B-12D3-A456-426652340000

**minlength(value)**

Matches with a string that has at least specified number of characters.

Eg: {username:minlength(4)} matches with John, Allen, William

**maxlength(value)**

Matches with a string that has less than or equal to the specified number of characters.

Eg: {username:maxlength(7)} matches with John, Allen, William

**length(min,max)**

Matches with a string that has number of characters between given minimum and maximum length (both numbers including).

Eg: {username:length(4, 7)} matches with John, Allen, William

**length(value)**

Matches with a string that has exactly specified number of characters.

Eg: {tin:length(9)} matches with 987654321

**min(value)**

Matches with an integer value greater than or equal to the specified value.

Eg: {age:min(18)} matches with 18, 19, 100

**max(value)**

Matches with an integer value less than or equal to the specified value.

Eg: {age:max(100)} matches with -1, 1, 18, 100

**range(min,max)**

Matches with an integer value between the specified minimum and maximum values (both numbers including).

Eg: {age:range(18,100)} matches with 18, 19, 99, 100

**alpha**

Matches with a string that contains only alphabets (A-Z) and (a-z).

Eg: {username:alpha} matches with rick, william

**regex(expression)**

Matches with a string that matches with the specified regular expression.

Eg 1: {age:regex(^[0-9]{2}$)} matches with any two-digit number, such as 10, 11, 98, 99

Eg 2: {age:regex(^\d{3}-\d{3}$)} matches with any three-digit number, then hyphen, and then three-digit number, such as 123-456

Custom Route Constraint Classes

Custom Route Constraint Class

public class ClassName : IRouteConstraint

{

public bool Match(HttpContext? HttpContext, IRouter? route, string routeKey, RouteValueDictionary values, RouteDirection routeDirection)

{

//return true or false

}

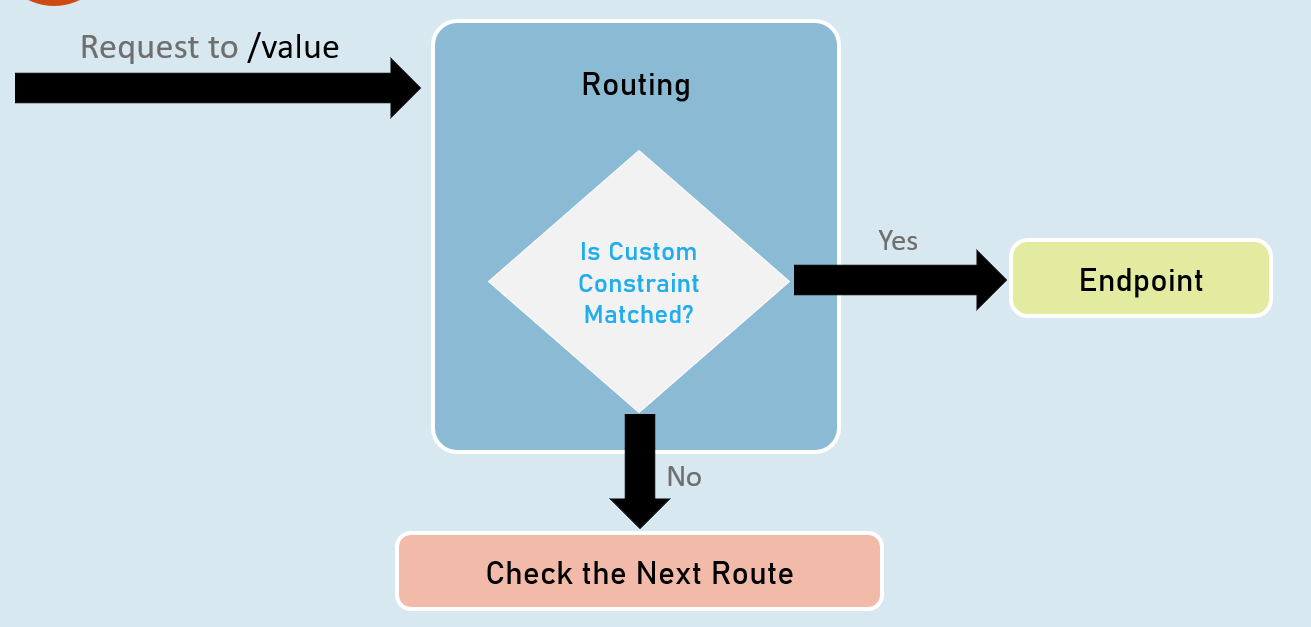
}

builder.Services.AddRouting(options =>

{

options.ConstraintMap.Add("name", typeof(ClassName));

}); //adding the custom constraint to routing



Endpoint Selection Order

Top is highest precedence (will be evaluated first)

**1:**URL template with more segments.

Eg: "a/b/c/d" is higher than "a/b/c".

**2:**URL template with literal text has more precedence than a parameter segment.

Eg: "a/b" is higher than "a/{parameter}".

**3:**URL template that has a parameter segment with constraints has more precedence than a parameter segment without constraints.

Eg: "a/b:int" is higher than "a/b".

**4:**Catch-all parameters (\*\*).

Eg: "a/{b}" is higher than "a/\*\*".

WebRoot

