ASP.NET CORE CHEATSHEET

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

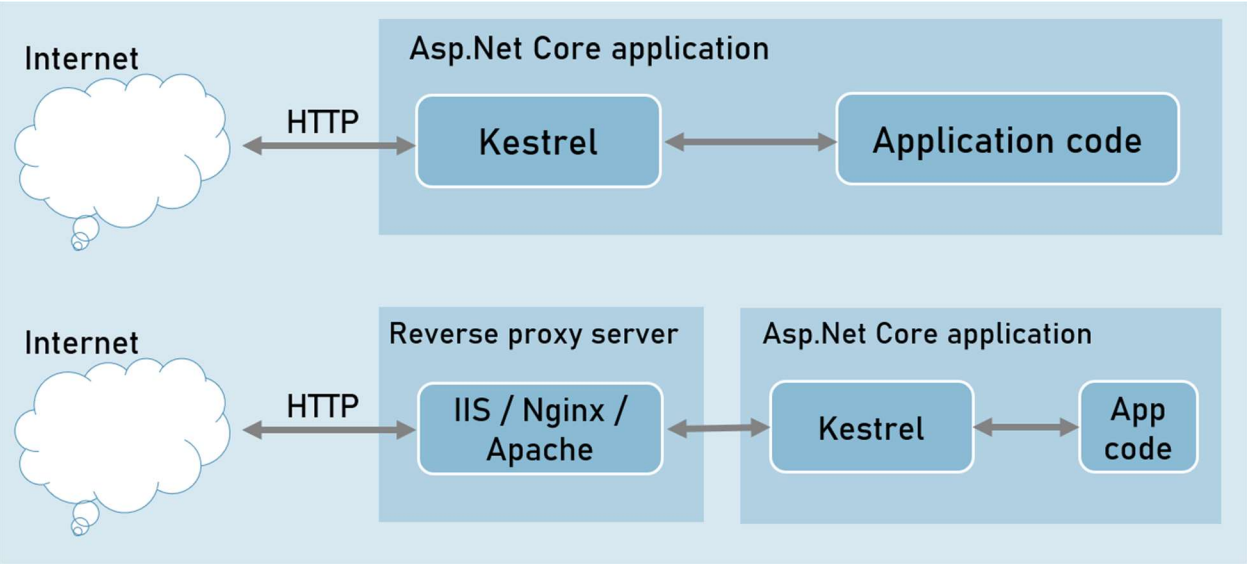
Kestrel and Other Servers:

Application Servers

- Kestrel

Reverse Proxy Servers

- IIS
- Nginx
- Apache



Benefits of Reverse Proxy Servers

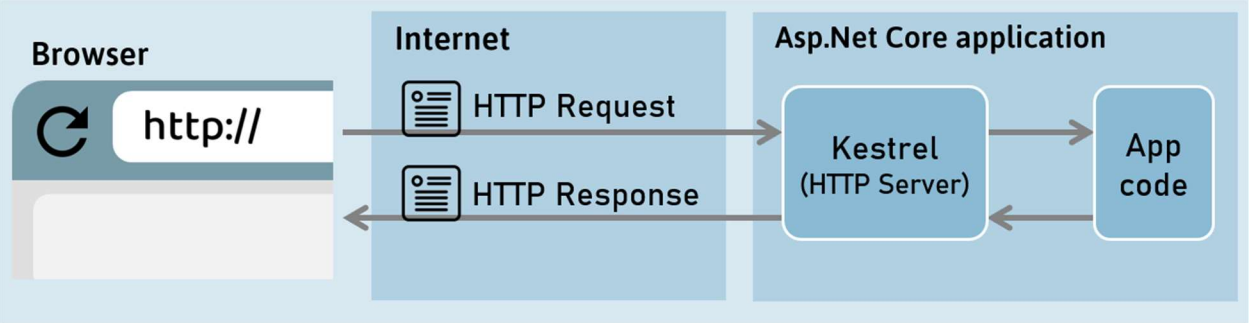
- Load Balancing
- Caching
- URL Rewriting
- Decompressing the requests
- Authentication
- Decryption of SSL Certificates

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

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

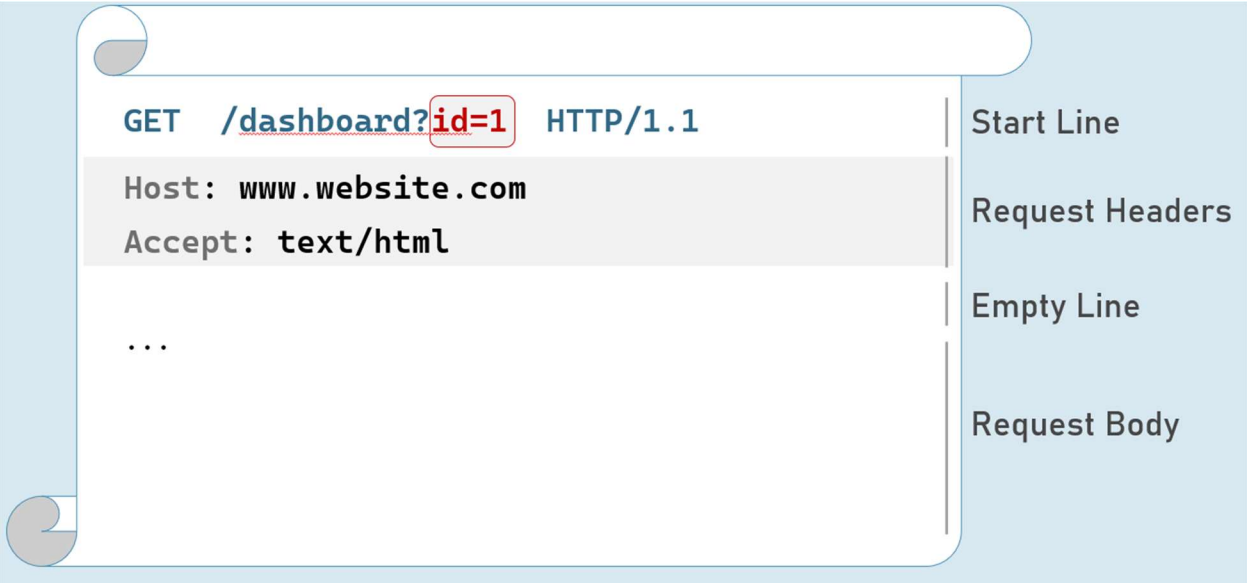
Contains url to redirect.

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

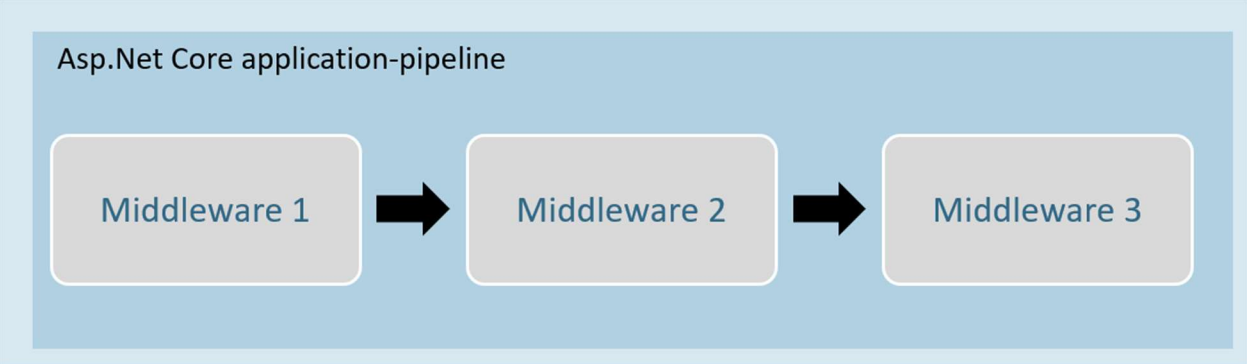
Get:

- 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.

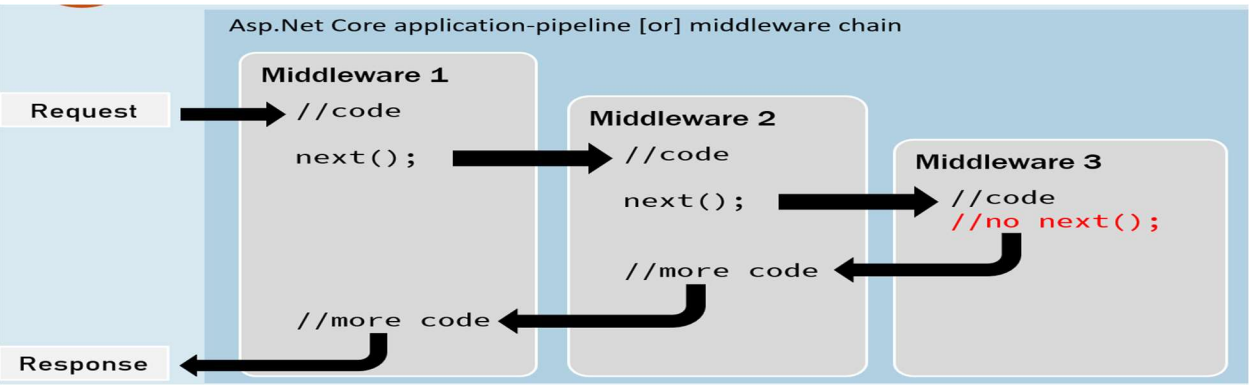
Middleware - Run

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( )

```
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();
```

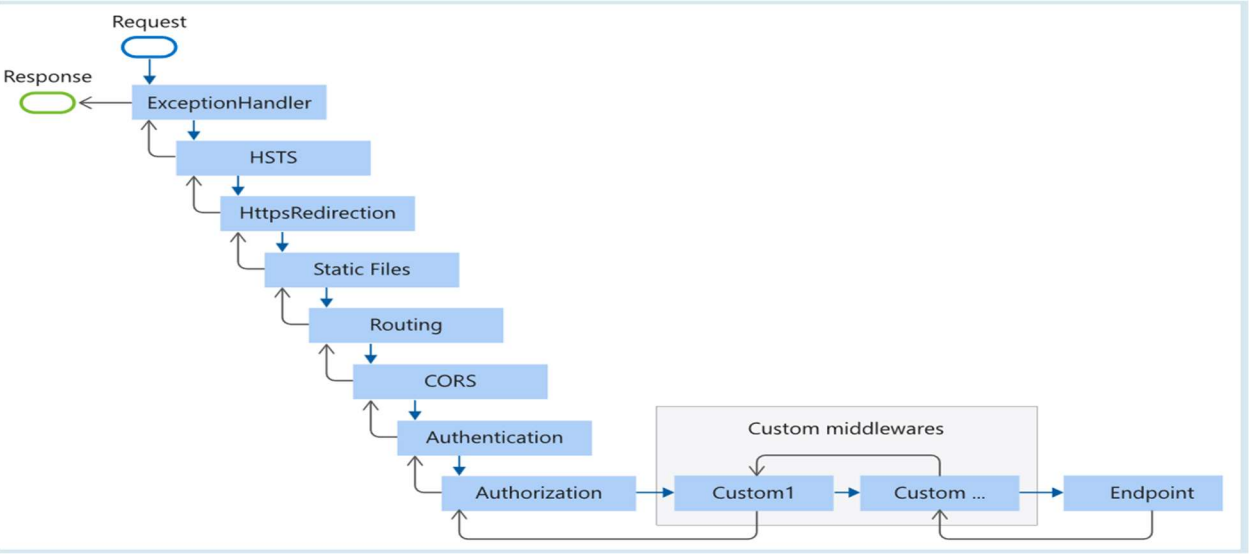
Conventional Middleware:

```
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>();
  }
}
```

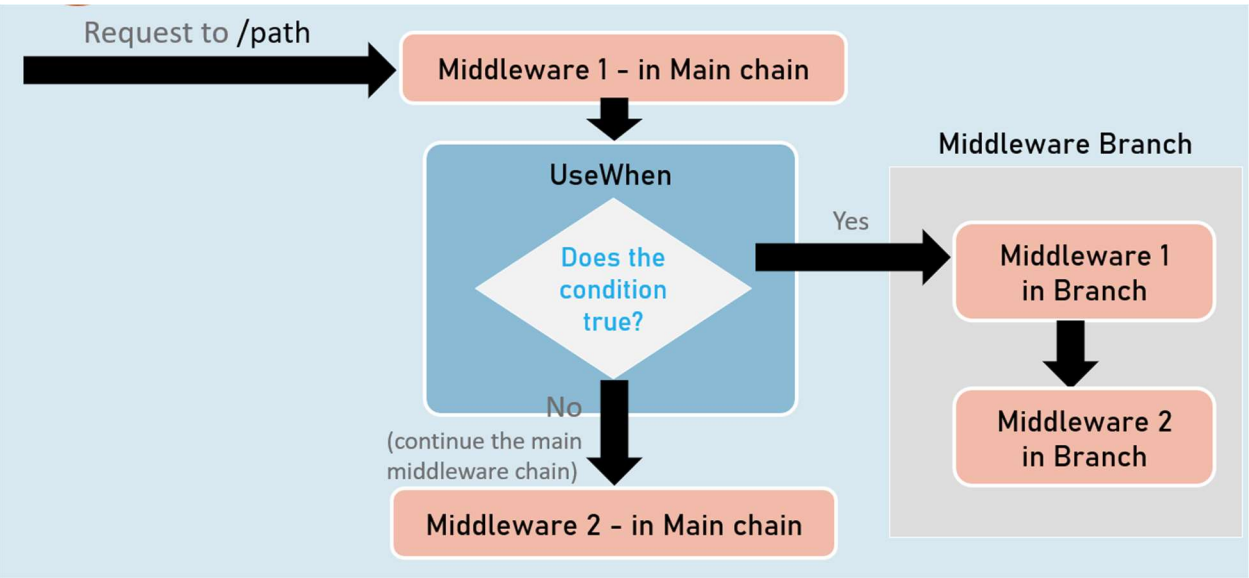
```
app.ExtensionMethodName();
```

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



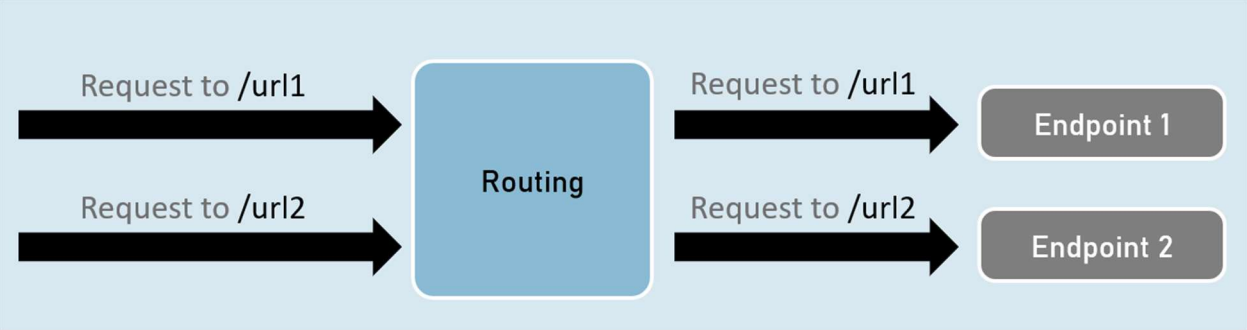
app.UseWhen( )

```
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( )

```
1. app.UseEndPoints(endpoints =>
2. {
3.  endpoints.Map(…);
4.  endpoints.MapGet(…);
5.  endpoints.MapPost(…);
6. );
```

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

Map, MapGet, MapPost

endpoints.Map( )

```
1. endpoints.Map("path", async (HttpContext context) =>
2. {
3.  //code
4. });
```

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

endpoints.MapGet( )

```
1. endpoints.MapGet("path", async (HttpContext context) =>
2. {
3.  //code
4. });
```

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

endpoints.MapPost( )

```
1. endpoints.MapPost("path", async (HttpContext context) =>
2. {
3.  //code
4. });
```

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

GetEndpoint( )


```
context.GetEndpoint();
```

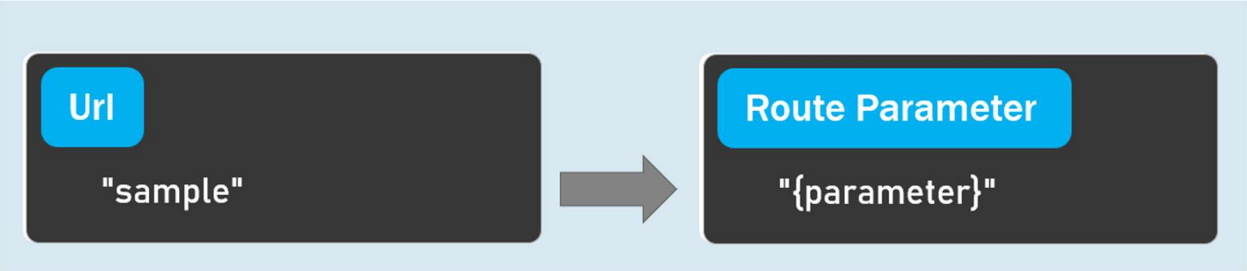
Returns an instance of Microsoft.AspNetCore.Http.Endpoint type, which represents an endpoint.

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.

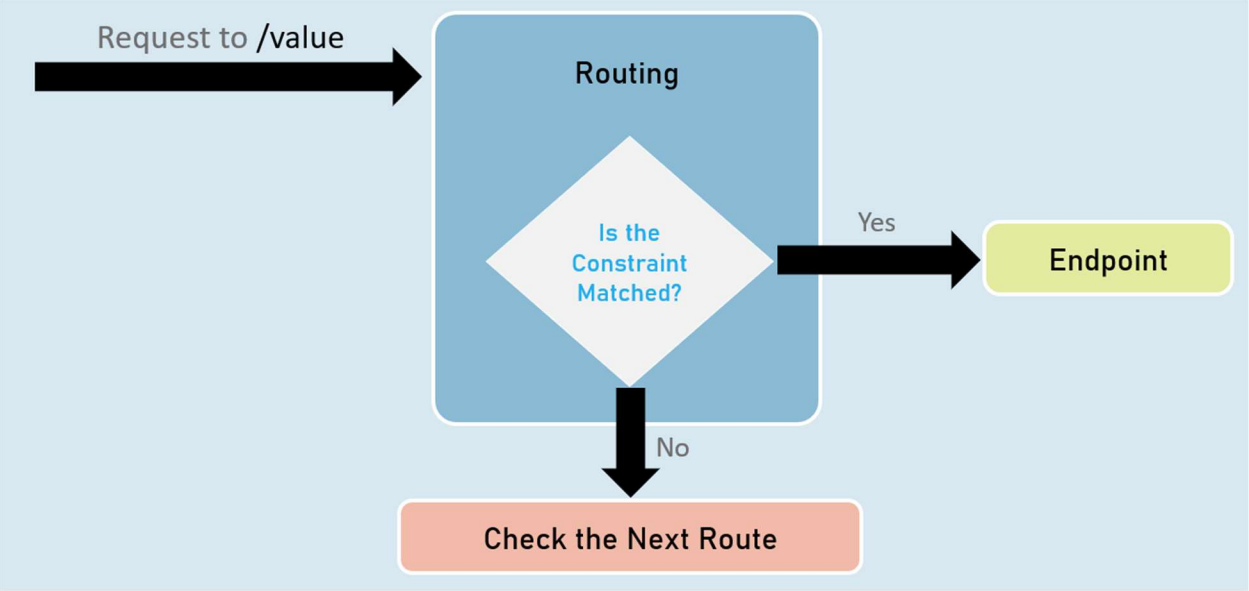
Optional Route Parameters

"{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

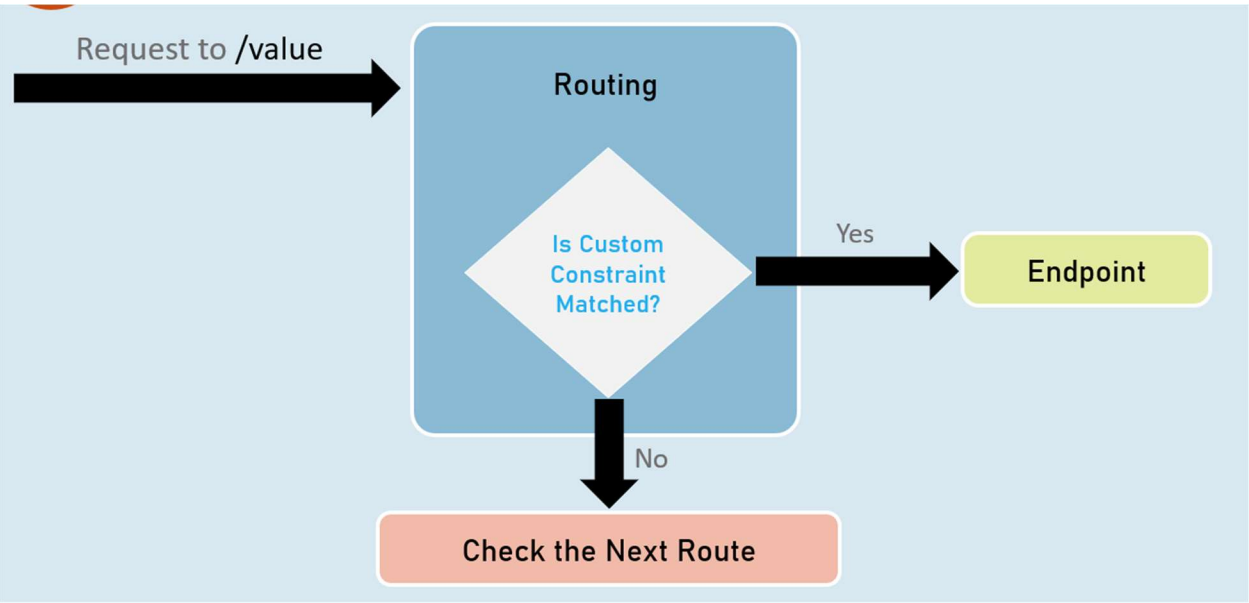
Eg 2: {age:regex(^\d{3}-\d{3}$)} matches with any three-digit number, then hyphen, and then three-digit number, such as 123-456

Custom Route Constraint Classes:

Custom Route Constraint Class

```
1. public class ClassName : IRouteConstraint
2. {
3.   public bool Match(HttpContext? HttpContext, IRouter? route, string routeKey,
   RouteValueDictionary values, RouteDirection routeDirection)
4.   {
5.     //return true or false
6.   }
7. }
```

```
1. builder.Services.AddRouting(options =>
2. {
3.   options.ConstraintMap.Add("name", typeof(ClassName));
4. }); //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.

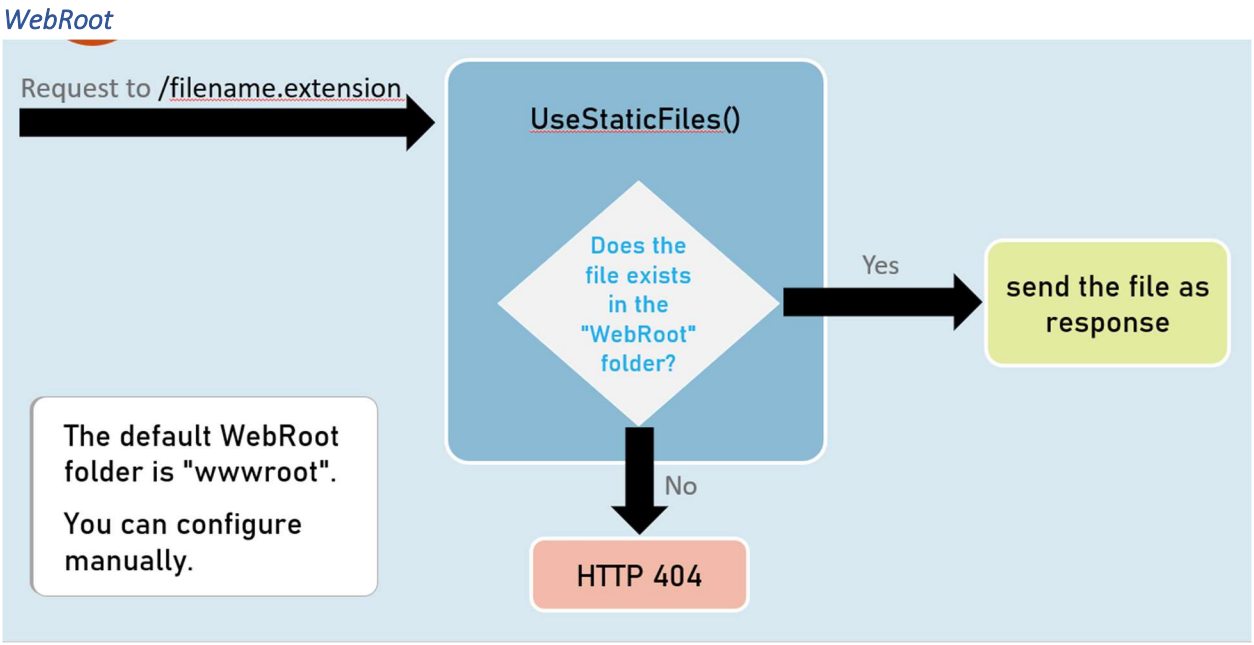
Eg: "a/b" is higher than "a/{parameter}".

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/**".



Introduction to Controllers:

Controller is a class that is used to group-up a set of actions (*or action methods*). Action methods do perform certain operation when a request is received & returns the result (response).

Creating Controllers:

Controllers should be either or both:

- The class name should be suffixed with "Controller". Eg: HomeController
- The [Controller] attribute is applied to the same class or to its base class.

Controller

```
1. [Controller]
2. class ClassNameController
3. {
4.   //action methods here
5. }
```

Optional:

- Is a public class.
- Inherited from Microsoft.AspNetCore.Mvc.Controller.

Enable 'routing' in controllers:

AddControllers( )

```
builder.Services.AddControllers();
```

Adds all controllers as services in the IServiceCollection.

So that, they can be accesed when a specific endpoint needs it.

MapControllers()

```
app.MapControllers();
```

Adds all action methods as endpoints.

So that, no need of using UseEndPoints() method for adding action methods as end points.

Responsibilities of Controllers:

Reading requests

Extracting data values from request such as query string parameters, request body, request cookies, request headers etc.

Invoking models

Calling business logic methods.

Generally business operations are available as 'services'.

Validation

Validate incoming request details (query string parameters, request body, request cookies, request headers etc.)

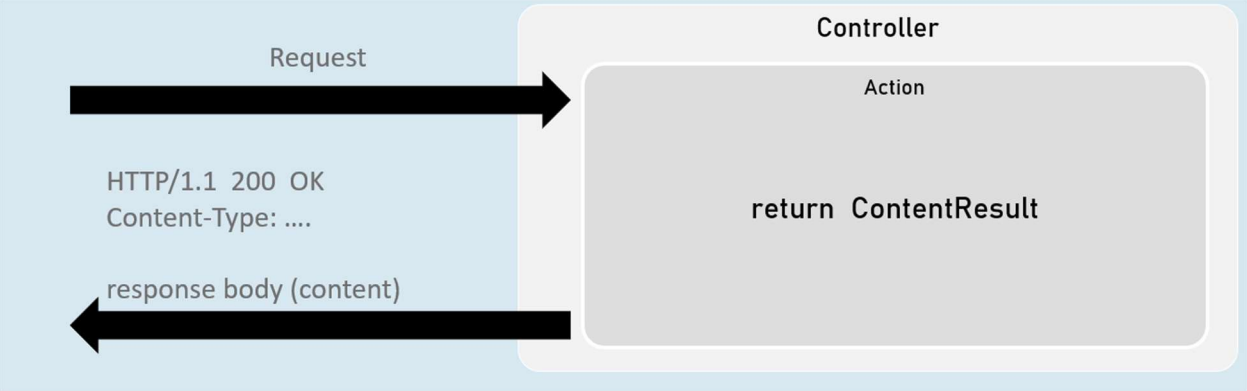
Preparing Response

Choosing what kind of response has to be sent to the client & also preparing the response (*action result*).

ContentResult

ContentResult can represent any type of response, based on the specified MIME type.

MIME type represents type of the content such as text/plain, text/html, application/json, application, xml, application/pdf etc.



```
return new ContentResult() { Content = "content", ContentType = "content type" };
```

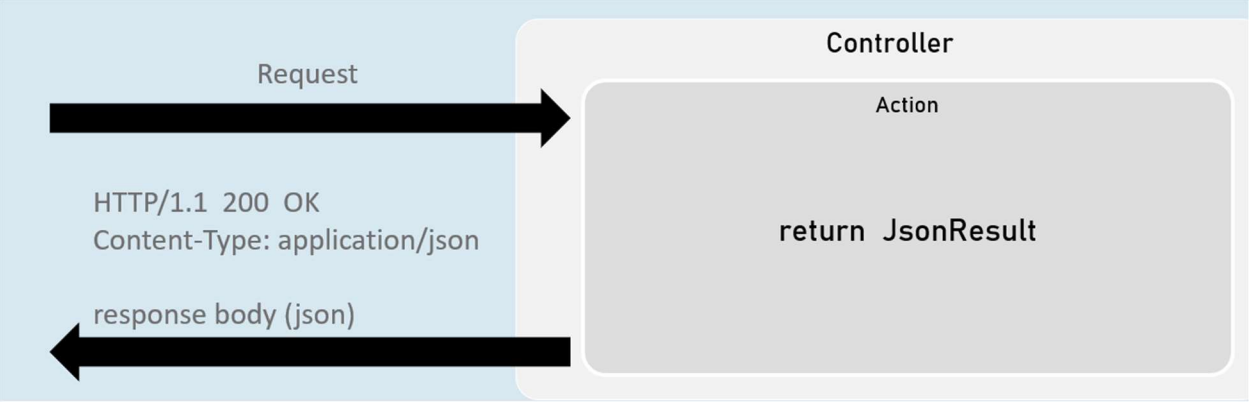
[or]

```
return Content("content", "content type");
```

JsonResult

JsonResult can represent an object in JavaScript Object Notation (JSON) format.

Eg: `{ "firstName": "James", "lastName": "Smith", "age": 25 }`




```
return new JsonResult(your_object);
```

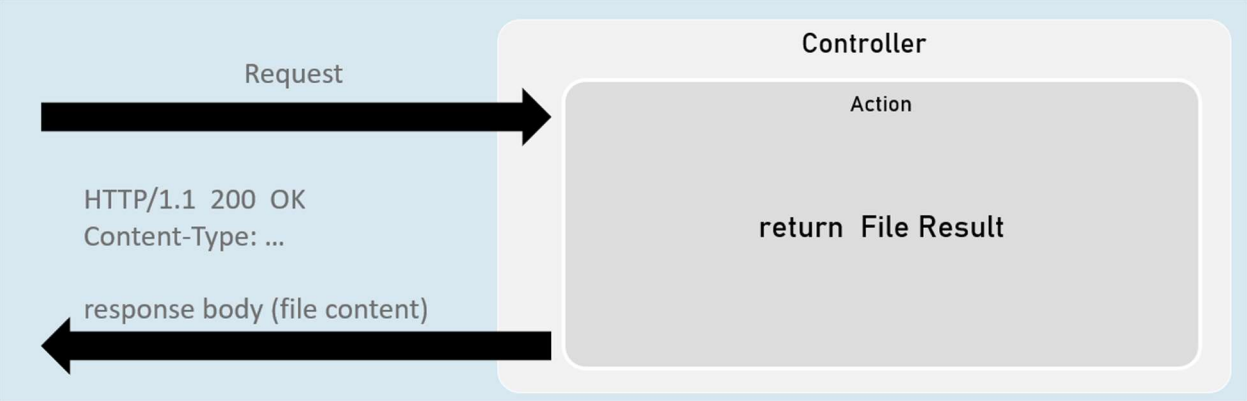
[or]

```
return Json(your_object);
```

File Results

File result sends the content of a file as response.

Eg: pdf file, txt file, exe file, zip file etc.



VirtualFileResult

```
return new VirtualFileResult("file relative path", "content type");
```

//or

```
return File("file relative path", "content type");
```

Represents a file within the WebRoot ('wwwroot' by default) folder.

Used when the file is present in the WebRoot folder.

PhysicalFileResult

Represents a file that is not necessarily part of the project folder.

Used when the file is present outside the WebRoot folder.

```
return new PhysicalFileResult("file absolute path", "content type");
```

//or

```
return PhysicalFile("file absolute path", "content type");
```

FileContentResult

Represents a file from the byte[ ].

Used when a part of the file or byte[ ] from other data source has to be sent as response.

```
return new FileContentResult(byte_array, "content type");
```

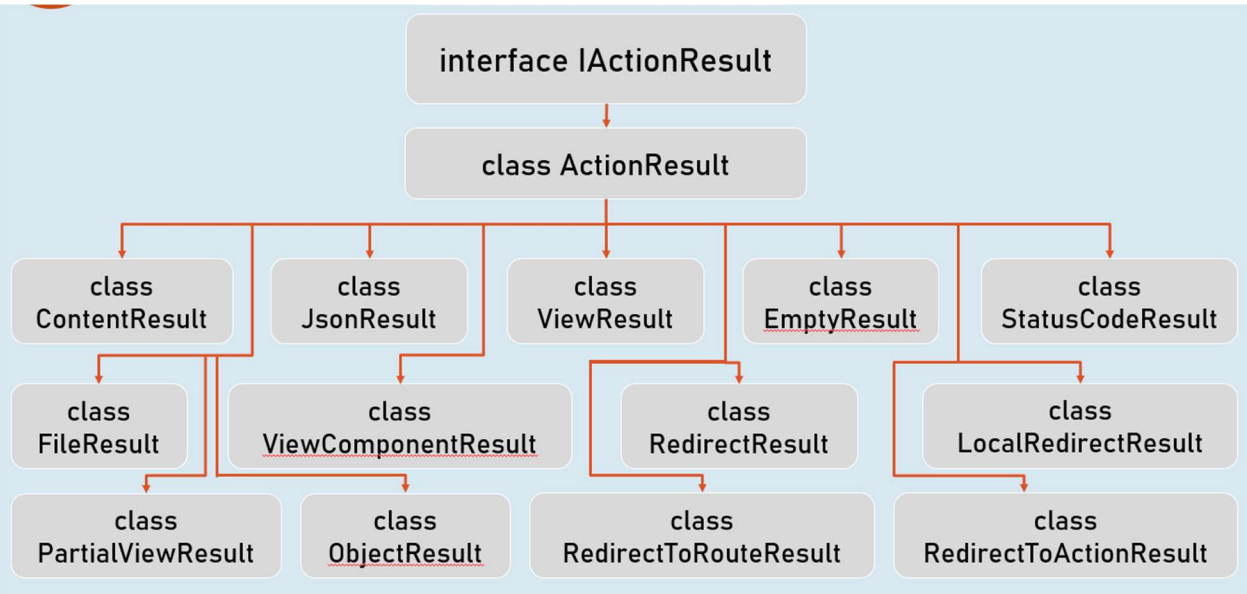
//or

```
return File(byte_array, "content type");
```

IActionResult

It is the parent interface for all action result classes such as ContentResult, JsonResult, RedirectResult, StatusCodeResult, ViewResult etc.

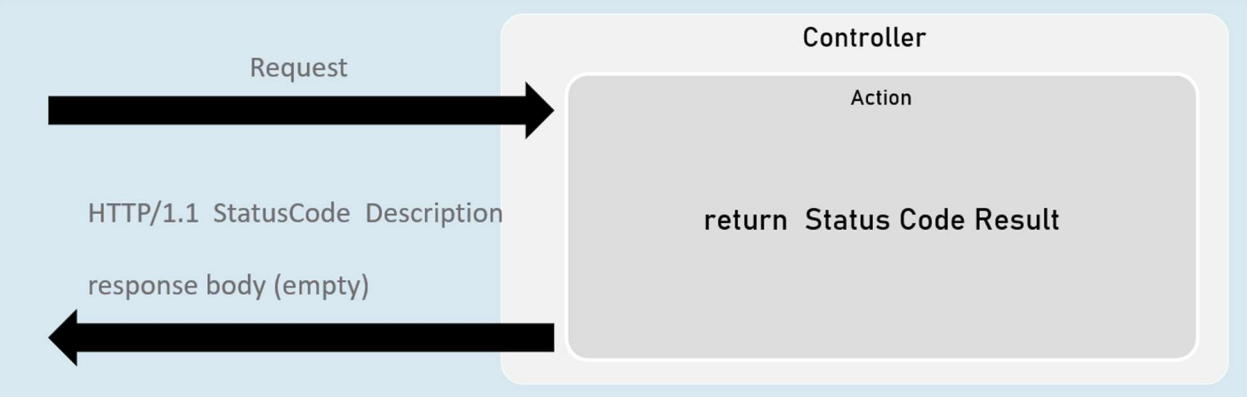
By mentioning the return type as IActionResult, you can return either of the subtypes of IActionResult



Status Code Results

Status code result sends an empty response with specified status code.

Eg: 200, 400, 401, 404, 500 etc.



StatusCodeResult

```
return new StatusCodeResult(status_code);
```

UnauthorizedResult

```
return new UnauthorizedResult();
```

BadRequestResult

```
return new BadRequestResult();
```

NotFoundResult

```
return new NotFoundResult();
```

StatusCodeResult

- Represents response with the specified status code.
- Used when you would like to send a specific HTTP status code as response.

```
return new StatusCodeResult(status_code);
```

//or

```
return StatusCode(status_code);
```

UnauthoriziedResult

- Represents response with HTTP status code '401 Unauthorized'.
- Used when the user is unauthorized (not signed in).

```
return new UnauthorizedResult();
```

//or

```
return Unauthorized();
```

BadRequestResult

- Represents response with HTTP status code '400 Bad Request'.
- Used when the request values are invalid (validation error).

```
return new BadRequestResult();
```

//or

```
return BadRequest();
```

NotFoundResult

- Represents response with HTTP status code '404 Not Found'.
- Used when the requested information is not available at server.

```
return new NotFoundResult();
```

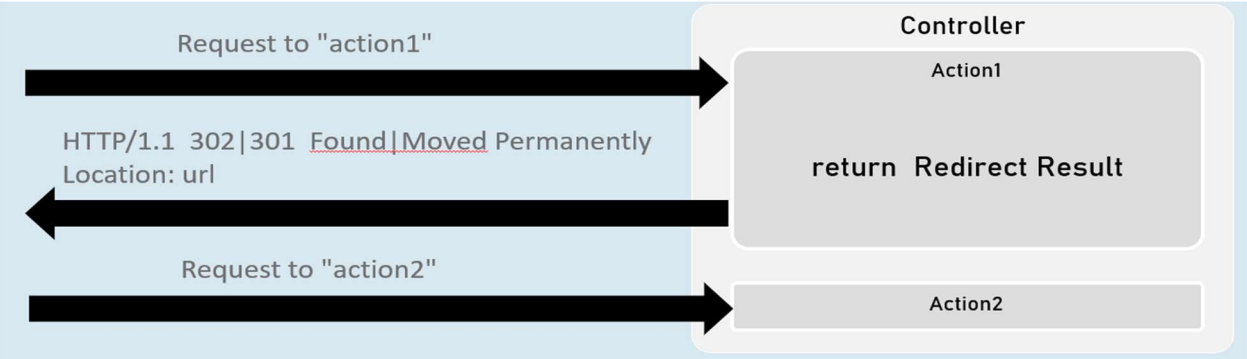
//or

```
return NotFound();
```

Redirect Results

Redirect result sends either HTTP 302 or 301 response to the browser, in order to redirect to a specific action or url.

Eg: redirecting from 'action1' to 'action2'.



RedirectToActionResult

```
return new RedirectToActionResult("action", "controller", new { route_values }, permanent);
```

LocalRedirectResult

```
return new LocalRedirectResult("local_url", permanent);
```

RedirectResult

```
return new RedirectResult("url", permanent);
```

RedirectToActionResult

Represents response for redirecting from the current action method to another action method, based on action name and controller name.

302 - Found

```
return new RedirectToActionResult("action", "controller", new { route_values });
```

//or

```
return RedirectToAction("action", "controller", new { route_values });
```

301 - Moved Permanently

```
return new RedirectToActionResult("action", "controller", new { route_values }, true);
```

//or

```
return RedirectToActionPermanent("action", "controller", new { route_values });
```

LocalRedirectResult

•Represents response for redirecting from the current action method to another action method, based on the specified url.

302 - Found

```
return new LocalRedirectResult("url");
```

//or

```
return LocalRedirect("url);
```

301 - Moved Permanently

```
return new LocalRedirectResult("url", true);
```

//or

```
return LocalRedirectPermanent("url");
```

RedirectResult

Represents response for redirecting from the current action method to any other url (either within the same web application or other web application).

302 - Found

```
return new RedirectResult("url");
```

//or

```
return Redirect("url);
```

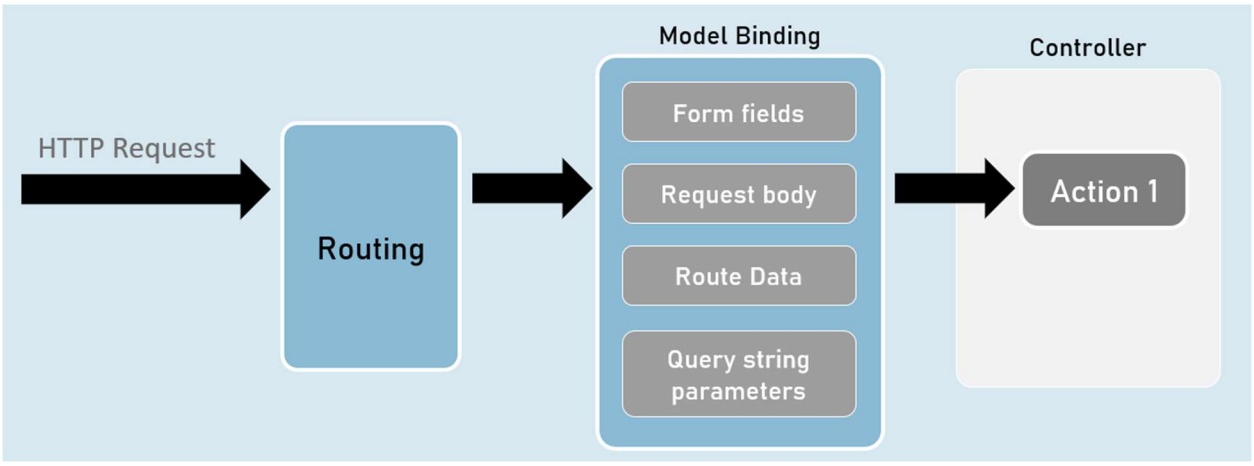
301 - Moved Permanently

```
return new RedirectResult("url", true);
```

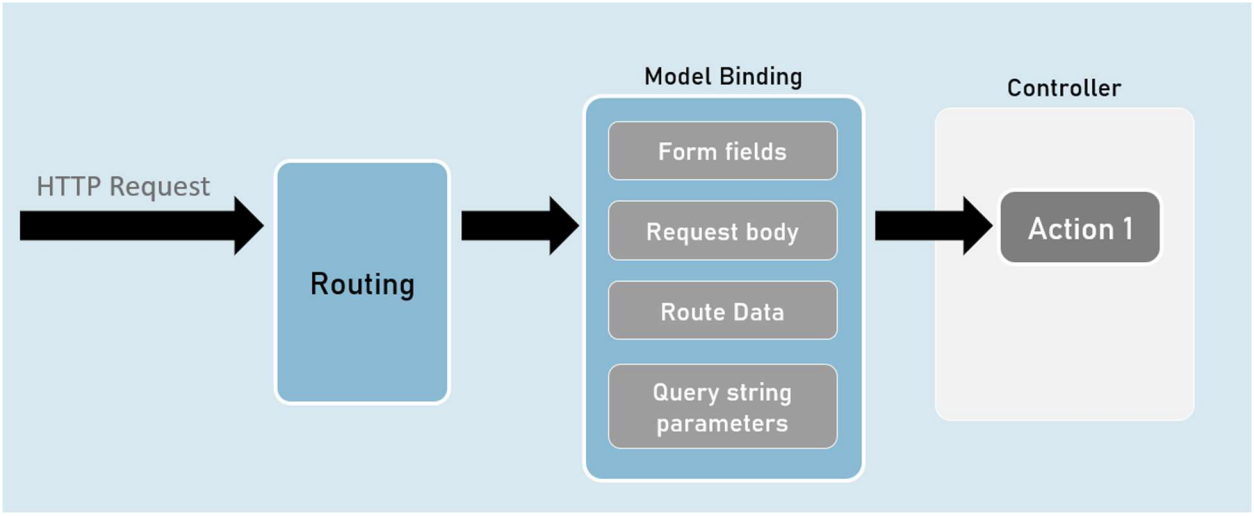
//or

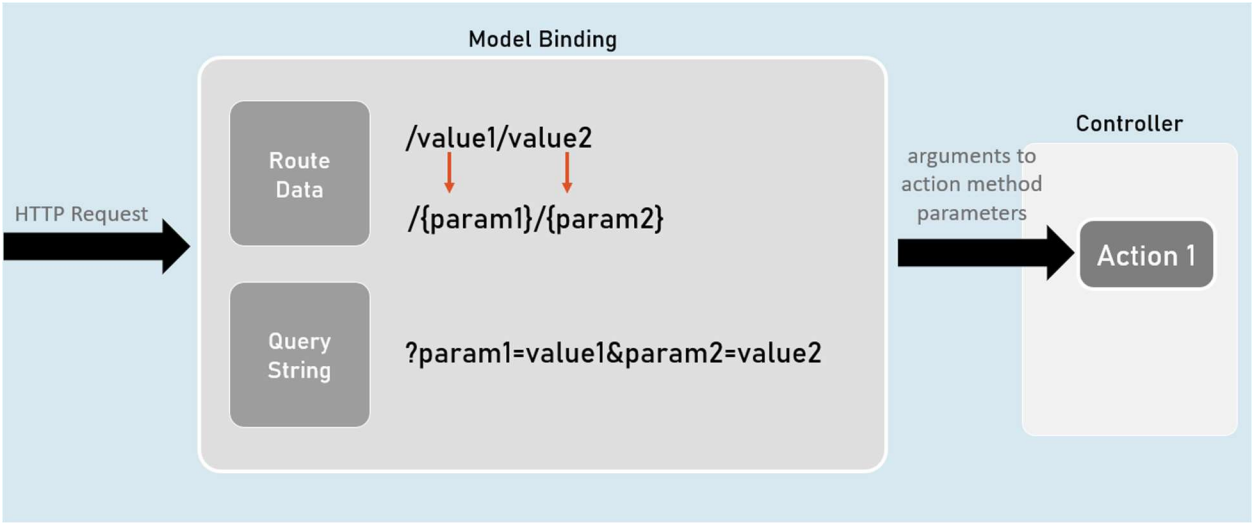
```
return RedirectPermanent("url");
```

Model Binding:
Model Binding is a feature of asp.net core that reads values from http requests and pass them as arguments to the action method.

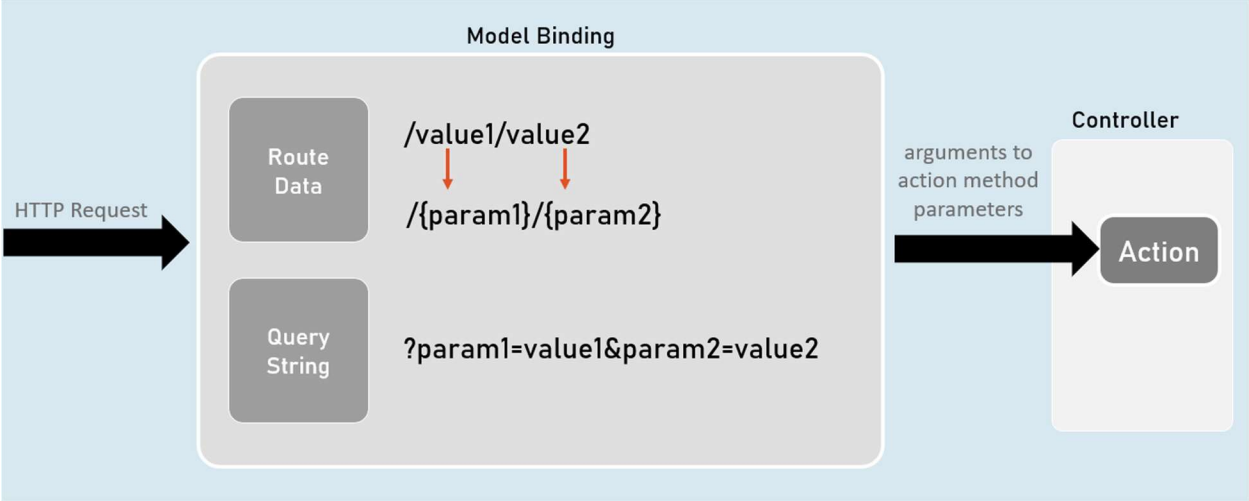


QueryString vs RouteData





[FromQuery] and [FromRoute]

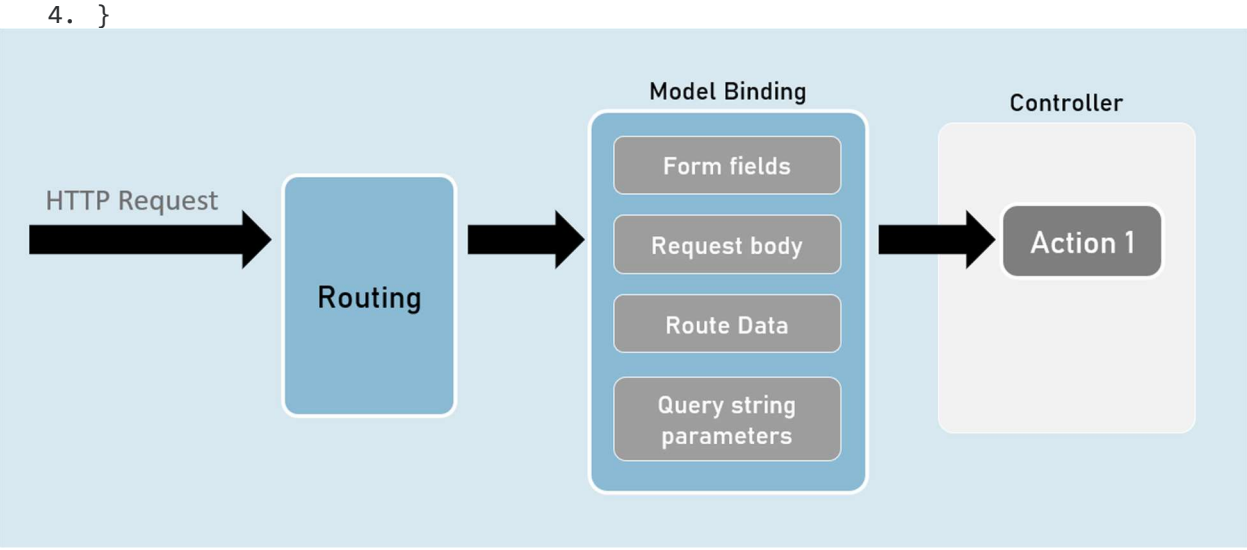


[FromQuery]

```
1. //gets the value from query string only
2. public IActionResult ActionMethodName( [FromQuery] type parameter)
3. {
4. }
```

[FromRoute]

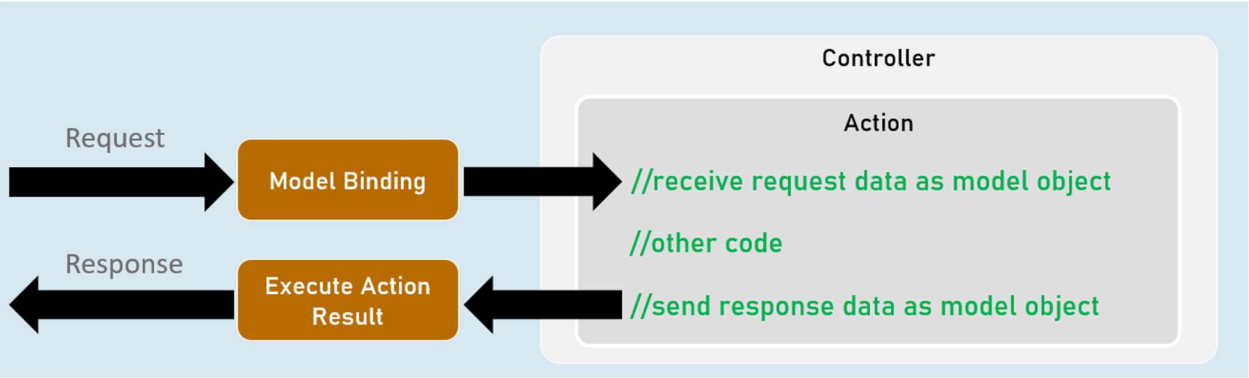
```
1. //gets the value from route parameters only
2. public IActionResult ActionMethodName( [FromRoute] type parameter)
3. {
4. }
```



Models

Model is a class that represents structure of data (as properties) that you would like to receive from the request and/or send to the response.

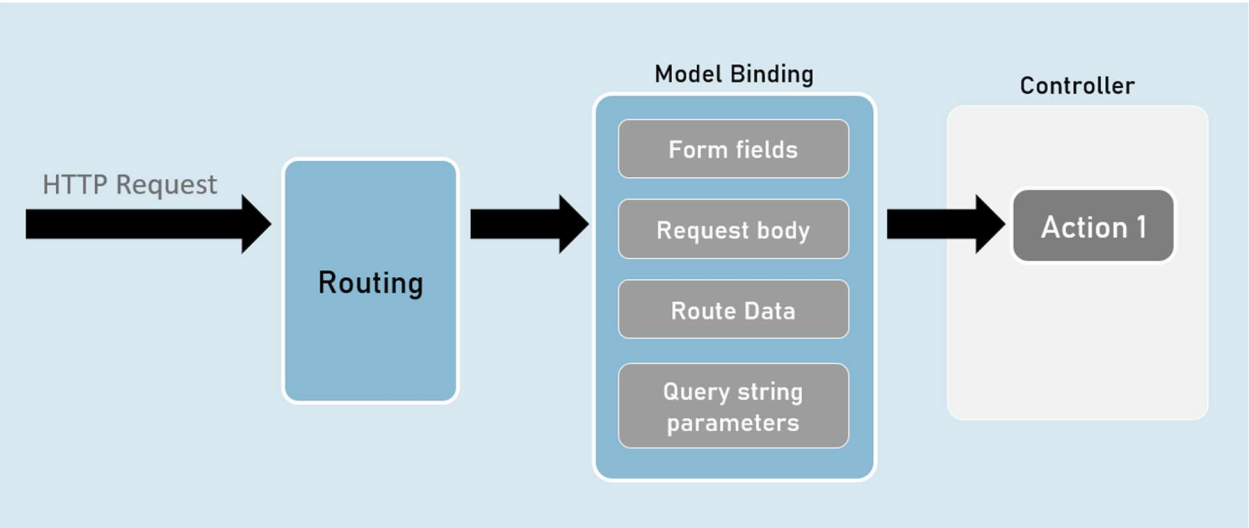
Also known as POCO (Plain Old CLR Objects).



Model

```
1. class ClassName
2. {
3.   public type PropertyName { get; set; }
4. }
```

form-urlencoded and form-data



form-urlencoded (default)

Request Headers

Content-Type: application/x-www-form-urlencoded

Request Body

param1=value1&param2=value2

form-data

Request Headers

Content-Type: multipart/form-data

Request Body

----------d74496d66958873e

Content-Disposition: form-data; name="param1"

value1

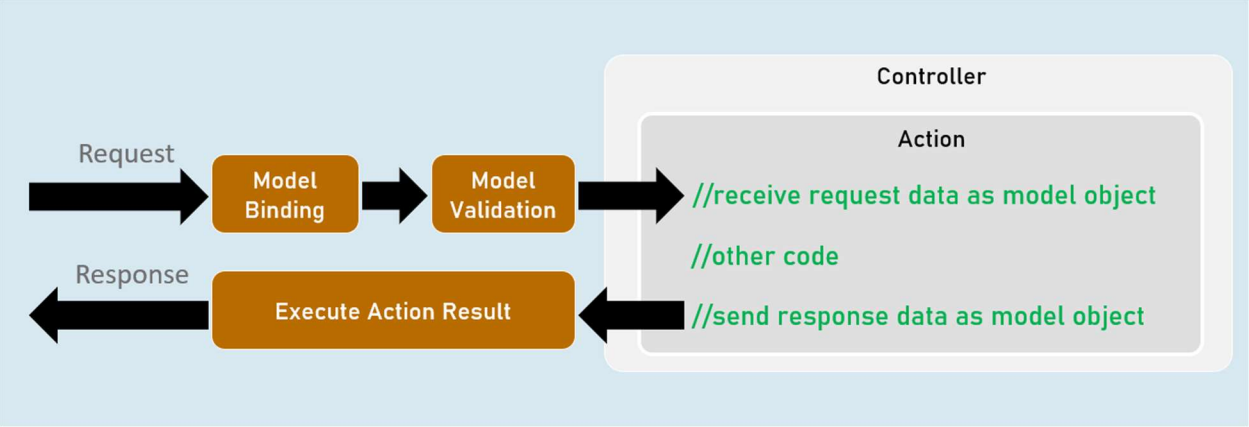
----------d74496d66958873e

Content-Disposition: form-data; name="param2"

value2

Model Validation

```
1. class ClassName
2. {
3.   [Attribute] //applies validation rule on this property
4.   public type PropertyName { get; set; }
5. }
```



ModelState

IsValid

Specifies whether there is at least one validation error or not (true or false).

Values

Contains each model property value with corresponding "Errors" property that contains list of validation errors of that model property.

ErrorCount

Returns number of errors.

Model Validation

[Required(ErrorMessage = "value")]

Specifies that the property value is required (can't be blank or empty).

[StringLength(int maximumLength, MinimumLength = value, ErrorMessage = "value")]

Specifies minimum and maximum length (number of characters) allowed in the string.

[Range(int minimum, int maximum, ErrorMessage = "value")]

Specifies minimum and maximum numerical value allowed.

[RegularExpression(string pattern, ErrorMessage = "value")]

Specifies the valid pattern (regular expression).

[EmailAddress(ErrorMessage = "value")]

Specifies that the value should be a valid email address.

[Phone(ErrorMessage = "value")]

Specifies that the value should be a valid phone number).

Eg: (999)-999-9999 or 9876543210

[Compare(string otherProperty, ErrorMessage = "value")]

Specifies that the values of current property and other property should be same.

[Url(ErrorMessage = "value")]

Specifies that the value should be a valid url (website address).

Eg: http://www.example.com

[ValidateNever]

Specifies that the property should not be validated (excludes the property from model validation).

Custom Validations

```
1. class ClassName : ValidationAttribute
2. {
3.   public override ValidationResult? IsValid(object? value, ValidationContext
   validationContext)
4.   {
5.     //return ValidationResult.Success;
6.     //[or] return new ValidationResult("error message");
7.   }
8. }
```

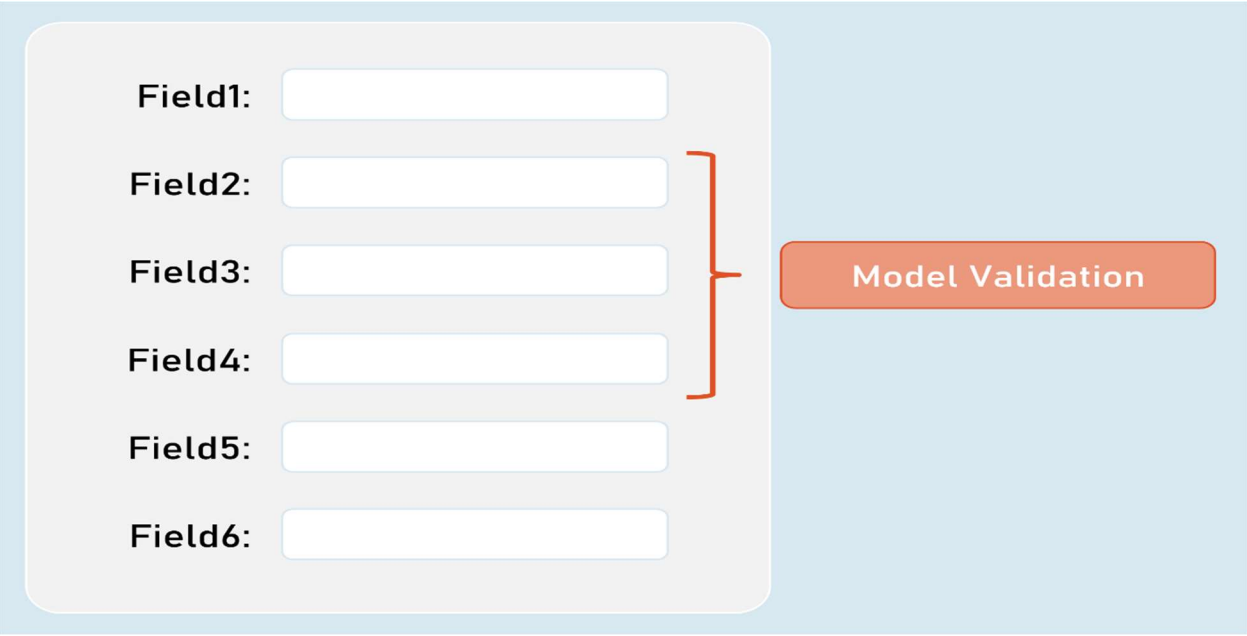
ValidationAttribute

- Base class for all validation attributes such as RequiredAttribute, RegularExpressionAttribute, RangeAttribute, StringLengthAttribute, CompareAttribute etc.
- Provides properties such as ErrorMessage & methods such as Validate(), IsValid() etc.

ValidationContext

- Acts as a parameter for "IsValid()" method of custom validation attribute classes.
- Provides properties such as ObjectType, ObjectInstance.

Custom Validations with Multiple Properties



IValidatableObject

```
1. class ClassName : IValidatableObject
2. {
3.   //model properties here
4.
5.   public IEnumerable<ValidationResult> Validate(ValidationContext
   validationContext)
6.   {
7.     if (condition)
8.     {
9.       yield return new ValidationResult("error message");
10.    }
11.  }
12.}
```

- Base class for model classes with validation.
- Provides a method called Validate() to define class level validation logic.
- The Validate() method executes after validating all property-level validations are executed; but doesn't execute if at least one property-level validations result error.

ValidationContext

- Acts as a parameter for "Validate()" method of model classes with IValidatableObject.
- Provides properties such as ObjectType, ObjectInstance.

[Bind] and [BindNever]

[Bind]

```
1. class ClassNameController
2. {
3.   public IActionResult ActionMethodName( [Bind(nameof(ClassName.PropertyName),
   nameof(ClassName.PropertyName) )]  ClassName parameterName)
4.   {
5.   }
6. }
```

- [Bind] attribute specifies that only the specified properties should be included in model binding.
- Prevents over-posting (post values into unexpected properties) especially in 'Create' scenarios.

[BindNever]

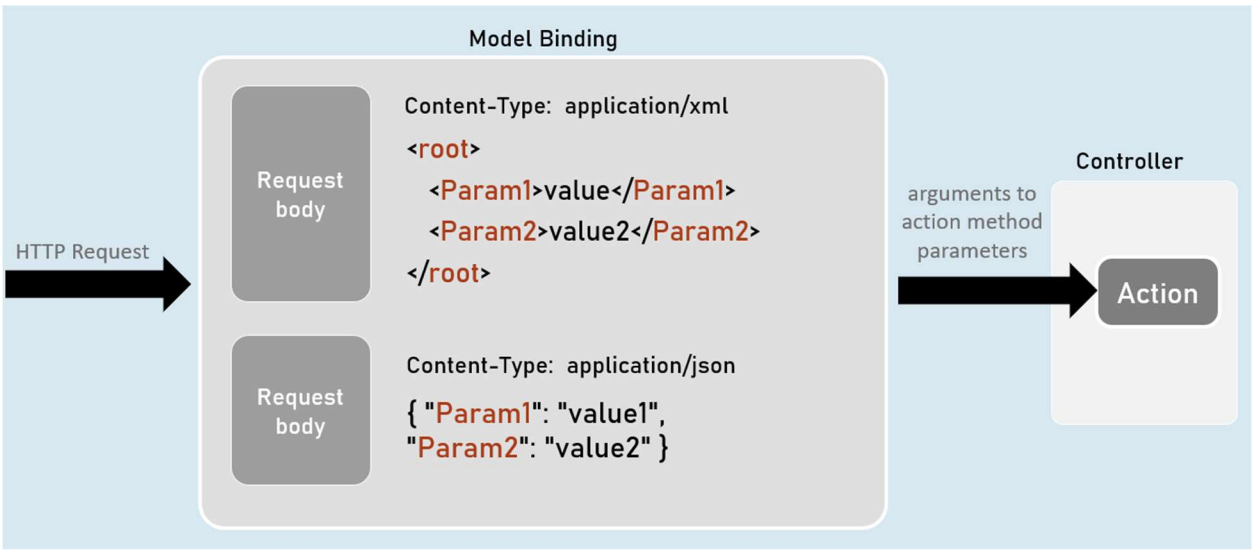
```
1. class ModelClassName
```

```
{
  [BindNever]
  public type PropertyName { get; set; }
}
```

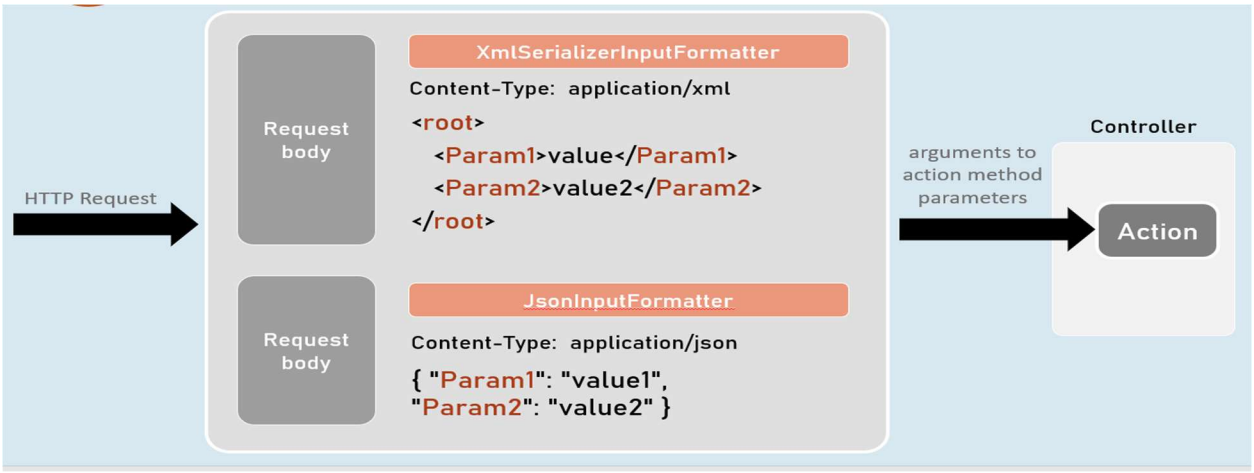
- [BindNever] attribute specifies that the specified property should NOT be included in model binding.
- Useful when you have fewer properties to eliminate from model binding.

[FromBody]

```
//enables the input formatters to read data from request body (as JSON or XML or custom) only
public IActionResult ActionMethodName( [FromBody] type parameter)
{
}
```



Input Formatters



Custom Model Binders

Custom Model Binder

```
class ClassName : IModelBinder
{
  public Task BindModelAsync(ModelBindingContext bindingContext)
  {
    //gets value from request
    bindingContext.ValueProvider.GetValue("FirstName");

    //returns model object after reading data from the request
    bindingContext.Result = ModelBindingResult.Success(your_object);
  }
}
```

IModelBinder

- Base interface for all custom model binders.
- Provides a method called BindModelAsync, to define logic for binding (reading) data from the request and creating a model object that has be received as parameter in the action method.

ModelBindingContext

- Acts as a parameter for "BindModelAsync()" method of custom model binder classes.
- Provides properties such as HttpContext, ModelState, ValueProvider, Result etc..

Custom Model Binder Providers

```
class ClassName : IModelBinderProvider
{
  public IModelBinder GetBinder(ModelBinderProviderContext providerContext)
  {
    //returns type of custom model binder class to be invoked
    return new BinderTypeModelBinder(typeof(YourModelBinderClassName));
  }
}
```

IModelBinderProvider

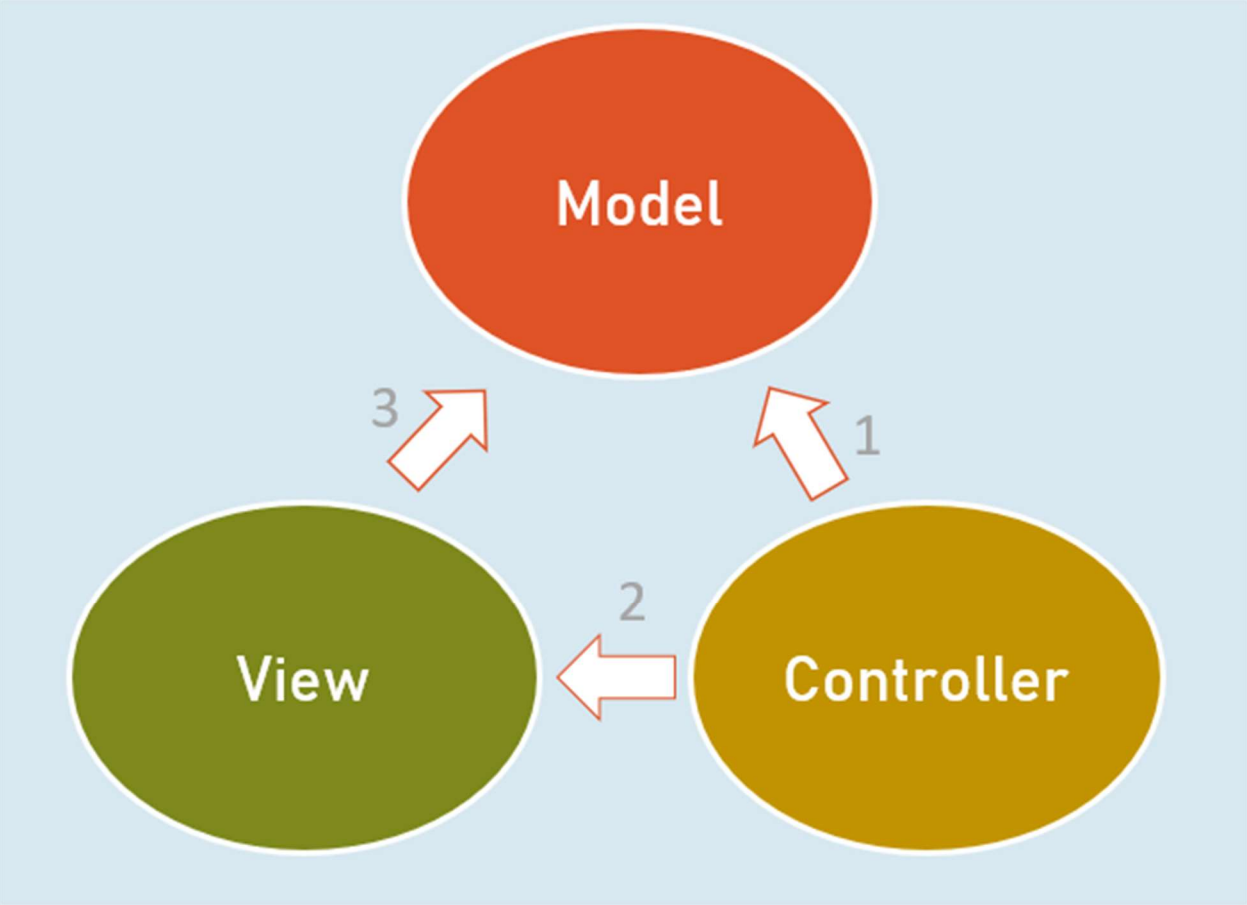
- Base interface for all custom model binder providers.
- Provides a method called GetBinder, to return the type of custom model binder class.

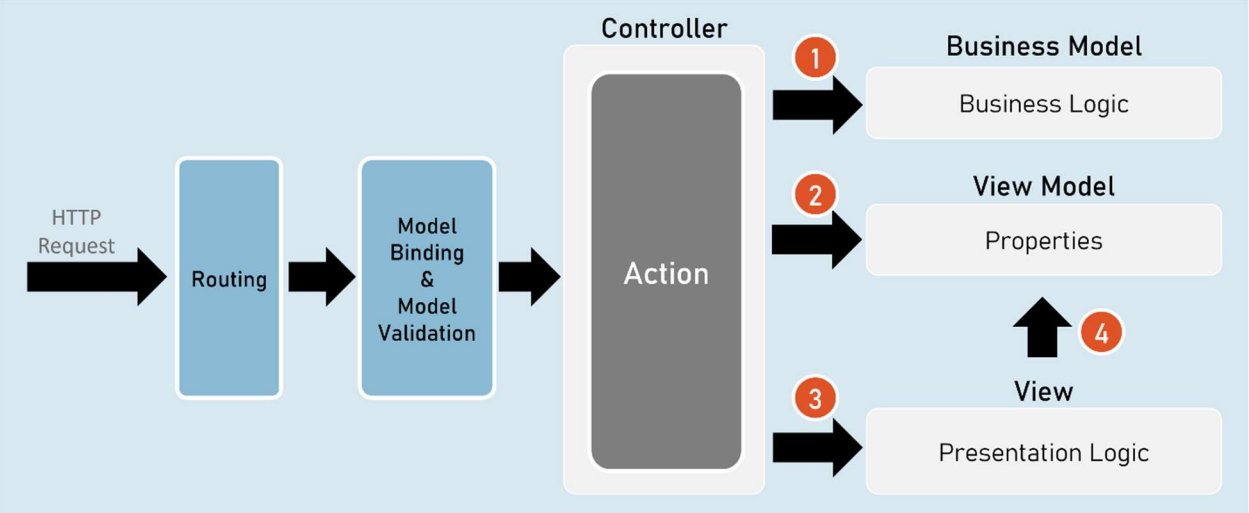
ModelBinderProviderContext

- Acts as a parameter for "GetBinder()" method of custom model binder provider classes.
- Provides properties such as BindingInfo, Services etc.

Model-View-Controller (MVC) Pattern

"Model-View-Controller" (MVC) is an architectural pattern that separates application code into three main components: Models, Views and Controllers.





1. Controller invokes Business Model.
2. Controller creates object of View Model.
3. Controller invokes View.
4. View accesses View Model.

Responsibilities of Model-View-Controller

Controller

- Receives HTTP request data.
- Invoke business model to execute business logic.

Business Model

- Receives input data from the controller.
- Performs business operations such as retrieving / inserting data from database.
- Sends data of the database back to the controller.

Controller

- Creates object of ViewModel and files data into its properties.
- Selects a view & invokes it & also passes the object of ViewModel to the view.

View

- Receives the object of ViewModel from the controller.
- Accesses properties of ViewModel to render data in html code.
- After the view renders, the rendered view result will be sent as response.

Benefits / Goals of MVC architectural pattern

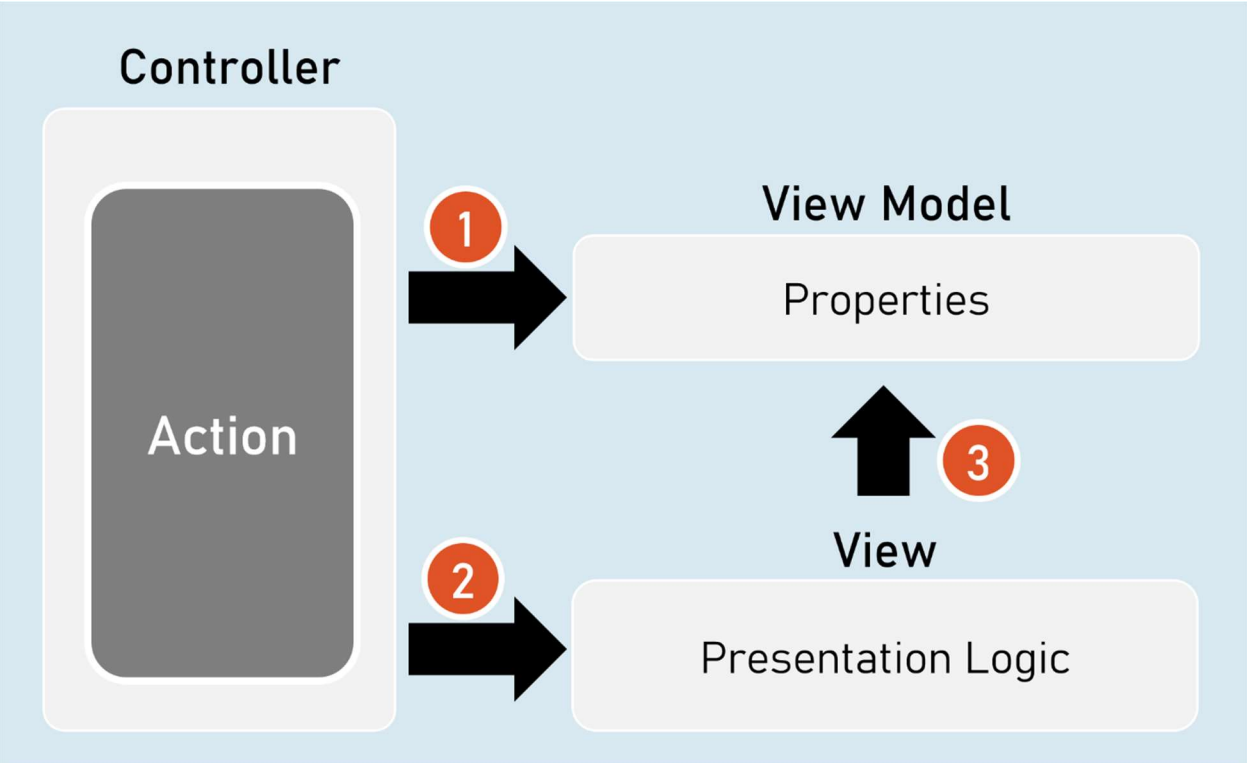
- Clean separation of concerns
- Each component (model, view and controller) performs single responsibility.
- Identifying and fixing errors will be easy.
- Each component (model, view and controller) can be developed independently.
- In practical, both view and controller depend on the model.

- Model doesn't depend on neither view nor the controller.
- This is one of the key benefits of the 'clean separation'.
- This separation allows the model to be built and tested independently.
- Unit testing each individual component is easier.

Views

View is a web page (.cshtml) that is responsible for containing presentation logic that merges data along with static design code (HTML).

- Controller creates an object of ViewModel and fills data in its properties.
- Controller selects an appropriate view and invokes the same view & supplies object of ViewModel to the View.
- View access the ViewModel.



- View contains HTML markup with Razor markup (C# code in view to render dynamic content).
- Razor is the view engine that defines syntax to write C# code in the view. @ is the syntax of Razor syntax.
- View is NOT supposed to have lots of C# code. Any code written in the view should relate to presenting the content (presentation logic).
- View should neither directly call the business model, nor call the controller's action methods. But it can send requests to controllers.

Razor View Engine

Razor Code Block

```
@{

  C# / html code here

}
```

Razor code block is a C# code block that contains one or more lines of C# code that can contain any statements and local functions.

Razor Expressions

```
@Expression
--or--
@(Expression)
```

Razor expression is a C# expression (accessing a field, property or method call) that returns a value.

Razor - If

```
@if (condition) {
  C# / html code here
}
```

Razor - if…else

```
@if (condition) {
  C# / html code here
}
else {
  C# / html code here
}
```

Else…if and nested-if also supported.

Razor - Switch

```
@switch (variable) {
  case value1: C# / html code here; break;
  case value2: C# / html code here; break;
  default: C# / html code here; break;
}
```

Razor - foreach

```
@foreach (var variable in collection ) {
  C# / html code here
}
```

Razor - for

```
@for (initialization; condition; iteration) {
  C# / html code here
}
```

Razor - Literal

```
@{
 @: static text
}
```

Razor - Literal

```
<text>static text</text>
```

Razor - Local Functions

```
@{
return_type method_name(arguments) {
  C# / html code here
 }
}
```

The local functions are callable within the same view.

Razor - Members

Razor - Methods, Properties, Fields

```
@functions {
 return_type method_name(arguments) {
   C# / html code here
 }

```

```
  data_type field_name;

  data_type property_name
  {
    set { … }
    get { … }
  }
}
```

The members of razor view can be accessible within the same view.

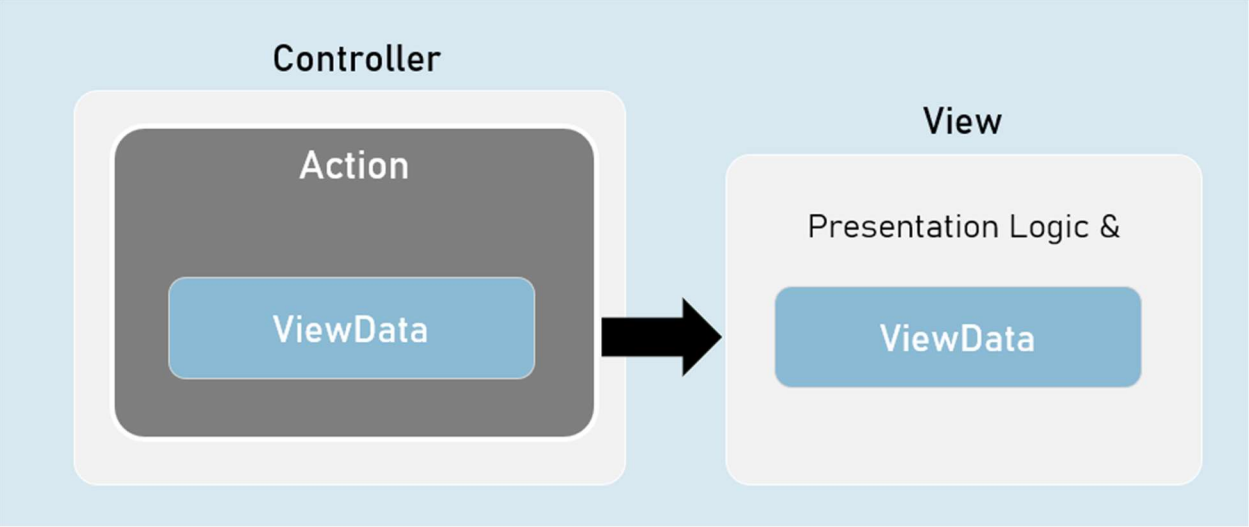
Html.Raw( )

```
@{
 string variable = "html code";
}

@Html.Raw(variable) //prints the html markup without encoding (converting html tags into plain text)
```

ViewData

ViewData is a dictionary object that is automatically created up on receiving a request and will be automatically deleted before sending response to the client.

It is mainly used to send data from controller to view.



ViewData is a property of Microsoft.AspNetCore.Mvc.Controller class and Microsoft.AspNetCore.Mvc.Razor.RazorPage class.

It is of Microsoft.AspNet.Mvc.ViewFeatures.ViewDataDictionary type.

```
namespace Microsoft.AspNetCore.Mvc
{
```

```
3.  public abstract class Controller : ControllerBase
4.  {
5.    public ViewDataDictionary ViewData { get; set; }
6.  }
7. }
```

- It is derived from IDictionary<KeyValuePair<string, object>> type.
- That means, it acts as a dictionary of key/value pairs.
- Key is of string type.
- Value is of object type.

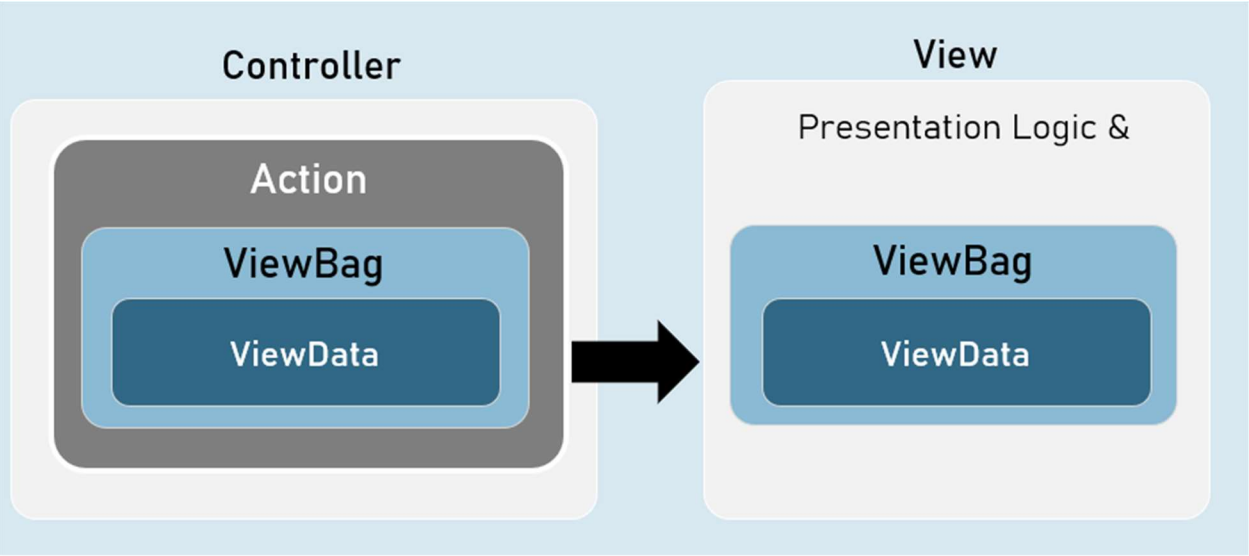
ViewData - Properties and Methods

- `int Count { get; set; }` //gets the number of elements.
- `[string Key]` //Gets or sets an element.
- `Add(string key, object value)` //Adds a new element.
- `ContainsKey(string key)` //Determines whether the specified key exists or not.
- `Clear()` //Clears (removes) all elements.

ViewBag

ViewBag is a property of Controller and View, that is used to access the ViewData easily.

ViewBag is 'dynamic' type.



ViewBag is a property of Microsoft.AspNetCore.Mvc.Controller class and Microsoft.AspNetCore.Mvc.Razor.RazorPageBase class.

It is of dynamic type.

```
1. namespace Microsoft.AspNetCore.Mvc
2. {
3.  public abstract class Controller : ControllerBase
4.  {
5.    public dynamic ViewBag { get; set; }
6.  }
7. }
```

The 'dynamic' type similar to 'var' keyword.

But, it checks the data type and at run time, rather than at compilation time.

If you try to access a non-existing property in the ViewBag, it returns null.

```
[string Key] //Gets or sets an element.
```

Benefits of 'ViewBag' over ViewData

ViewBag's syntax is easier to access its properties than ViewData.

Eg: `ViewBag.property` [vs] `ViewData["key"]`

You need NOT type-cast the values while reading it.

Eg: `ViewBag.object_name.property`

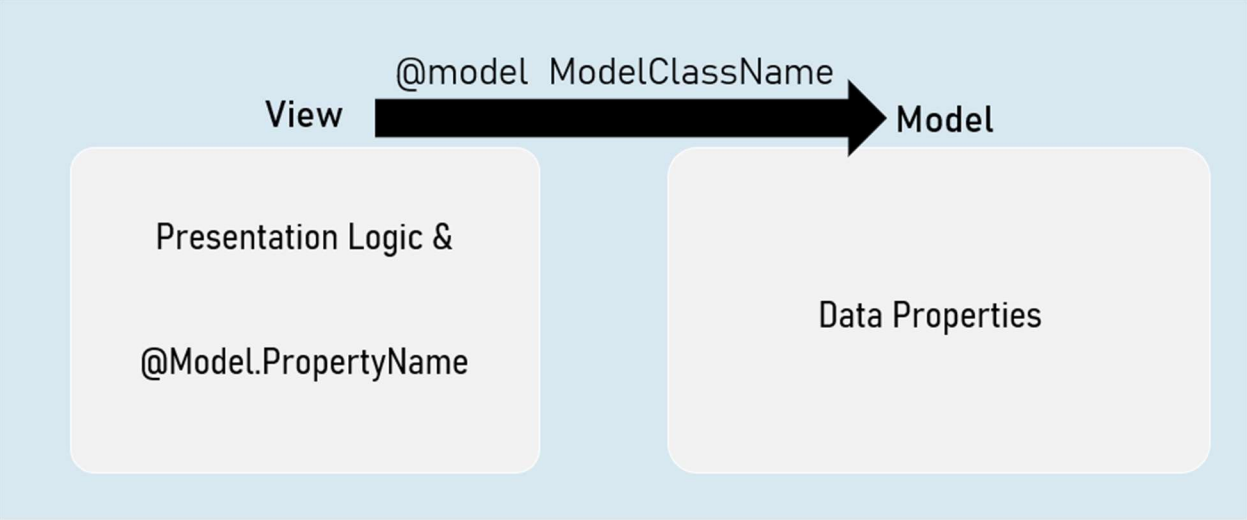
[vs]

```
(ViewData["key"] as ClassName).Property
```

Strongly Typed Views

Strongly Typed View is a view that is bound to a specified model class.

It is mainly used to access the model object / model collection easily in the view.



Benefits of Strongly Typed Views

- You will get Intellisense while accessing model properties in strongly typed views, since the type of model class was mentioned at @model directive.
- Property names are compile-time checked; and shown as errors in case of misspelled / non-existing properties in strongly typed views.
- You will have only one model per one view in strongly typed views.
- Easy to identify which model is being accessed in the view.

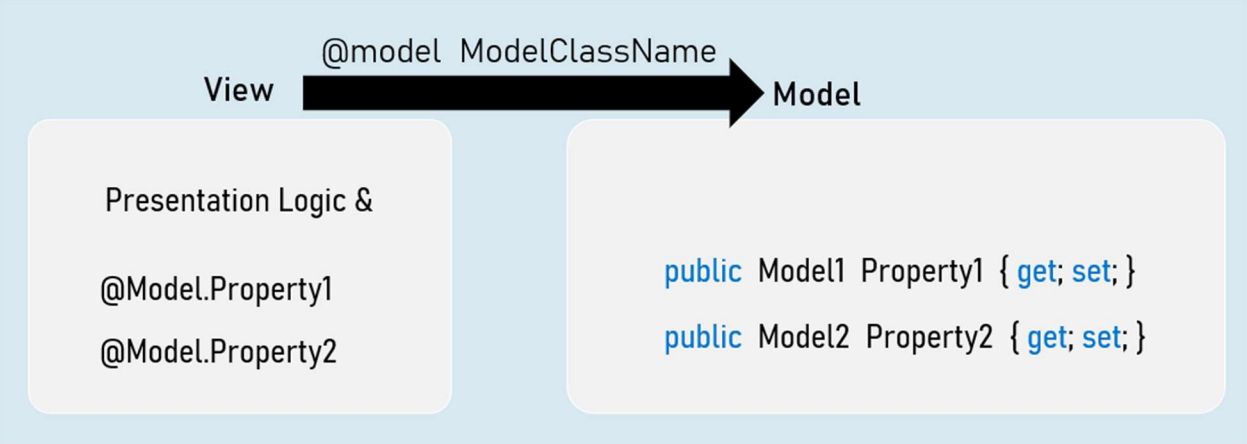
Helper methods in Controller to invoke a View

- `return View( );` //View name is the same name as the current action method.
- `return View(object Model );` //View name is the same name as the current action method & the view can be a strongly-typed view to receive the supplied model object.
- `return View(string ViewName);` //View name is explicitly specified.
- `return View(string ViewName, object Model );` //View name is explicitly specified & the view can be a strongly-typed view to receive the supplied model object.

Strongly Typed Views

Strongly Typed View can be bound to a single model directly.

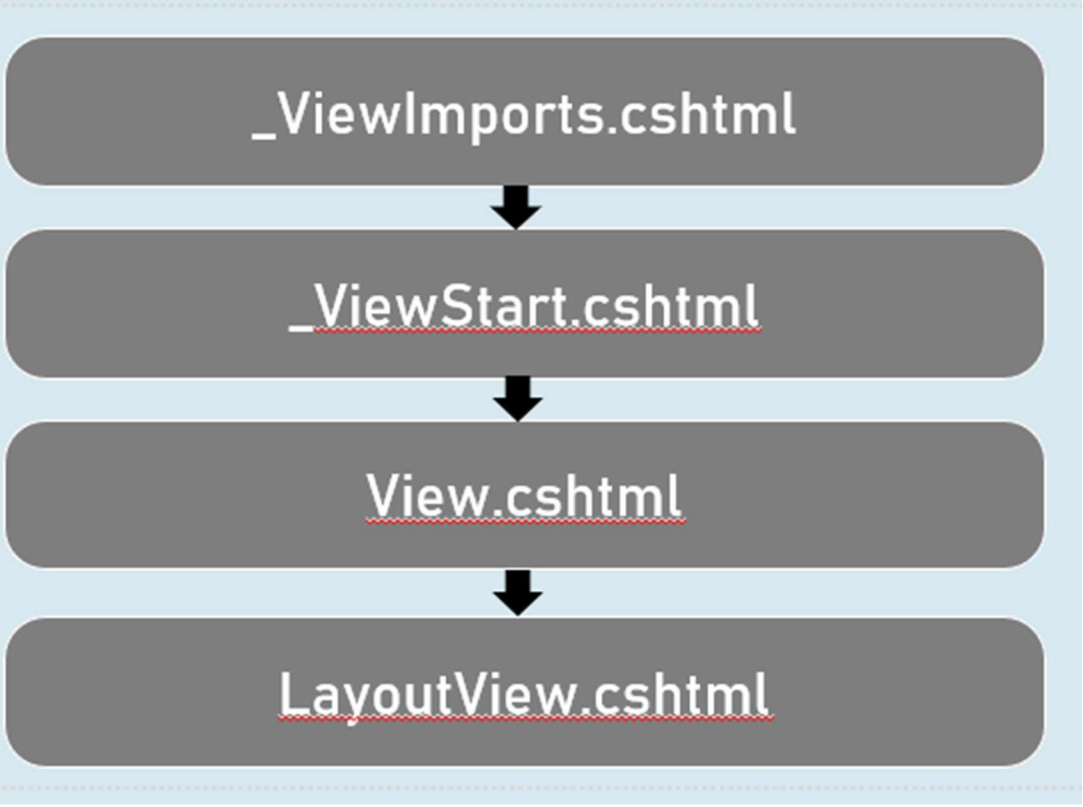
But that model class can have reference to objects of other model classes.

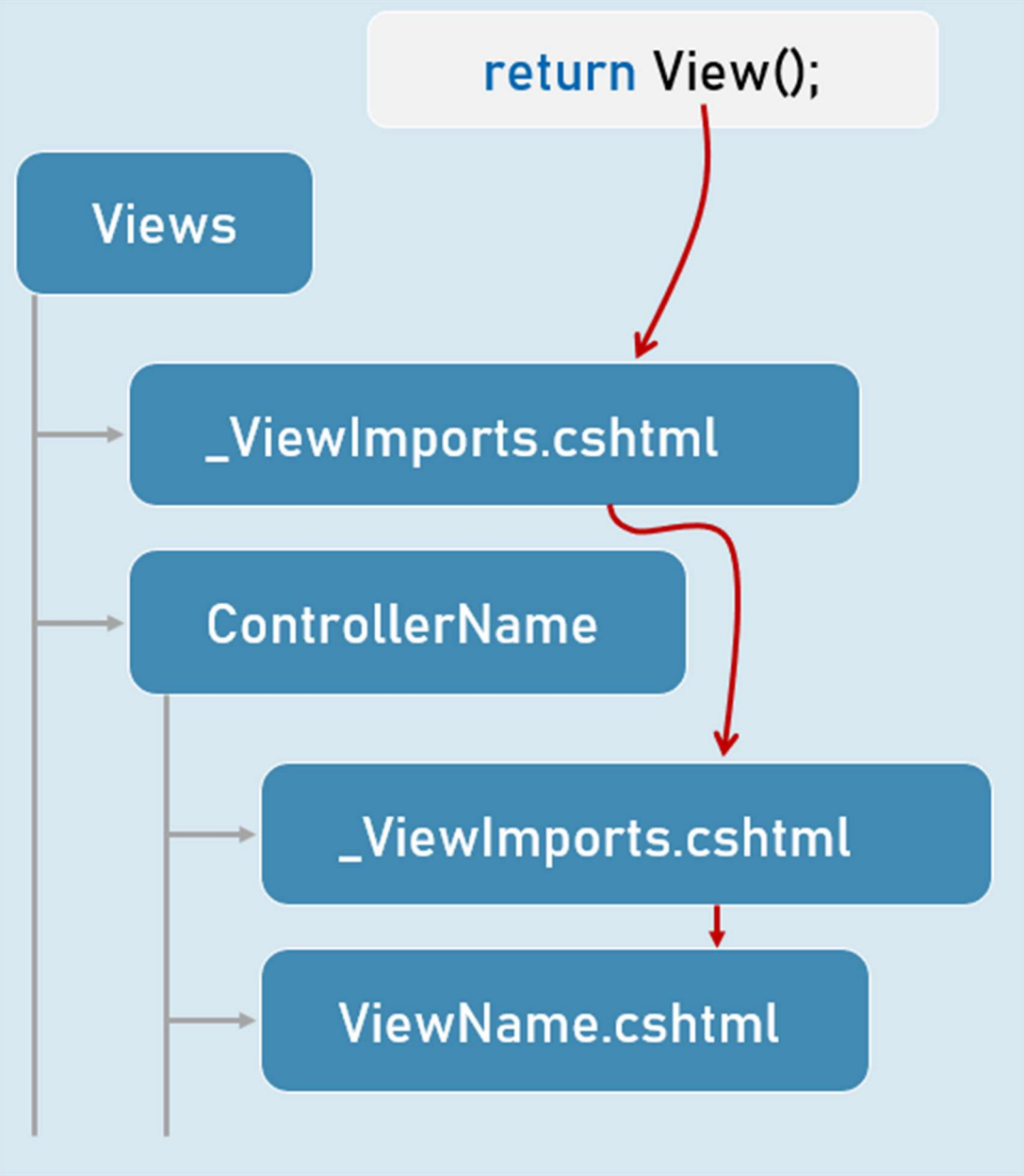


ViewImports.cshtml

ViewImports.cshtml is a special file in the "Views" folder or its subfolder, which executes automatically before execution of a view.

It is mainly used to import common namespaces that are to imported in a view.

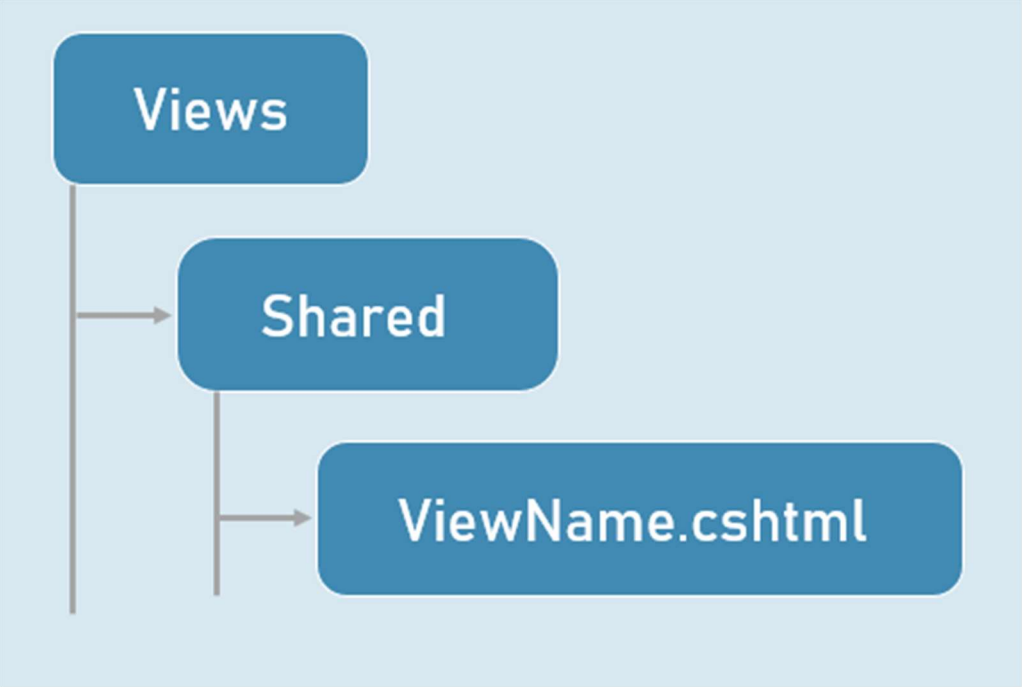




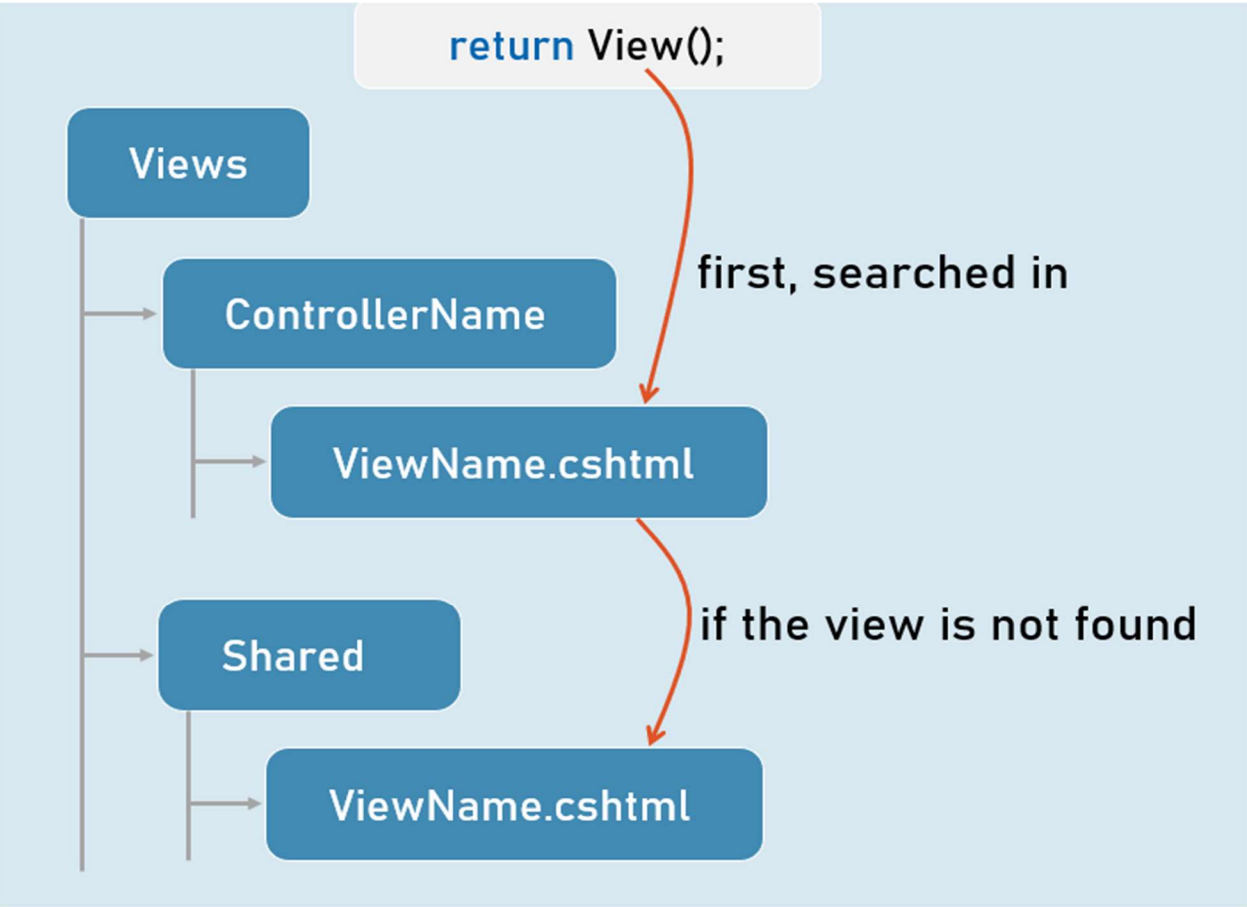
Shared Views

Shared views are placed in "Shared" folder in "Views" folder.

They are accessible from any controller, if the view is NOT present in the "Views\ControllerName" folder.

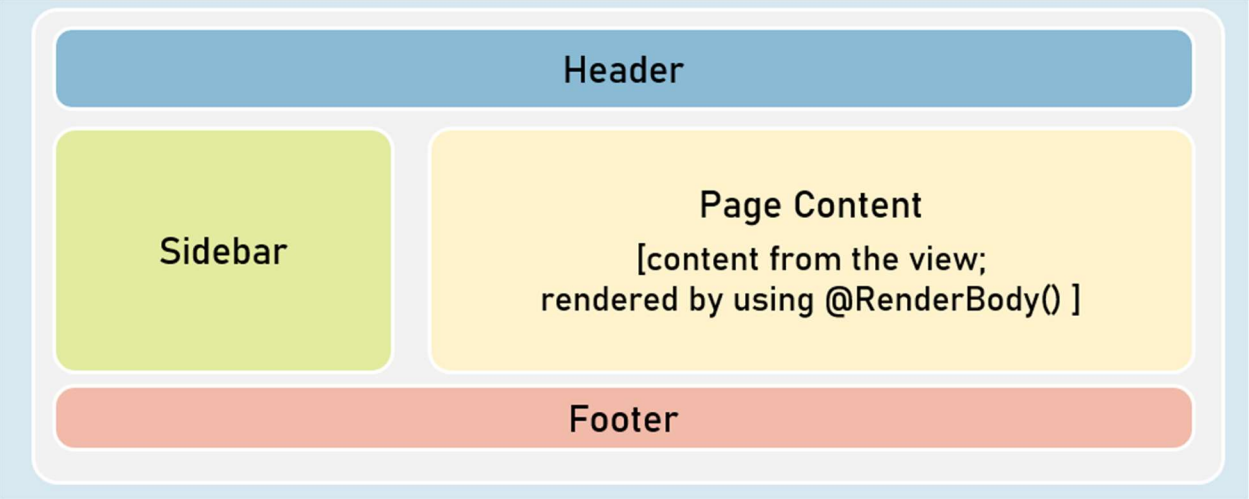


View Resolution

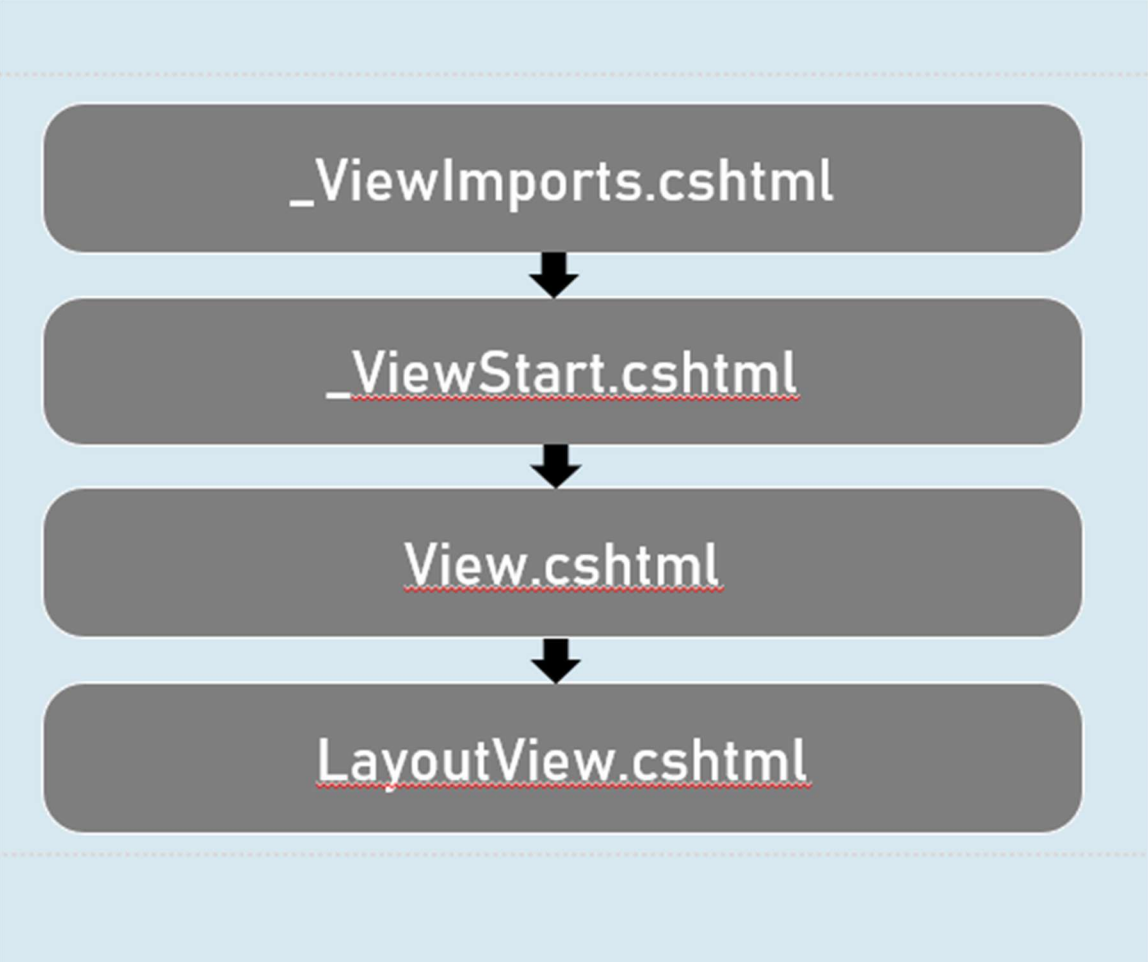


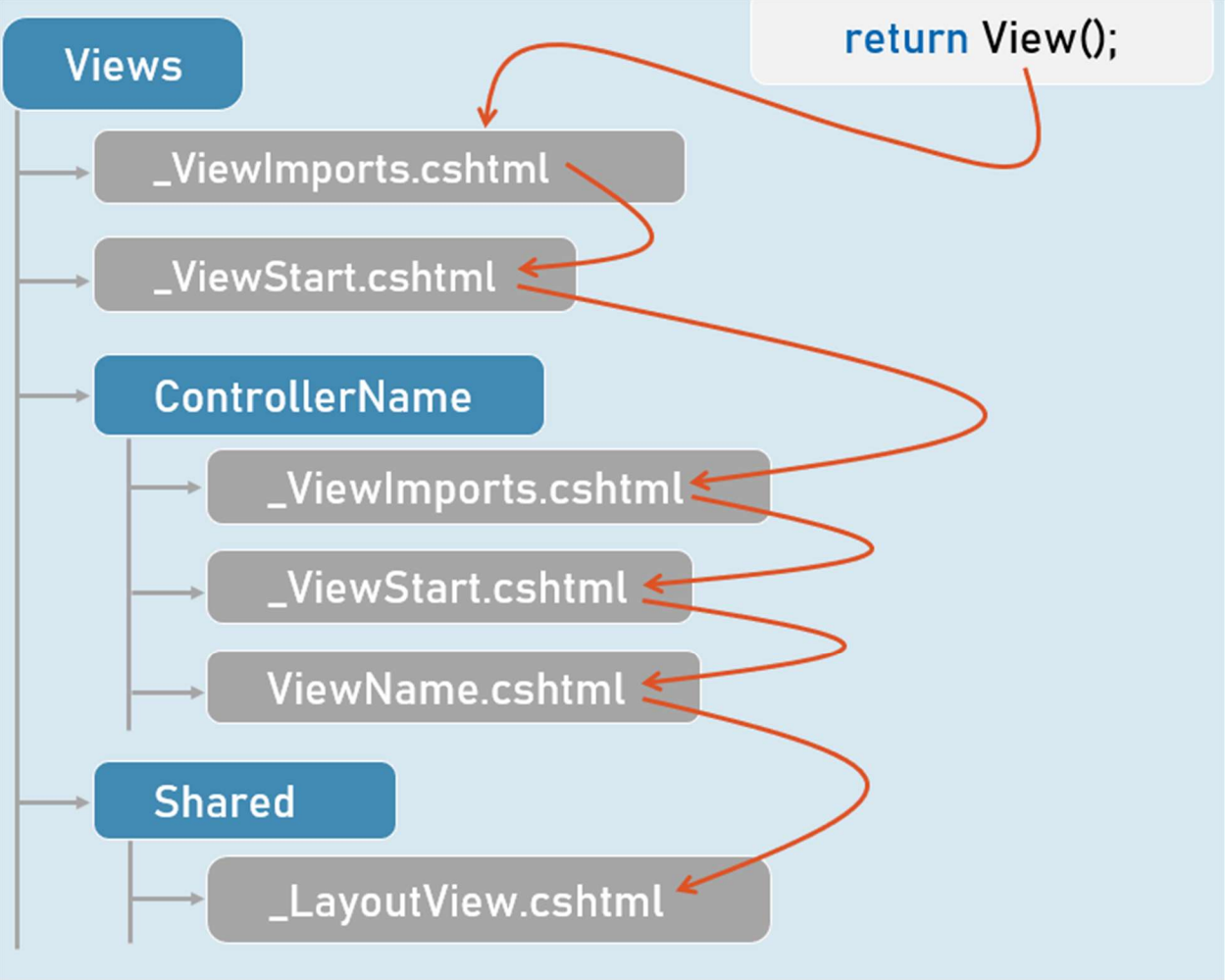
Layout Views

Layout View is a web page (.cshtml) that is responsible for containing presentation logic template (commonly the html template with header, sidebar, footer etc.)



Order of Views Execution



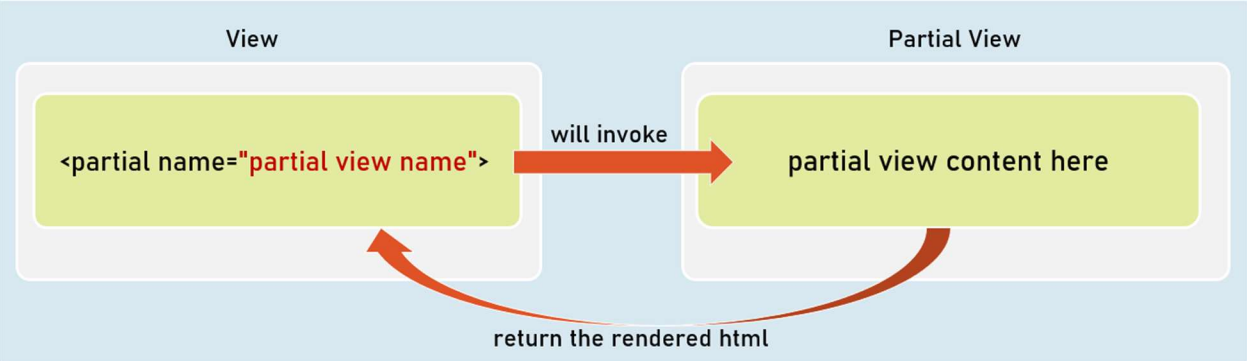


Layout Views

- The @RenderBody() method presents only in layout view to represent the place where exactly the content from the view has to be rendered.
- The "Layout" property of the view specifies path of the layout view.
- It can be dynamically set in the view.
- Both View and Layout View shares the same ViewData object.
- So it is possible to send data from view to layout, since the view executes first.
- The css files / js files imported in layout view will be applicable to view also, because the content of view will be merged into the layout view at run time.

Partial Views

Partial view is a razor markup file (.cshtml) that can't be invoked individually from the controller; but can be invoked from any view within the same web application.



Invoking Partial Views

```
<partial name="partial view name" />
```

Returns the content to the parent view.

```
@await Html.PartialAsync("partial view name")
```

Returns the content to the parent view.

```
@{ await Html.RenderPartialAsync("partial view name"); }
```

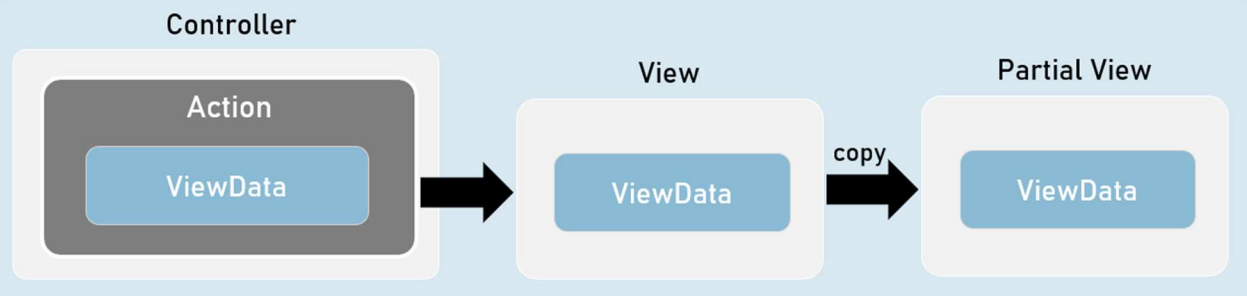
Streams the content to the browser.

Partial Views with ViewData

When partial view is invoked, it receives a *copy* of the parent view's ViewData object.

So, any changes made in the ViewData in the partial view, do NOT effect the ViewData of the parent view.

Optionally, you can supply a custom ViewData object to the partial view, if you don't want the partial view to access the entire ViewData of the parent view.



Invoking Partial Views with View Data

```
@{ await Html.RenderPartialAsync("partial view name", ViewData);
}
```

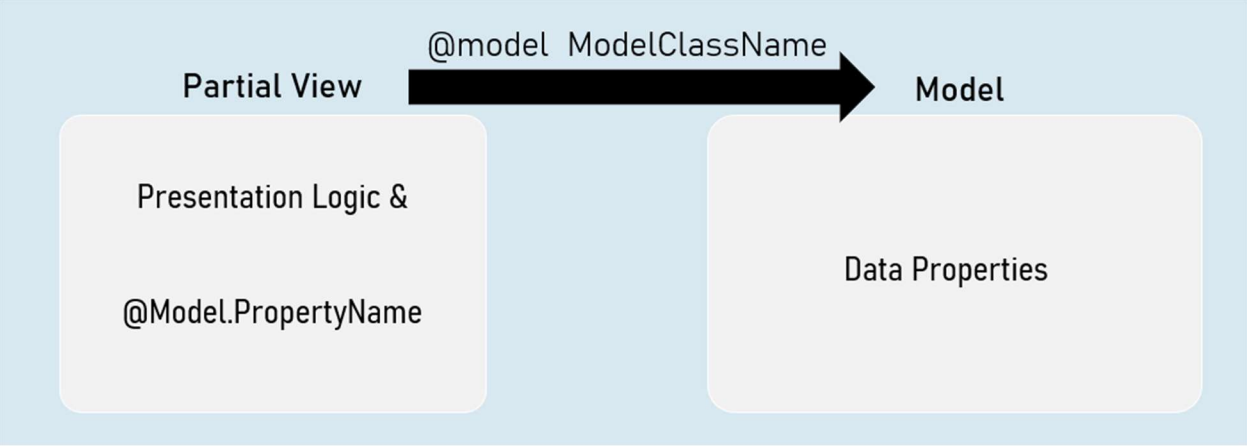
-- or --

```
<partial name="partial view name" view-data="ViewData" />
```

Strongly Typed Partial Views

Strongly Typed Partial View is a partial view that is bound to a specified model class.

So, it gets all the benefits of a strongly typed view.



Invoking Strongly-Typed Partial View

```
@{ await Html.RenderPartialAsync("partial view name", Model); }
```

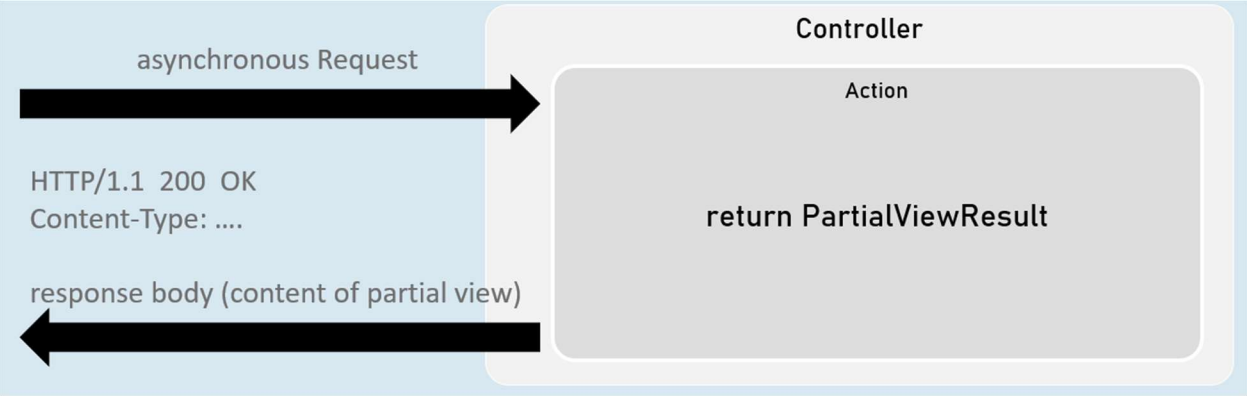
-- or --

```
<partial name="partial view name" model="Model" />
```

PartialViewResult

PartialViewResult can represent the content of a partial .

Generally useful to fetch partial view's content into the browser, by making an asynchronous request (XMLHttpRequest / fetch request) from the browser.



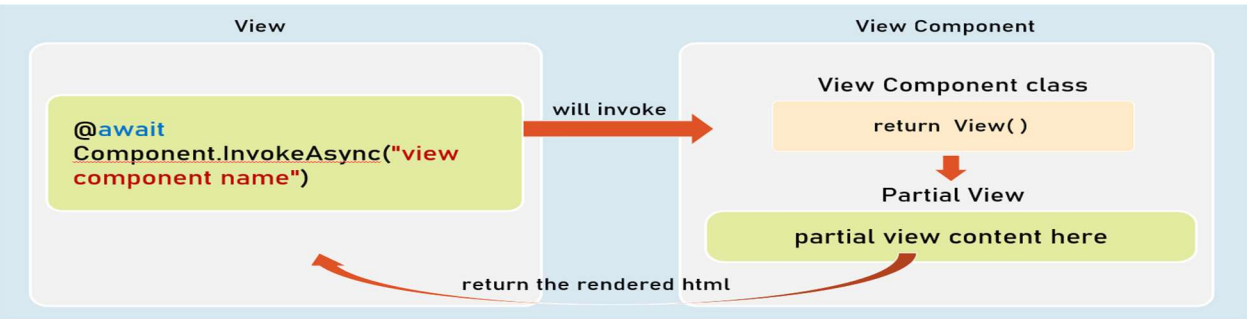

```
return new PartialViewResult() { ViewName = "partial view name", Model = model };
```

[or]

```
return PartialView("partial view name", model);
```

View Components

View Component is a combination of a class (derived from Microsoft.AspNetCore.ViewComponent) that supplied data, and a partial view to render that data.



Invoking View Component

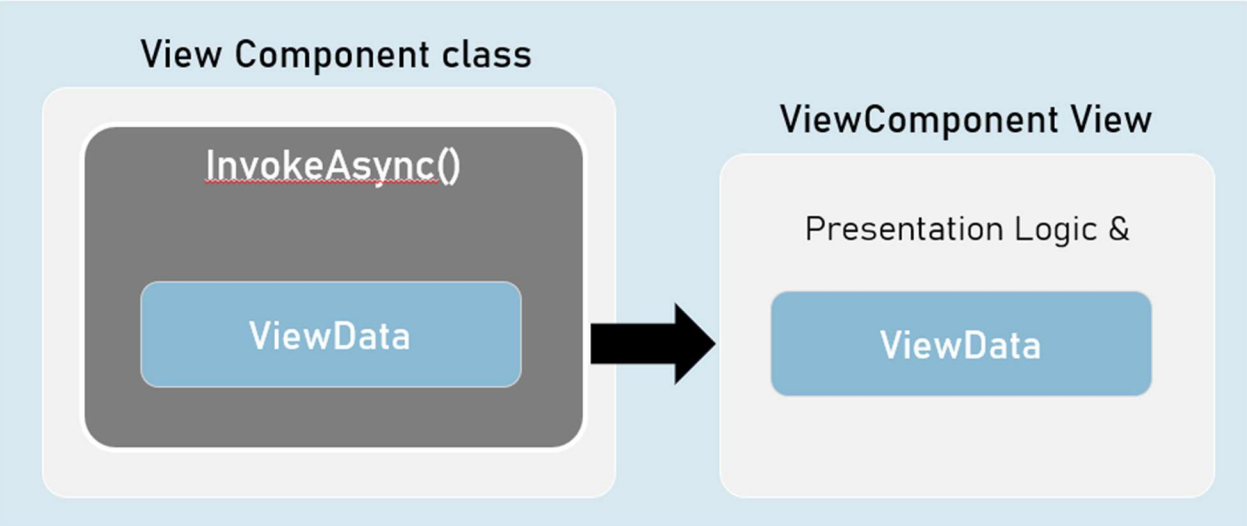
```
1. @await Component.InvokeAsync("view component name");
2. --or--
3. <vc:view-component-name />
```

View Components

- View component renders a chunk rather than a whole response.
- Includes the same separation-of-concerns and testability benefits found with a controller and view.
- Should be either suffixed with the word "ViewComponent" or should have [ViewComponent] attribute.
- Optionally, it can inherit from System.AspNetCore.Mvc.ViewComponent.

View Components with ViewData

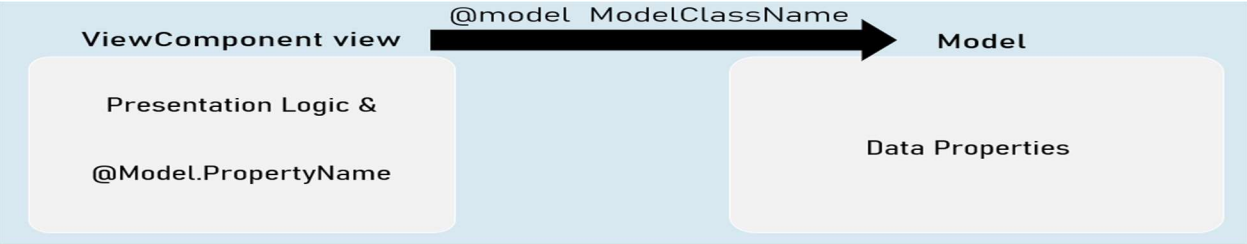
The ViewComponent class can share ViewData object to the ViewComponent view.



Strongly Typed ViewComponent

Strongly Typed ViewComponent's view is tightly bound to a specified model class.

So, it gets all the benefits of a strongly typed view.

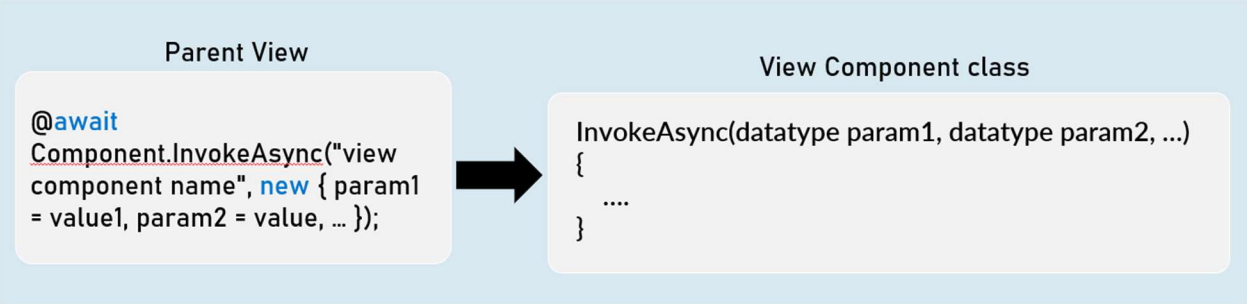


ViewComponents with Parameters

You can supply one or more parameters to the view component class.

The parameters are received by InvokeAsync method of the view component class.

All the parameters of view component are mandatory (must supply a value).



Invoking ViewComponent with parameters

```
@await Component.InvokeAsync("view component name", new { param = value });
```

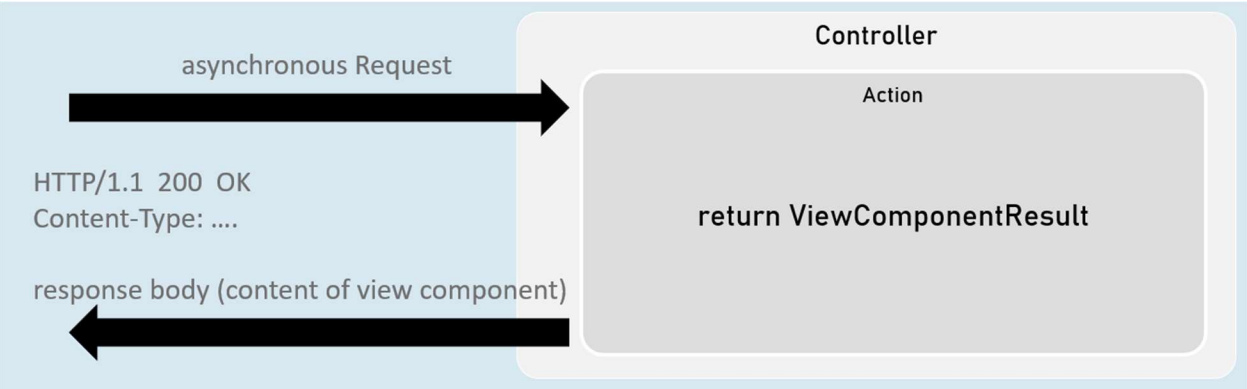
-- or --

```
<vc:view-component-name param="value" />
```

ViewComponentResult

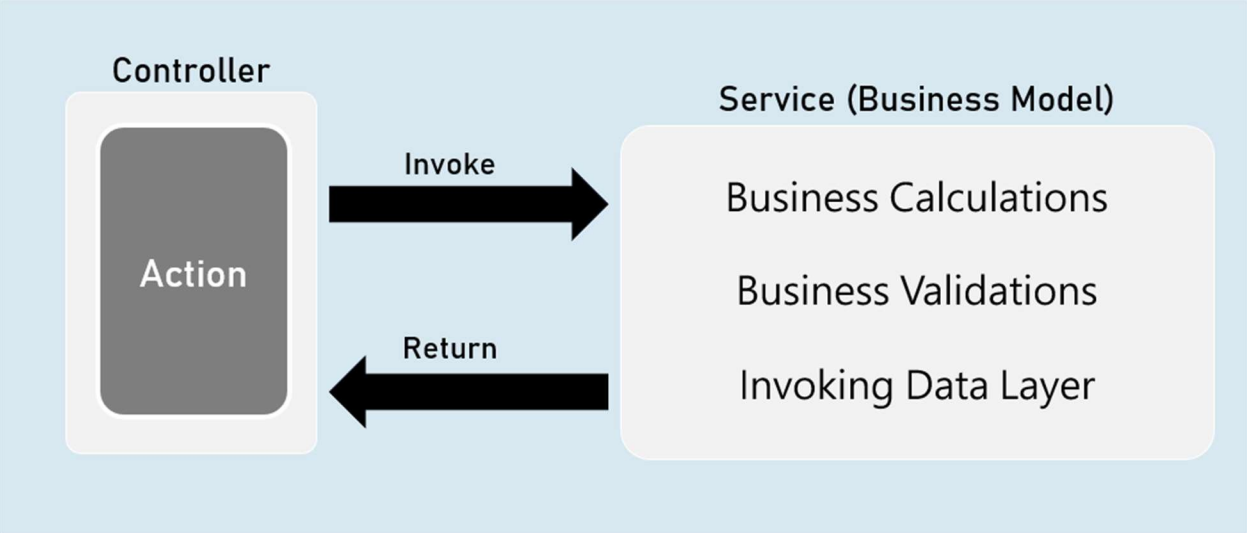
ViewComponent can represent the content of a view component .

Generally useful to fetch view component's content into the browser, by making an asynchronous request (XMLHttpRequest / fetch request) from the browser.



```
1. return new ViewComponentResult() { ViewName = "view component name", Arguments
   = new { param1 = value, param2 = value } };
2. [or]
3. return ViewComponent("view component name", new { param1 = value, param2 =
   value } });
```

Services



'Service' is a class that contains business logic such as business calculations, business validations that are specific to the domain of the client's business.

Service is an abstraction layer (middle layer) between presentation layer (or application layer) and data layer.

It makes the business logic separated from presentation layer and data layer.

It makes the business logic to be unit testable easily.

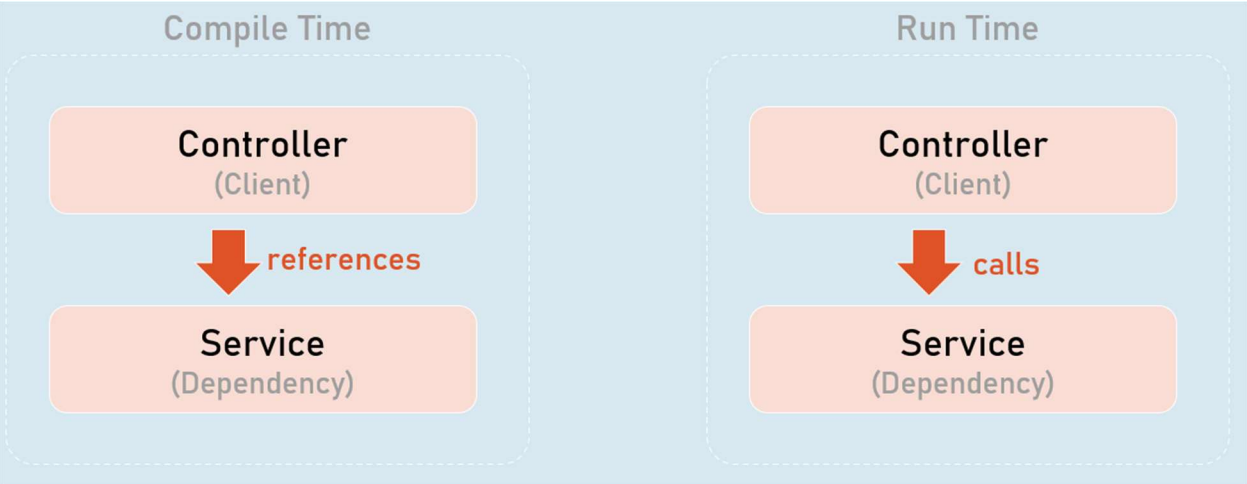
Will be invoked by controller.

Direct Dependency

Controller (Client) → Service (Dependency)

```
public class MyController : Controller
{
  private readonly MyService _service;
  public MyController()
  {
     _service = new MyService();
  }
  public IActionResult ActionMethod()
  {
     _service.ServiceMethod();
  }
}
```

```
public class MyService
{
  public void ServiceMethod()
  {
     …
  }
}
```



Higher-level modules depend on lower-level modules.

Dependency Problem

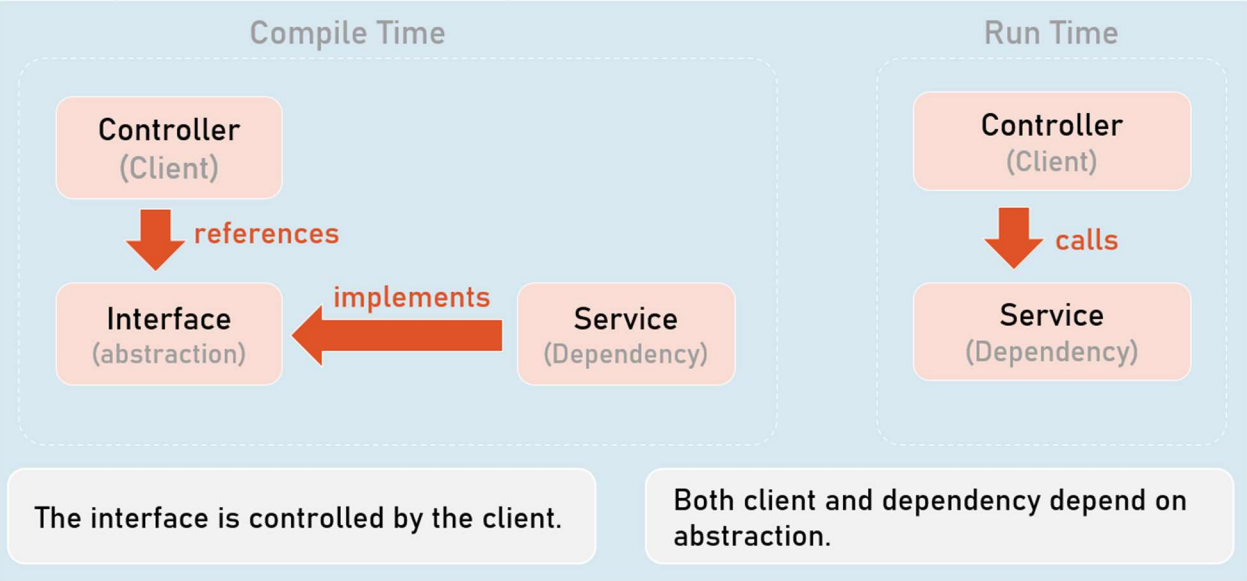
Higher-level modules depend on lower-level modules.

- Means, both are tightly-coupled.
- The developer of higher-level module SHOULD WAIT until the completion of development of lower-level module.
- Requires much code changes in to interchange an alternative lower-level module.
- Any changes made in the lower-level module effects changes in the higher-level module.
- Difficult to test a single module without effecting / testing the other module.

Dependency Inversion Principle

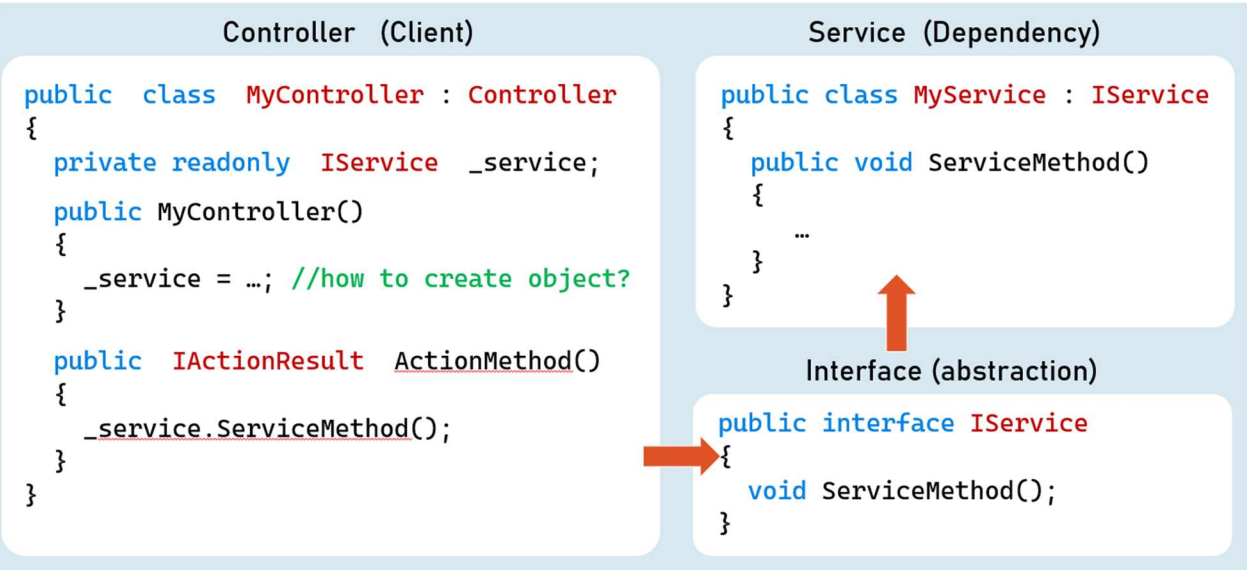
Dependency Inversion Principle (DIP) is a design principle (guideline), which is a solution for the dependency problem.

"The higher-level modules (clients) SHOULD NOT depend on low-level modules (dependencies).
Both should depend on abstractions (interfaces or abstract class)."
"Abstractions should not depend on details (both client and dependency).
Details (both client and dependency) should depend on abstractions."



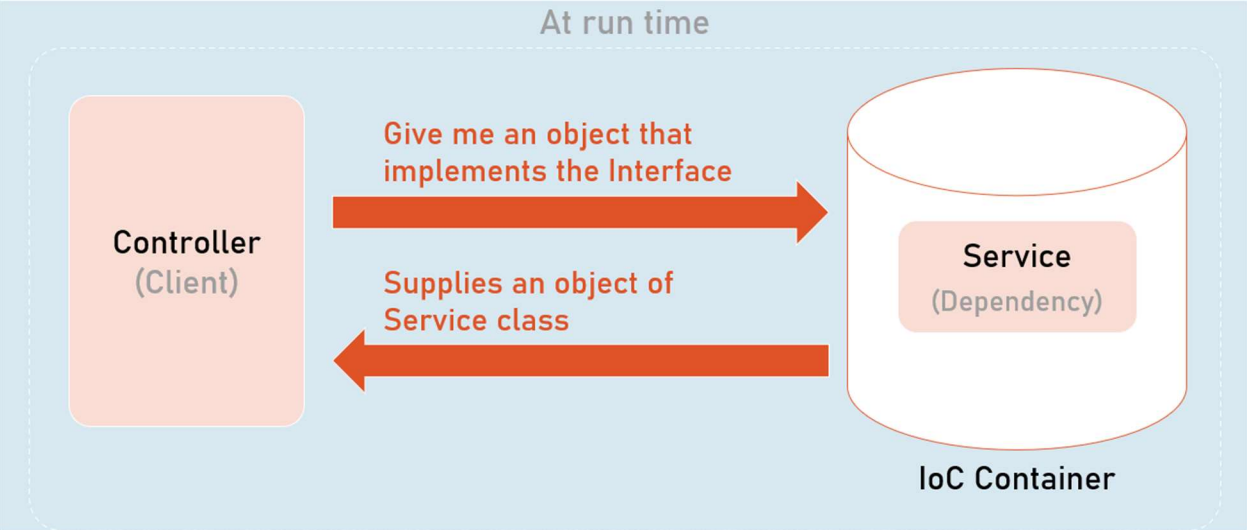
The interface is controlled by the client.

Both client and dependency depend on abstraction.



Inversion of Control (IoC)

- Inversion of Control (IoC) is a design pattern (reusable solution for a common problem), which suggests "IoC container" for implementation of Dependency Inversion Principle (DIP).
- It inverses the control by shifting the control to IoC container.
- "Don't call us, we will call you" pattern.
- It can be implemented by other design patterns such as events, service locator, dependency injection etc.

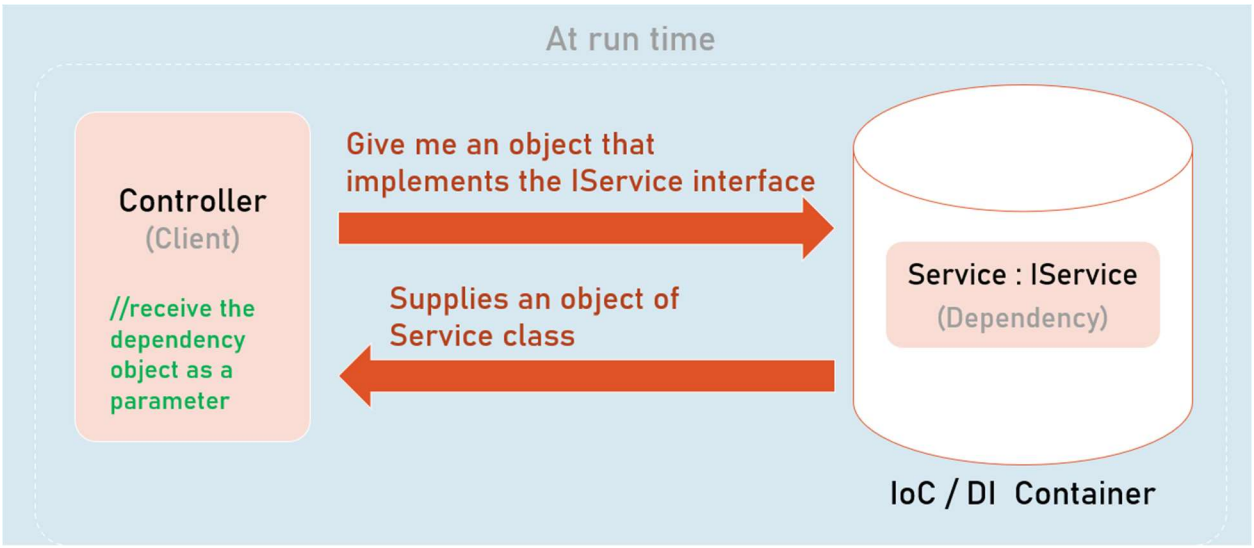


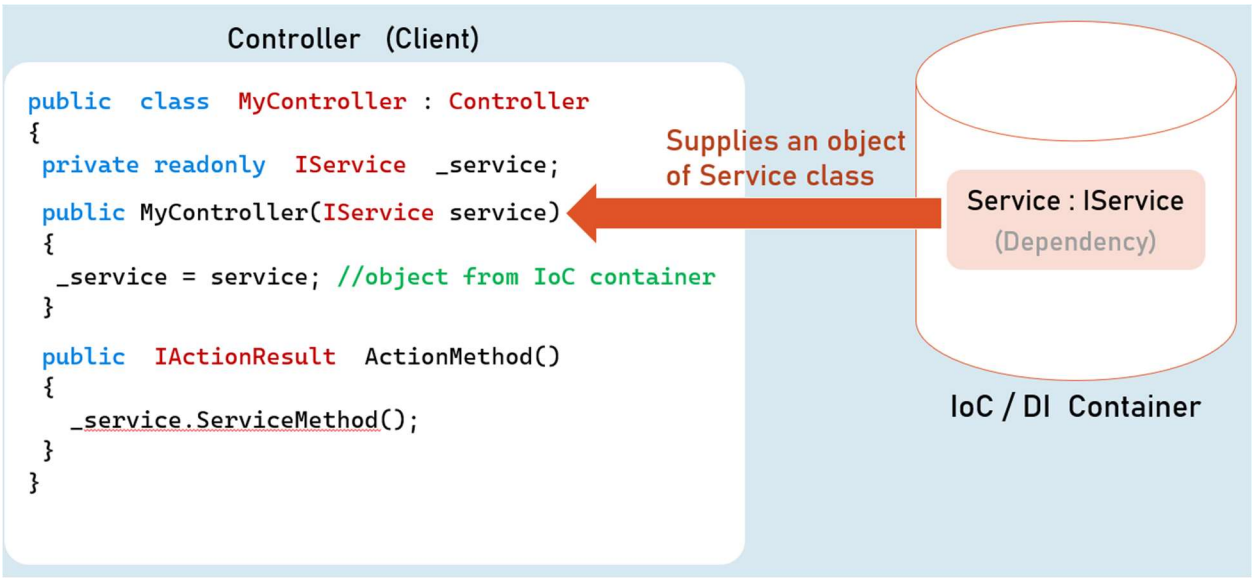
All dependencies should be added into the IServiceCollection (acts as IoC container).

```
builder.Services.Add(
  new ServiceDescriptor(
    typeof (Interface),
    typeof (Service)
    ServiceLifetime.LifeTime //Transient, Scoped, Singleton
  )
);
```

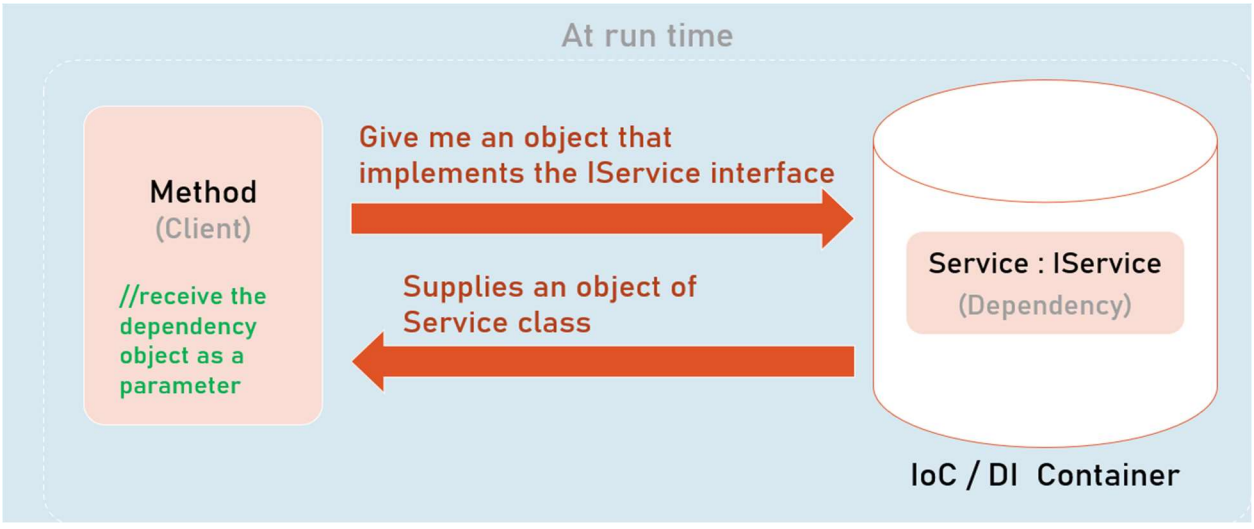
Dependency Injection (DI)

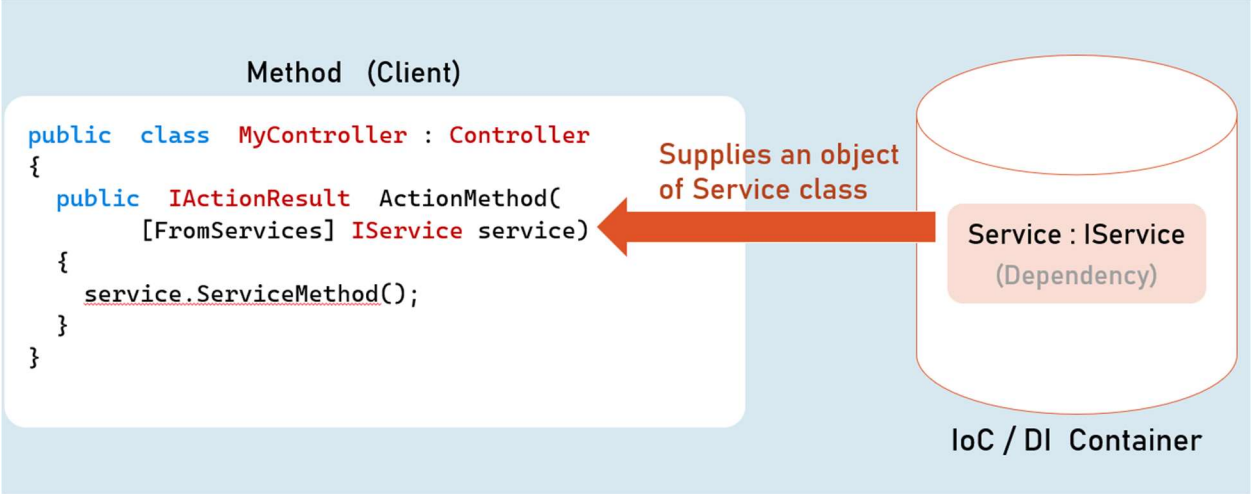
- Dependency injection (DI) is a design pattern, which is a technique for achieving "Inversion of Control (IoC)" between clients and their dependencies.
- It allows you to inject (supply) a concrete implementation object of a low-level component into a high-level component.
- The client class receives the dependency object as a parameter either in the constructor or in a method.





Method Injection



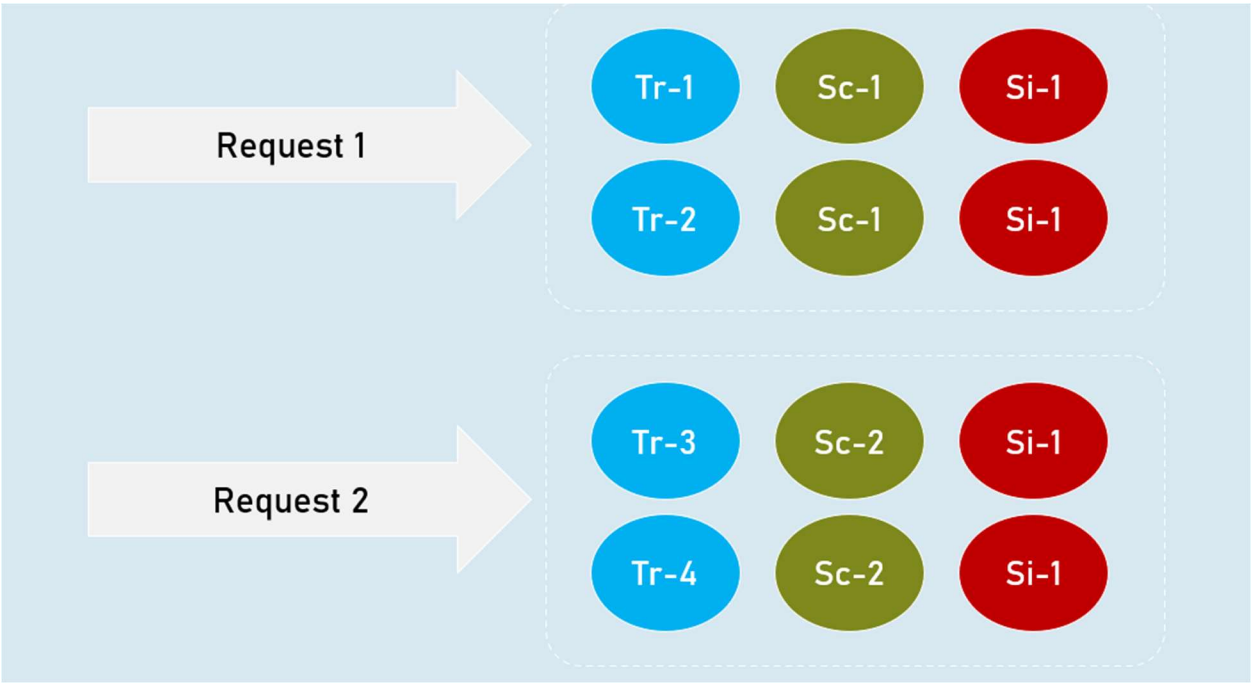


Service Lifetime

(Transient, Scoped, Singleton)

A service lifetime indicates when a new object of the service has to be created by the IoC / DI container.

1. **Transient:** Per injection
2. **Scoped:** Per scope (browser request)
3. **Singleton:** For entire application lifetime.



Transient

Transient lifetime service objects are created each time when they are injected.

Service instances are disposed at the end of the scope (usually, a browser request)

Scoped

Scoped lifetime service objects are created once per a scope (usually, a browser request).

Service instances are disposed at the end of the scope (usually, a browser request).

Singleton

Singleton lifetime service objects are created for the first time when they are requested.

Service instances are disposed at application shutdown.

Transient

```
builder.Services.AddTransient<IService, Service>(); //Transient Service
```

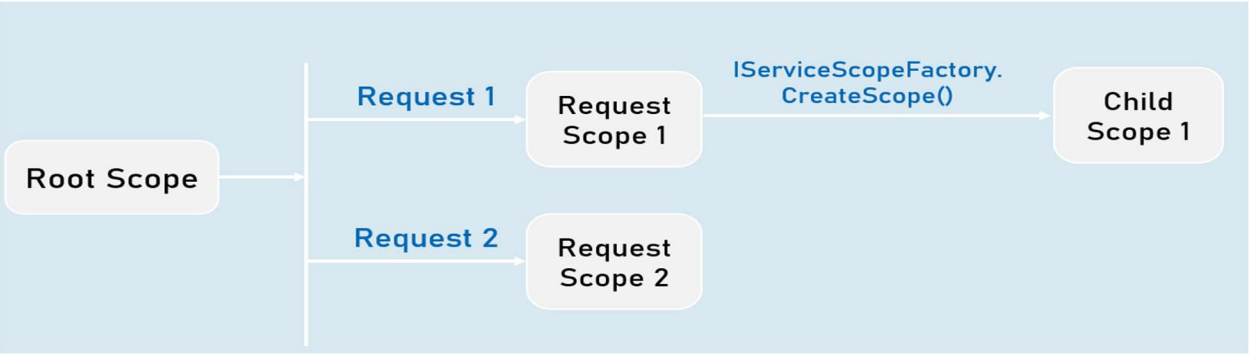
Scoped

```
builder.Services.AddScoped<IService, Service>(); //Scoped Service
```

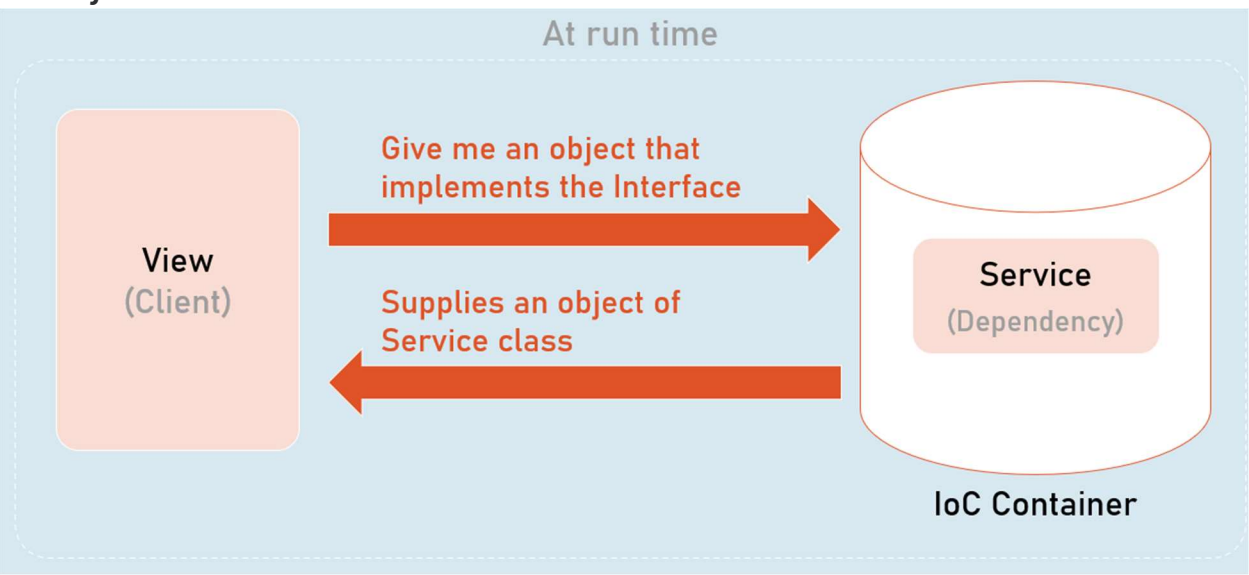
Singleton

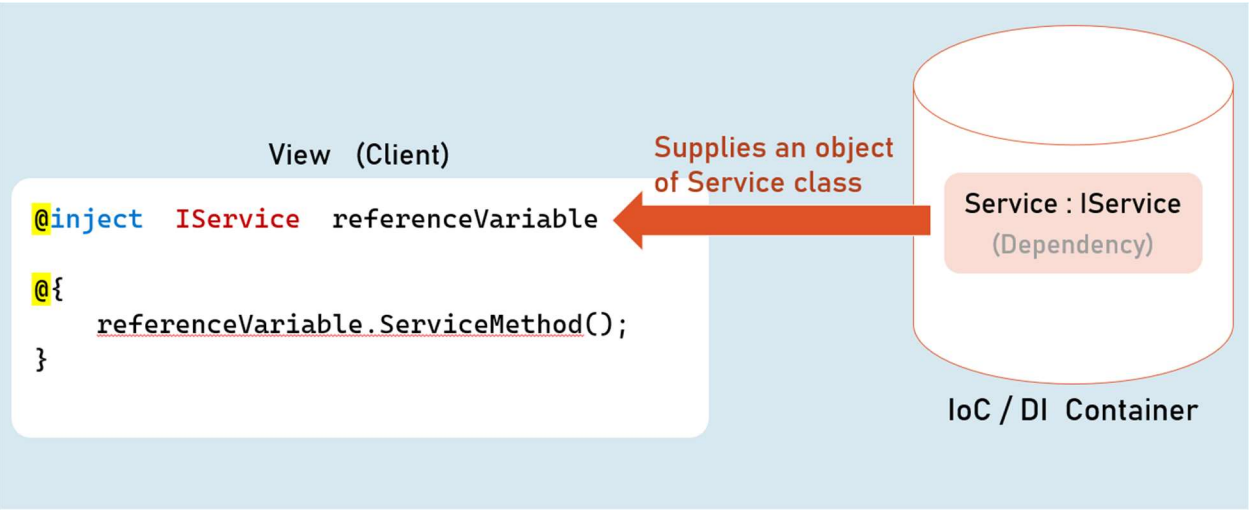
```
builder.Services.AddSingleton<IService, Service>(); //Singleton Service
```

Service Scope



View Injection





Best Practices in DI

Global state in services
Avoid using **static classes** to store some data globally for all users / all requests.
You may use **Singleton** services for simple scenarios / simple amount of data. In this case, prefer ConcurrentDictionary instead of Dictionary, which better handles concurrent access via multiple threads.
Alternatively, prefer to use **Distributed Cache / Redis** for any significant amount of data or complex scenarios.

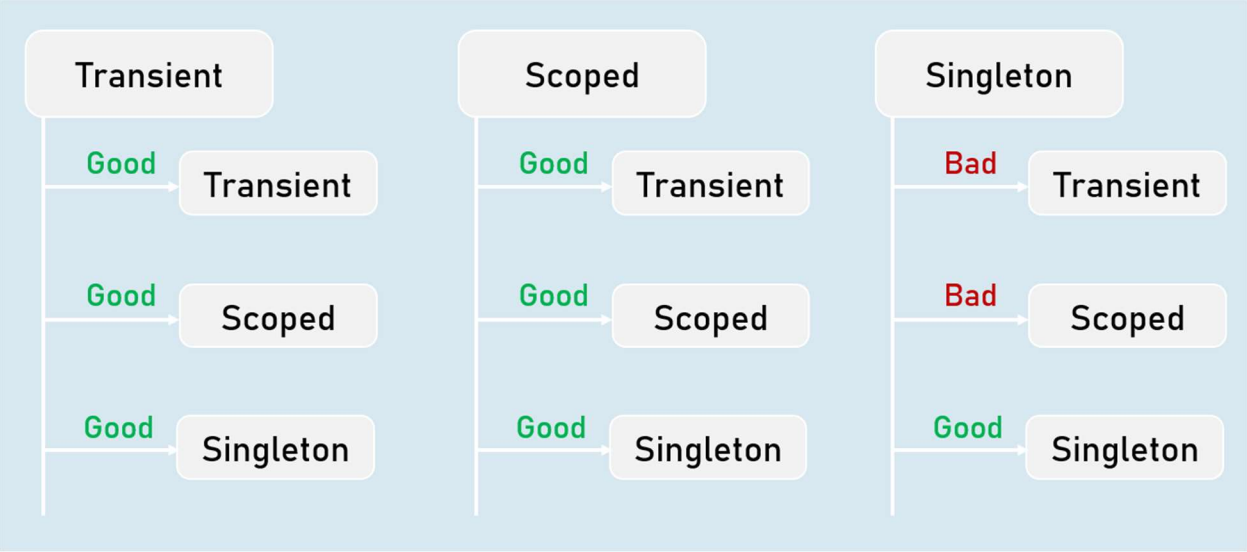
Request state in services
Don't use scoped services to share data among services within the same request, because they are NOT thread-safe.
Use **HttpContext.Items** instead.

Service Locator Pattern
Avoid using service locator pattern, without creating a child scope, because it will be harder to know about dependencies of a class.
For example, don't invoke **GetService()** in the default scope that is created when a new request is received.
But you can use the **IServiceScopeFactory.ServiceProvider. GetService()** within a child scope.

Calling Dispose() method
Don't invoke the Dispose() method manually for the services injected via DI.
The IoC container automatically invoke Dispose(), at the end of its scope.

Captive Dependencies
Don't inject scoped or transient services in singleton services.
Because, in this case, transient or scoped services act as singleton services, inside of singleton service.

Storing reference of service instance
Don't hold the reference of a resolved service object.
It may cause memory leaks and you may have access to a disposed service object.



Autofac

- Autofac is another IoC container library for .Net Core.
- Means, both are tightly-coupled.
- Microsoft.Extensions.DependencyInjection [vs] Autofac
- https://autofac.readthedocs.io/en/latest/getting-started/index.html

Microsoft.Extensions.DependencyInjection

- Built-in IoC container in asp.net core
- Lifetimes: Transient, Scoped, Singleton
- Metadata for services: Not supported
- Decorators: Not supported

Autofac

- Alternative to the Microsoft.Extensions
- Lifetimes: InstancePerDependency, InstancePerLifetimeScope, SingleInstance, InstancePerOwned, InstancePerMatchingLifetimeScope
- Metadata for services: Supported
- Decorators: Supported

Introduction to Environments

An environment represents is a system in which the application is deployed and executed.

Development

The environment, where the developer makes changes in the code, commits code to the source control.

Staging

The environment, where the application runs on a server, from which other developers and quality controllers access the application

Production

The environment, where the real end-users access the application.

Shortly, it's where the application "live" to the audience.

Environment Setting

Set Environment in launchSettings.json

in launchSettings.json

```
{
 "profiles":
 {
  "profileName":
  {
   "environmentVariables":
   {
    "DOTNET_ENVIRONMENT": "EnvironmentNameHere",
    "ASPNETCORE_ENVIRONMENT": "EnvironmentNameHere"
   }
  }
 }
}
```

Access Environment in Program.cs

app.Environment

IWebHostEnvironment

EnvironmentName

Gets or sets name of the environment.

By default it reads the value from either DOTNET_ENVIRONMENT or ASPNETCORE_ENVIRONMENT.

ContentRootPath

Gets or sets absolute path of the application folder.

IsDevelopment()

Returns Boolean true, if the current environment name is "Development".

IsStaging()

Returns Boolean true, if the current environment name is "Staging".

IsProduction()

Returns Boolean true, if the current environment name is "Production".

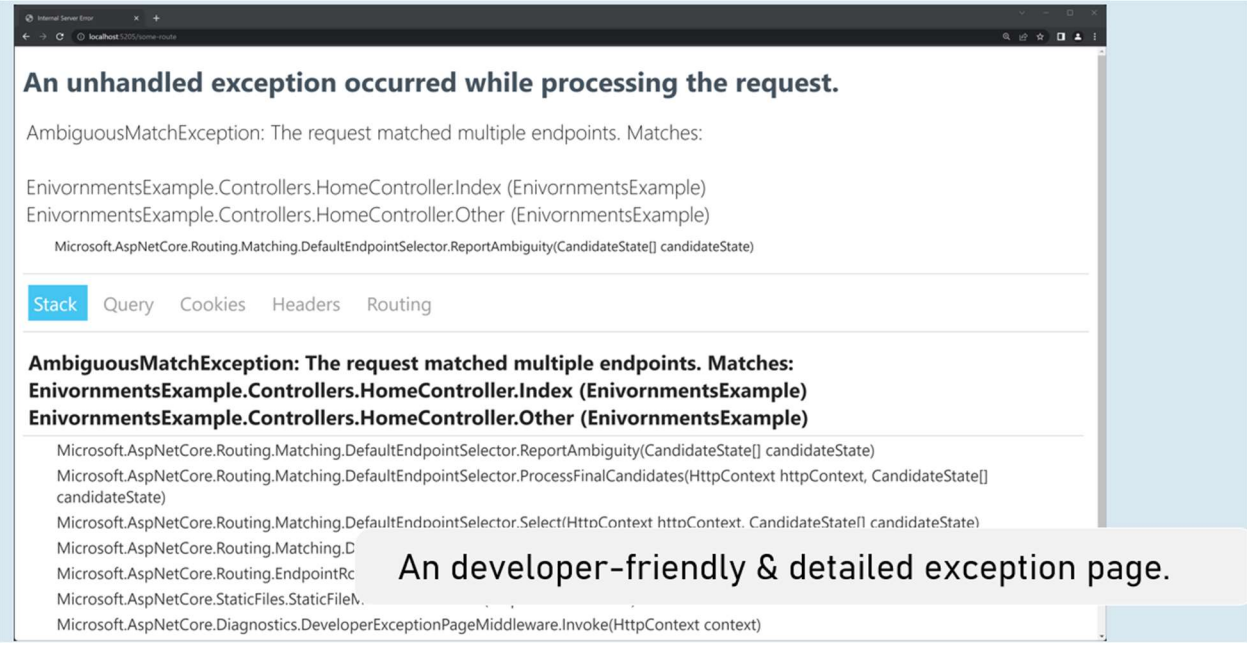
IsEnvironment(string environmentName)

Returns Boolean true, if the current environment name matches with the specified environment.

Access Environment in Controller and other classes

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Hosting;

public class ControllerName : Controller
{
 private readonly IWebHostEnvironment _webHost;

 public ControllerName(IWebHostEnvironment webHost)
 {
  _webHost = webHost;
 }
}
```

Developer Exception Page

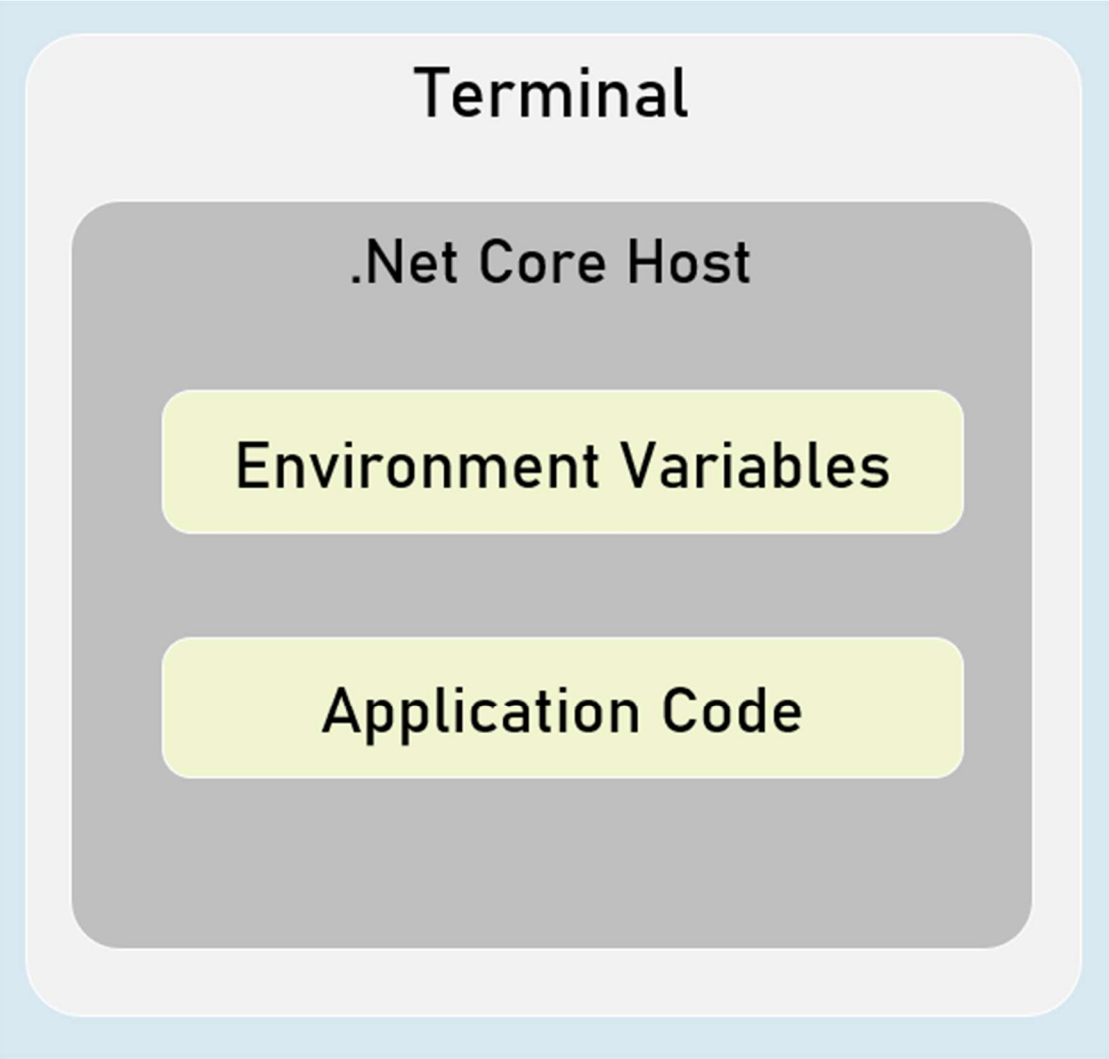


Enable developer exception page

in Program.cs

```
1. if (app.Environment.IsDevelopment()
2. {
3.  app.UseDeveloperExceptionPage();
4. }
```

Process-Level Environment



The environment variables are stored & accessible within the same process only.

Setting Environment Variables in Process

in "Windows PowerShell" / "Developer PowerShell in VS"

$Env:ASPNETCORE_ENVIRONMENT="EnvironmentName"

dotnet run --no-launch-profile

<environment> tag helper

include

```
<environment include="Environment1,Environment2">
  html content here
</environment>
```

It renders the content only when the current environment name matches with either of the specified environment names in the "include" property.

exclude

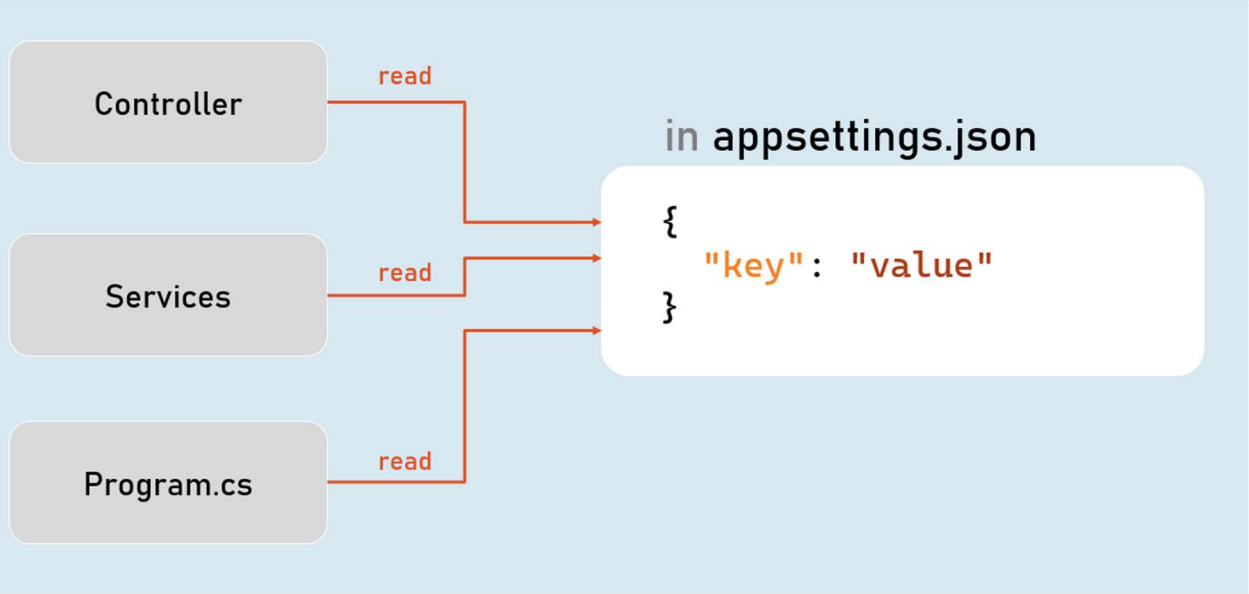
```
<environment exclude="Environment1,Environment2">
  html content here
</environment>
```

It renders the content only when the current environment name doesn't match with either of the specified environment names in the "exclude" property.

Configuration Settings

Configuration (or configuration settings) are the constant key/value pairs that are set at a common location and can be read from anywhere in the same application.

Examples: connection strings, Client ID & API keys to make REST-API calls, Domain names, Constant email addresses etc.



Configuration Sources

1. appsettings.json
2. Environment Variables
3. File Configuration (JSON, INI or XML files)
4. In-Memory Configuration
5. Secret Manager

Access Configuration

in Program.cs:

```
app.Configuration
```

IConfiguration

[string key]

Gets or sets configuration value at the specified key.

GetValue<T>(string key, object defaultValue)

Gets the configuration value at the specified key; returns the default value if the key doesn't exists.

IConfiguration in Controller

in Controller and other classes

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;

public class ControllerName : Controller
{
  private readonly IConfiguration _configuration;

  public ControllerName(IConfiguration configuration)
  {
   _configuration = configuration;
  }
}
```

Hierarchical Configuration

in appsettings.json

```
{
 "MasterKey":
 {
  "Key1": "value"
  "Key2": "value"
 }
}
```

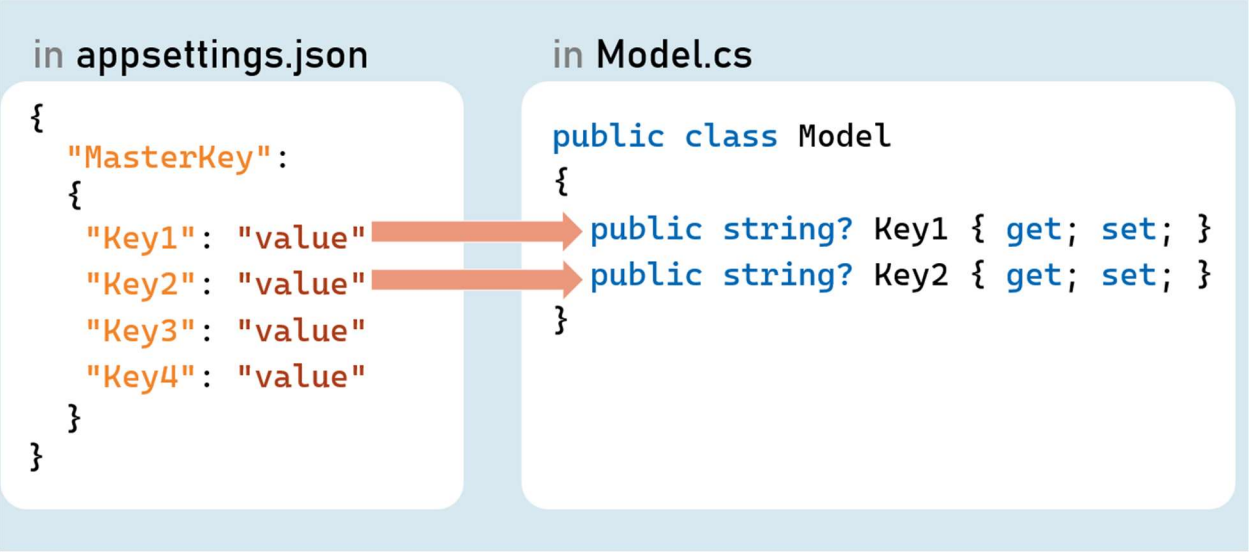
to read configuration

`Configuration["MasterKey:Key1"]`

IConfiguration.GetSection(string key)

Returns an IConfigurationSection based on the specified key.

Options Pattern



Options pattern uses custom classes to specify what configuration settings are to be loaded into properties.

Examples: Reading the specific connections strings out of many configuration settings.

The option class should be a non-abstract class with a public parameterless constructor.

Public read-write properties are bound.

Fields are not bound.

IConfiguration.GetSection(string key)

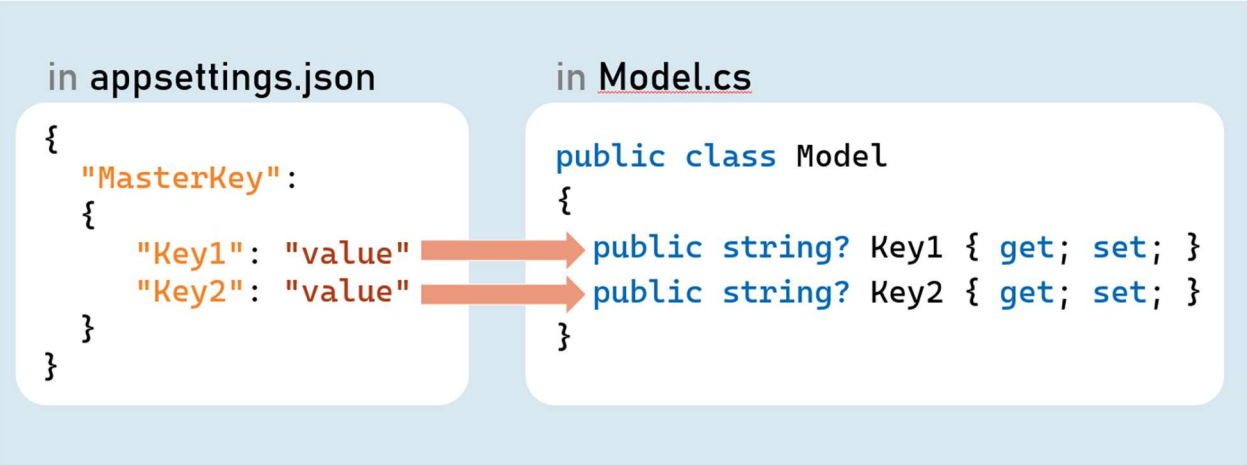
Returns an IConfigurationSection based on the specified key.

IConfiguration.Bind(object instance) and IConfiguration.Get<T>()

Binds (loads) configuration key/value pairs into a new object of the specified type.

Configuration as Service

Inject Configuration as Service



Add Configuration as Service

in Program.cs:

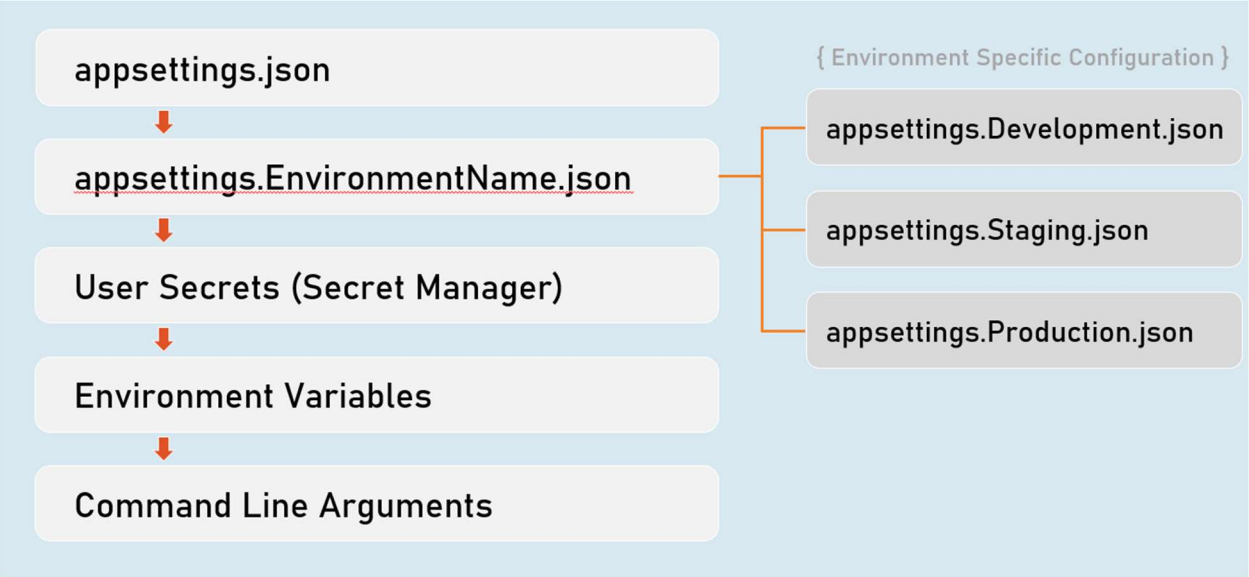
```
builder.Services.Configure<Model>(builder.Configuration.GetSectio
n("MasterKey"));
```

Inject Configuration as Service in Controller in Controller and other classes

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Options;

public class ControllerName : Controller
{
  private readonly Model _options;

  public ControllerName(IOptions<Model> options)
  {
   _options = options.Value;
  }
}
```

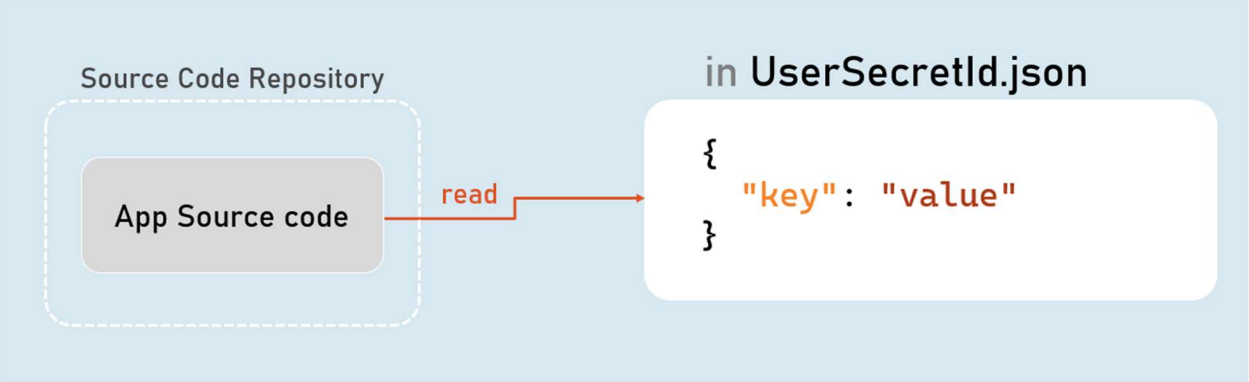
Environment Specific Configuration

Order of Precedence of Configuration Sources



Secrets Manager

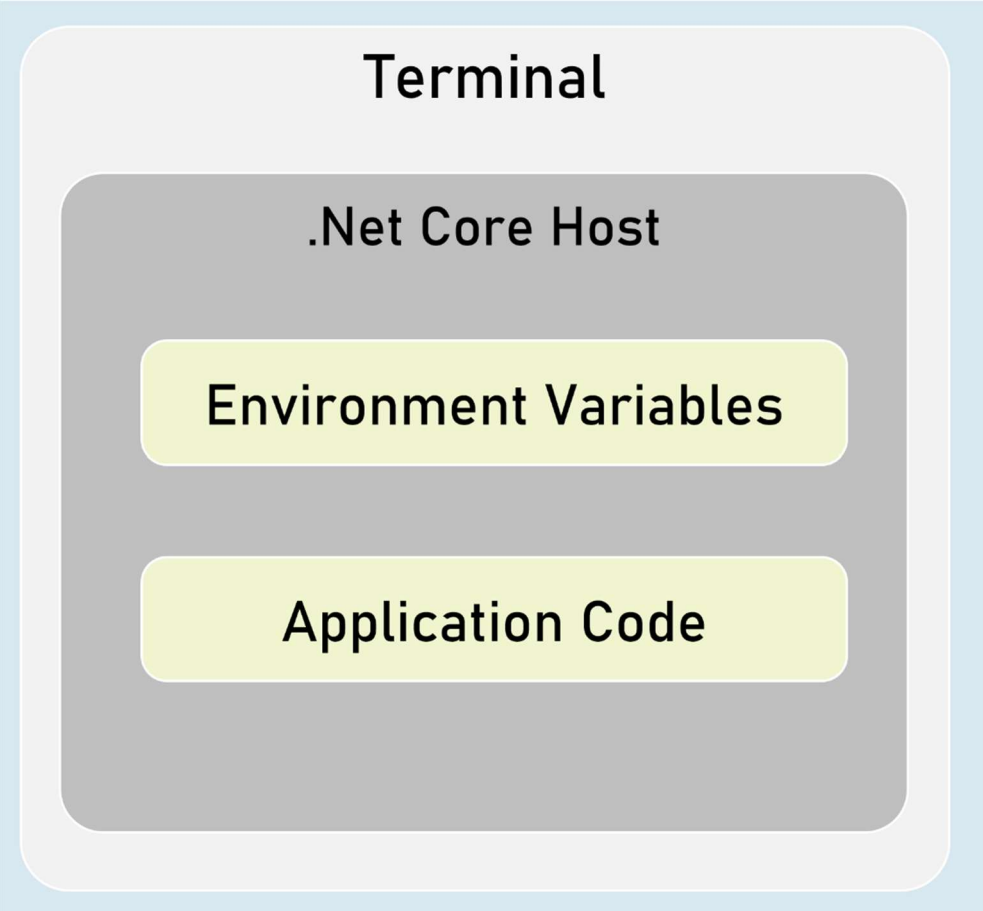
The 'secrets manager ' stores the user secrets (sensitive configuration data) in a separate location on the developer machine.



Enable Secrets Manager in "Windows PowerShell" / "Developer PowerShell in VS"

```
1. dotnet user-secrets init
2. dotnet user-secrets set "Key" "Value"
3. dotnet user-secrets list
```

Environment Variables Configuration



You can set configuration values as in-process environment variables.

Set Configuration as Environment Variables

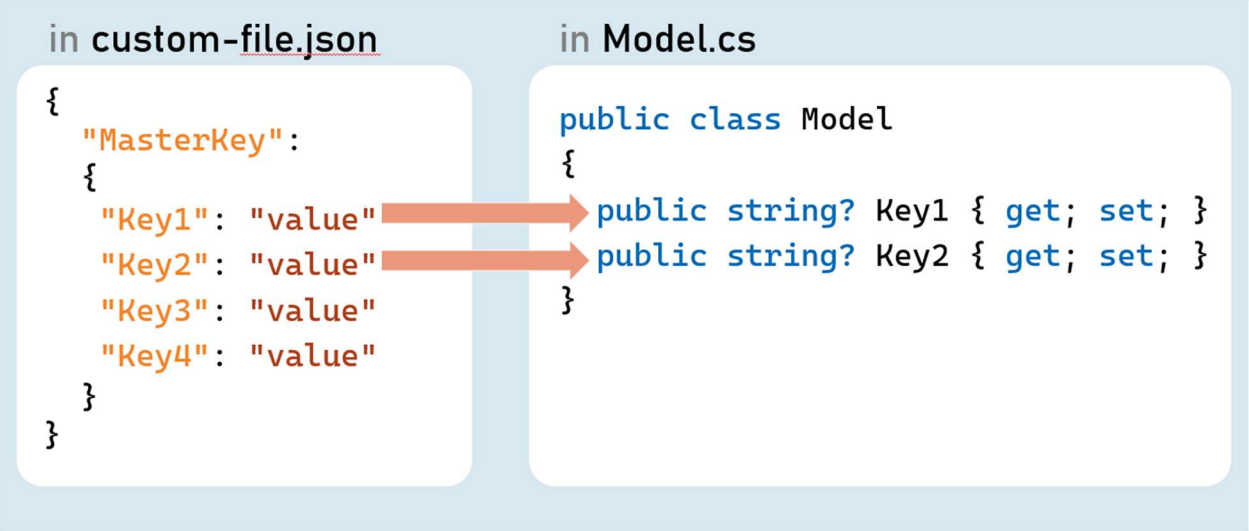
in "Windows PowerShell" / "Developer PowerShell in VS":

```
1. $Env:ParentKey__ChildKey="value"
2. dotnet run --no-launch-profile
```

It is one of the most secured way of setting-up sensitive values in configuration.

__ (underscore and underscore) is the separator between parent key and child key.

Custom Json Configuration



Add Custom Json file as Configuration Source

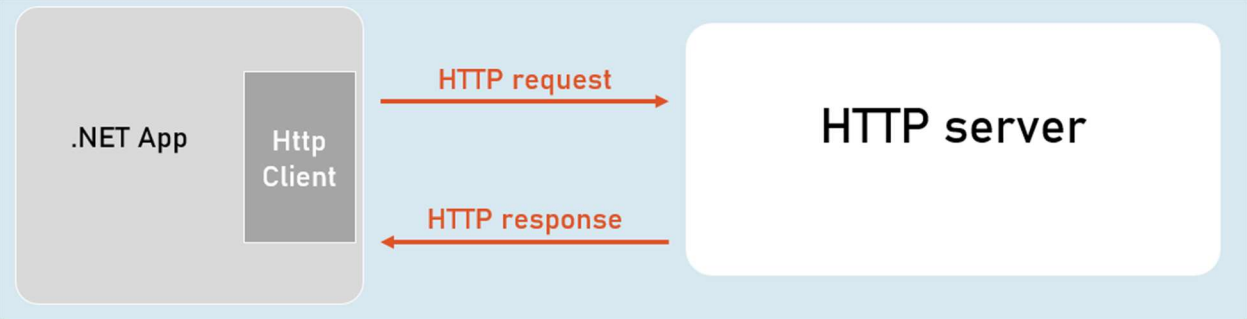
in Program.cs:

```
1. builder.Host.ConfigureAppConfiguration( (hostingContext, config) => {
2.     config.AddJsonFile("filename.json", optional: true, reloadOnChange: true);
3. });
```

Http Client

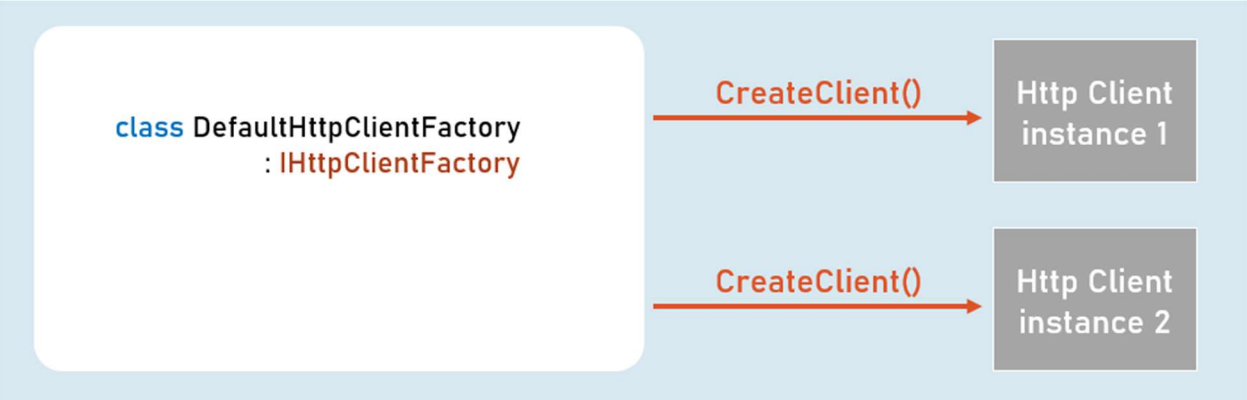
HttpClient is a class for sending HTTP requests to a specific HTTP resource (using its URL) and receiving HTTP responses from the same.

Examples: Making a request to a third-party weather API, ChatGPT etc.



IHttpClientFactory

IHttpClientFactory is an interface that provides a method called CreateClient() that creates a new instance of HttpClient class and also automatically disposes the same instance (closes the connection) immediately after usage.



HttpClient

Properties

- BaseAddress
- DefaultRequestHeaders

Methods

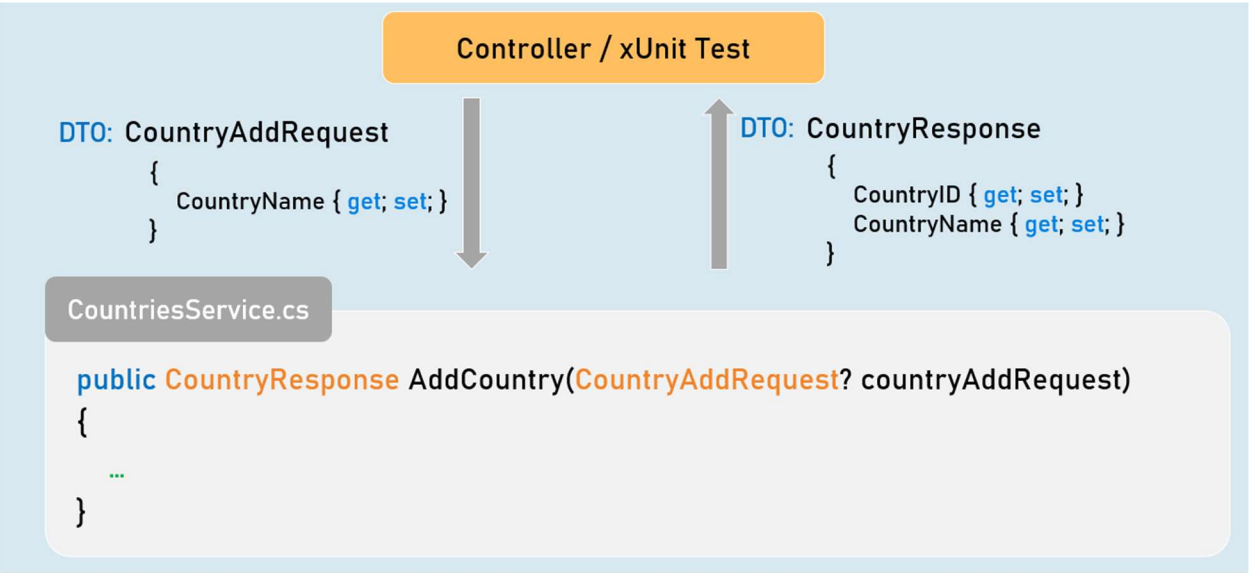
- GetAsync()
- PostAsync()
- PutAsync()
- DeleteAsync()

Introduction to xUnit

xUnit is the free, open source unit testing tool for .NET Framework.

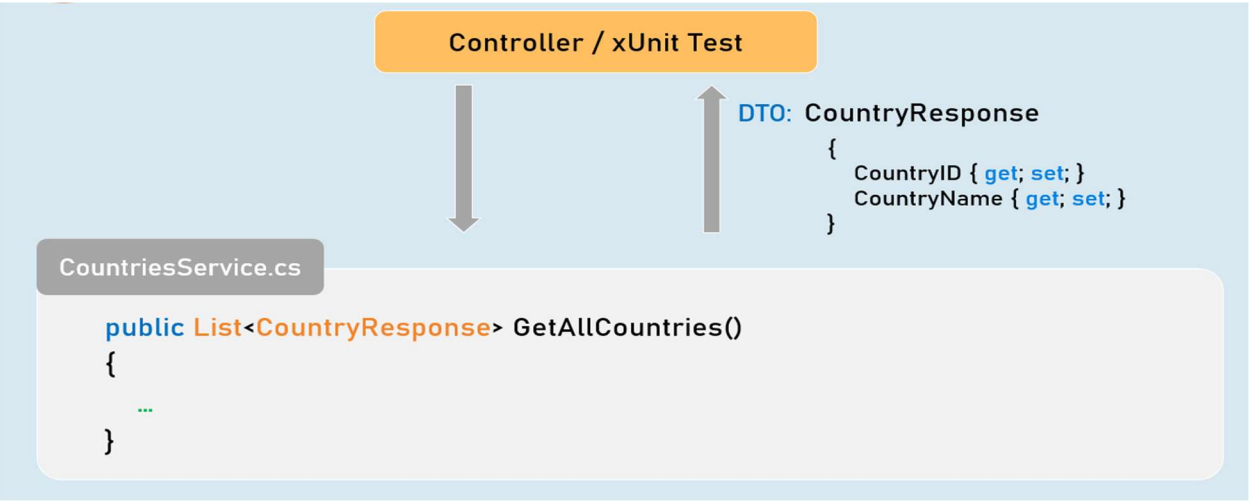
- Easy and extensible.
- Best to use with a mocking framework called "Moq".

Add Country - xUnit Test



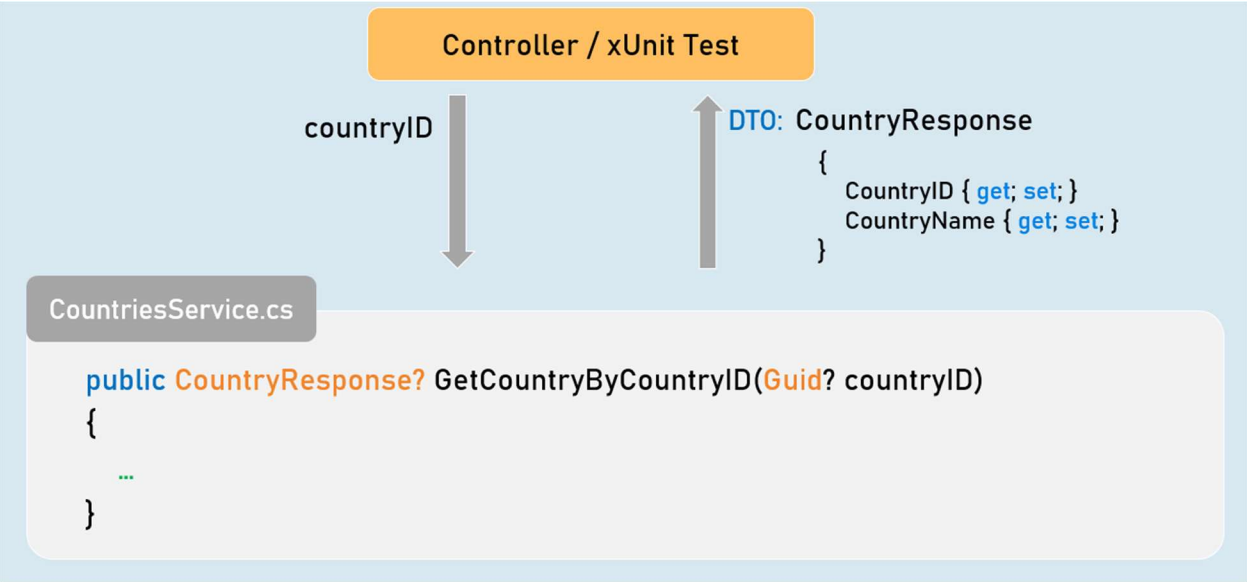

```
1. public CountryResponse AddCountry(CountryAddRequest? countryAddRequest)
2. {
3.  //Check if "countryAddRequest" is not null.
4.  //Validate all properties of "countryAddRequest"
5.  //Convert "countryAddRequest" from "CountryAddRequest" type to "Country".
6.  //Generate a new CountryID
7.  //Then add it into List<Country>
8.  //Return CountryResponse object with generated CountryID
9. }
```

Get All Countries - xUnit Test



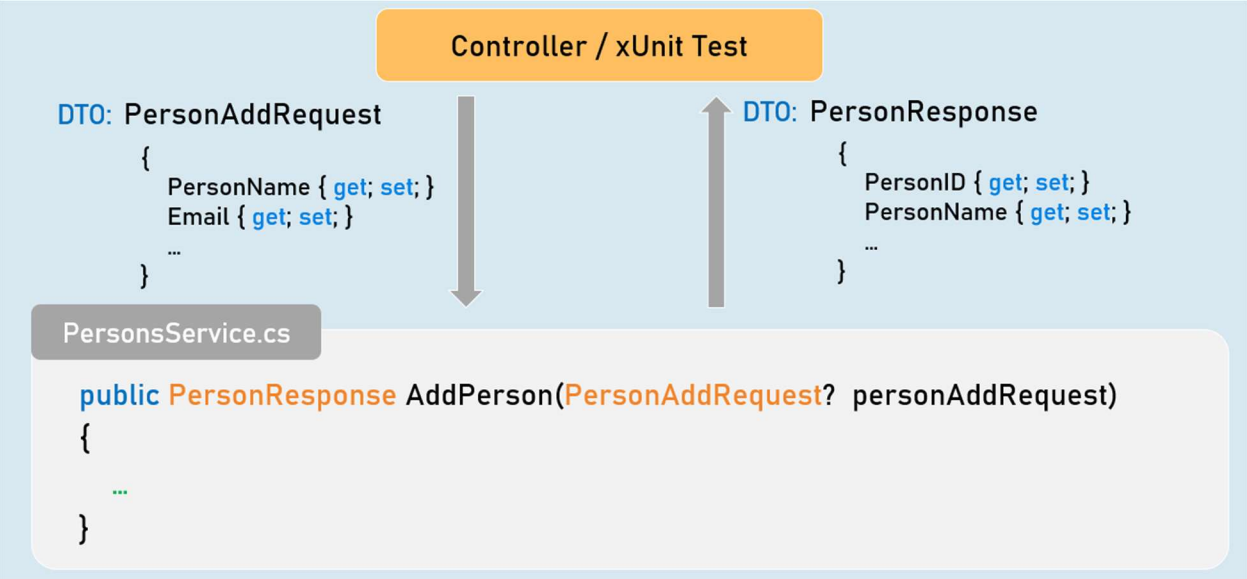
```
1. public List<CountryResponse> GetAllCountries()
2. {
3.  //Convert all countries from "Country" type to "CountryResponse" type.
4.  //Return all CountryResponse objects
5. }
```

Get Country by Country ID - xUnit Test




```
1. public CountryResponse? GetCountryByCountryID(Guid? countryID)
2. {
3.  //Check if "countryID" is not null.
4.  //Get matching country from List<Country> based countryID.
5.  //Convert matching country object from "Country" to "CountryResponse" type.
6.  //Return CountryResponse object
7. }
```

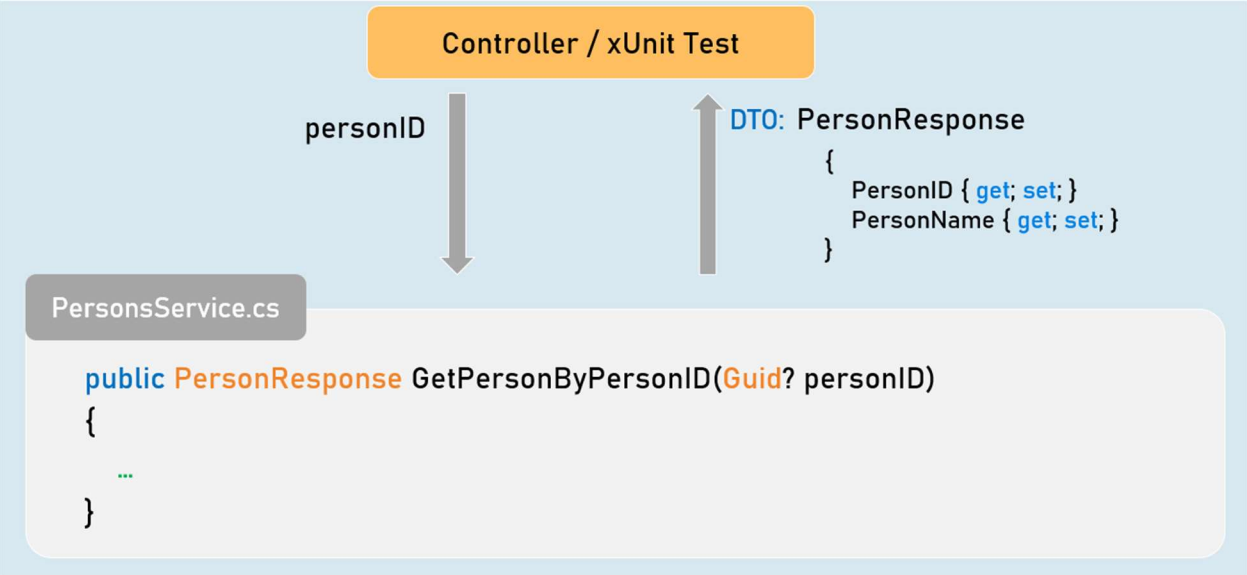
Add Person - xUnit Test




```
1. public PersonResponse AddPerson(PersonAddRequest? personAddRequest)
```

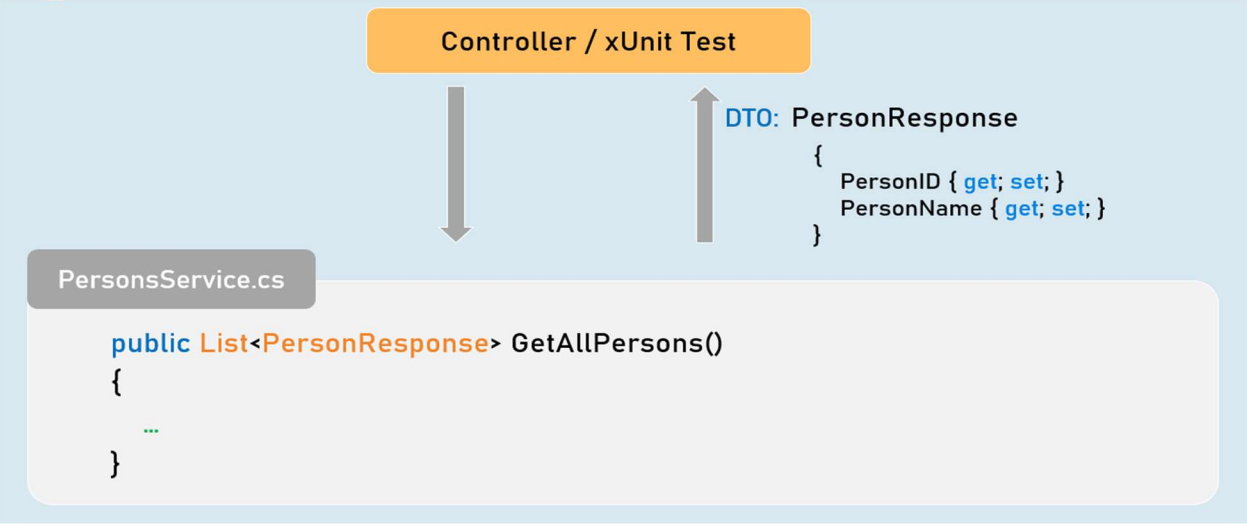
```
{
 //Check if "personAddRequest" is not null.
 //Validate all properties of "personAddRequest".
 //Convert "personAddRequest" from "PersonAddRequest" type to "Person".
 //Generate a new PersonID.
 //Then add it into List<Person>.
 //Return PersonResponse object with generated PersonID.
}
```

Get Person by Person ID - xUnit Test



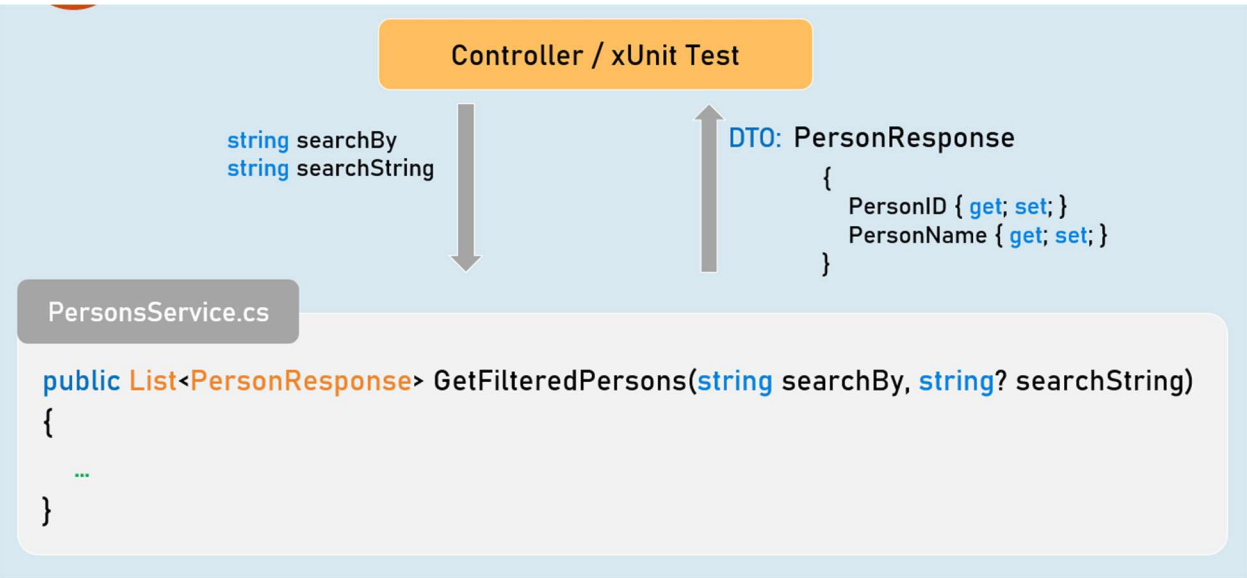

```
public PersonResponse GetPersonByPersonID(Guid? personID)
{
 //Check if "personID" is not null.
 //Get matching person from List<Person> based personID.
 //Convert matching person object from "Person" to "PersonResponse" type.
 //Return PersonResponse object
}
```

Get All Persons - xUnit Test



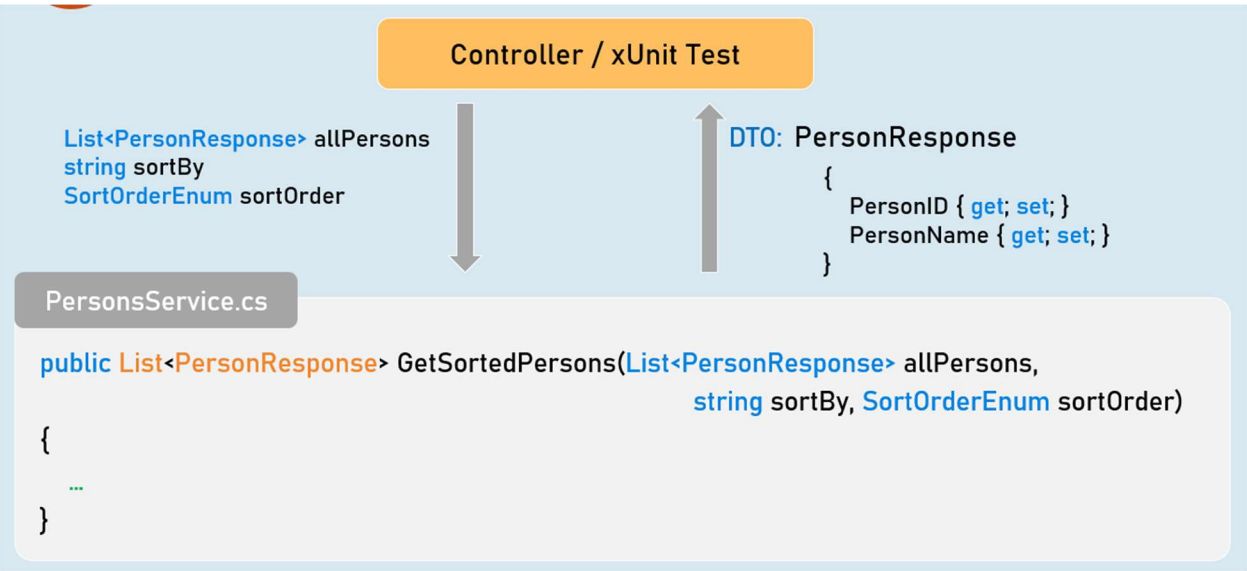
```
1. public List<PersonResponse> GetAllPersons()
2. {
3.  //Convert all persons from "Person" type to "PersonResponse" type.
4.  //Return all PersonResponse objects
5. }
```

Get Filtered Persons - xUnit Test



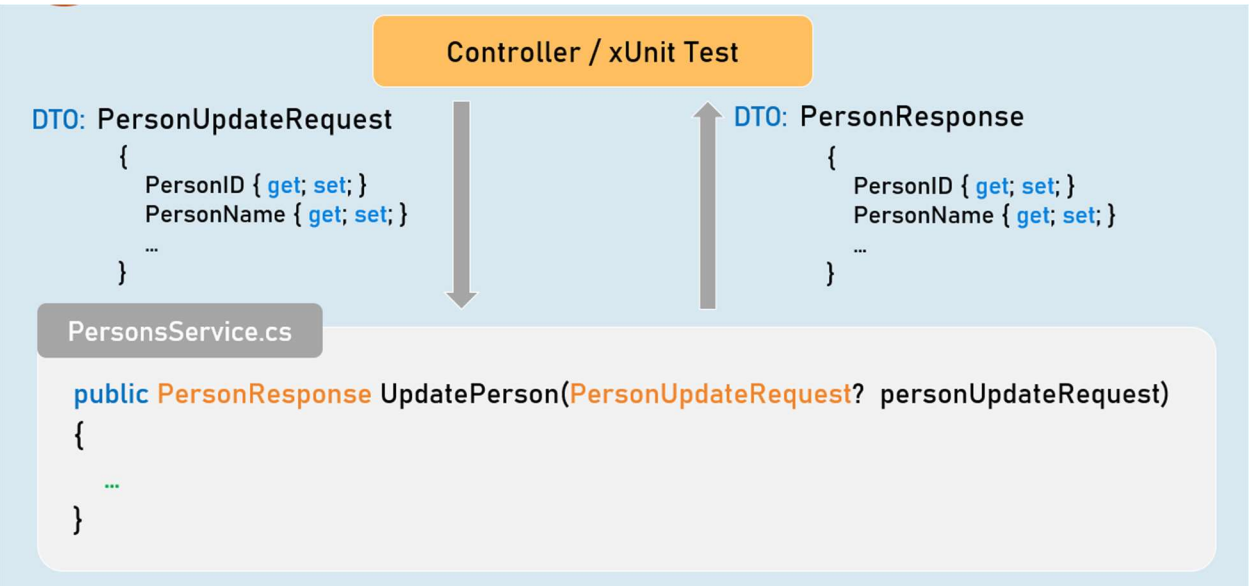

```
1. public List<PersonResponse> GetFilteredPersons(string searchBy, string?
   searchString)
2. {
3.  //Check if "searchBy" is not null.
4.  //Get matching persons from List<Person> based on given searchBy and
   searchString.
5.  //Convert the matching persons from "Person" type to "PersonResponse" type.
6.  //Return all matching PersonResponse objects
7. }
```

Get Sorted Persons - xUnit Test



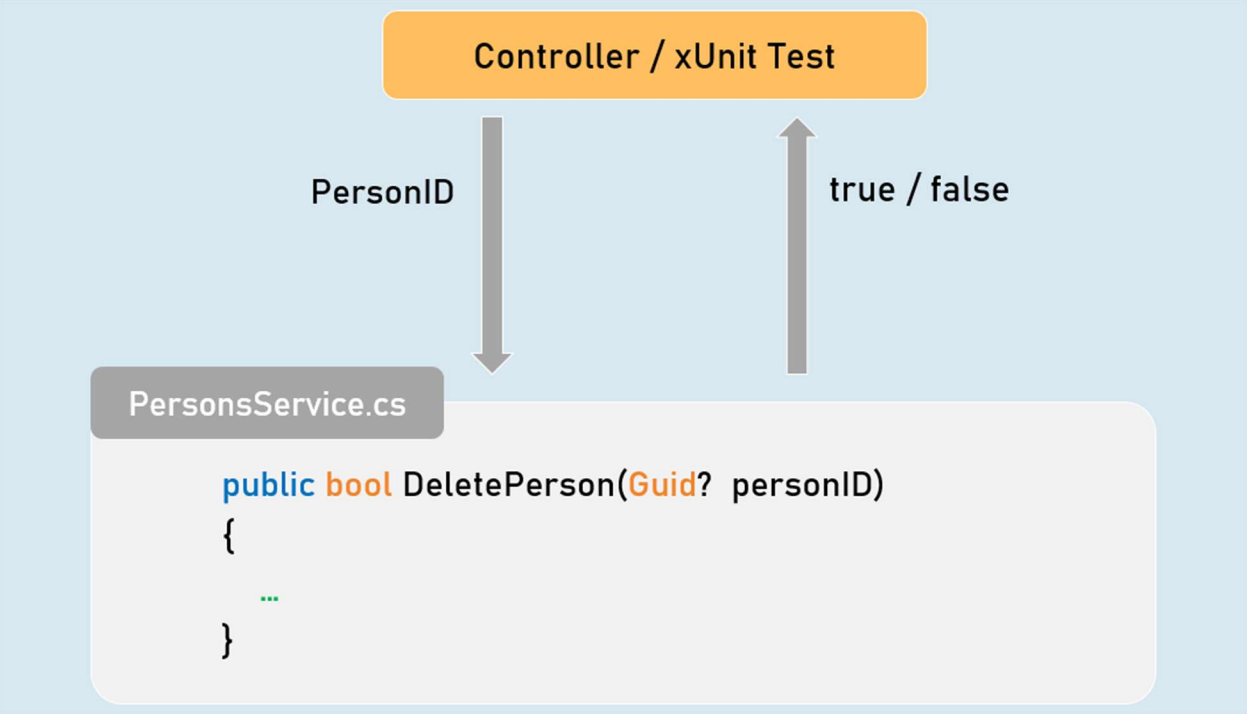
```
1. public List<PersonResponse> GetSortedPersons(List<PersonResponse> allPersons,
2. string sortBy, SortOrderEnum sortOrder)
3. {
4.  //Check if "sortBy" is not null.
5.  //Get sorted persons from "allPersons" based on given "sortBy" and
   "sortOrder".
6.  //Convert the sorted persons from "Person" type to "PersonResponse" type.
7.  //Return all sorted PersonResponse objects
8. }
```

Update Person - xUnit Test




```
1. public PersonResponse UpdatePerson(PersonUpdateRequest? personUpdateRequest)
2. {
3.  //Check if "personUpdateRequest" is not null.
4.  //Validate all properties of "personUpdateRequest"
5.  //Get the matching "Person" object from List<Person> based on PersonID.
6.  //Check if matching "Person" object is not null
7.  //Update all details from "PersonUpdateRequest" object to "Person" object
8.  //Convert the person object from "Person" to "PersonResponse" type
9.  //Return PersonResponse object with updated details
10.}
11.
```

Delete Person - xUnit Test

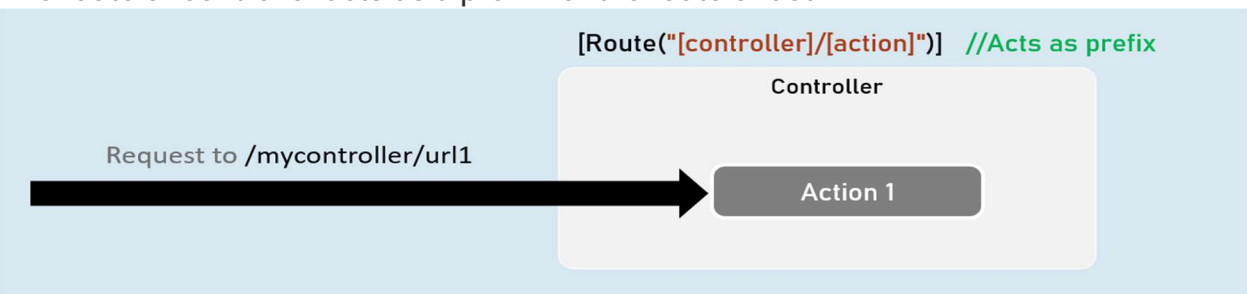



```
1. public bool DeletePerson(Guid? personID)
2. {
3.  //Check if "personID" is not null.
4.  //Get the matching "Person" object from List<Person> based on PersonID.
5.  //Check if matching "Person" object is not null
6.  //Delete the matching "Person" object from List<Person>
7.  //Return Boolean value indicating whether person object was deleted or not
8. }
```

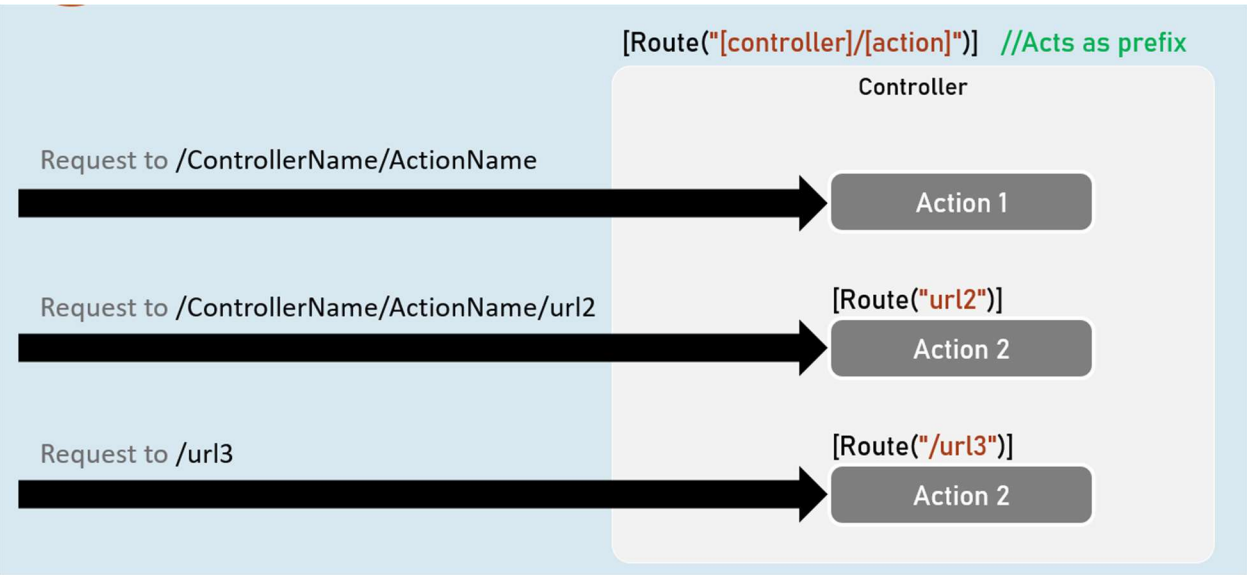
Route Tokens

The route tokens [controller], [action] can be used to apply common-patterned routes for all action methods.

The route of controller acts as a prefix for the route of acti



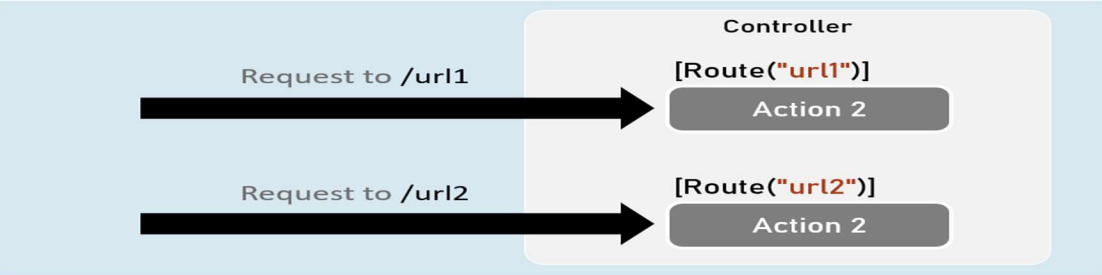
ons

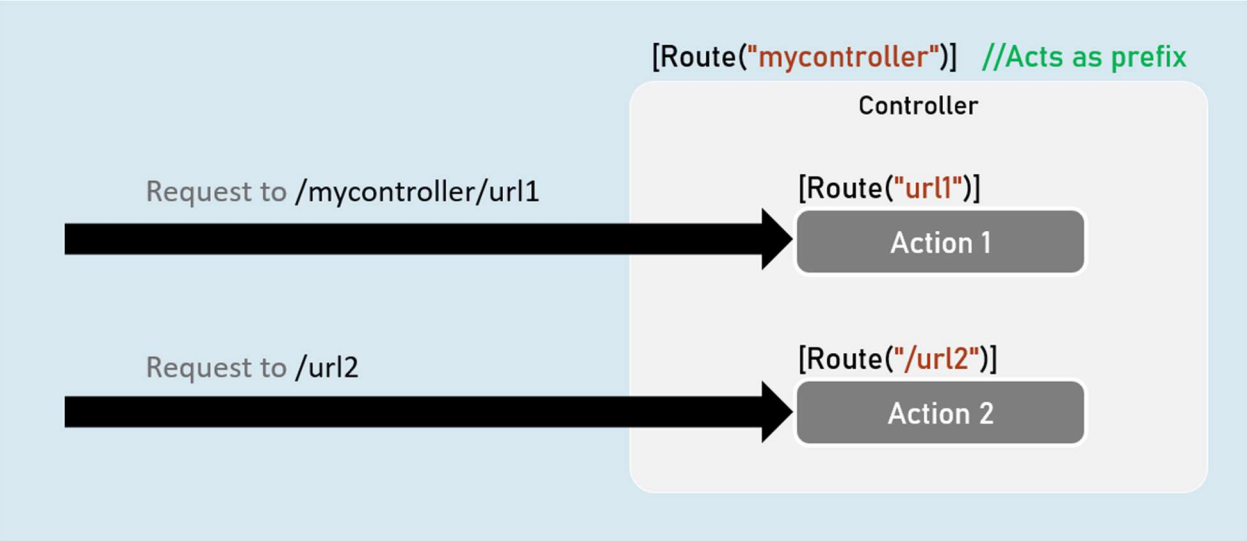


Attribute Routing

[Route] attribute specifies route for an action method or controller.
The route of controller acts as a prefix for the route of actions.

[Route] - Attribute Routing





Tag Helpers

Tag helpers are the classes that can be invoked as an html tag or html attribute.
They generate a html tag or adds values into attributes of existing html tags.

```
1. <input asp-for="ModelProperty">
2. <input type="text" name="ModelProperty" id="ModelProperty" value="ModelValue"
   />
```

Tag Helpers for <a>, <form>

- asp-controller
- asp-action
- asp-route-x
- asp-route
- asp-area

Tag Helpers for <input>, <textarea>, <label>

- asp-for

Tag Helpers for <select>

- asp-for
- asp-items

Binding

The form tags such as <input>, <label>, <textarea>, <select> can be bound with specific model properties.

It applies model property name to "name" and "id" attributes of <input> tag.

Url Generation

Route URLs' will be re-generated for <a> and <form> tags;

It generates the url as "controller/action" pattern.

Tag Helpers for <img>

- asp-append-version

**Tag Helpers for **

- asp-validation-for

Tag Helpers for <script>

- asp-fallback-src
- asp-fallback-test

**Tag Helpers for **

- asp-validation-summary

Tag Helpers for <form>

```
<form asp-controller="ControllerName" asp-action="ActionName">
</form>
```

will be converted as:

```
<form action="~/ControllerName/ActionName">
</form>
```

asp-controller and asp-action

Generates route url for the specified action method with "controller/action" route pattern.

Tag Helpers for <a>

```
<a asp-controller="ControllerName" asp-action="ActionName"> </a>
```

will be converted as:

```
<a href="~/ControllerName/ActionName"> </a>
```

asp-controller and asp-action

Generates route url for the specified action method with "controller/action" route pattern.

Tag Helpers for <a> and <form>

```
<a asp-controller="ControllerName" asp-action="ActionName" asp-route-parameter="value" > </a>
```

will be converted as:

```
<a href="~/ControllerName/ActionName/value-of-parameter"> </a>
```

asp-route-x

Specifies value for a route parameter, which can be a part of the route url.

Tag Helpers for <input>, <textarea>, <select>

```
<input asp-for="ModelProperty" />
```

will be converted as:

```
<input type="text" name="ModelProperty" id="ModelProperty"
value="ModelValue" data-val-rule="ErrorMessage" />
```

asp-for

Generates "type", "name", "id", "data-validation" attributes for the <input>, <textarea>, <select> tags.

Tag Helpers for <label>

```
<label asp-for="ModelProperty"> </label>
```

will be converted as:

```
<label for="ModelProperty"> </label>
```

asp-for

Generates "for" attribute for the <label>.

Client Side Validations

Data annotations on model properties

```
1. [Required]
2. publicDataTypePropertyName { get; set; }
```

"data-*" attributes in html tags [auto-generated with "asp-for" helper]

```
<input data-val="true" data-required="ErrorMessage" />
```

Import jQuery Validation Scripts

```
1. https://cdnjs.cloudflare.com/ajax/libs/ jquery/3.6.0/jquery.min.js
2. https://cdnjs.cloudflare.com/ajax/libs/ jquery-
   validate/1.19.3/jquery.validate.min.js
```

```
3. https://cdnjs.cloudflare.com/ajax/libs/ jquery-validation-unobtrusive/3.2.12/
   jquery.validate.unobtrusive.min.js
```

Tag Helpers for <img>

```
1. <img src="~/FilePath" asp-append-version="true" />
2. <img src="/FilePath?v=HashOfImageFile" />
```

asp-append-version

- Generates SHA512 hash of the image file as query string parameter appended to the file path.
- It REGENERATES a new hash every time when the file is changed on the server. If the same file is requested multiple times, file hash will NOT be regenerated.

Tag Helpers for <script>

```
1. <script src="CDNUrl" asp-fallback-src="~/LocalUrl" asp-fallback-test="object">
   </script>
2. <script src="CDNUrl"> </script>
3. <script> object || document.write("<script src='/LocalUrl'></script>");
   </script>
```

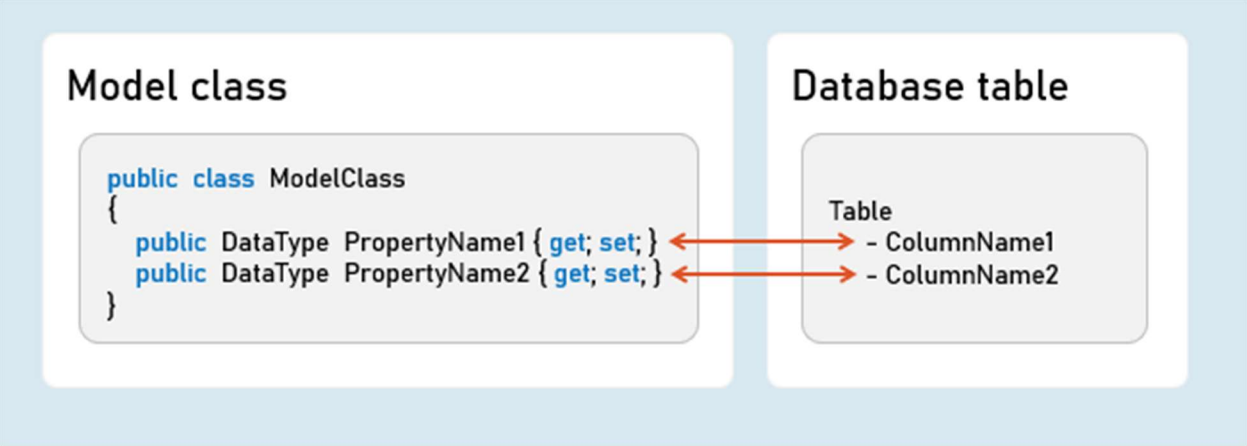
asp-fallback-src

- It makes a request to the specified CDNUrl at the "src" attribute.
- It checks the value of the specified object at the "asp-fallback-test" tag helper.
- If its value is null or undefined (means, the script file at CDNUrl is not loaded), then it makes another request to the LocalUrl through another script tag.

Introduction to EntityFrameworkCore

EntityFrameworkCore is light-weight, extensible and cross-platform framework for accessing databases in .NET applications.
It is the most-used database framework for Asp.Net Core Apps.

EFCore Models



Pros & Cons of EntityFrameworkCore

Shorter Code

The CRUD operations / calling stored procedures are done with shorter amount of code than ADO.NET.

Performance

EFCore performs slower than ADO.NET.

So ADO.NET or its alternatives (such as Dapper) are recommended for larger & high-traffic applications.

Strongly-Typed

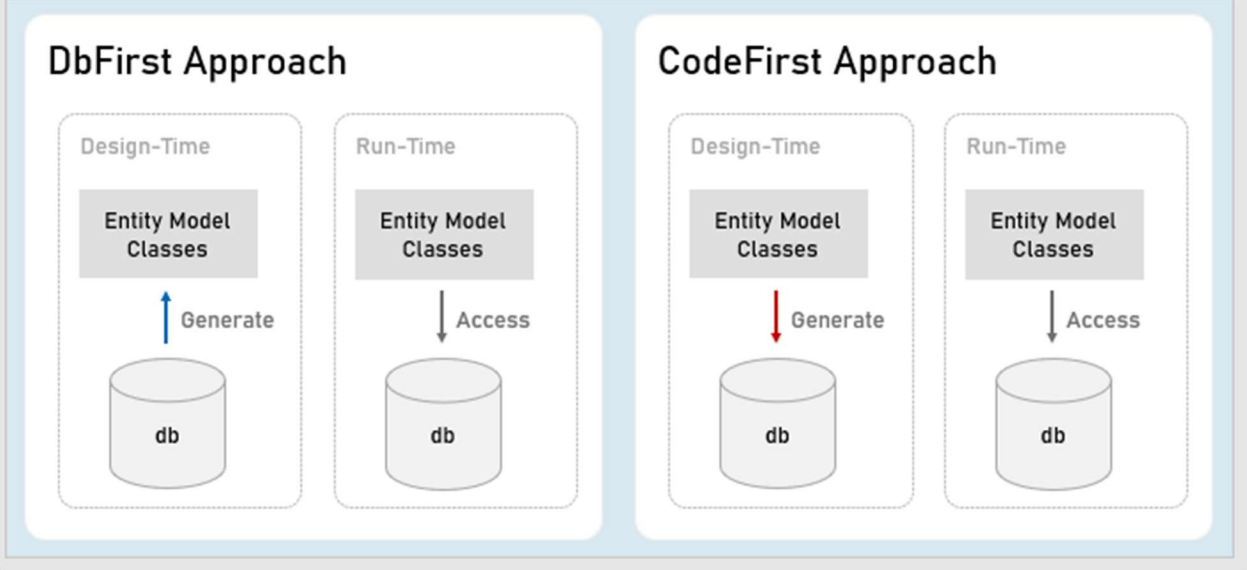
The columns as created as properties in model class.

So the Intellisense offers columns of the table as properties, while writing the code.

Plus, the developer need not convert data types of values; it's automatically done by EFCore itself.

Approaches in Entity Framework Core

EFCore Approaches



Pros and Cons of EFCore Approaches

CodeFirst Approach

Suitable for newer databases.

Manual changes to DB will be most probably lost because your code defines the database.

Stored procedures are to be written as a part of C# code.

Suitable for smaller applications or prototype-level applications only; but not for larger or high data-intense applications.

DbFirst Approach

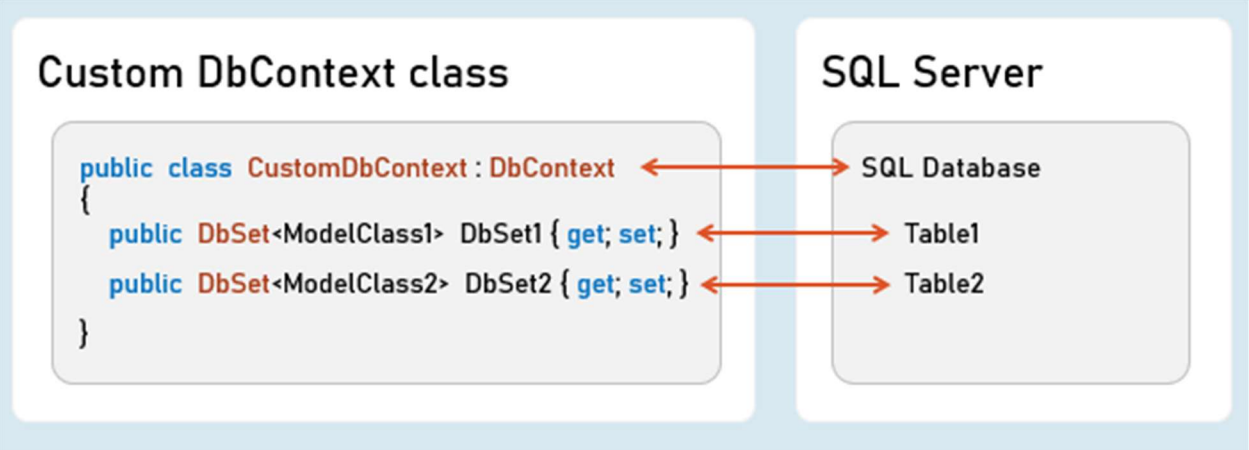
Suitable if you have an existing database or DB designed by DBAs, developed separately.

Manual changes to DB can be done independently.

Stored procedures, indexes, triggers etc., can be created with T-SQL independently.

Suitable for larger applications and high data-intense applications.

DbContext and DbSet



DbContext

An instance of DbContext is responsible to hold a set of DbSets' and represent a connection with database.

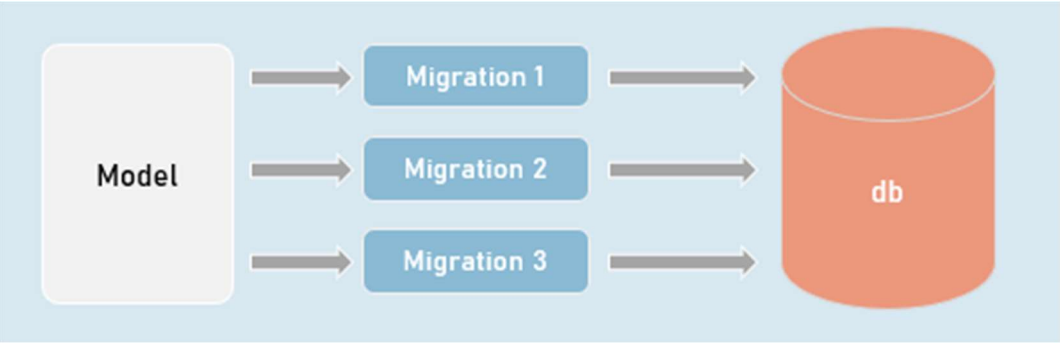
DbSet

Represents a single database table; each column is represented as a model property.

Add DbContext as Service in Program.cs:

```
builder.Services.AddDbContext<DbContextClassName>( options => {
 options.UseSqlServer();
}
);
```

Code-First Migrations



Migrations

Creates or updates database based on the changes made in the model.

in **Package Manager Console (PMC):**

`Add-Migration MigrationName`

//Adds a migration file that contains C# code to update the database

`Update-Database -Verbose`

//Executes the migration; the database will be created or table schema gets updated as a result.

Seed Data

in DbContext:

`modelBuilder.Entity<ModelClass>().HasData(entityObject);`

It adds initial data (initial rows) in tables, when the database is newly created

EF CRUD Operations - Query

SELECT - SQL

```
SELECT Column1, Column2 FROM TableName
 WHERE Column = value
 ORDER BY Column
```

LINQ Query:

```
_dbContext.DbSetName
 .Where(item => item.Property == value)
 .OrderBy(item => item.Property)
 .Select(item => item);

//Specifies condition for where clause
//Specifies condition for 'order by' clause
//Expression to be executed for each row
```

EF CRUD Operations - Insert

INSERT - SQL

`INSERT INTO TableName(Column1, Column2) VALUES (Value1, Value2)`

Add:

```
_dbContext.DbSetName.Add(entityObject);
//Adds the given model object (entity object) to the DbSet.
```

SaveChanges()

```
1. _dbContext.SaveChanges();
2. //Generates the SQL INSERT statement based on the model object data and
   executes the same at database server.
```

EF CRUD Operations - Delete

DELETE - SQL

```
DELETE FROM TableName WHERE Condition
```

Remove:

```
1. _dbContext.DbSetName.Remove(entityObject);
2. //Removes the specified model object (entity object) to the DbSet.
```

SaveChanges()

```
1. _dbContext.SaveChanges();
2. //Generates the SQL DELETE statement based on the model object data and
   executes the same at database server.
```

EF CRUD Operations - Update

UPDATE - SQL

```
UPDATE TableName SET Column1 = Value1, Column2 = Value2 WHERE
PrimaryKey = Value
```

Update:

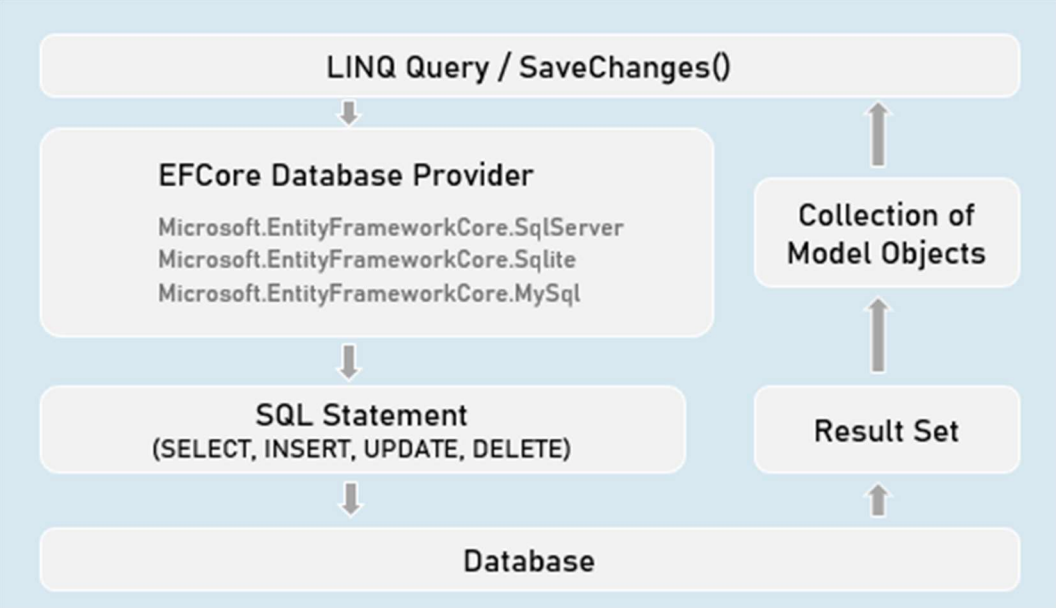
```
1. entityObject.Property = value;
2. //Updates the specified value in the specific property of the model object
   (entity object) to the DbSet.
```

SaveChanges()

```
1. _dbContext.SaveChanges();
2. //Generates the SQL UPDATE statement based on the model object data and
   executes the same at database server.
```

How EF Query Works?

Workflow of Query Processing in EF



EF - Calling Stored Procedures

Stored Procedure for CUD (INSERT | UPDATE | DELETE):

```
int DbContext.Database.ExecuteSqlRaw(
 string sql,
 params object[] parameters)

//Eg: "EXECUTE [dbo].[StoredProcName] @Param1 @Parm2
//A list of objects of SqlParameter type
```

Stored Procedure for Retrieving (Select):

```
IQueryable<Model> DbSetName.FromSqlRaw(
 string sql,
 paramsobject[] parameters)

//Eg: "EXECUTE [dbo].[StoredProcName] @Param1 @Parm2"
//A list of objects of SqlParameter type
```

Creating Stored Procedure (SQL Server)

```
CREATE PROCEDURE [schema].[procedure_name]
(@parameter_name data_type, @parameter_name data_type)
AS BEGIN
 statements
END
```

Advantages of Stored Procedure

Single database call

You can execute multiple / complex SQL statements with a single database call.

As a result, you'll get:

- Better performance (as you reduce the number of database calls)
- Complex database operations such as using temporary tables / cursors becomes easier.

Maintainability

The SQL statements can be changed easily WITHOUT modifying anything in the application source code (as long as inputs and outputs doesn't change)

[Column] Attribute

Model class

```
1. public class ModelClass
2. {
3.   [Column("ColumnName", TypeName = "datatype")]
4.   public DataType PropertyName { get; set; }
5.
6.   [Column("ColumnName", TypeName = "datatype")]
7.   publicDataTypePropertyName { get; set; }
8. }
```

Specifies column name and data type of SQL Server table.

EF - Fluent API

DbContext class

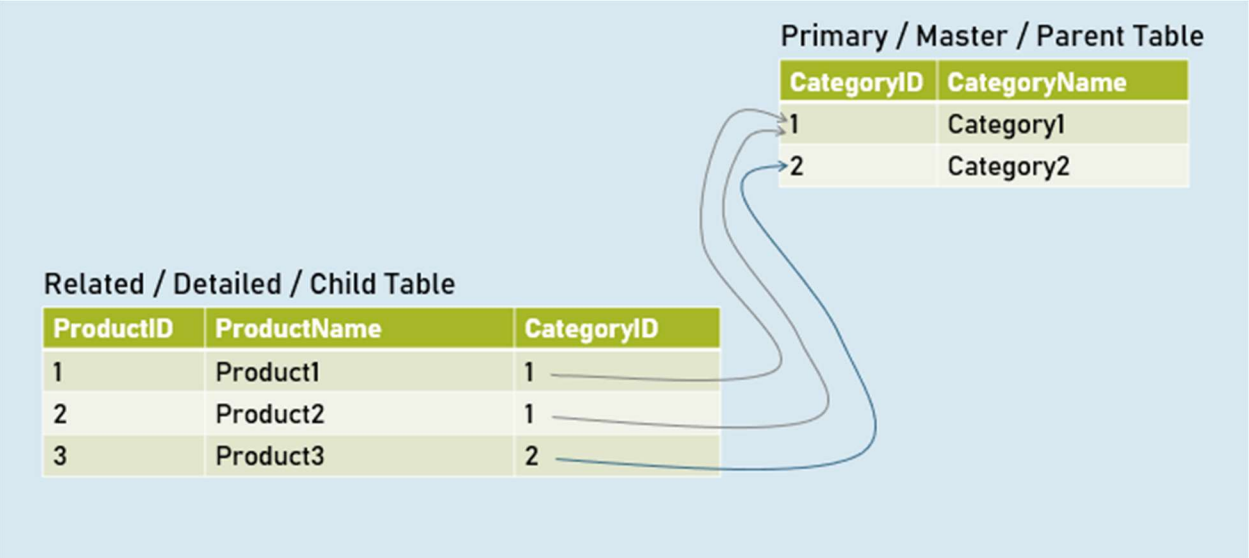
```
1. public class CustomDbContext : DbContext
2. {
3.   protected override void OnModelCreating(ModelBuilder modelBuilder)
4.   {
5.     //Specify table name (and schema name optionally) to be mapped to the
   model class
6.     modelBuilder.Entity<ModelClass>( ).ToTable("table_name", schema:
   "schema_name");
7.
8.     //Specify view name (and schema name optionally) to be mapped to the model
   class
9.     modelBuilder.Entity<ModelClass>( ).ToView("view_name", schema:
   "schema_name");
10.
11.    //Specify default schema name applicable for all tables in the DbContext
12.    modelBuilder.HasDefaultSchema("schema_name");
13.  }
14. }
```

```
1.  public class CustomDbContext : DbContext
2.  {
3.    protected override void OnModelCreating(ModelBuilder modelBuilder)
4.    {
5.      modelBuilder.Entity<ModelClass>( ).Property(temp => temp.PropertyName)
6.        .HasColumnName("column_name") //Specifies column name in table
7.        .HasColumnType("data_type") //Specifies column data type in table
8.        .HasDefaultValue("default_value") //Specifies default value of the
    column
9.    }
10. }
```

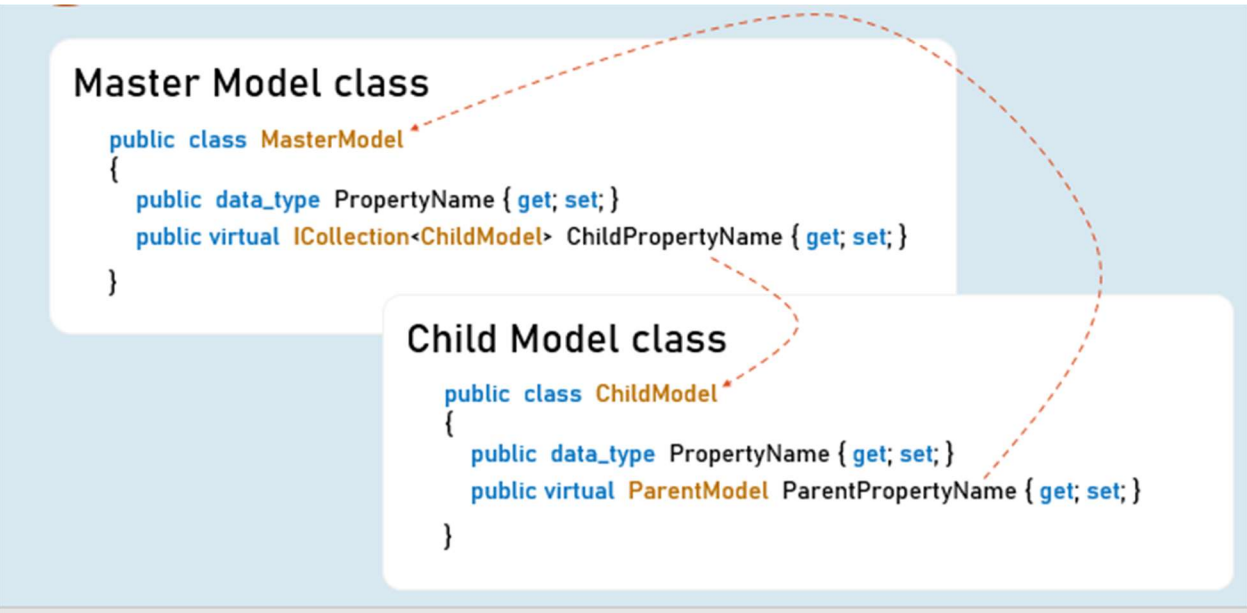
```
1.  public class CustomDbContext : DbContext
2.  {
3.    protected override void OnModelCreating(ModelBuilder modelBuilder)
4.    {
5.      //Adds database index for the specified column for faster searches
6.      modelBuilder.Entity<ModelClass>( ).HasIndex("column_name").IsUnique();
7.
8.      //Adds check constraint for the specified column - that executes for
    insert & update
9.      modelBuilder.Entity<ModelClass>( ).HasCheckConstraint("constraint_name",
    "condition");
10.   }
11. }
```

EF - Table Relations with Fluent API

Table Relations



EF - Table Relations with Navigation Properties



EF - Table Relations with Fluent API

DbContext class

```
1. public class CustomDbContext : DbContext
2. {
3.   protected override void OnModelCreating(ModelBuilder modelBuilder)
4.   {
5.     //Specifies relation between primary key and foreign key among two tables
6.     modelBuilder.Entity<ChildModel>( )
7.      .HasOne<ParentModel>(parent =>
   parent.ParentReferencePropertyInChildModel)
8.      .WithMany(child => child.ChildReferencePropertyInParentModel) //optional
9.      .HasForeignKey(child => child.ForeignKeyPropertyInChildModel)
10.  }
11. }
```

EF - Async Operations

async

- The method is awaitable.
- Can execute I/O bound code or CPU-bound code

await

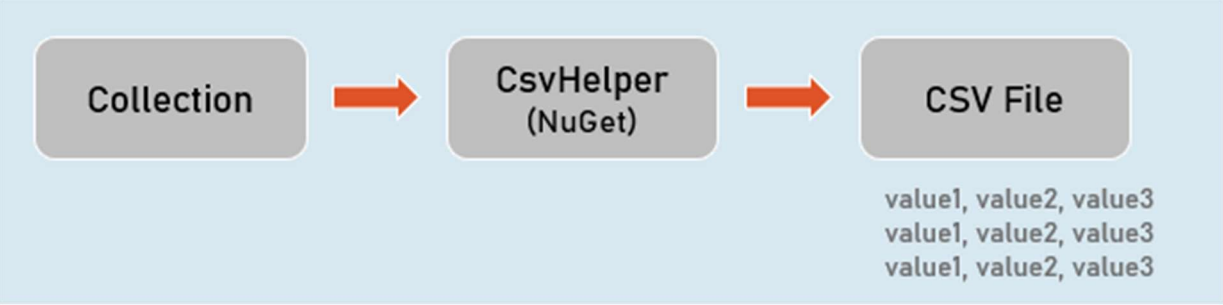
- Waits for the I/O bound or CPU-bound code execution gets completed.
- After completion, it returns the return value.

Generate PDF Files

Rotativa.AspNetCore:

```
using Rotativa.AspNetCore;
using Rotativa.AspNetCore.Options;

return new ViewAsPdf("ViewName", ModelObject, ViewData)
{
  PageMargins = new Margins() { Top = 1, Right = 2, Bottom = 3, Left = 4 },
  PageOrientation = Orientation.Landscape
}
```

Generate CSV Files (CSVHelper)



CsvWriter:

WriteRecords(records)

Writes all objects in the given collection.

Eg:

```
1,abc
2,def
```

WriteHeader<ModelClass>( )

Writes all property names as headings.

Eg:

```
Id, Name
```

WriteRecord(record)

Writes the given object as a row.

Eg:

```
1, abc
```

WriteField(value)

Writes given value.

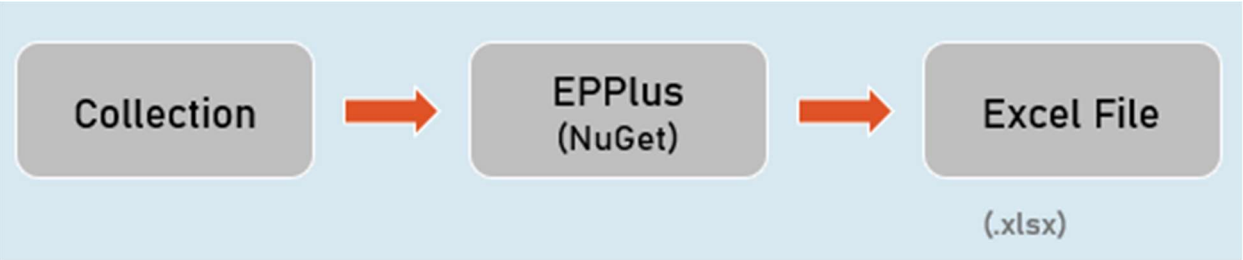
NextRecord( )

Moves to the next line.

Flush( )

Writes the current data to the stream.

Generate Excel Files (EPPlus)



ExcelWorksheet

```
["cell_address"].Value
```

Sets or gets value at the specified cell.

```
["cell_address"].Style
```

Sets or gets formatting style of the specific cell.

Best Practices of Unit Tests

Isolated / Stand-alone

(separated from any other dependencies such as file system or database)

Test single method at-a-time

(should not test more than one method in a single test case)

Unordered

(can be executed in any order)

Fast

(Tests should take little time to run (about few milliseconds))

Repeatable

(Tests can run repeatedly but should give same result, if no changes in the actual source code)

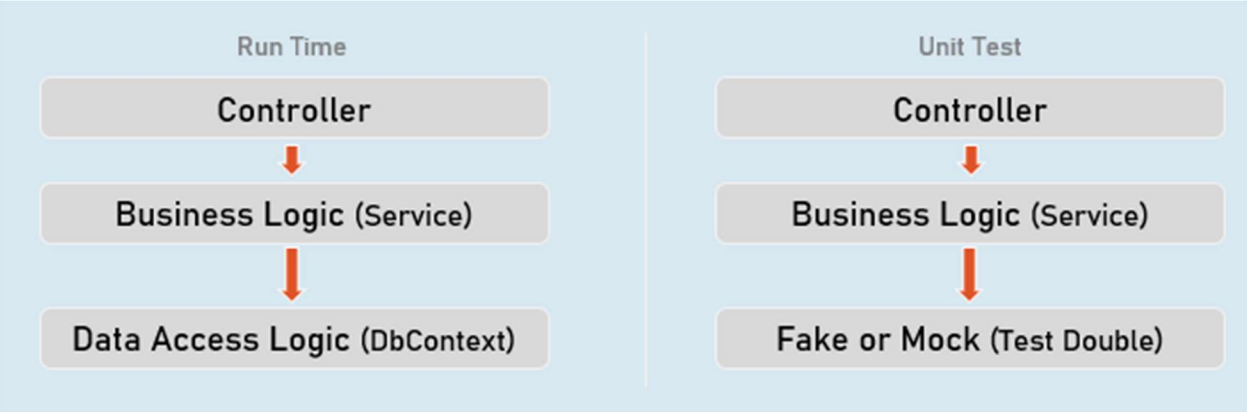
Timely

(Time taken for writing a test case should not take longer time, than then time taken for writing the code that is being tested)

Mocking the DbContext

Test Double

A "test double" is an object that look and behave like their production equivalent objects.



A "test double" is an object that look and behave like their production equivalent objects.

Fake

An object that providers an alternative (dummy) implementation of an interface

Mock

An object on which you fix specific return value for each individual method or property, without actual / full implementation of it.

Mocking the DbContext

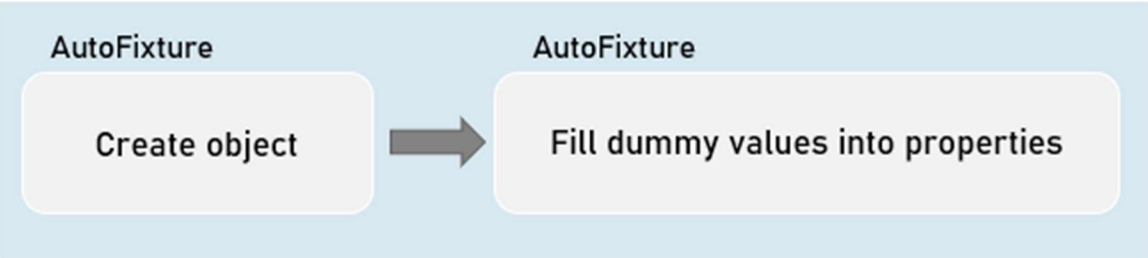
```
1. Install-Package Moq
2. Install-Package EntityFrameworkCoreMock.Moq
```

Mocking the DbContext:

```
1. var dbContextOptions = new
   DbContextOptionsBuilder<DbContextClassName>().Options;
2.
3. //mock the DbContext
4. DbContextMock<DbContextClass> dbContextMock = new
   DbContextMock<DbContextClass>(dbContextOptions);
5. var initialData = new List<ModelClass>() { … };
6.
7. //mock the DbSet
8. var dbSetMock = dbContextMock.CreateDbSetMock(temp => temp.DbSetName,
   initialData);
9.
10.//create service instance with mocked DbContext
11.var service = newServiceClass(dbContextMock.Object);
```

AutoFixture

AutoFixture generates objects of the specified classes and their properties with some fake values based their data types.



Normal object creation

```
1. new ModelClass() {
2.  Property1 = value,
3.  Property2 = value
4. }
```

With AutoFixture

`Fixture.Create<ModelClass>(); //initializes all properties of the specified model class with dummy values`

AutoFixture

`Install-Package AutoFixture`

Working with AutoFixture:

```
var fixture = new Fixture();

//Simple AutoFixture
var obj1 = fixture.Create<ModelClass>();

//Customization with AutoFixture
var obj2 = fixture.Build<ModelClass>()
 .With(temp => temp.Property1, value)
 .With(temp => temp.Property2, value)
.Create();
```

Fluent Assertions

Fluent Assertions are a set of extension methods to make the assertions in unit testing more readable and human-friendly.

`Install-Package FluentAssertions`

Assert

```
//Equal
Assert.Equal(expected, actual);

//Not Equal
Assert.NotEqual(expected, actual);

//Null
Assert.Null(actual);

//Not Null
Assert.NotNull(actual);

//True
Assert.True(actual);

//False
Assert.False(actual);

//Empty
Assert.Empty(actual);

//Not Empty
```

```
Assert.NotEmpty(actual);

//Null or empty
Assert.True(string.IsNullOrEmpty(actual)); //string
Assert.True(actual == null || actual.Length == 0); //collection

//Should not be null or empty
Assert.False (string.IsNullOrEmpty(actual)); //string
Assert.False(actual == null || actual.Length == 0); //collection

//number should be positive
Assert.True(actual > 0);

//number should be negative
Assert.True(actual < 0);

//number should be >= expected
Assert.True(actual >= expected);

//number should be <= expected
Assert.True(actual <= expected);

//number should be in given range
Assert.True(actual >= minimum && actual <= maximum);

//number should not be in given range
Assert.True(actual < minimum || actual > maximum);

//check data type
Assert.IsType<ExpectedType>(actual);

//Compare properties of two objects (Equals method SHOULD BE overridden)
Assert.Equal(expected, actual);

//Compare properties (should not be equal) of two objects (Equals method
   SHOULD BE overridden)
Assert.NotEqual(expected, actual);
```

Fluent Assertion

```
//Equal
actual.Should().Be(expected);

//Not Equal
actual.Should().NotBe(expected);

//Null
actual.Should().BeNull();

//Not Null
actual.Should().NotBeNull();

//True
actual.Should().BeTrue();
```

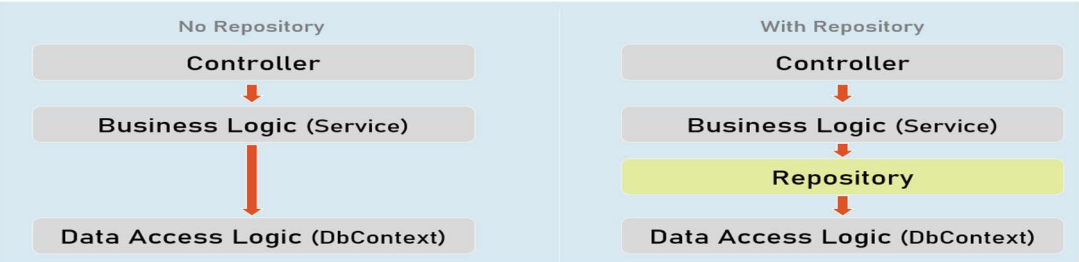
```
15.
16. //False
17. actual.Should().BeFalse();
18.
19. //Empty
20. actual.Should().BeEmpty();
21.
22. //Not Empty
23. actual.Should().NotBeEmpty();
24.
25. //Null or empty
26. actual.Should().BeNullOrEmpty();
27.
28. //Should not be null or empty
29. actual.Should().NotBeNullOrEmpty();
30.
31. //number should be positive
32. actual.Should().BePositive();
33.
34. //number should be negative
35. actual.Should().BeNegative();
36.
37. //number should be >= expected
38. actual.Should().BeGreaterThanOrEqualTo(expected);
39.
40. //number should be <= expected
41. actual.Should().BeLessThanOrEqualTo(expected);
42.
43. //number should be in given range
44. actual.Should().BeInRange(minimum, maximum);
45.
46. //number should not be in given range
47. actual.Should().NotBeInRange(minimum, maximum);
48.
49. //number should be in given range
50. actual.Should().BeInRange(minimum, maximum);
51.
52. //number should not be in given range
53. actual.Should().NotBeInRange(minimum, maximum);
54.
55. //check data type (same type)
56. actual.Should().BeOfType<ExpectedType>();
57.
58. //check data type (same type or derived type)
59. actual.Should().BeAssignableTo<ExpectedType>();
60.
61. //Compare properties of two objects (Equals method NEED NOT be overridden)
62. actual.Should().BeEquivalentTo(expected);
63.
64. //Compare properties (should not equal) of two objects (Equals method NEED NOT
    be overridden)
65. actual.Should().BeNotEquivalentTo(expected);
```

Fluent Assertions - Collections:

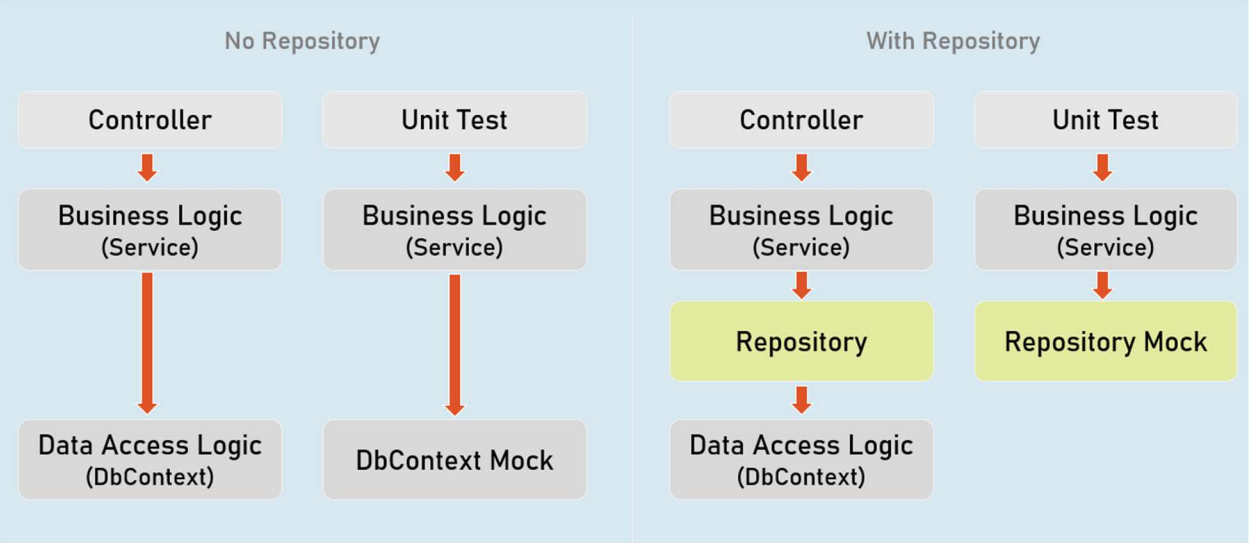
```
actualCollection.Should().BeEmpty();
actualCollection.Should().NotBeEmpty();

actualCollection.Should().HaveCount(expectedCount);
actualCollection.Should().NotHaveCount(expectedCount);

actualCollection.Should().HaveCountGreaterThanOrEqualTo(expectedCount);
actualCollection.Should().HaveCountLessThanOrEqualTo(expectedCount);

actualCollection.Should().HaveSameCount(expectedCollection);
actualCollection.Should().NotHaveSameCount(expectedCollection);

actualCollection.Should().BeEquivalentTo(expectedCollection);
actualCollection.Should().NotBeEquivalentTo(expectedCollection);

actualCollection.Should().ContainInOrder(expectedCollection);
actualCollection.Should().NotContainInOrder(expectedCollection);

actualCollection.Should().OnlyHaveUniqueItems(expectedCount);
actualCollection.Should().OnlyContain(temp => condition);

actualCollection.Should().BeInAscendingOrder(temp => temp.Property);
actualCollection.Should().BeInDescendingOrder(temp => temp.Property);

actualCollection.Should().NotBeInAscendingOrder(temp => temp.Property);
actualCollection.Should().NotBeInDescendingOrder(temp => temp.Property);

delegateObj.Should().Throw<ExceptionType>();
delegateObj.Should().NotThrow<ExceptionType>();

await delegateObj.Should().ThrowAsync<ExceptionType>();
await delegateObj.Should().NotThrowAsync<ExceptionType>();
```

Repository

Repository (or Repository Pattern) is an abstraction between Data Access Layer (EF DbContext) and business logic layer (Service) of the application.



Unit Testing



Benefits of Repository Pattern

Loosely-coupled business logic (service) & data access.

(You can independently develop them).

Changing data store

(You can create alternative repository implementation for another data store, when needed).

Unit Testing

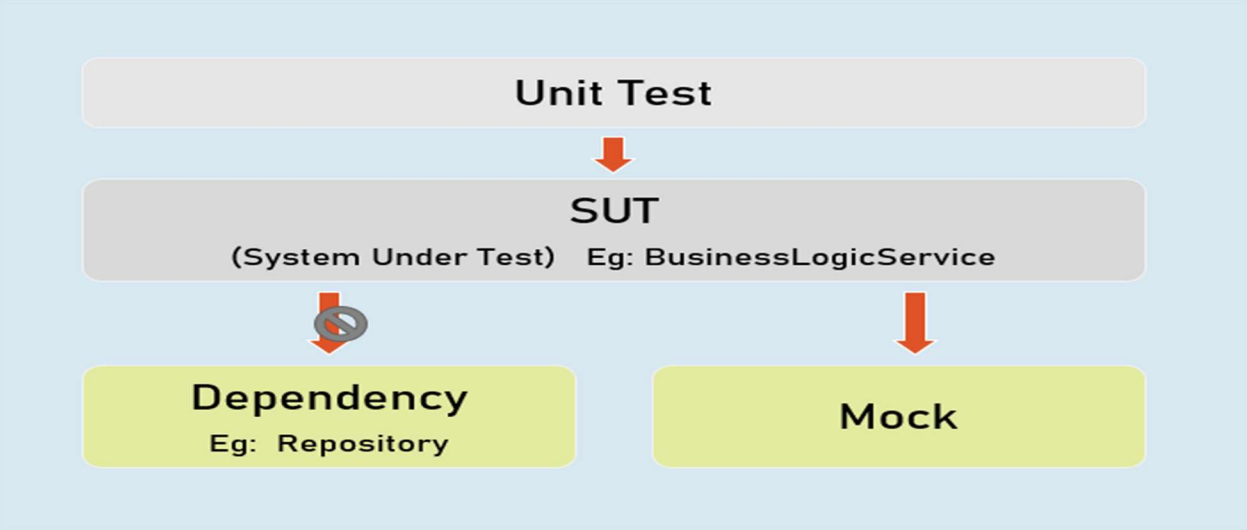
(Mocking the repository is much easier (and preferred) than mocking DbContext).

Mocking the Repository


```
Install-Package Moq
```

Mocking the Repository:

```
1. //mock the repository
2. Mock<IRepository> repositoryMock = new Mock<IRepository>();
3.
4. //mock a method repository method
5. repositoryMock.Setup(temp => temp.MethodName(It.Any<ParameterType>()))
6.  .Returns(return_value);
7.
8. //create service instance with mocked repository
9. var service = newServiceClass(repositoryMock.Object);
```



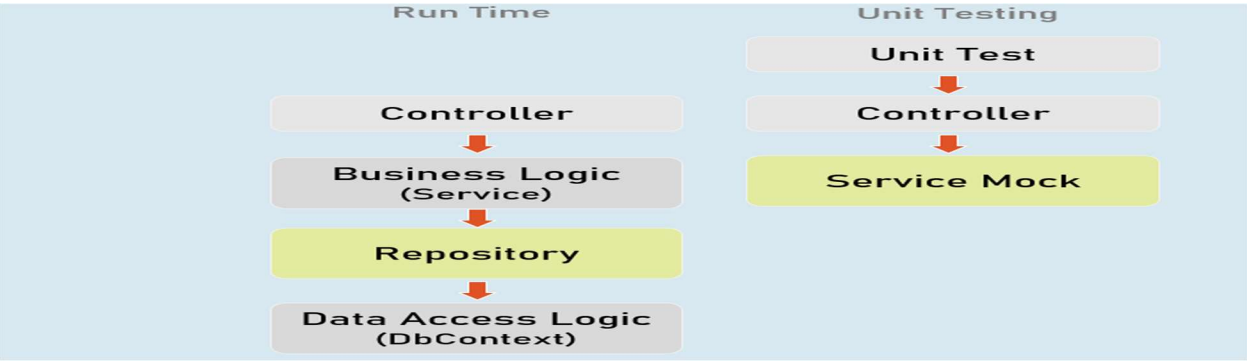
Mock<IPersonsRepository>

Used to mock the methods of IPersonsRepository.

IPersonsRepository

Represents the mocked object that was created by Mock<T>.

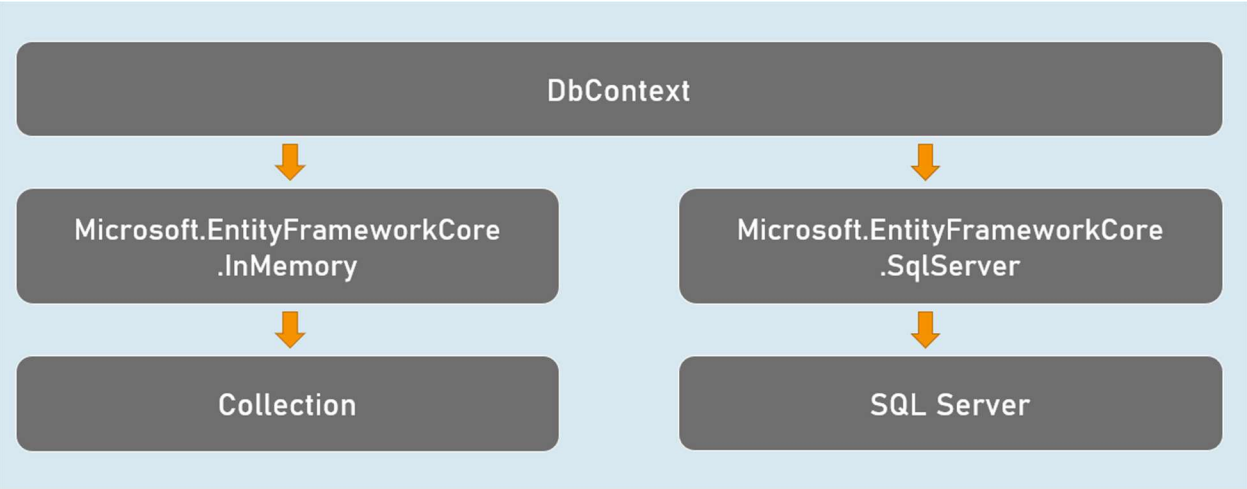
Unit Testing the Controller



Unit Testing the Controller:

```
1. //Arrange
2. ControllerName controller = new ControllerName();
3.
4. //Act
5. IActionResult result = controller.ActionMethod();
6.
7. //Assert
8. result.Should().BeAssignableTo<ActionResultType>(); //checking type of action
   result
9. result.ViewData.Model.Should().BeAssignableTo<ExpectedType>(); //checking type
   of model
10. result.ViewData.Model.Should().Be(expectedValue); //you can also use any other
   assertion
```

EFCore In-Memory Provider

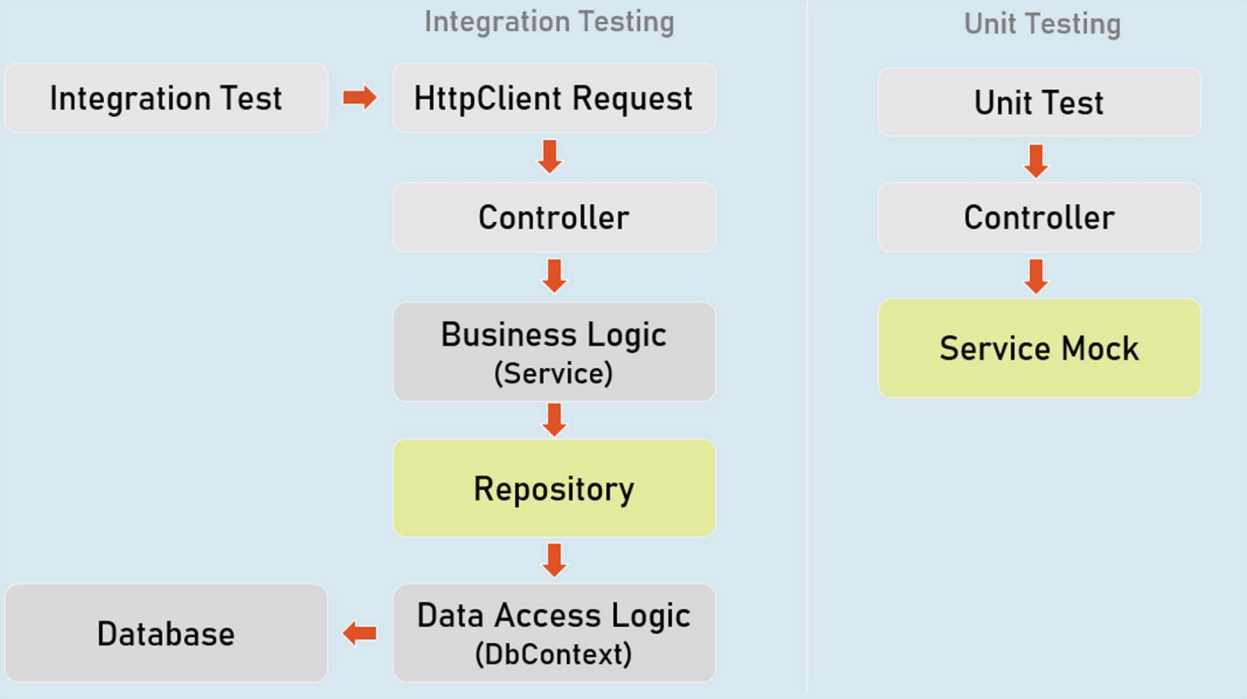



```
Install-Package Microsoft.EntityFrameworkCore.InMemory
```

Using In-memory provider:

```
1. var dbContextOptions =
2.  new DbContextOptionsBuilder<DbContextClassName>()
3.  .UseInMemoryDatabase("database_name");
4.  .Options;
5.
6. var dbContext = newDbContextClassName(dbContextOptions);
```

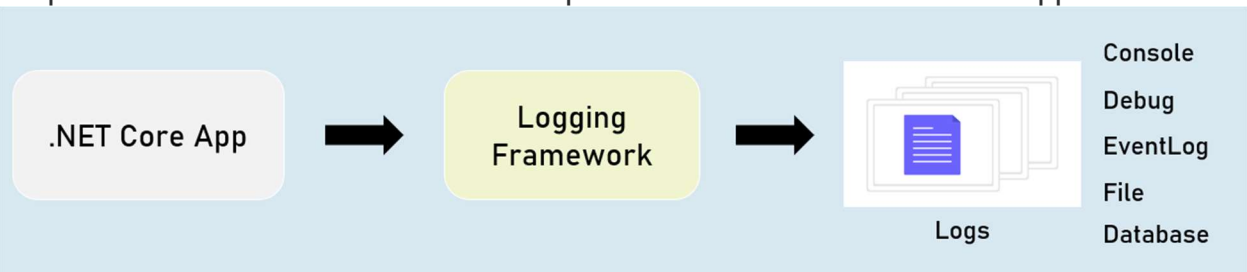
Integration Test

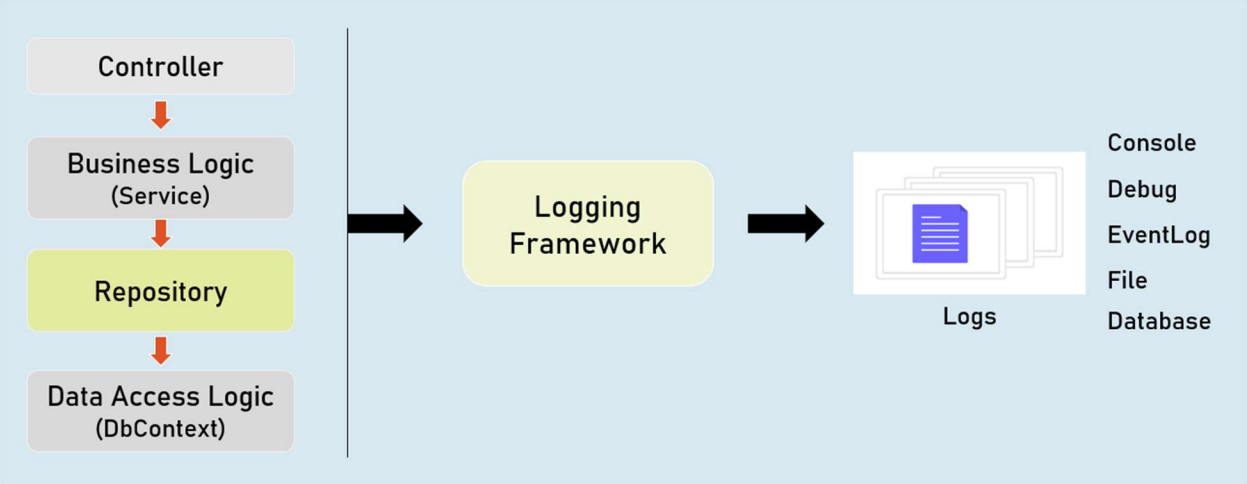



```
1. //Create factory
2. WebApplicationFactory factory = new WebApplicationFactory();
3.
4. //Create client
5. HttpClient client = factory.CreateClient();
6.
7. //Send request client
8. HttpResponseMessage response = await client.GetAsync("url");
9.
10. //Assert
11. result.Should().BeSuccessful(); //Response status code should be 200 to 299
```

Logging

Logging is the process of recording run-time actions as they happen in real-time.
Helps us to understand the failures and performance bottlenecks of the application.





ILogger

Debug

```
ILogger.LogDebug("log_message");
```

Logs that provide details & values of variables for debugging purpose.

Information

```
ILogger.LogInformation("log_message");
```

Logs that track the general flow of the application execution.

Warning

```
ILogger.LogWarning("log_message");
```

Logs that highlight an abnormal or unexpected event.

Error

```
ILogger.LogError("log_message");
```

Logs to indicate that flow of execution is stopped due to a failure.

Critical

```
ILogger.LogCritical("log_message");
```

Logs to indicate an unrecoverable application crash.

Logging Configuration

appsettings.json

```
{
 "Logging": {
  "LogLevel": {
   "Default": "Debug | Information | Warning | Error| Critical"
   "Microsoft.AspNetCore": "Debug | Information | Warning | Error | Critical"
  }
 }
}
```

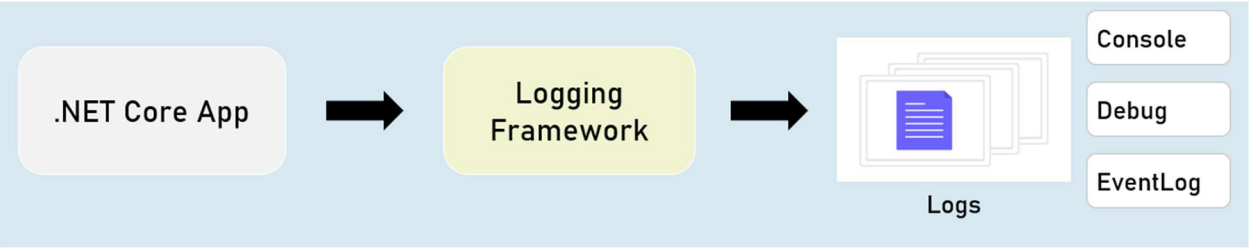
Controller and other classes

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;

public class ControllerName : Controller
{
 private readonly ILogger<ClassName> _logger;

 public ControllerName(ILogger<ClassName> logger)
 {
  _logger = logger;
 }
}
```

Logging Providers

Logging provider specifies where to store / display logs.

The built-in logging providers in asp.net core doesn't support file / database logging providers.



in **Program.cs:**

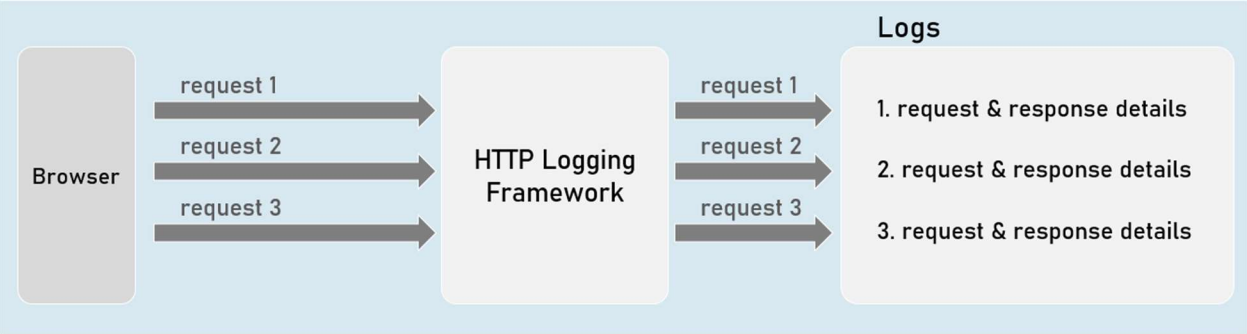
```
builder.Host.ConfigureLogging(logging =>
{
  logging.ClearProviders();
  logging.AddConsole();
```

```
5.    logging.AddDebug();
6.    logging.AddEventLog();
7. });
```

HTTP Logging

Logs details all HTTP requests and responses.

You need to set a value of "HttpLoggingFields" enum to set specify desired details.



HTTP Logging Options

"HttpLoggingFields" enum:

RequestMethod

Method of request. Eg: GET

RequestPath

Path of request. Eg: /home/index

RequestProtocol

Protocol of request. Eg: HTTP/1.1

RequestScheme

Protocol Scheme of request. Eg: http

RequestQuery

Query string Scheme of request. Eg: ?id=1

RequestHeaders

Headers of request. Eg: Connection: keep-alive

RequestPropertiesAndHeaders

Includes all of above (default)

RequestBody

Entire request body. [has performance drawbacks; not recommended]

Request

Includes all of above

"HttpLoggingFields" enum

ResponseStatusCode

Status code of response. Eg: 200

ResponseHeaders

Headers of response. Eg: Content-Length: 20

ResponsePropertiesAndHeaders

Includes all of above (default)

ResponseBody

Entire response body. [has performance drawbacks; not recommended]

Response

Includes all of above

All

Includes all from request and response

HTTP Logging Options

Program.cs:

```
1. builder.Serices.AddHttpLogging(options =>
2. {
3.   options.LoggingFields =
   Microsoft.AspNetCore.HttpLogging.HttpLoggingFields.YourOption;
```

```
4. });
```

Serilog

Serilog is a structured logging library for Asp.Net Core.
Supports variety of logging destinations, referred as "Sinks" - starts with Console, Azure, DataDog, ElasticSearch, Amazon CloudWatch, Email and Seq.

Serilog - Configuration

appsettings.json

```
1. {
2.  "Serilog": {
3.   "Using": [
4.     "Serilog.Sinks.YourSinkHere"
5.   ],
6.   "MinimumLevel": "Debug | Information | Warning | Error | Critical",
7.   "WriteTo": [
8.   {
9.     "Name": "YourSinkHere",
10.    "Args": "YourArguments"
11.  }
12.  ]
13. }
14.}
```

Serilog - Options

Program.cs:

```
1. builder.Host.UseSerilog(HostBuilderContext context,
2. IServiceProvider services, LoggerConfiguration configuration) =>
3. {
4.   configuration
5.   .ReadFrom.Configuration(context.Configuration) //read configuration settings
   from built-in IConfiguration
6.   .ReadFrom.Services(services); //read services from built-in IServiceProvider
7. });
```

Serilog - File Sink

The "Serilog.Sinks.File" logs into a specified file.

You can configure the filename, rolling interval, file size limit etc., using configuration settings.

Serilog - "File Sink" Configuration

appsettings.json

```
{
 "Serilog": {
    "Using": [ "Serilog.Sinks.File" ],
    "MinimumLevel": "Debug | Information | Warning | Error | Critical",
    "WriteTo": [
    {
     "Name": "File",
     "Args": [
       "path": "folder/filename.ext",
       "rollingInterval": "Minute | Hour | Day | Month | Year | Infinite",
     ]
    }
   ]
  }
}
```

Serilog - Database Sink

The "Serilog.Sinks.MSSqlServer" logs into a specified SQL Server database.

You can configure the connection string using configuration settings.

Serilog - 'MSSqlServer' Sink Configuration

appsettings.json

```
{
  "Serilog": {
    "Using": [ "Serilog.Sinks.MSSqlServer" ],
    "MinimumLevel": "Debug | Information | Warning | Error | Critical",
    "WriteTo": [
    {
      "Name": "MSSqlServer",
      "Args": [
        "connectionString": "your_connection_string_here",
        "tableName": "table_name",
      ]
    }
   ]
  }
}
```

Serilog - Seq Sink

The "Serilog.Sinks.Seq" is a real-time search and analysis server for structured application log data.

Seq server can run on Windows, Linux or Docker.

Serilog - 'Seq' Sink - Configuration

appsettings.json

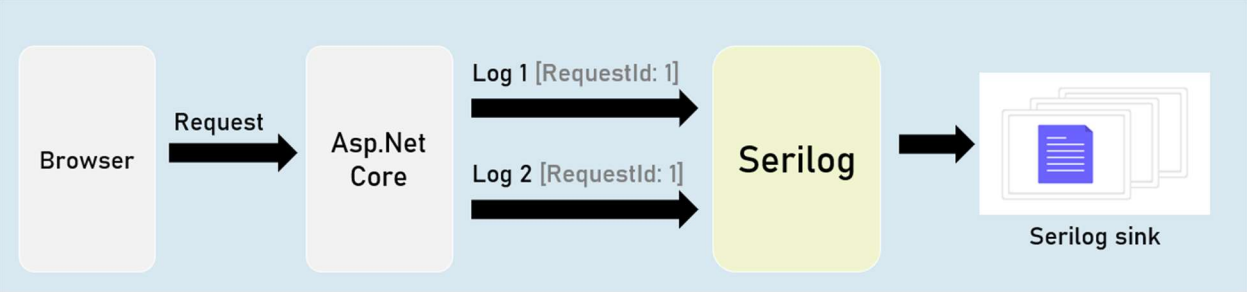
```
{
  "Serilog": {
    "Using": [ "Serilog.Sinks.Seq" ],
    "MinimumLevel": "Debug | Information | Warning | Error | Critical",
    "WriteTo": [
     {
       "Name": "Seq",
       "Args": [
          "serverUrl": "http://localhost:5341"
       ]
```

```
    }
  ]
 }
}
```

Serilog - RequestId

"RequestId" is the unique number (guid) of each individual requests, used to identify to which request the log belongs to.

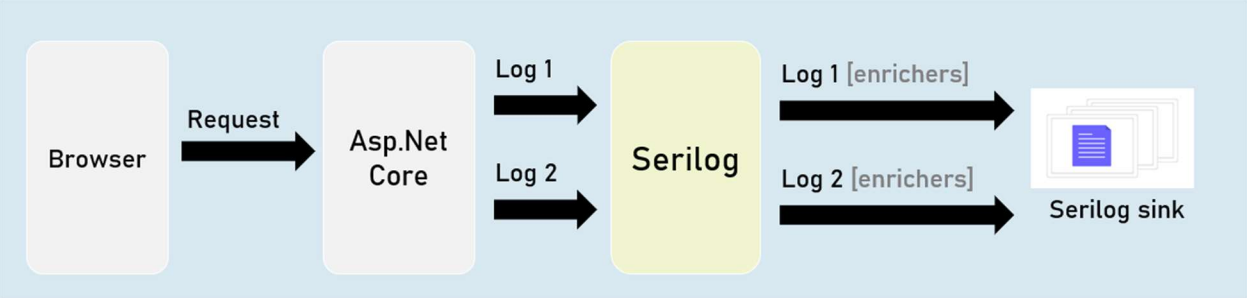
RequestId is "TraceIdentifier" internally, that is generated by Asp.Net Core.



Serilog - Enrichers

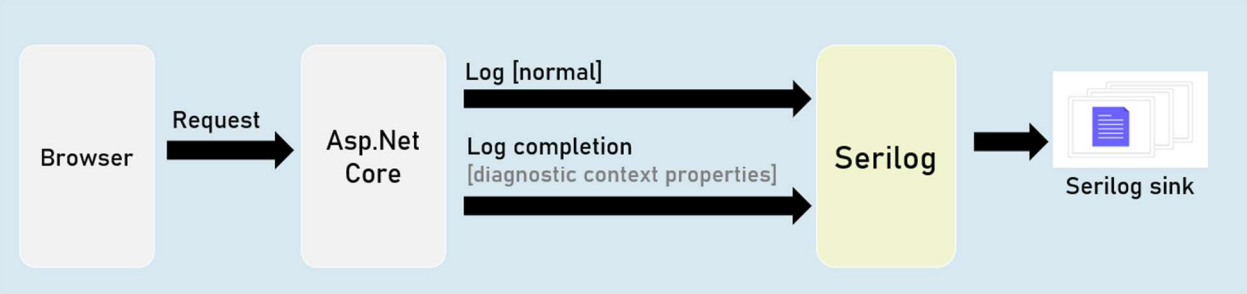
Enrichers are additional details that are added to LogContext; so they're stored in logs.

Eg: MachineName[or]Custom Properties.



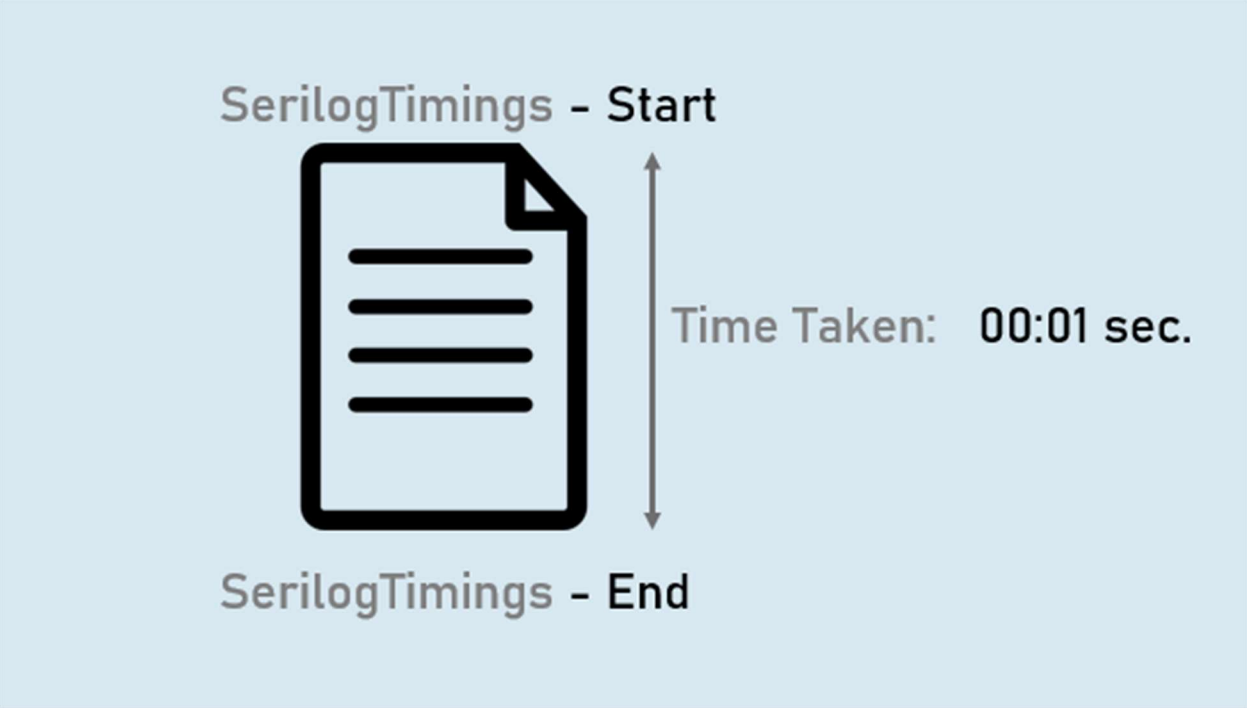
Serilog - IDiagnosticContext

Diagnostic context allows you to add additional enrichment properties to the context; and all those properties are logged at once in the final "log completion event" of the request.



Serilog Timings

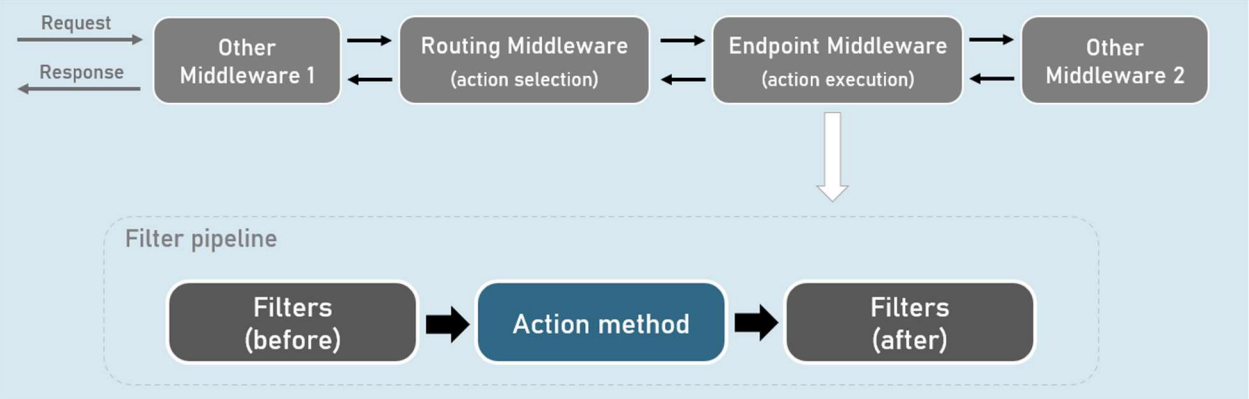
"SerilogTimings" package records timing of a piece of your soure code, indicating how much time taken for executing it.



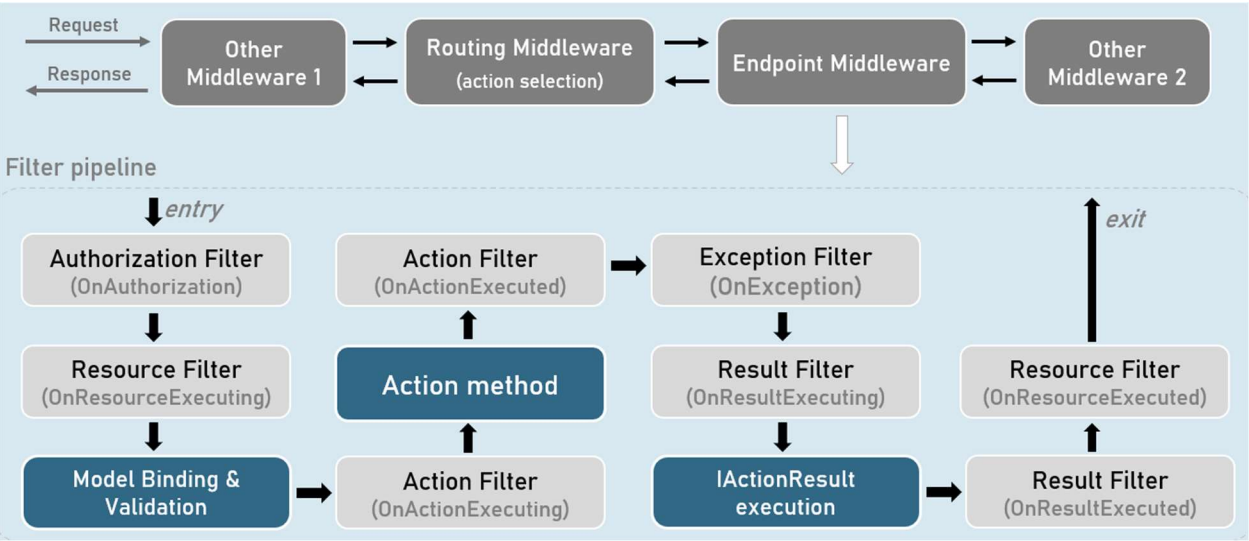
Filters

Filters are the code blocks that execute before / after specific stages in "Filter Pipeline".

Filters perform specific tasks such as authorization, caching, exeption handling etc.



Filter Pipeline



Overview of Types of Filters

Authorization Filter

Determines whether the user is authorized to access the action method.

Resource Filter

Invoking custom model binder explicitly

Caching the response.

Action Filter

Manipulating & validating the action method parameters.

Manipulating the ViewData.

Overriding the IActionResult provided by action method.

Exception Filter

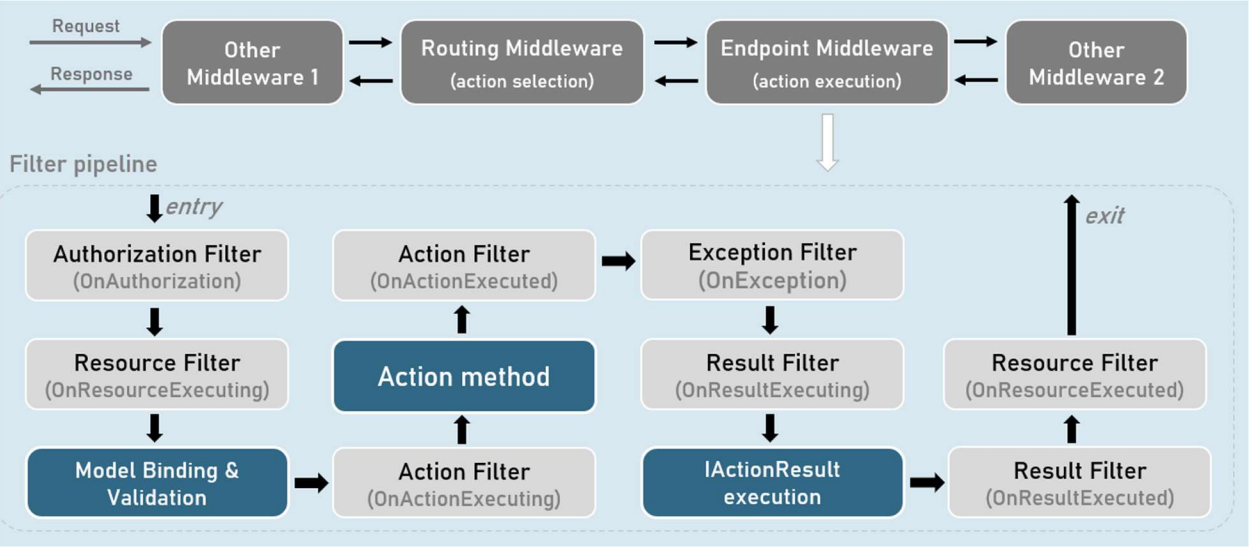
Handling unhandled exception that occur in model binding, action filters or action methods.

Result Filter

Preventing IActionResult from execution.

Adding last-moment changes to response (such as adding response headers).

Action Filter



When it runs

Runs immediately before and after an action method executes.

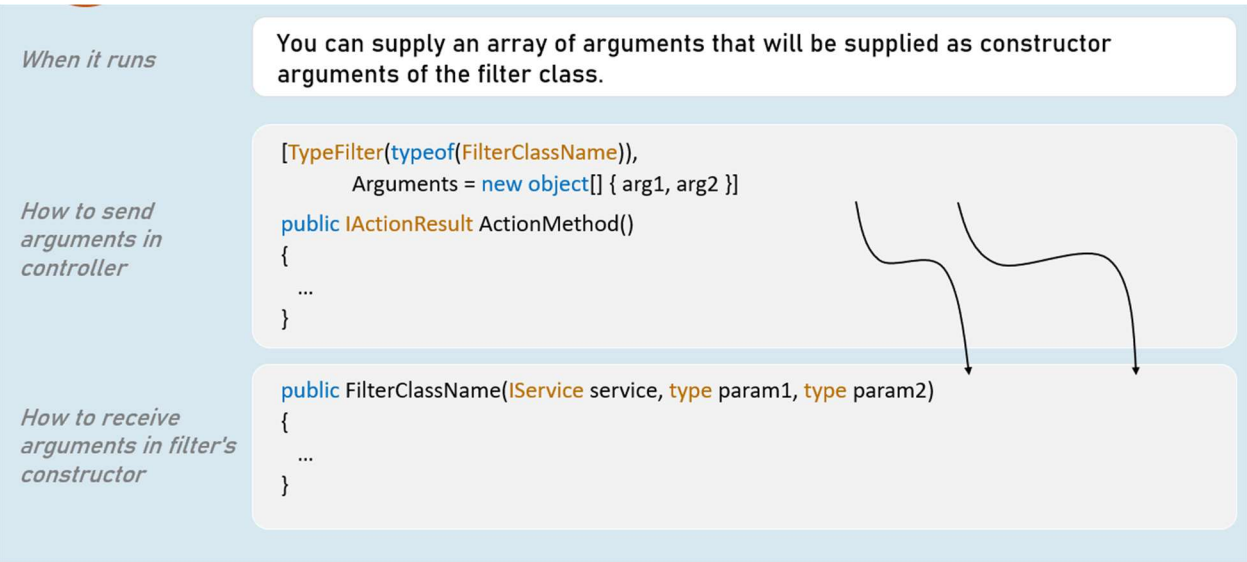
'OnActionExecuting' method

- It can access the action method parameters, read them & do necessary manipulations on them.
- It can validate action method parameters.
- It can short-circuit the action (prevent action method from execution) and return a different IActionResult.

'OnActionExecuted' method

- It can manipulate the ViewData.
- It can change the result returned from the action method.

Filter Arguments



Global Filters

Filter Scopes

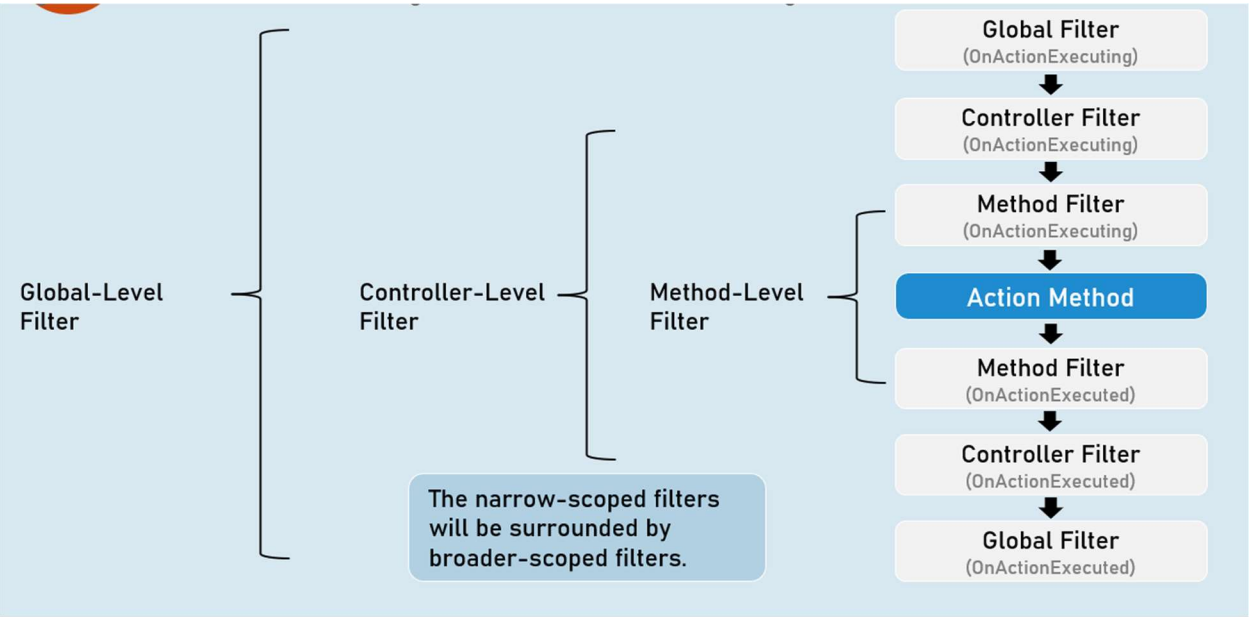
What are global filters?

Global filters are applied to all action methods of all controllers in the project.

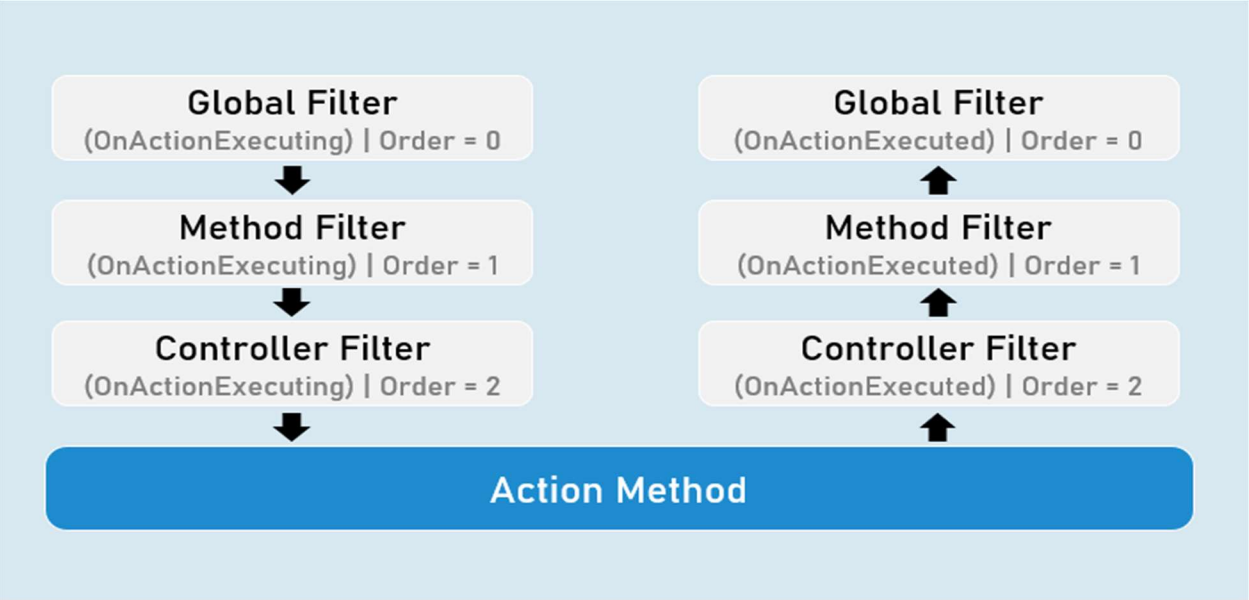
How to add global filters in Program.cs?

```
builder.Services.AddControllersWithViews(options => {
  options.Filters.Add<FilterClassName>(); //add by type
  //or
  options.Filters.Add(new FilterClassName()); //add filter instance
});
```

Default Order of Filter Execution

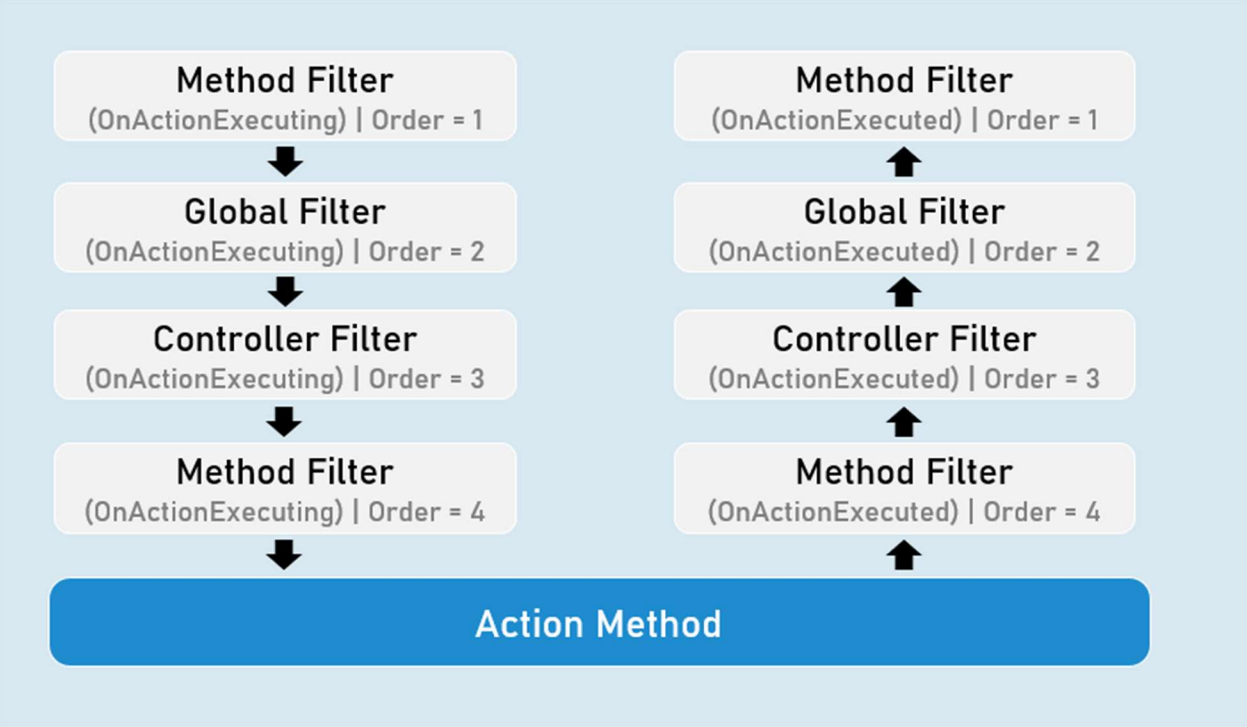


Custom Order of Filters



IOrderedFilter

Example



IOrderedFilter

Action filter with IOrderedFilter

```
1. public class FilterClassName : IActionFilter, IOrderedFilter
2. {
3.   public int Order { get; set; } //Defines sequence of execution
4.
5.   public FilterClassName(int order)
6.   {
7.     Order = order;
8.   }
9.
10.  public void OnActionExecuting(ActionExecutingContext context)
11.  {
12.    //TO DO: before logic here
13.  }
14.
15.  public void OnActionExecuted(ActionExecutedContext context)
16.  {
17.    //TO DO: after logic here
18.  }
19.}
```

Async Filters

Asynchronous Action Filter

```
1. public class FilterClassName : IAsyncActionFilter, IOrderedFilter
2. {
3.   public int Order { get; set; } //Defines sequence of execution
4.
5.   public FilterClassName(int order)
6.   {
7.     Order = order;
8.   }
9.
10.  public async Task OnActionExecutionAsync(ActionExecutingContext context,
   ActionExecutionDelegate next)
11.  {
12.    //TO DO: before logic here
13.    await next();
14.    //TO DO: after logic here
15.  }
16.}
```

Short-circuiting Filters

Action Filters

When it runs

Runs immediately before and after an action method executes.

'OnActionExecuting' method

- It can access the action method parameters, read them & do necessary manipulations on them.
- It can validate action method parameters.
- It can short-circuit the action (prevent action method from execution) and return a different IActionResult.

'OnActionExecuted' method

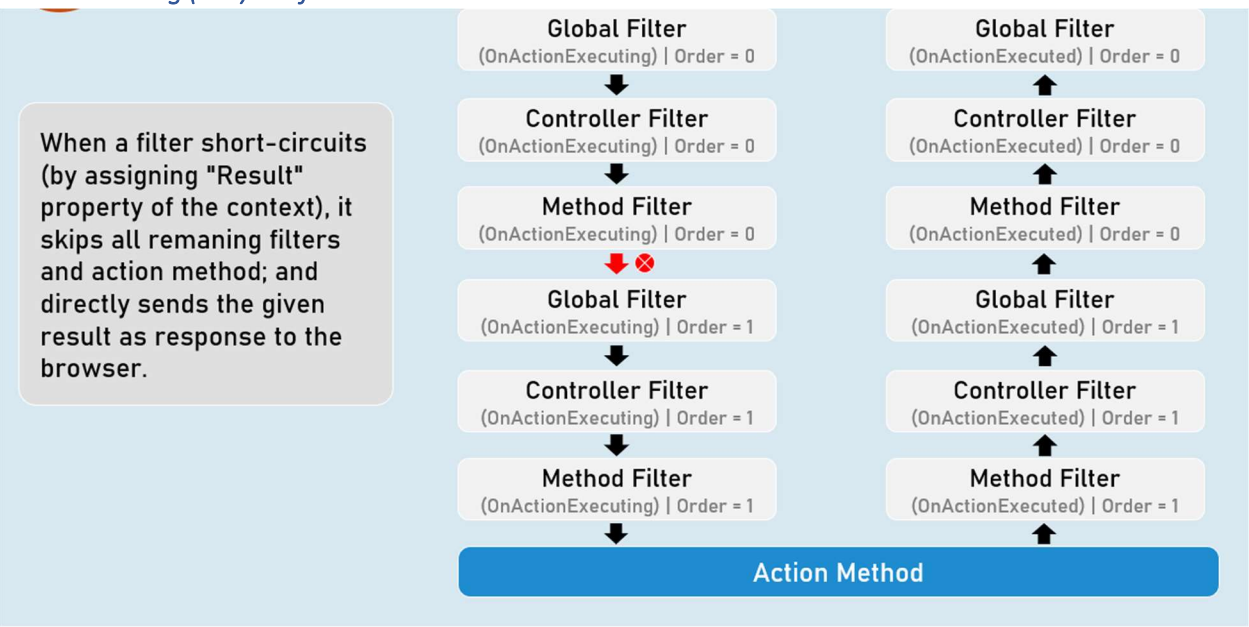
- It can manipulate the ViewData.
- It can change the result returned from the action method.
- It can throw exceptions to either return the exception to the exception filter (if exists); or return the error response to the browser.

Short-Circuiting Action Filter

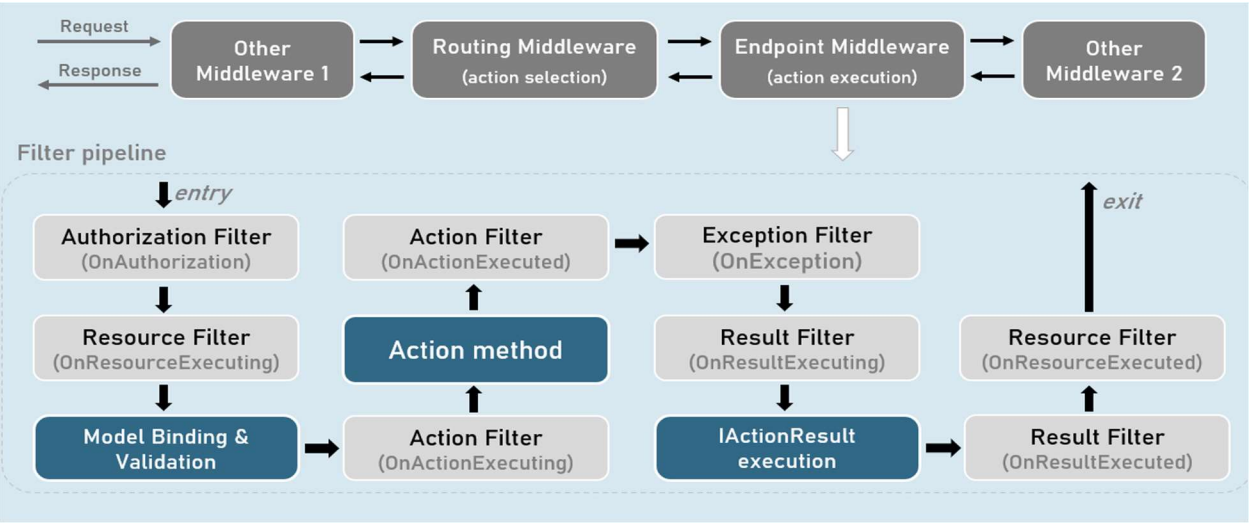
```
1. public class FilterClassName : IAsyncActionFilter, IOrderedFilter
2. {
3.  public int Order { get; set; } //Defines sequence of execution
4.
```

```
5.  public FilterClassName(int order)
6.  {
7.   Order = order;
8.  }
9.
10. public async Task OnActionExecutionAsync(ActionExecutingContext context,
    ActionExecutionDelegate next)
11. {
12.  //TO DO: before logic here
13.  context.Result = some_action_result; //you can return any type of
    IActionResult
14.  //Not calling next(). So it leads remaining filters & action method short-
    circuited.
15. }
16.}
```

Short-Circuiting (exit) the filters



Result Filter



When it runs

- Runs immediately before and after an IActionResult executes.
- It can access the IActionResult returned by the action method.

'OnResultExecuting' method

- It can continue executing the IActionResult normally, by not assigning "Result" property of the context.
- It can short-circuit the action (prevent IActionResult from execution) and return a different IActionResult.

'OnResultExecuted' method

- It can manipulate the last-moment changes in the response, such as adding necessary response headers.
- It should not throw exceptions because, exceptions raised in result filters would not be caught by the exception filter.

Synchronous Result Filter

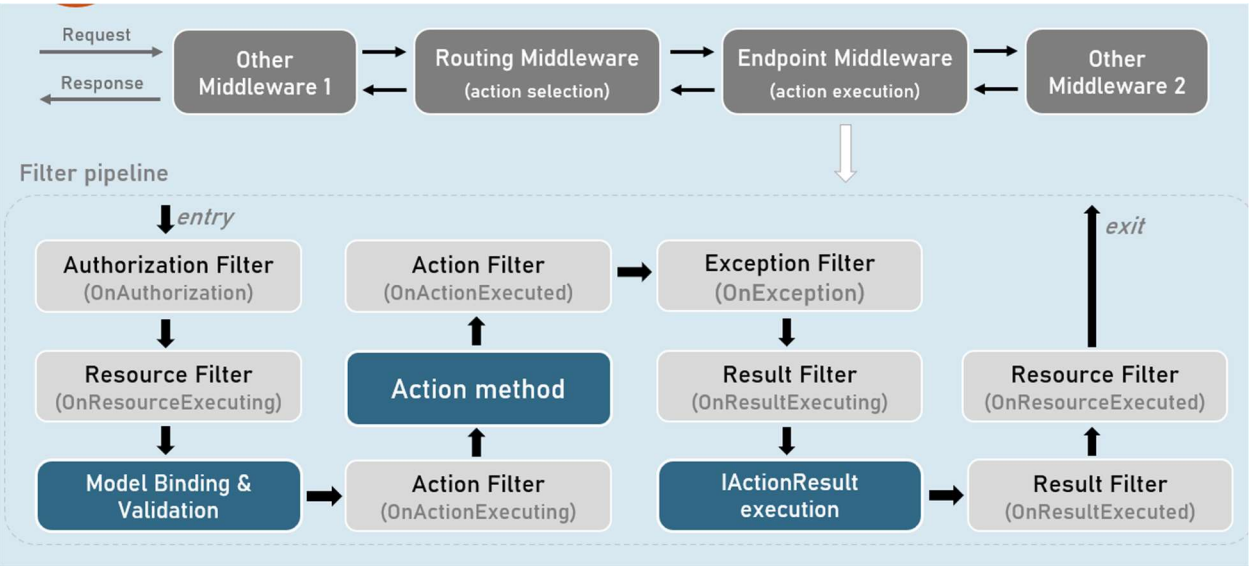
```
1. public class FilterClassName : IResultFilter, IOrderedFilter
2. {
3.   public int Order { get; set; } //Defines sequence of execution
4.
5.   public FilterClassName(int order)
6.   {
7.     Order = order;
8.   }
9.   public void OnResultExecuting(ResultExecutingContext context)
10.  {
11.    //TO DO: before logic here
12.  }
13.  public void OnResultExecuted(ResultExecutedContext context)
14.  {
15.    //TO DO: after logic here
16.  }
17.}
```

Asynchronous Result Filter

```
1. public class FilterClassName : IAsyncResultFilter, IOrderedFilter
2. {
3.   public int Order { get; set; } //Defines sequence of execution
4.
5.   public FilterClassName(int order)
6.   {
7.     Order = order;
8.   }
9.
```

```
10.  public async Task OnResultExecutionAsync(ResultExecutingContext context,
   ResultExecutionDelegate next)
11.  {
12.  //TO DO: before logic here
13.  await next();
14.  //TO DO: after logic here
15.  }
16. }
```

Resource Filter



When it runs

Runs immediately after Authorize Filter and after Result Filter executes.

'OnResourceExecuting' method

- It can do some work before model binding. Eg: Adding metrics to an action method.
- It can change the way how model binding works (invoking a custom model binder explicitly).
- It can short-circuit the action (prevent IActionResult from execution) and return a different IActionResult.

  Eg: Short-circuit if an unsupported content type is requested.

'OnResourceExecuted' method

- It can read the response body and store it in cache.

Synchronous Resource Filter

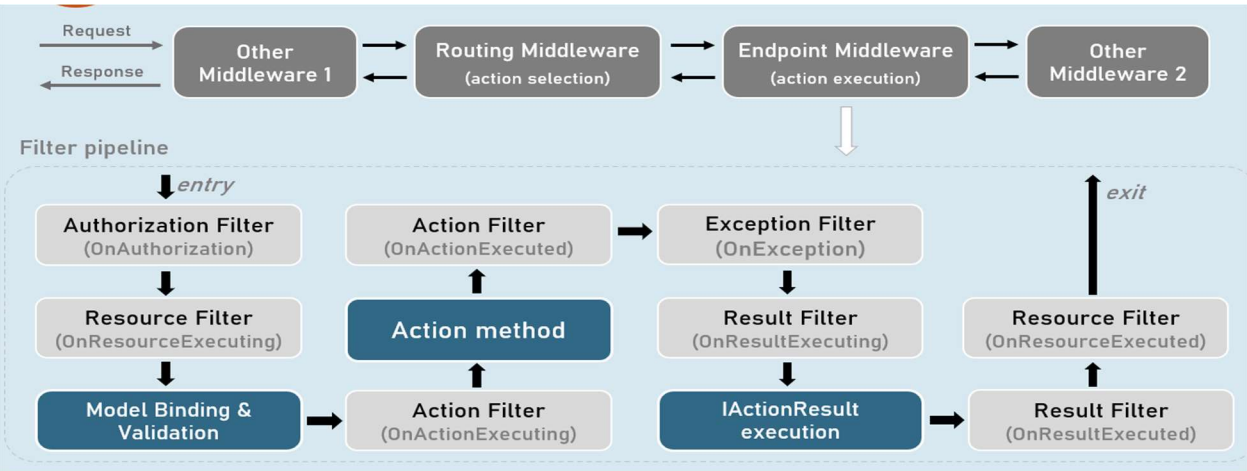
```
1. public class FilterClassName : IResourceFilter, IOrderedFilter
2. {
```

```
3.    public int Order { get; set; } //Defines sequence of execution
4.
5.    public FilterClassName(int order)
6.    {
7.      Order = order;
8.    }
9.
10.   public void OnResourceExecuting(ResourceExecutingContext context)
11.   {
12.     //TO DO: before logic here
13.   }
14.
15.   public void OnResourceExecuted(ResourceExecutedContext context)
16.   {
17.     //TO DO: after logic here
18.   }
19. }
```

Asynchronous Resource Filter

```
1.  public class FilterClassName : IAsyncResourceFilter, IOrderedFilter
2.  {
3.    public int Order { get; set; } //Defines sequence of execution
4.
5.    public FilterClassName(int order)
6.    {
7.      Order = order;
8.    }
9.
10.   public async Task OnResourceExecutionAsync(ResourceExecutingContext context,
    ResourceExecutionDelegate next)
11.   {
12.     //TO DO: before logic here
13.     await next();
14.     //TO DO: after logic here
15.   }
16. }
```

Authorization Filter



Authorization Filters

Runs before any other filters in the filter pipeline.

'OnAuthorize' method

- Determines whether the user is authorized for the request.
- Short-circuits the pipeline if the request is NOT authorized.
- Don't throw exceptions in OnAuthorize method, as they will not be handled by exception filters.

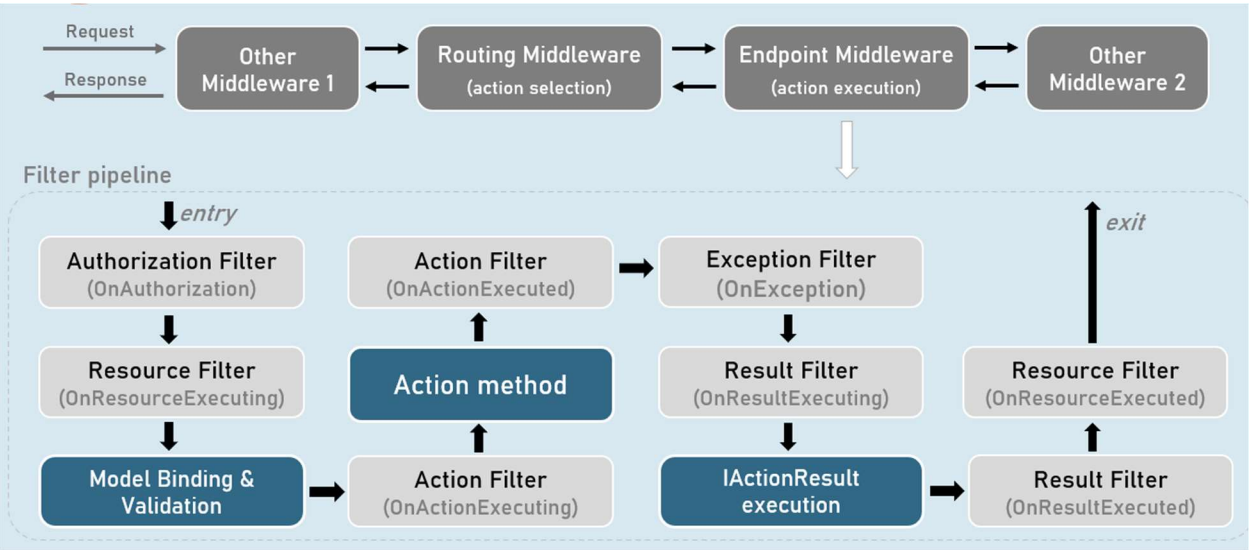
Synchronous Authorization Filter

```
1. public class FilterClassName : IAuthorizationFilter
2. {
3.   public void OnAuthorization(AuthorizationFilterContext context)
4.   {
5.     //TO DO: authorization logic here
6.   }
7. }
```

Asynchronous Authorization Filter

```
1. public class FilterClassName : IAsyncAuthorizationFilter
2. {
3.   public async Task OnAuthorizationAsync(AuthorizationFilterContext context)
4.   {
5.     //TO DO: authorization logic here
6.   }
7. }
```

Exception Filter



When it runs

Runs when an exception is raised during the filter pipeline.

'OnException method

- Handles unhandled exceptions that occur in controller creation, model binding, action filters or action methods.
- Doesn't handle the unhandled exceptions that occur in authorization filters, resource filters, result filters or IActionResult execution.
- Recommended to be used only when you want a different error handling and generate different result for specific controllers; otherwise, ErrorHandlingMiddleware is recommended over Exception Filters.

Synchronous Exception Filter

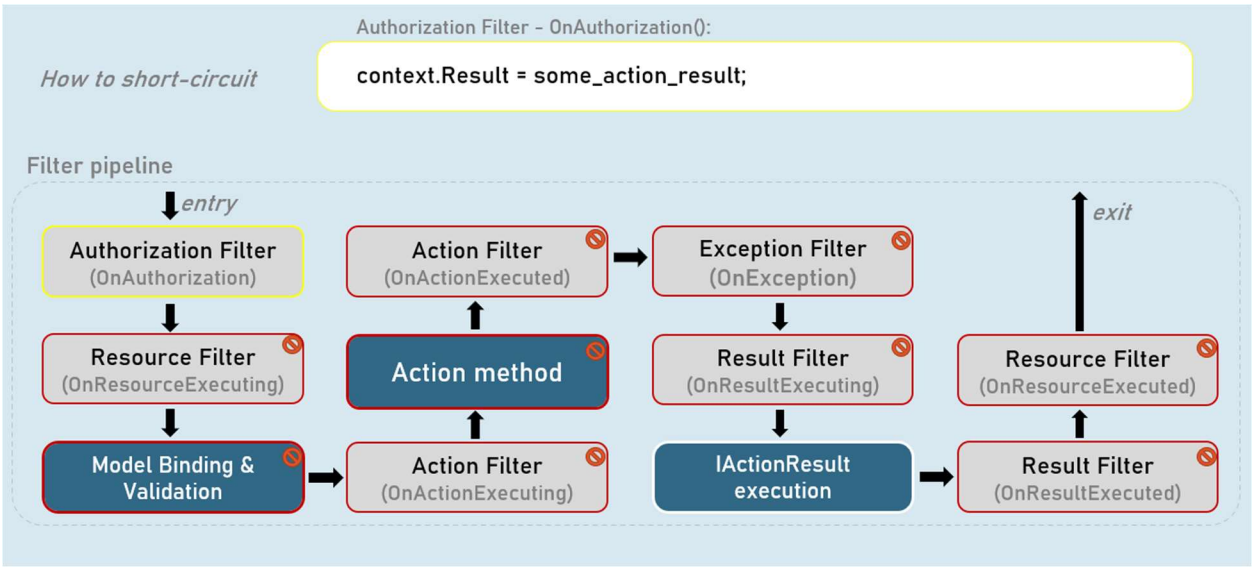
```
1. public class FilterClassName : IAsyncExceptionFilter
2. {
3.   public async Task OnExceptionAsync(ExceptionFilterContext context)
4.   {
5.     //TO DO: exception handling logic here, as follows
6.     context.Result = some_action_result;
7.     //or
8.     context.ExceptionHandled = true;
9.     return Task.CompletedTask;
10.  }
11.}
```

Asynchronous Exception Filter

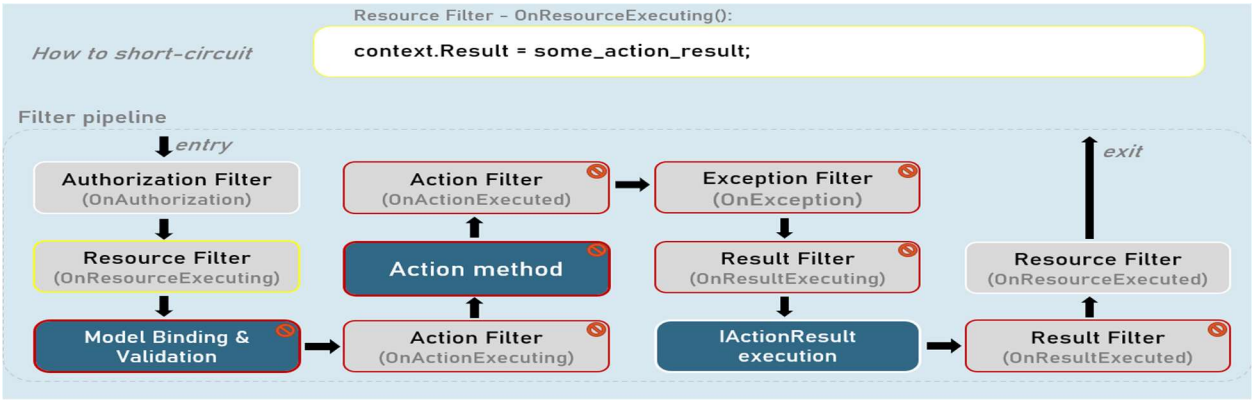
```
1. public class FilterClassName : IAsyncExceptionFilter
2. {
3.   public async Task OnExceptionAsync(ExceptionFilterContext context)
4.   {
5.     //TO DO: exception handling logic here, as follows
6.     context.Result = some_action_result;
7.     //or
8.     context.ExceptionHandled = true;
9.     return Task.CompletedTask;
10.  }
11.}
```

Impact of Short-Circuiting

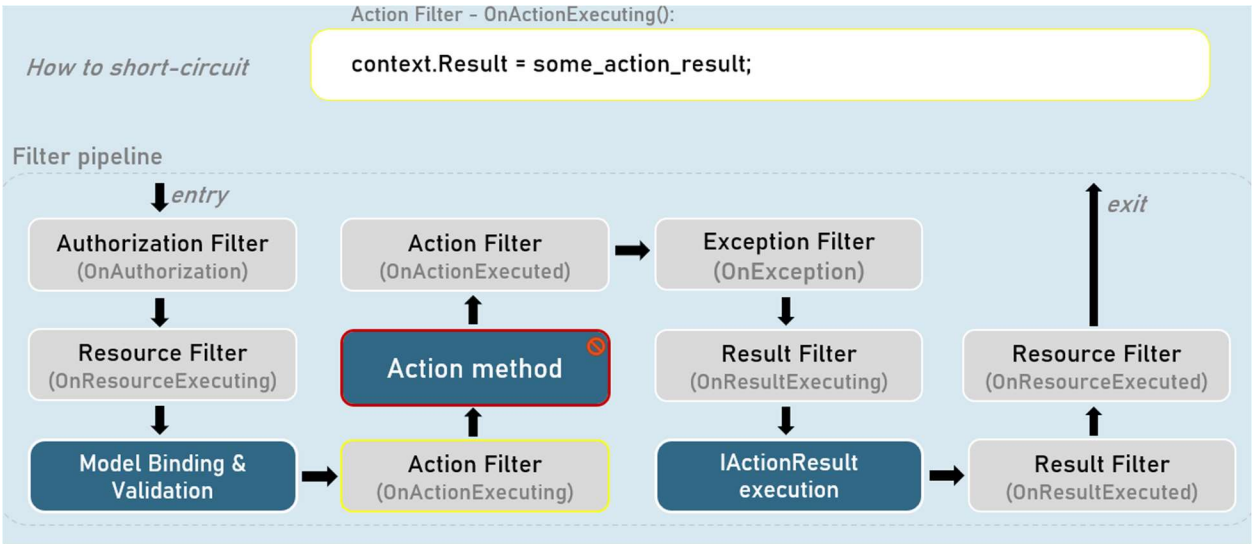
Short-circuiting Authorization Filter



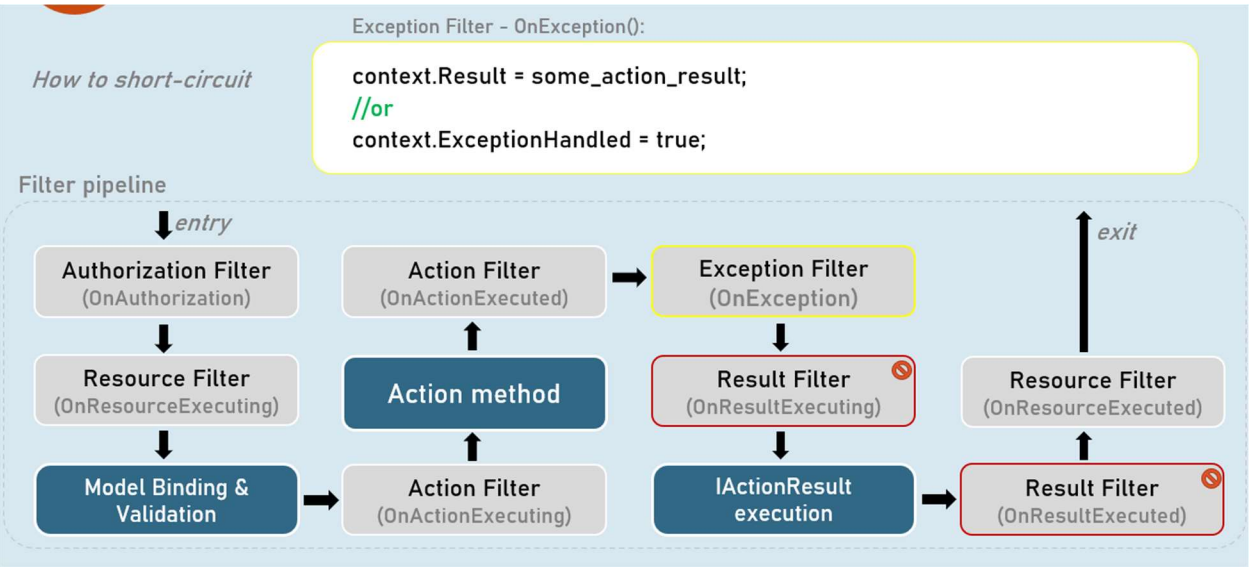
Short-circuiting Resource Filter



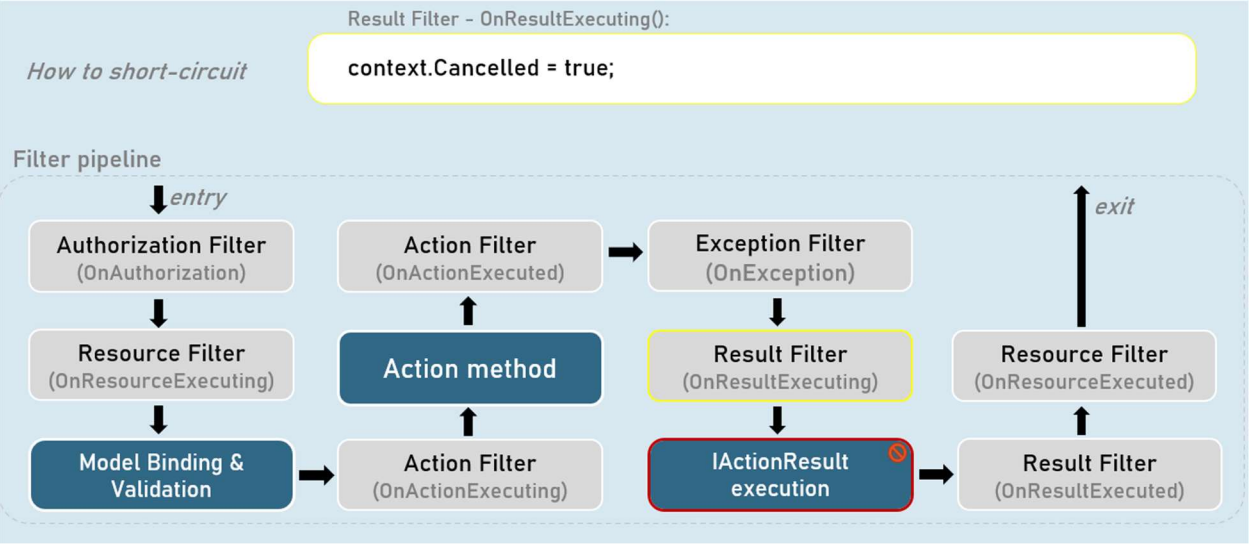
Short-circuiting Action Filter



Short-circuiting Exception Filter



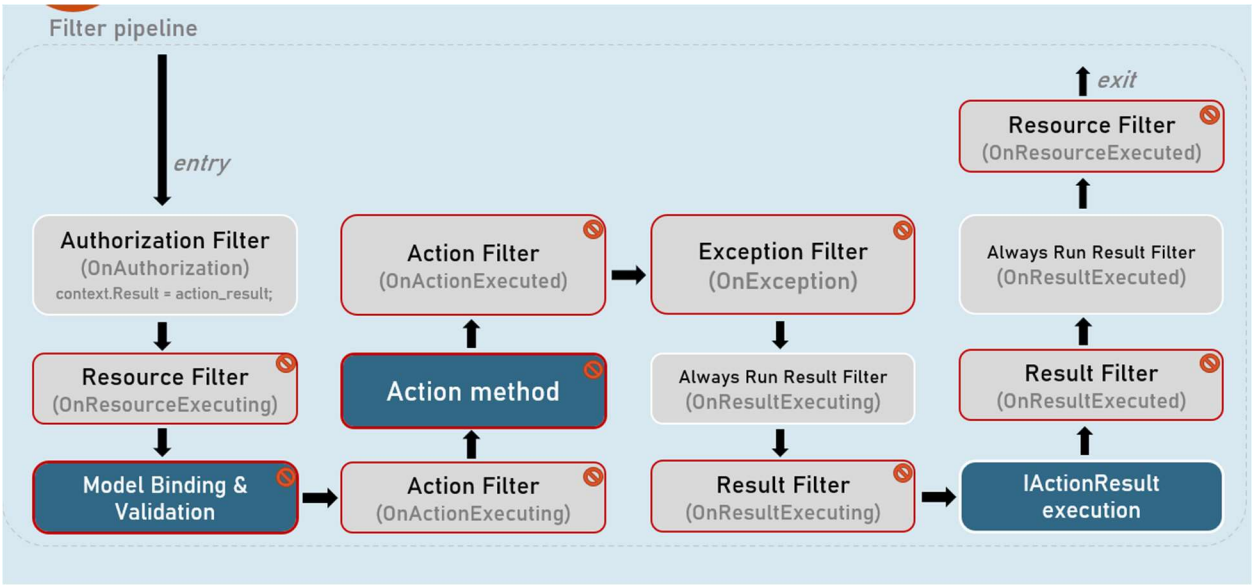
Short-circuiting Result Filter



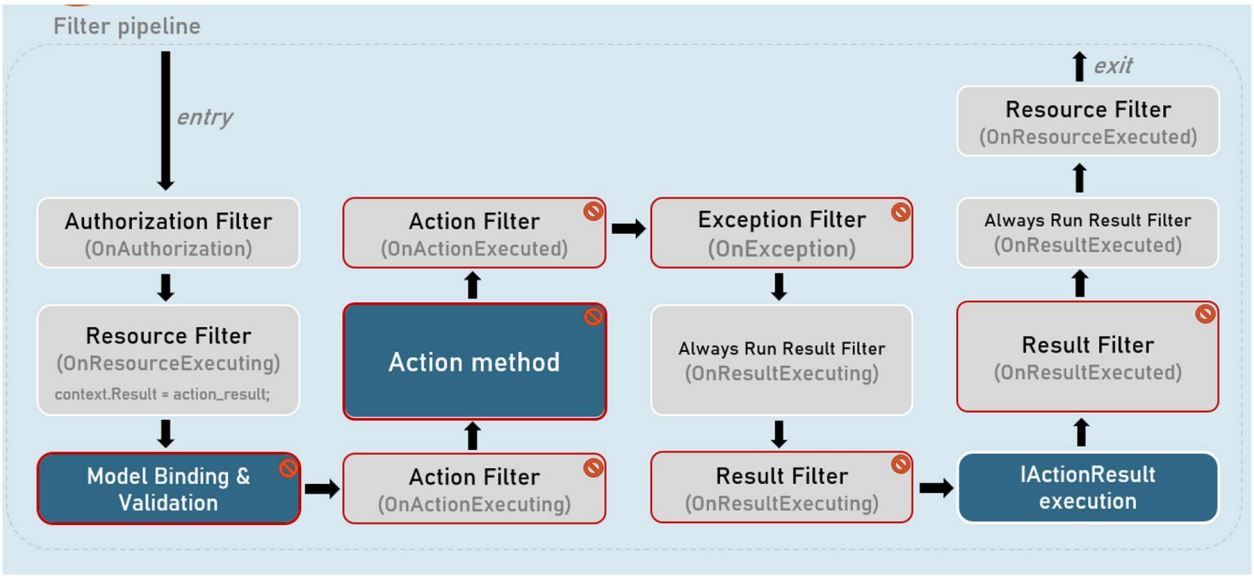
Short-circuiting the filters

AlwaysRun Result Filter

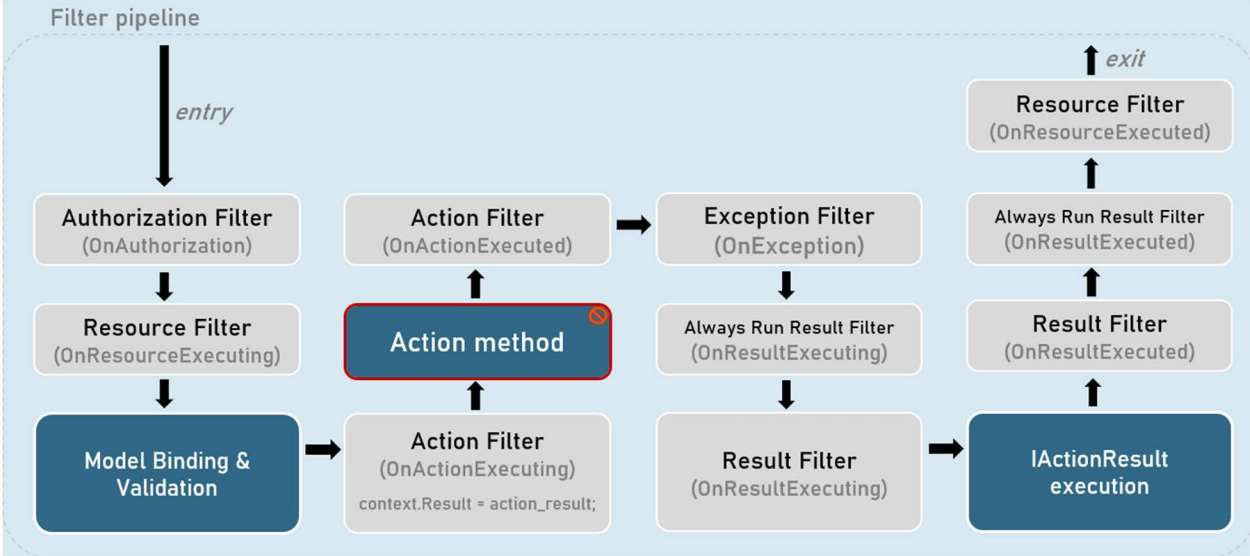
Short-circuiting Authorization Filter



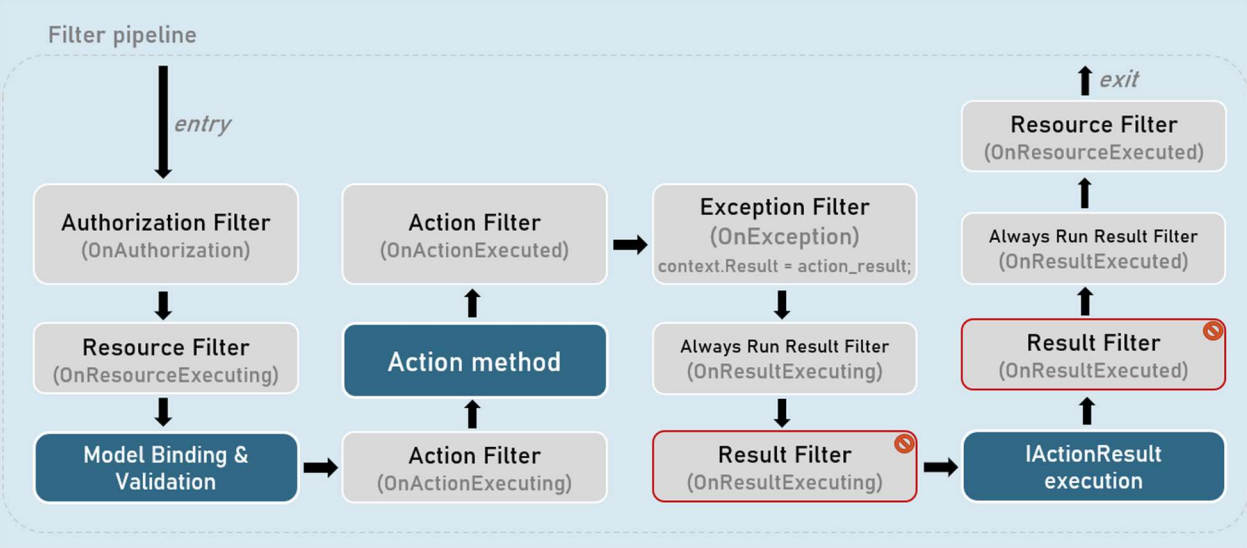
Short-circuiting Resource Filter



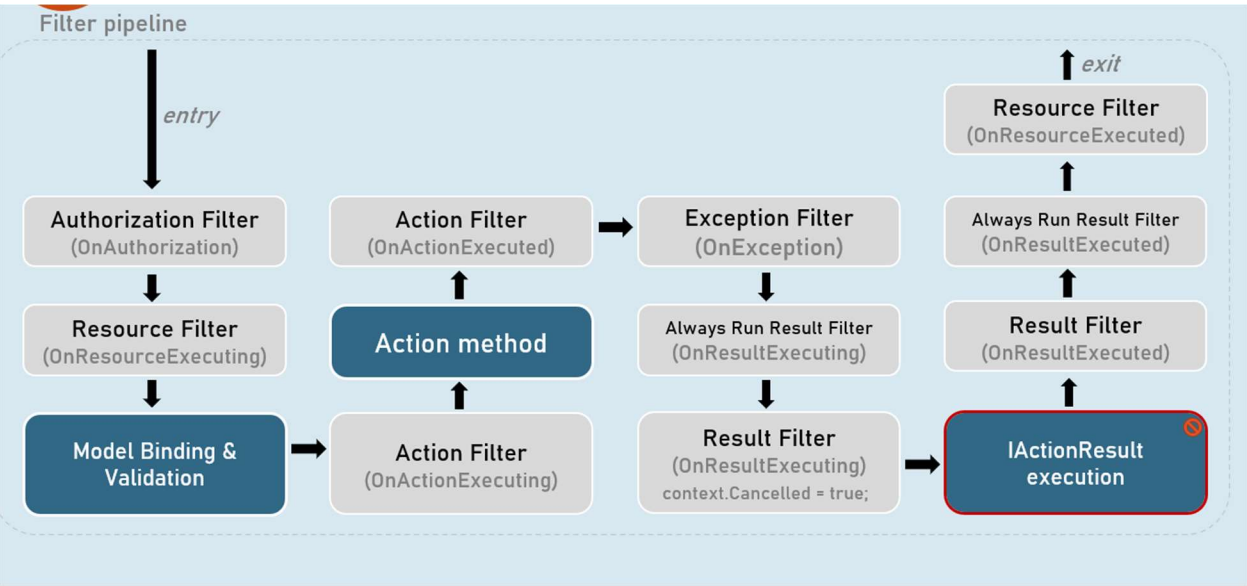
Short-circuiting Action Filter



Short-circuiting Exception Filter



Short-circuiting Result Filter



When AlwaysRunResultFilter runs

Runs immediately before and after result filters.

Result filters:

Doesn't execute when authorization filter, resource filter or exception filter short-circuits.

AlwaysRunResult filter:

Execute always even when authorization filter, resource filter or exception filter short-circuits.

'OnResultExecuting' method

Same as Result filter

'OnResultExecuted' method

Same as Result filter

Synchronous Always Run Result Filter

```
1. public class FilterClassName : IAlwaysRunResultFilter
2. {
3.   public void OnResultExecuting(ResultExecutingContext context)
4.   {
5.     //TO DO: before logic here
6.   }
7.   public void OnResultExecuted(ResultExecutedContext context)
8.   {
9.     //TO DO: after logic here
```

```
  }
}
```

Asynchronous Always Run Result Filter

```
public class FilterClassName : IAsyncAlwaysRunResultFilter
{
  public async Task OnResultExecutionAsync(ResultExecutingContext context,
ResultExecutionDelegate next)
  {
   //TO DO: before logic here
   await next();
   //TO DO: after logic here
  }
}
```

Filter Overrides

```
[TypeFilter(typeof(FilterClassName))] //filter applied at controller level
public class ControllerName : Controller
{
  public IActionResult Action1() //requirement: The filter SHOULD execute
  {
  }
  public IActionResult Action2() //requirement: The filter SHOULD NOT execute.
But how??
  {
  }
}
```

Attribute to be applied to desired action method

```
public class SkipFilterAttribute : Attribute, IFilterMetadata
{
}
```

Action method

```
[SkipFilter]
public IActionResult ActionMethod()
{
}
```

Filter that respects 'SkipFilterAttribute'

```
public class FilterClassName : IActionFilter //or any other filter interface
{
  public void OnActionExecuting(ActionExecutingContext context)
  {
    //get list of filters applied to the current working action method
    if (context.Filters.OfType<SkipResultFilter>().Any())
    {
      return;
    }
```

```
    //TO DO: before logic here
  }

  public void OnActionExecuted(ActionExecutedContext context)
  {
    //TO DO: after logic here
  }
}
```

It skips execution of code of a filter, for specific action methods.

[ServiceFilter]

Common purpose

Both are used to apply a filter a controller or action method.

Type Filter Attribute

```
//can supply arguments to filter
[TypeFilter(typeof(FilterClassName), Arguments = new object[] { arg1, arg2 })]
public IActionResult ActionMethod()
{
  …
}
```

Service Filter Attribute

```
//can't supply arguments to filter
[ServiceFilter(typeof(FilterClassName))]
public IActionResult ActionMethod()
{
  …
}
```

Type Filter

- Can supply arguments to the filter.
- Filter instances are created by using Microsoft.Extensions.DependencyInjection. ObjectFactory.
- They're NOT created using DI (Dependency Injection).
- The lifetime of filter instances is by default transient (a new filter instance gets created every time when it is invoked).
- But optionally, you can re-use the same instance of filter class across multiple requests, by setting a property called TypeFilterAttribute.IsReusable to 'true'.
- Filter classes NEED NOT be registered (added) to the IoC container.
- Filter classes CAN inject services using both constructor injection or method injection.

Service Filter

- Can't supply arguments to the filter.
- Filter instances are created by using ServiceProvider (using DI).
- The lifetime of filter instances is the actual lifetime of the filter class added in the IoC container.
- Eg: If the filter class is added to the IoC container with AddScoped() method, then its instances are scoped.
- Filter class SHOULD be registered (added) to the IoC container, much like any other service.
- Filter classes CAN inject services using both constructor injection or method injection.

Filter attribute classes

IActionFilter [vs] ActionFilterAttribute

[versus]

Action filter that implements 'IActionFilter'

```
1. public class FilterClassName : IActionFilter, IOrderedFilter
2. {
3.   //supports constructor DI
4. }
```

Action filter that inherits 'ActionFilterAttribute'

```
1. public class FilterClassName : ActionFilterAttribute
2. {
3.   //doesn't support constructor DI
4. }
```

Filter interfaces:

- IAuthorizationFilter
- IResourceFilter
- IActionFilter
- IExceptionFilter
- IResultFilter
- IAsyncAuthorizationFilter
- IAsyncResourceFilter
- IAsyncActionFilter
- IAsyncExceptionFilter
- IAsyncResultFilter

Filter attributes:

- ActionFilterAttribute
- ExceptionFilterAttribute

- ResultFilterAttribute

Action filter that implements 'IActionFilter'

```
public class FilterClassName : IActionFilter, IOrderedFilter
{
  public int Order { get; set; }

  public FilterClassName(IService service, type arg)
  {
  }

  public void OnActionExecuting(ActionExecutingContext context)
  {
  }

  public void OnActionExecuted(ActionExecutedContext context)
  {
  }
}
```

```
[TypeFilter(typeof(FilterClassName),
Arguments = new object[] { arg1, … })]
```

Action filter that inherits 'ActionFilterAttribute'

```
public class FilterClassName : ActionFilterAttribute
{
  public FilterClassName(type arg)
  {
  }

  public override void OnActionExecuting(ActionExecutingContext context)
  {
  }

  public override void OnActionExecuted(ActionExecutedContext context)
  {
  }
}
```

```
[FilterClassName(arg1, … )]
```

Internal definitions of IActionFilter and ActionFilterAttribute

IActionFilter

```
namespace Microsoft.AspNetCore.Mvc.Filters
{
  public interface IActionFilter : IFilterMetadata
  {
    void OnActionExecuting(ActionExecutingContext context);
    void OnActionExecuted(ActionExecutedContext context);
```

```
7.    }
8.  }
```

ActionFilterAttribute

```
1.  namespace Microsoft.AspNetCore.Mvc.Filters
2.  {
3.    public class ActionFilterAttribute : Attribute, IActionFilter,
    IAsyncActionFilter, IOrderedFilter, IResultFilter, IAsyncResultFilter
4.    {
5.      public virtual void OnActionExecuting(ActionExecutingContext context) { }
6.      public virtual void OnActionExecuted(ActionExecutedContext context) { }
7.      public virtual void OnResultExecuting(ActionExecutingContext context) { }
8.      public virtual void OnResultExecuted(ActionExecutedContext context) { }
9.      public virtual Task OnActionExecutionAsync(ActionExecutingContext context,
    ActionExecutionDelegate next) { }
10.     public virtual Task OnResultExecutionAsync(ResultExecutingContext context,
    ResultExecutionDelegate next) { }
11.     public int Order { get; set; }
12.   }
13. }
```

Filter interface [vs] FilterAttribute class

Filter interface [such as IActionFilter, IResultFilter etc.]

- Filter class MUST implement all methods - both "Executing" and "Executed" methods.
- Filter class CAN have DI with either constructor injection or method injection.
- Doesn't implement "Attribute" class.
- Filter should be applied to controller or action methods by using [ServiceFilter] or [TypeFilter] attributes; otherwise can be applied as global filter in the Program.cs.

  Eg: [TypeFilter(typeof(FilterClassName))] //lengthy

- Filter class can receive arguments only through constructor parameters; but only with [TypeFilter] attribute; not with [ServiceFilter] attribute.

FilterAttribute class [such as ActionFilterAttribute etc.]

- Filter class MAY override desired (either or both methods - "Executing" and "Executed") methods.
- Filter class CAN'T have DI with neither constructor injection nor method injection.
- FilterAttribute class [such as ActionFilterAttribute etc.]
- Filter can be applied to controller or action methods by directly using the filter class name itself (without using [ServiceFilter] or [TypeFilter] attributes); otherwise can be applied as global filter in the Program.cs.
- Eg: [FilterClassName] //simple
- Filter class can receive arguments either through constructor parameters or filter class's properties.

IFilterFactory

Filter factory that inherits 'IFilterFactory'

```
1. public class FilterClassNameAttribute : Attribute,
2. IFilterFactory
3. {
4.   public type Prop1 { get; set; }
5.
6.   public FilterClassName(type arg1, type arg2)
7.   {
8.     this.Prop1 = arg1; this.Prop2 = arg2;
9.   }
10.
11.  public IFilterMetadata CreateInstance(IServiceProvider serviceProvider)
12.  {
13.    FilterClassName filter =
   serviceProvider.GetRequiredService<FilterClassName>(); //instantiate the
   filter
14.    filter.Property1 = Prop1;
15.    …
16.    return filter;
17.  }
18.}
```

```
[FilterClassName(arg1, arg2, … )]
```

Action filter that inherits 'ActionFilterAttribute'

```
1. public class FilterClassName : ActionFilterAttribute
2. {
3.   public FilterClassName(type arg1, type arg2)
4.   {
5.   }
6.
7.   public override void OnActionExecuting(ActionExecutingContext context)
8.   {
9.   }
10.
11.  public override void OnActionExecuted(ActionExecutedContext context)
12.  {
13.  }
14.}
```

```
[FilterClassName(arg1, arg2, … )]
```

IFilterFactory

```
1. namespace Microsoft.AspNetCore.Mvc.Filters
2. {
3.   public interface IFilterFactory : IFilterMetadata
4.   {
```

```
5.      IFilterMetadata CreateInstance(IServiceProvider serviceProvider);
6.      bool IsReusable { get; }
7.   }
8. }
```

FilterAttribute class [such as ActionFilterAttribute etc.]

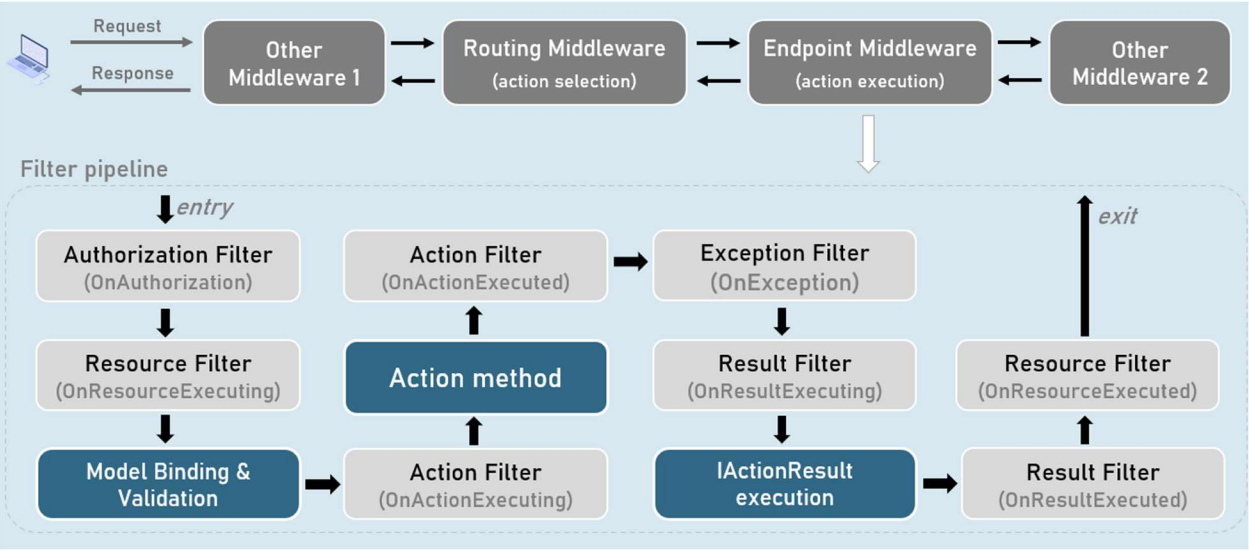
- Filter CAN be applied as an attribute to the controller or action method. Eg: [FilterClassName]
- Filter class CAN'T have DI with neither constructor injection nor method injection.
- Filter class CAN receive arguments either through constructor parameters or filter class's properties.

IFilterFactory

- Filter CAN be applied as an attribute to the controller or action method. Eg: [FilterClassName]
- Filter class CAN have DI with either constructor injection or method injection.
- Filter class CAN receive arguments only through filter class's properties, if it is instantiated through ServiceProvider (using DI).

Alternatively, if you don't need to inject services using DI in the filter class; you can instantiate the filter class with 'new' keyword, in the CreateInstance() method of IFilterFactory; then the filter class can receive arguments either as constructor parameters or properties.

Filters [vs] Middleware



Middleware

Middleware pipeline is a superset of Filter pipeline, which contains the full-set of all middlewares added to the ApplicationBuilder in the application's startup code (Program.cs).

Middleware pipeline execute for all requests.

Middleware handles application-level functionality such as Logging, HTTPS redirection, Performance profiling, Exception handling, Static files, Authentication etc., by accessing low-level abstractions such as HttpContext.

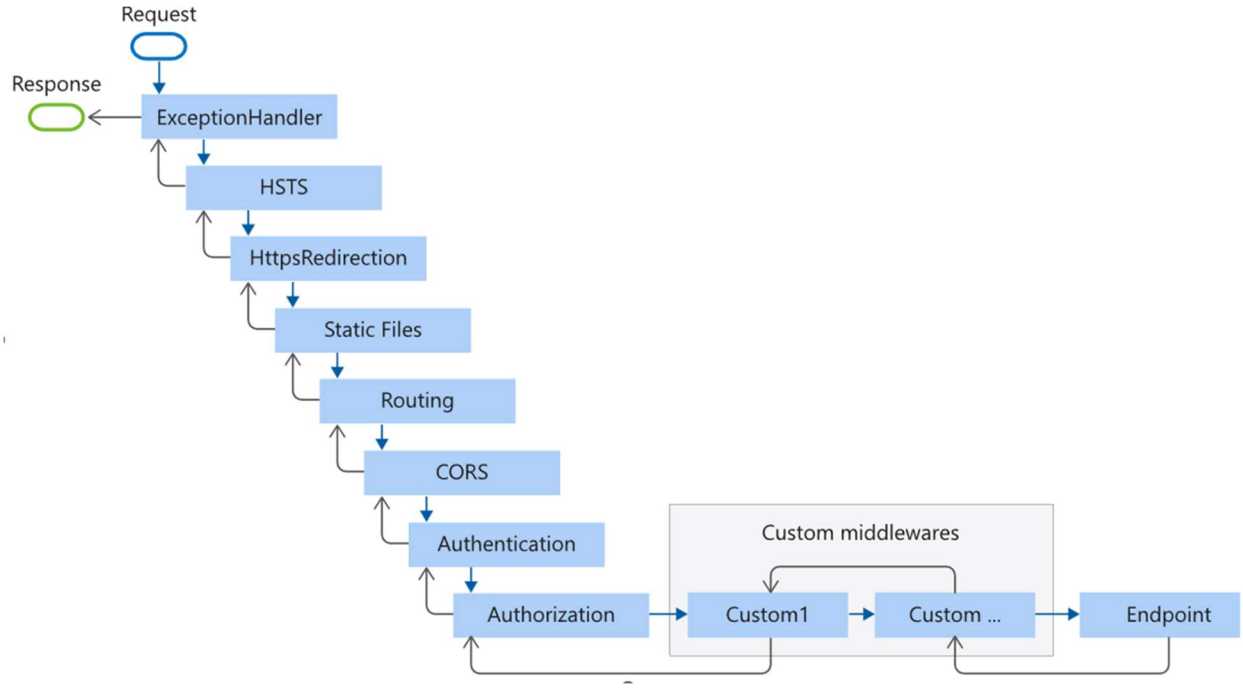
Filter

Filter pipeline is a subset of Middleware pipeline which executes under "EndPoint Middleware".

In addition, filter pipeline executes for requests that reach "EndPoint Middleware".

Filters handle MVC-specific functionality such as manipulating or accessing ViewData, ViewBag, ModleState, Action result, Action parameters etc.

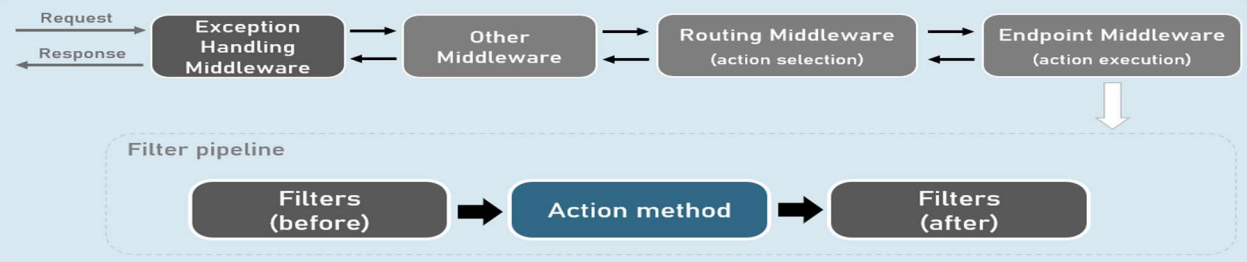
Middleware Pipeline



Exception Handling Middleware

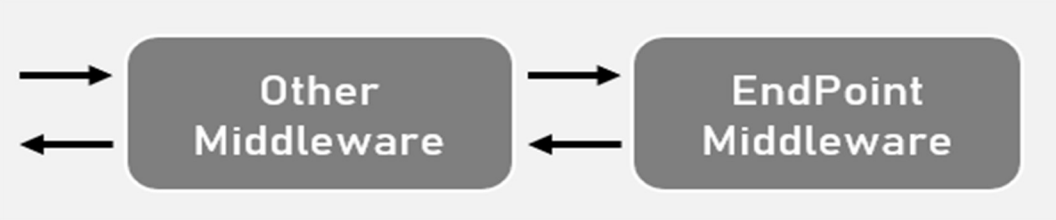
Handles all errors occurred in filter pipeline (including model binding, controllers and filters).

Should be added to the application pipeline, before RoutingMiddleware.



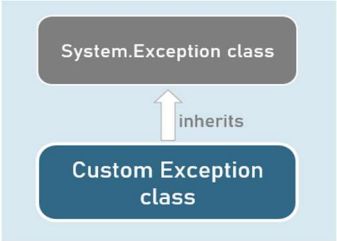
Custom Exception Handling Middleware

```
1. public class ExceptionHandlingMiddleware
2. {
3.   private readonly RequestDelegate _next; //Stores reference of subsequent
   middleware
4.
5.   public ExceptionHandlingMiddleware(RequestDelegate next)
6.   {
7.     _next = next;
8.   }
9.   public async Task InvokeAsync(HttpContext context)
10.  {
11.    try
12.    {
13.      await _next(context);
14.    }
15.    catch(Exception ex)
16.    {
17.      …
18.    }
19.  }
20. }
```



Custom Exceptions

A custom exception class is an exception class that inherits from System.Exception class & represents a domain-specific exception



Used to represent the domain-specific errors stand-out of system-related (.NET) related exceptions.

Custom Exception class

```
1. public class CustomException : Exception
2. {
3.   public CustomException() : base()
4.   {
5.   }
6.
7.   public CustomException(string? message) : base(message)
8.   {
9.   }
10.
11.  public CustomException(string? message, Exception? innerException) :
   base(message, innerException)
12.  {
13.  }
14.}
```

UseExceptionHandler()

The built-in UseExceptionHandler() middleware redirects to the specified route path, when an unhandled exception occurs during the application execution.

Can be used as an alternative to custom exception handling middleware.

Catches and logs unhandled exceptions.

Re-executes the request in an alternative pipeline using the specified route path.

Overview of SOLID Principles

"SOLID" is a set of five design patterns, whose main focus is to create loosely coupled, flexible, maintainabile code.

Broad goal of SOLID Principles:Reduce dependencies of various classes / modules of the application.

Single Responsibility Principle (SRP)

A software module or class should have one-and-only reason to change.

Liskov Subsitution Principle (LSP)

Subtypes must be substitutable for their base types.

Open-Closed Principle (OCP)

A class is closed for modifications; but open for extension.

Interface Segregation Principle (ISP)

No client class should be forced to depend on methods it does not use.

Dependency Inversion Principle (DIP)

High-level modules should not depend on low-level modules. Both should depend upon abstractions.

Dependency Inversion Principle (DIP)

Direct Dependency

Controller (Client)

```
1. public class MyController : Controller
2. {
3.   private readonly MyService _service;
4.   public MyController()
5.   {
6.     _service = new MyService(); //direct
7.   }
8.   public IActionResult ActionMethod()
9.   {
10.    _service.ServiceMethod();
11.  }
12.}
```

Service (Dependency)

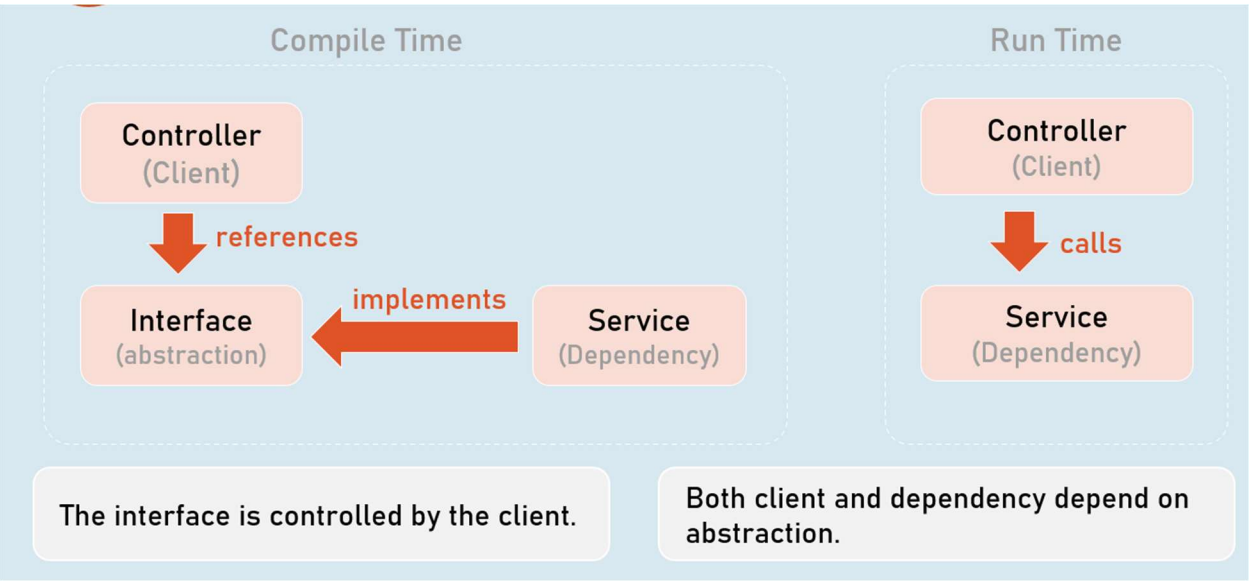
```
1. public class MyService
2. {
3.   public void ServiceMethod()
4.   {
5.     …
6.   }
7. }
```

Dependency Problem

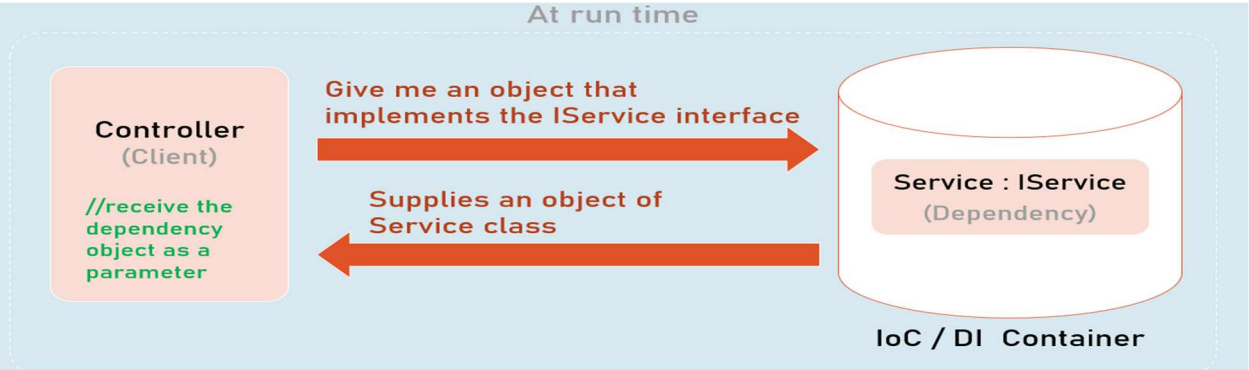
- Higher-level modules depend on lower-level modules.
- Means, both are tightly-coupled.
- The developer of higher-level module SHOULD WAIT until the completion of development of lower-level module.
- Requires much code changes in to interchange an alternative lower-level module.
- Any changes made in the lower-level module effects changes in the higher-level module.
- Difficult to test a single module without effecting / testing the other module.

Dependency Inversion Principle

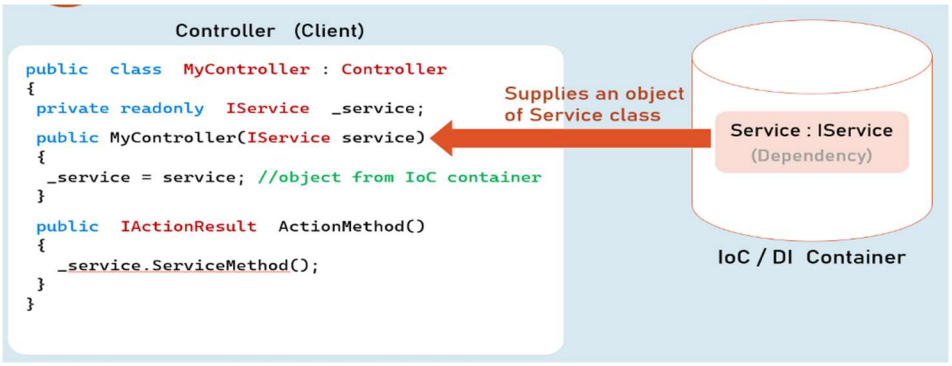
- Dependency Inversion Principle (DIP) is a design principle (guideline), which is a solution for the dependency problem.
- "The higher-level modules (clients) SHOULD NOT depend on low-level modules (dependencies).
- Both should depend on abstractions (interfaces or abstract class)."
- "Abstractions should not depend on details (both client and dependency).
- Details (both client and dependency) should depend on abstractions."



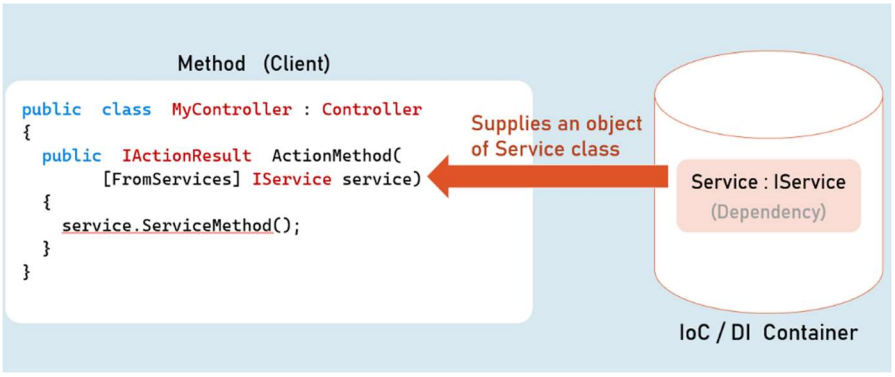
Dependency Injection



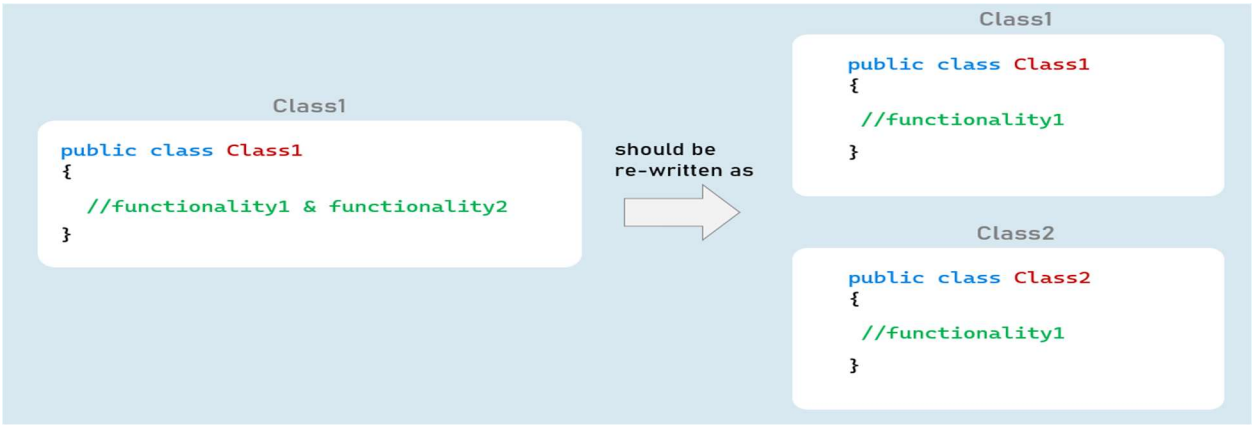
Constructor Injection



Method Injection



Single Responsibility Principle (SRP)



- A class should have one-and-only reason to change.
- A class should implement only one functionality.
- Avoid multiple / tightly coupled functionalities in a single class.
- Eg: A class that performs validation should only involve in validation.
- But it should not read configuration settings from a configuration file.
- But instead, it call a method of another class that reads configuration settings.

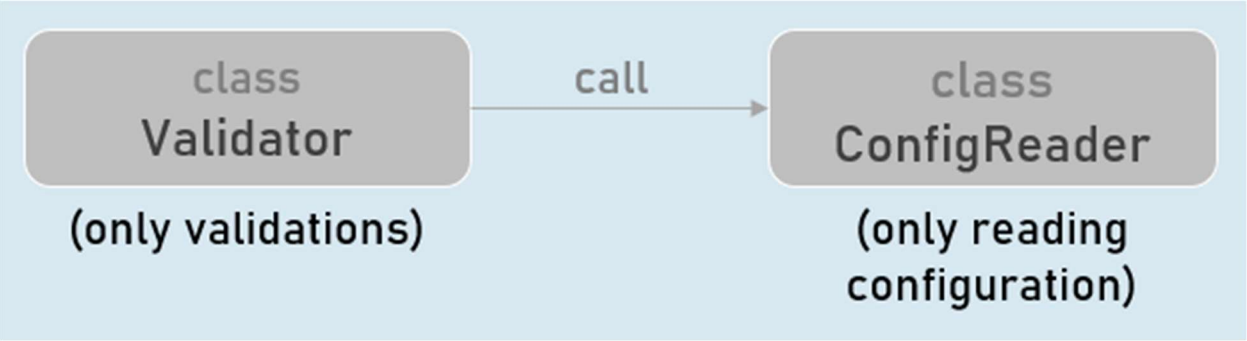
Interfaces

Create alternative implementation of the class by implementing the same interface.

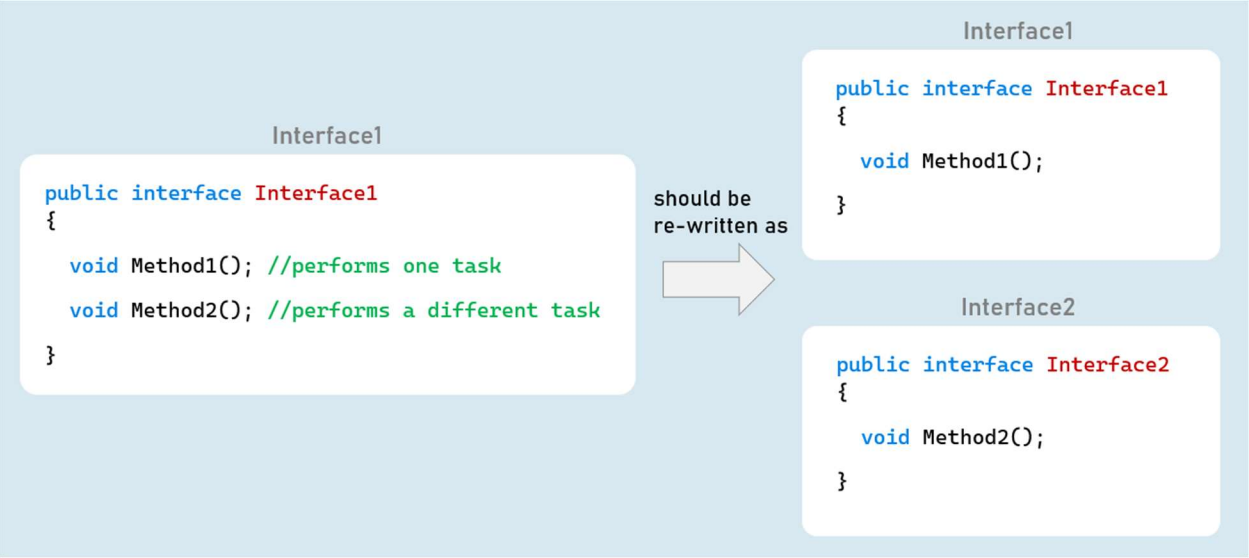
Benefit: Makes the class independent of other classes, in terms of its purpose / functionality.

So that, the classes become easier to design, write, debug, maintain and test.

Eg:



Interface Segregation Principle (ISP)



- No client class should be forced to depend on methods it doesn't use.
- We should prefer to make many smaller interfaces rather than one single big interface.
- The client classes may choose one or more interfaces to implement.
- Benefit: Makes it easy to create alternative implementation for a specific functionality, rather than recreating entire class.

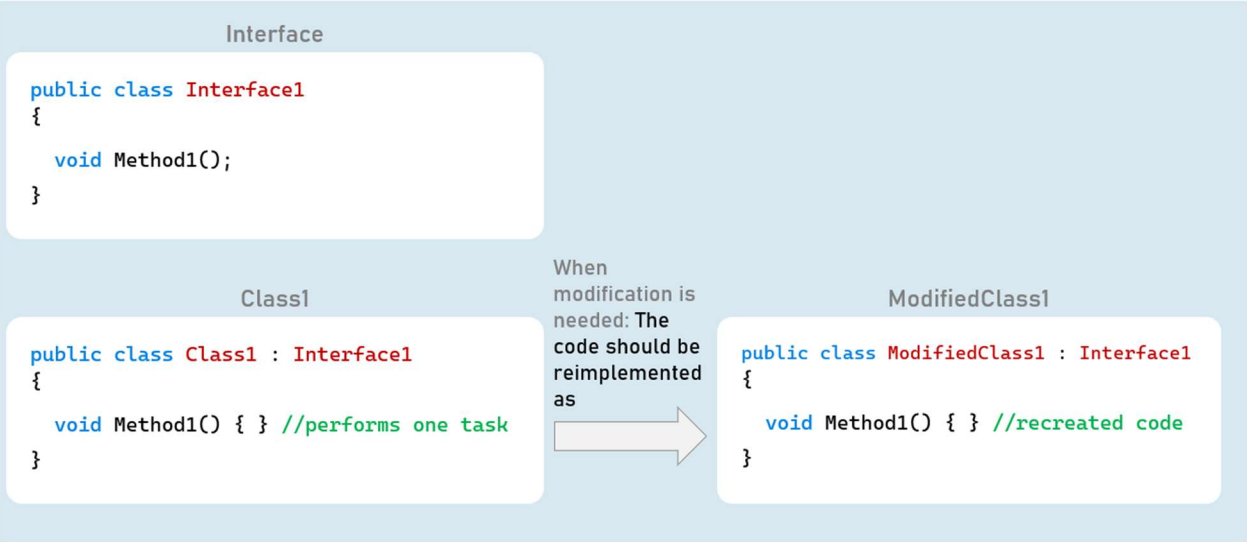
Eg:

Assume, a class has two methods: GetPersons() and AddPerson().

Instead of creating both methods in a single interface, create them as two different interfaces: IPersonGetter, IPersonAdder

1. interface IPersonsGetter (methods to get persons data)
2. interface IPersonsAdder(methods to create person)

Open/Closed Principle (OCP)



A class is closed for modifications; but open for extension.

You should treat each class as readonly for development means; unless for bug-fixing.

If you want to extend / modify the functionality of an existing class; you need to recreate it as a separate & alternative implementation; rather than modifying existing code of the class.

Eg:

Assume, a class has a method: GetPersons().

The new requirement is to get list of sorted persons.

Instead of modifying existing GetPersons() method, you need to create an alternative class that gets sorted persons list.

Benefit: Not modifying existing code of a class doesn't introduce new bugs; and keeps the existing unit tests stay relavant and needs no changes.

1. `class PersonGetter : IPersonGetter (GetPersons() method retrieves list of persons)`
2. `class SortedPersonGetter : IPersonGetter (GetPersons() method retrieves sorted list of persons)`

Interfaces

Create alternative implementation of the class by implementing the same interface.

Inheritance

Create a child class of the existing class and override the required methods that needs changes.

Liskov Substitution Principle (LSP)

Parent Class

```
1. public class ParentClass
2. {
3.   public virtual int Calculate(int? a, int? b)
4.   {
5.     //if 'a' or 'b' is null, throw ArgumentNullException
6.     //return sum of 'a' and 'b'
7.   }
8. }
```

Child Class

```
1. public class ChildClass : ParentClass
2. {
3.   public override int Calculate(int? a, int b)
4.   {
5.     //if 'a' or 'b' is null, throw ArgumentNullException
6.     //if 'a' or 'b' is negative, throw ArgumentException
7.     //return product of 'a' and 'b'
8.   }
9. }
```

[Violates LSP]

Functions that use references of base classes must be able to use objects of derived classes without breaking / changing its functionality.

The child classes that override methods of base class, should provide same behavior.

Using object of parent class

ParentClass variable = new ParentClass();

variable.Method(); //executes ParentClass.Method

Using object of child class

ParentClass variable = new ChildClass();

variable.Method(); //executes ChildClass.Method

[Both methods should offer same functionality]

Functions that use references of base classes must be able to use objects of derived classes without breaking / changing its functionality.

The child classes that override methods of base class, should provide same behavior.

If a derived class overrides a method of base class; then the method of derived class should provide same behavior:

With same input, it should provide same output (return value).

The child class's method should not introduce (throw) any new exceptions than what were thrown in the base implementation.

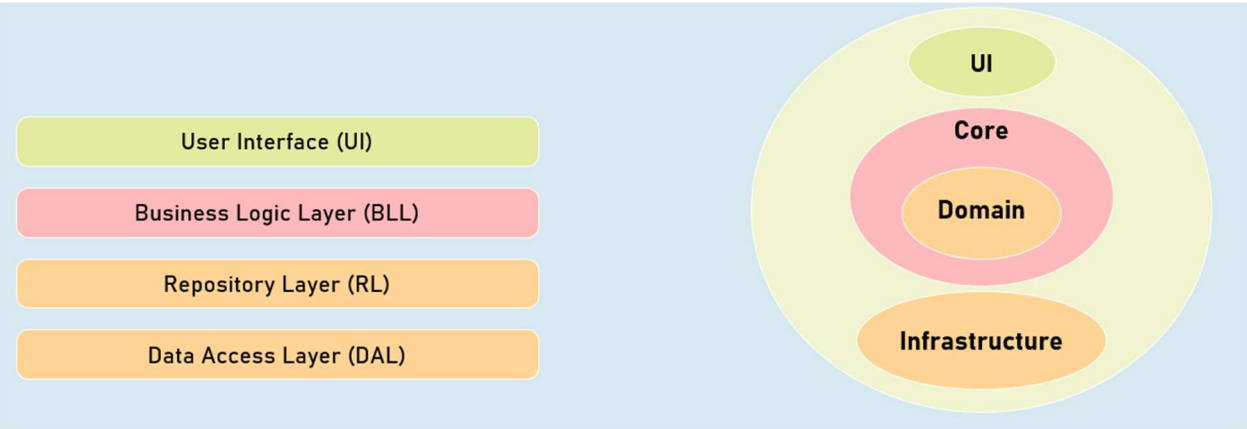
The child class's method should not implement stricter rules than base class's implementation.

Benefit: Prevents code to break - if by mistake or wantedly, someone has replaced the derived class with its base class (or even vice versa), as its behavior doesn't change.

Overview of Clean Architecture

Instead of "business logic" depend on "data access logic", this dependency is inverted; that means, the "data access logic" depend on "business logic".

Benefit: The business logic is highly clean-separated, independent of data storage and UI, unit-testable.



Traditional Three-Tier / N-Tier Architecture

1. User Interface (UI)
2. Business Logic Layer (BLL)
3. Repository Layer (RL)
4. Data Access Layer (DAL)

Clean Architecture

UI

1. Controllers, Views, View Models
2. Filters, Middleware

Core

1. Business Logic Services
2. Business Logic Interfaces
3. Data Transfer Objects (DTO)

Domain

1. Repository Interfaces
2. Entity Classes

Infrastructure

1. DbContext, Repositories
2. External API Calls

Clean Architecture

Changing external system

Allows you to change external systems (external APIs / third party services) easily, without affecting the application core.

Scalable

You can easily scale-up or scale-out, without really affecting overall architecture of the application.

Database independent

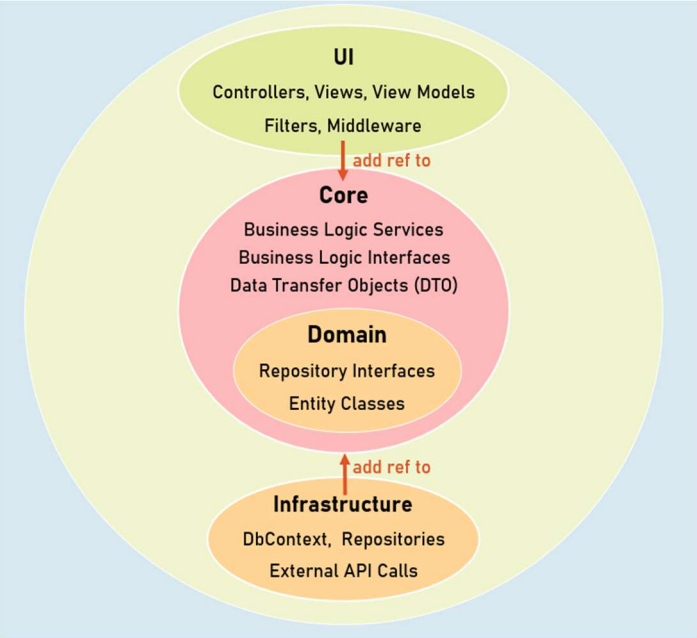
The application core doesn't depend on specific databases; so you can change it any time, without affecting the application core.

Testable

The application core doesn't depend on any other external APIs or repositories; so that you can write unit tests against business logic services easily by mocking essential repositories.

Clean architecture is earlier named as "Hexagonal architecture", "Onion architecture", "Domain-Driven Design", "Vertical Slice Architecture". Over time, it is popular as "clean architecture".

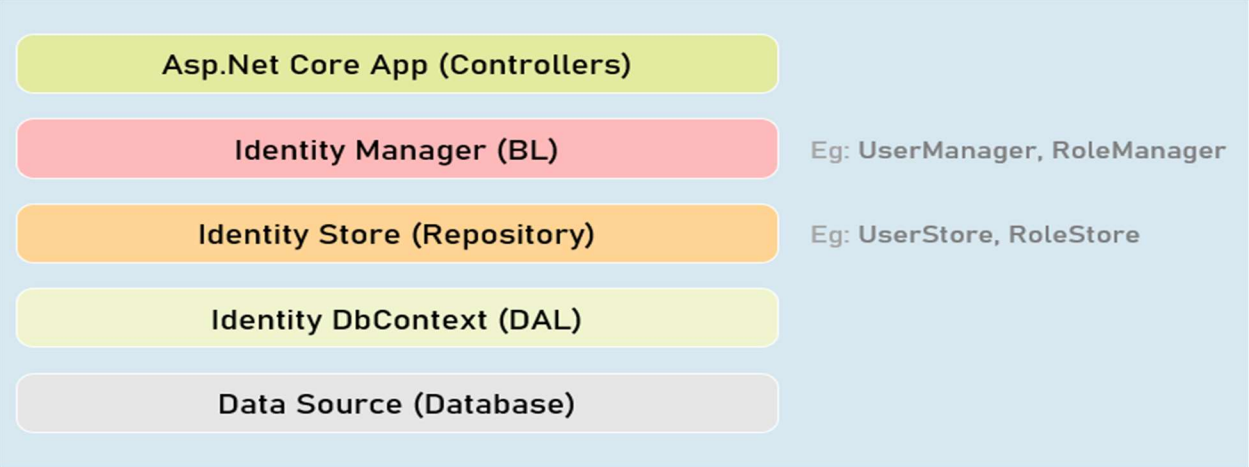
Clean Architecture



Introduction to Identity

It is an API that manages users, passwords, profile data, roles, tokens, email confirmation, external logins etc.

It is by default built on top of EntityFrameworkCore; you can also create custom data stores.



IdentityUser<T>

Acts as a base class for ApplicationUser class that acts as model class to store user details.

You can add additional properties to the ApplicationUser class.

Built-in Properties:

1. Id
2. UserName
3. PasswordHash
4. Email
5. PhoneNumber

IdentityRole<T>

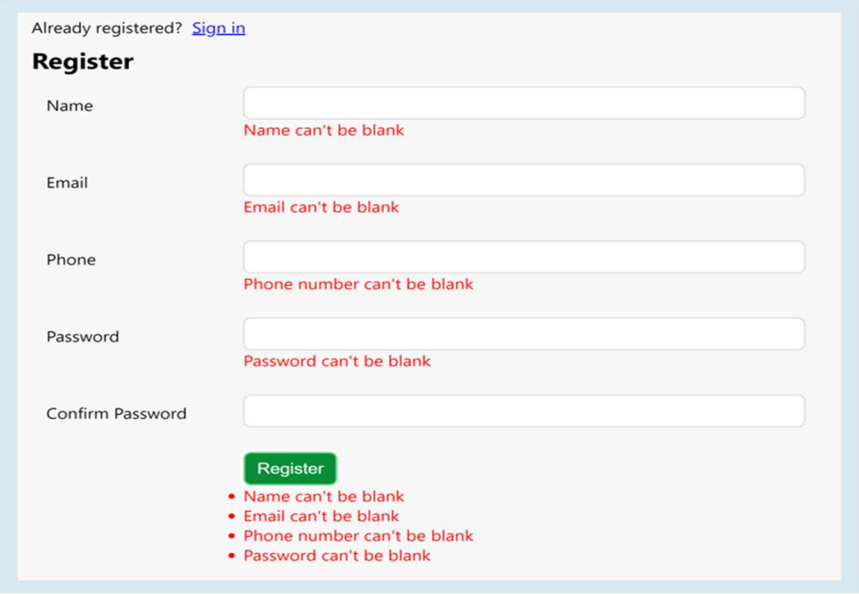
Acts as a base class for ApplicationRole class that acts as model class to store role details. Eg: "admin"

You can add additional properties to the ApplicationRole class.

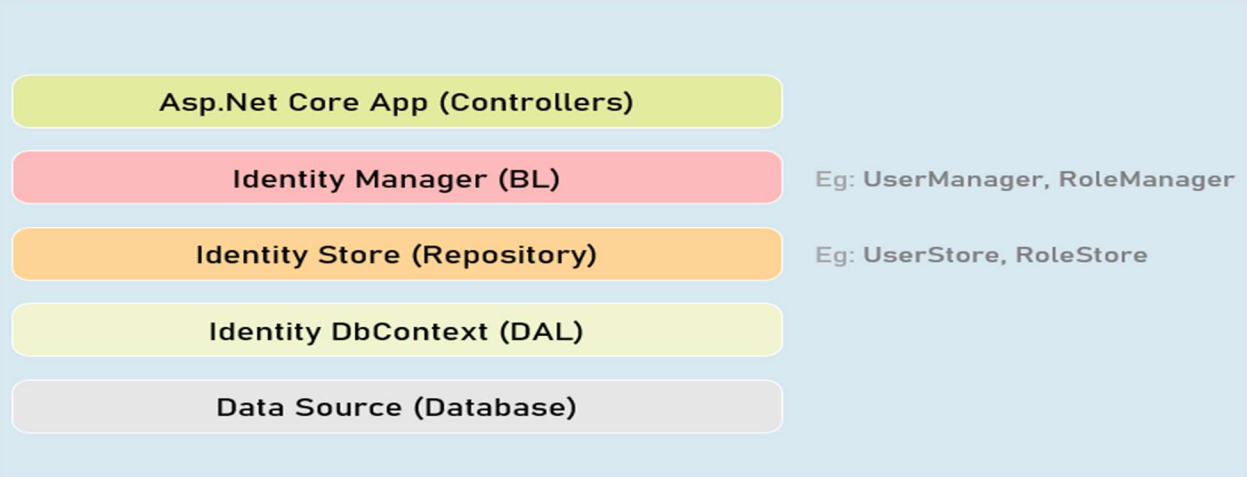
Built-in Properties:

1. Id
2. Name

Register View



Managers



UserManager

Provides business logic methods for managing users.

It provides methods for creating, searching, updating and deleting users.

Methods:

- CreateAsync()
- DeleteAsync()
- UpdateAsync()
- IsInRoleAsync()
- FindByEmailAsync()
- FindByIdAsync()
- FindByNameAsync()

SignInManager

Provides business logic methods for sign-in and sign-in functionality of the users.

It provides methods for creating, searching, updating and deleting users.

Methods:

SignInAsync()

PasswordSignInAsync()

SignOutAsync()

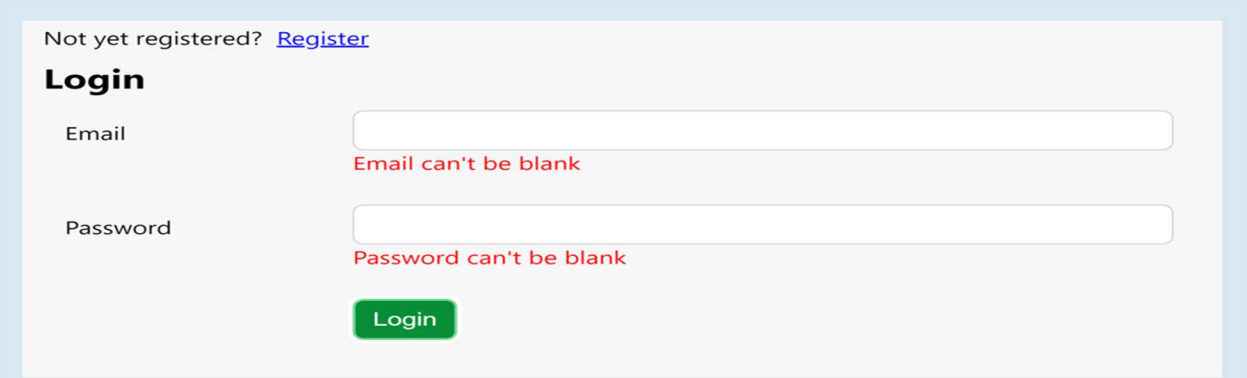
IsSignedIn()

Password Complexity Configuration

```
1. services.AddIdentity<ApplicationUser, ApplicationRole>(options => {
2.   options.Password.RequiredLength = 6; //number of characters required in
   password
3.   options.Password.RequireNonAlphanumeric = true; //is non-alphanumeric
   characters (symbols)
4. required in password
5.   options.Password.RequireUppercase = true; //is at least one upper case
   character required in password
6.   options.Password.RequireLowercase = true; //is at least one lower case
   character required in password
7.   options.Password.RequireDigit = true; //is at least one digit required in
   password
8.   options.Password.RequiredUniqueChars = 1; //number of distinct characters
   required in password
9. })
10. .AddEntityFrameworkStores<ApplicationDbContext>()
11. .AddDefaultTokenProviders()
12. .AddUserStore<UserStore<ApplicationUser, ApplicationRole,
   ApplicationDbContext, Guid>>()
13. .AddRoleStore<RoleStore<ApplicationRole, ApplicationDbContext, Guid>>();
14.
```

Login/Logout Buttons

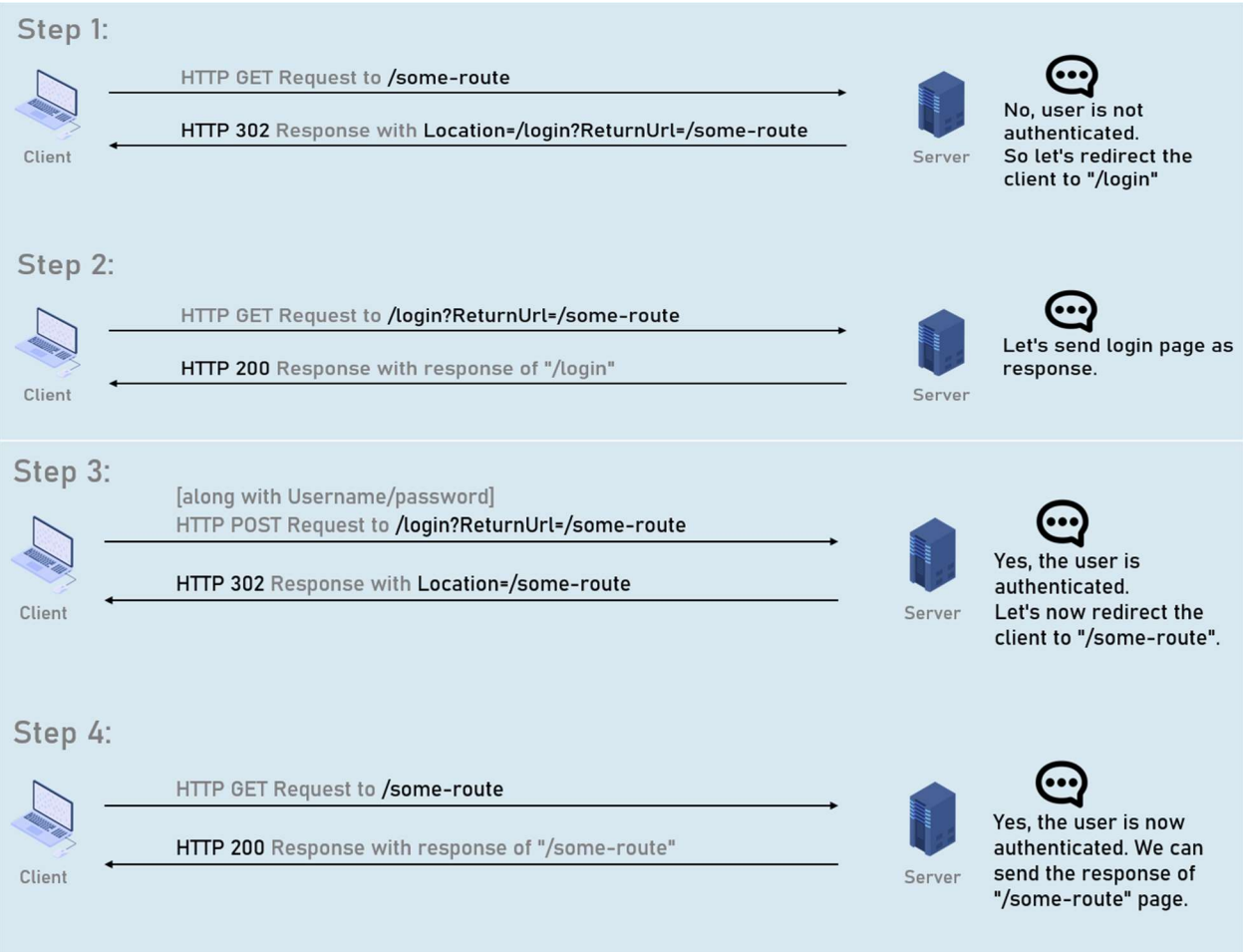
Login View



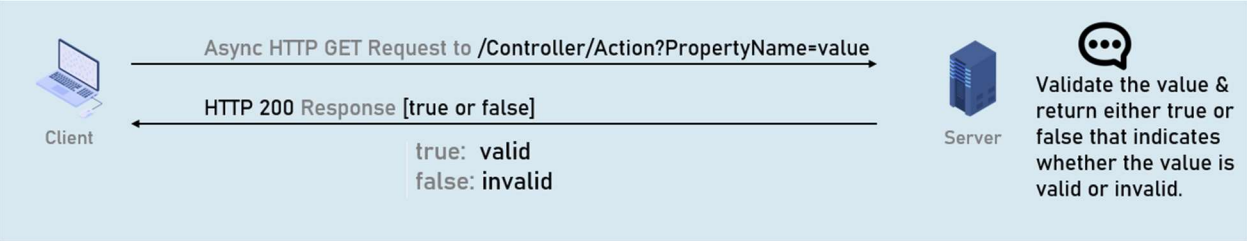
Authorization Policy

```
1. services.AddAuthorization(options =>
2. {
3.   var policy = new
   AuthorizationPolicyBuilder().RequireAuthenticatedUser().Build();
4.   options.FallbackPolicy = policy;
5. });
```

ReturnUrl



Remote Validation



Model class

```
1. public class ModelClassName
2. {
3.   [Remote(action: "action name", controller: "controller name", ErrorMessage =
   "error message")]
4.   public type PropertyName { get; set; }
5. }
```

Conventional Routing

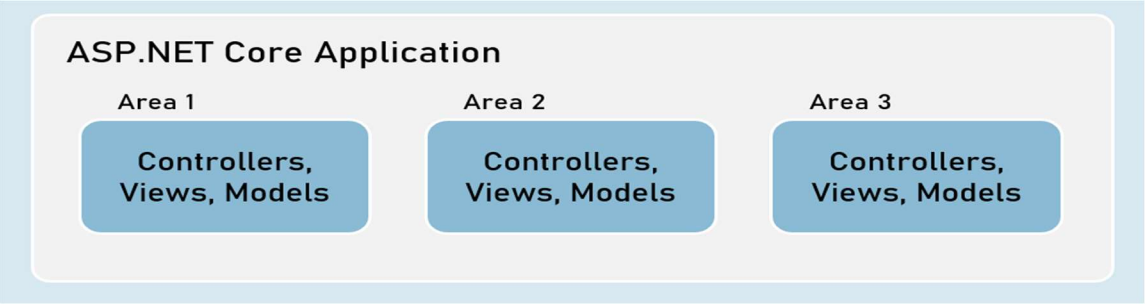
Conventional routing is a type of routing system in asp.net core that defines route templates applied on all controllers in the entire application.

You can override this using attribute routing on a specific action method.

```
1. endpoints.MapControllerRoute(
2.   name: "default",
3.   pattern: "{controller=Persons}/{action=Index}/{id?}"
4. );
```

Areas

Area is a group of related controllers, views and models that are related to specific module or specific user.



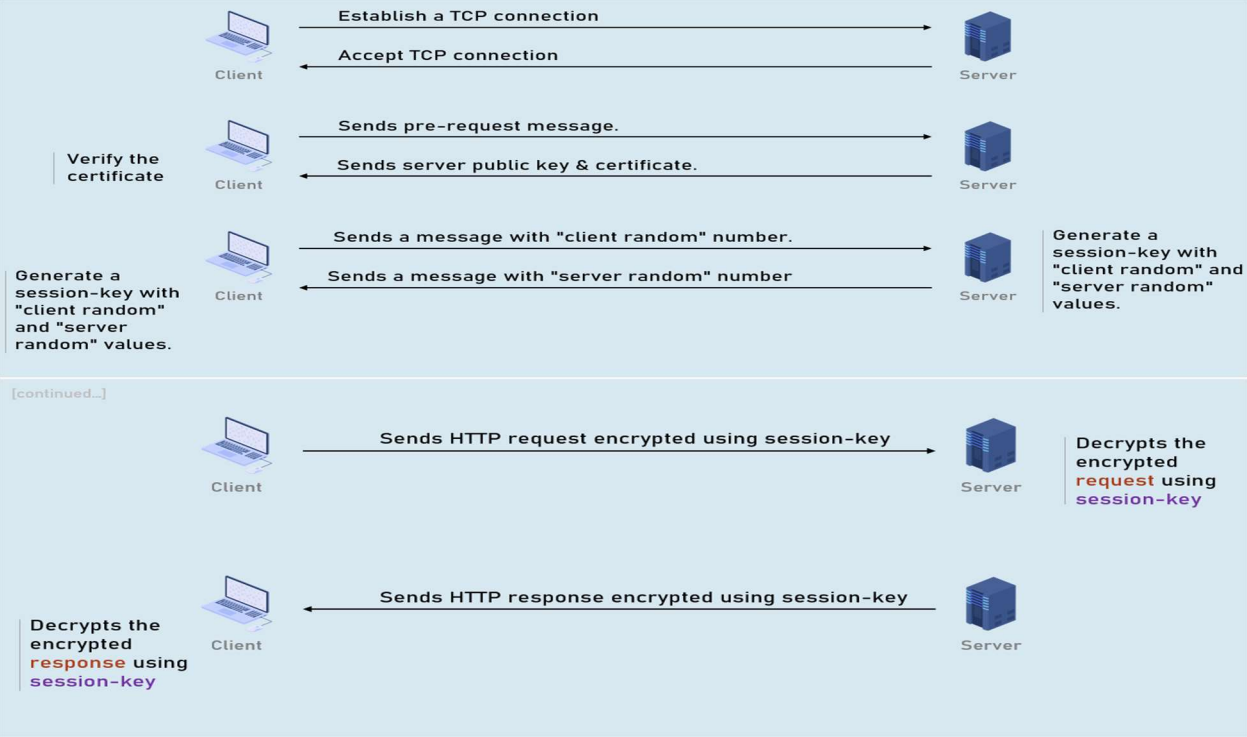
User Roles

Role Based Authentication

User-role defines type of the user that has access to specific resources of the application.

Examples: Administrator role, Customer role etc.

HTTPS

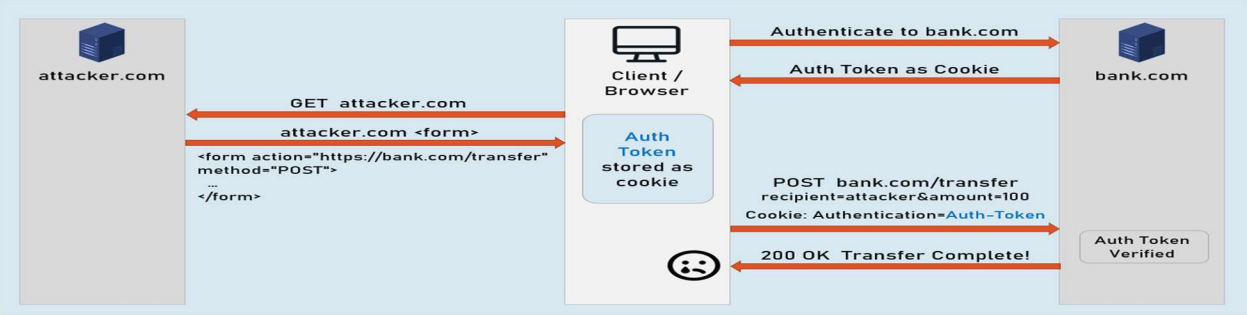


XSRF

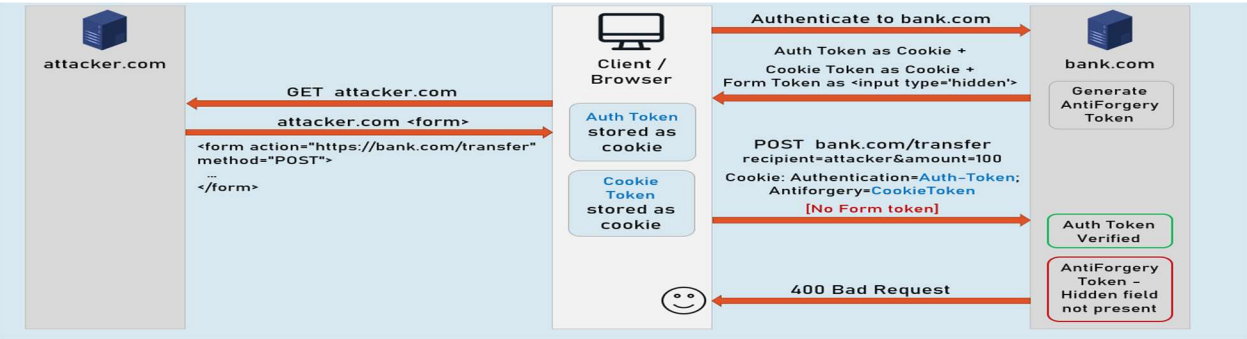
XSRF (Cross Site Request Forgery - CSRF) is a process of making a request to a web server from another domain, using an existing authentication of the same web server.

Eg: attacker.com creates a form that sends malicious request to original.com.

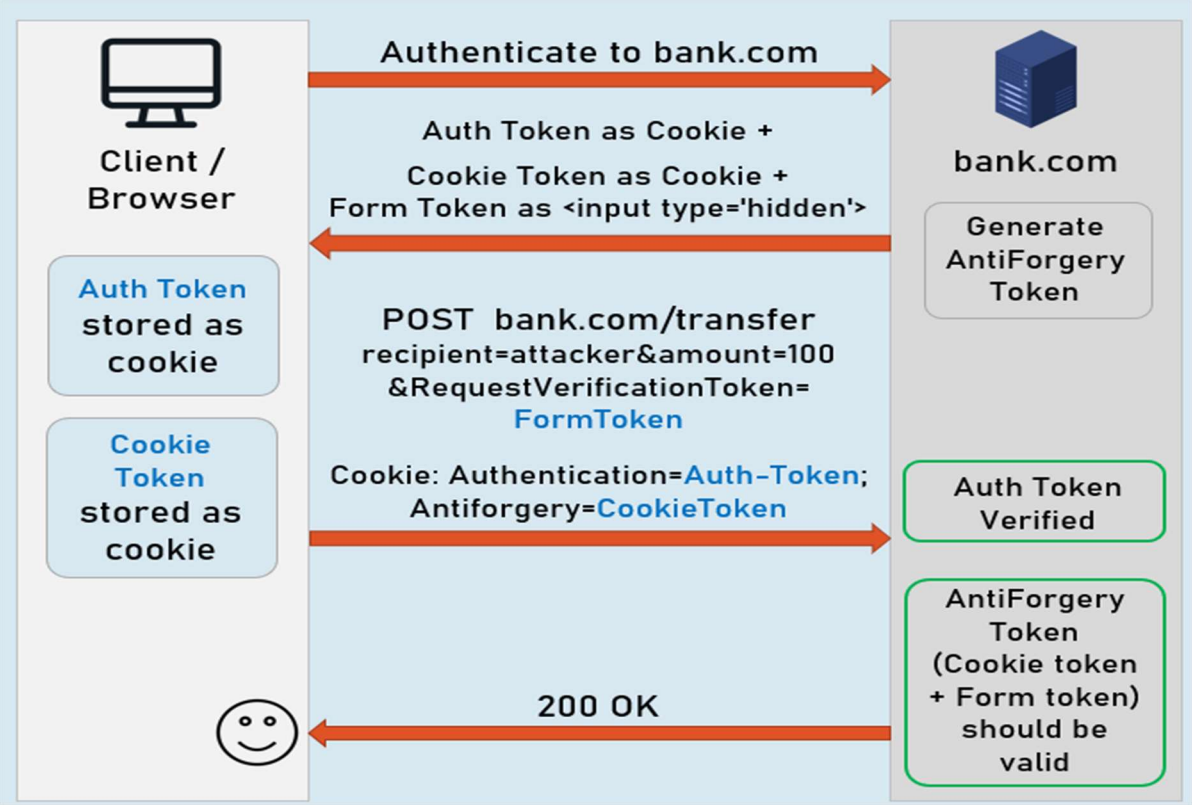
Attacker's request without AntiForgeryToken



Attacker's request



Legit request [No attacker.com]



Introduction to Web API

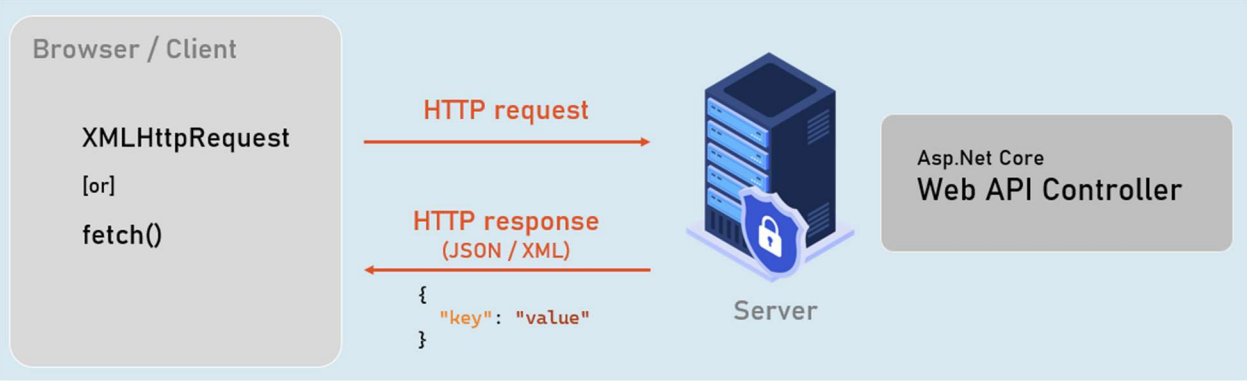
ASP.NET Core Web API is a component (part) of ASP.NET Core, which is used create HTTP-based RESTful services (also known as HTTP services) that can be consumed (invoked) by wide range of client applications such as single-page web applications, mobile applications etc.

Asp.Net Core:

- Asp.Net Core MVC
- Asp.Net Core Web API
- Asp.Net Core Blazor
- Asp.Net Core Razor Pages

RESTful / Web API Services

RESTful services (Representational State Transfer) is an architecture style that defines to create HTTP services that receives HTTP GET, POST, PUT, DELETE requests; perform CRUD operations on the appropriate data source; and returns JSON / XML data as response to the client.



Web API Controllers

Should be either or both:

- The class name should be suffixed with "Controller". Eg: ProductsController
- The [ApiController] attribute is applied to the same class or to its base class.

Controller

```
1. [ApiController]
2. class ClassNameController
3. {
4.     //action methods here
5. }
```

Optional:

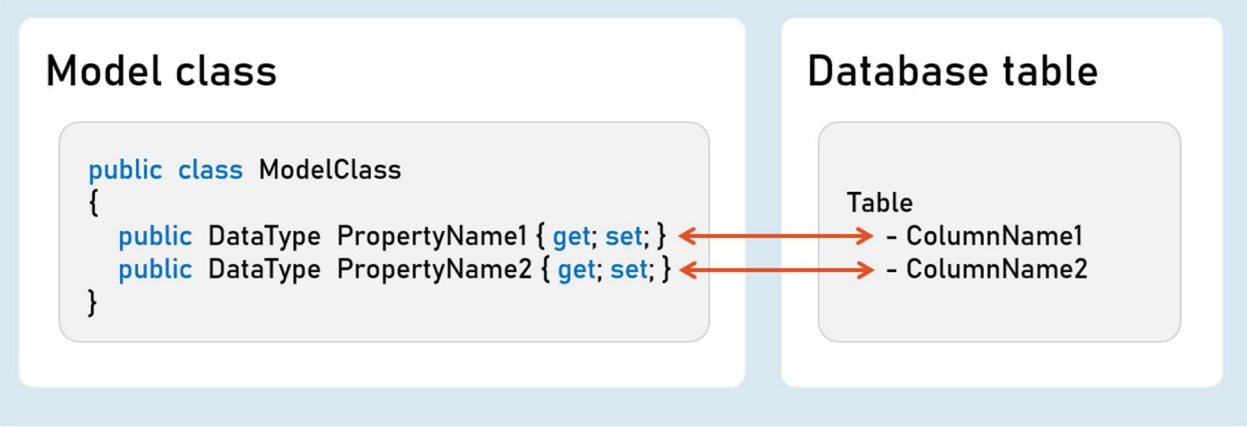
- Is a public class.
- Inherited from Microsoft.AspNetCore.Mvc.ControllerBase.

Introduction to EntityFrameworkCore

EntityFrameworkCore is light-weight, extensible and cross-platform framework for accessing databases in .NET applications.

It is the most-used database framework for Asp.Net Core Apps.

EFCore Models



Pros & Cons of EntityFrameworkCore

1. Shorter Code

The CRUD operations / calling stored procedures are done with shorter amount of code than ADO.NET.

2. Performance

EFCore performs slower than ADO.NET.

So ADO.NET or its alternatives (such as Dapper) are recommended for larger & high-traffic applications.

3. Strongly-Typed

The columns as created as properties in model class.

So the Intellisense offers columns of the table as properties, while writing the code.

Plus, the developer need not convert data types of values; it's automatically done by EFCore itself.

ProblemDetails

ProblemDetails

```
1. public class ProblemDetails
2. {
3.     string? Type { get; set; } //URI references that identifies the problem
   type
4.     string? Title { get; set; } //Summary of the problem type
5.     int? Status { get; set; } //HTTP response status code
6.     string? Detail { get; set; } //Explanation of the problem
7. }
```

ValidationProblemDetails

```
1. public class ValidationProblemDetails : ProblemDetails
2. {
3.     string? Type { get; set; } //URI references that identifies the problem
   type
4.     string? Title { get; set; } //Summary of the problem type
5.     int? Status { get; set; } //HTTP response status code
6.     string? Detail { get; set; } //Explanation of the problem
7.     IDictionary<string, string[]> Errors { get; set; } //List of validation
   errors
8. }
```

IActionResult [vs] ActionResult

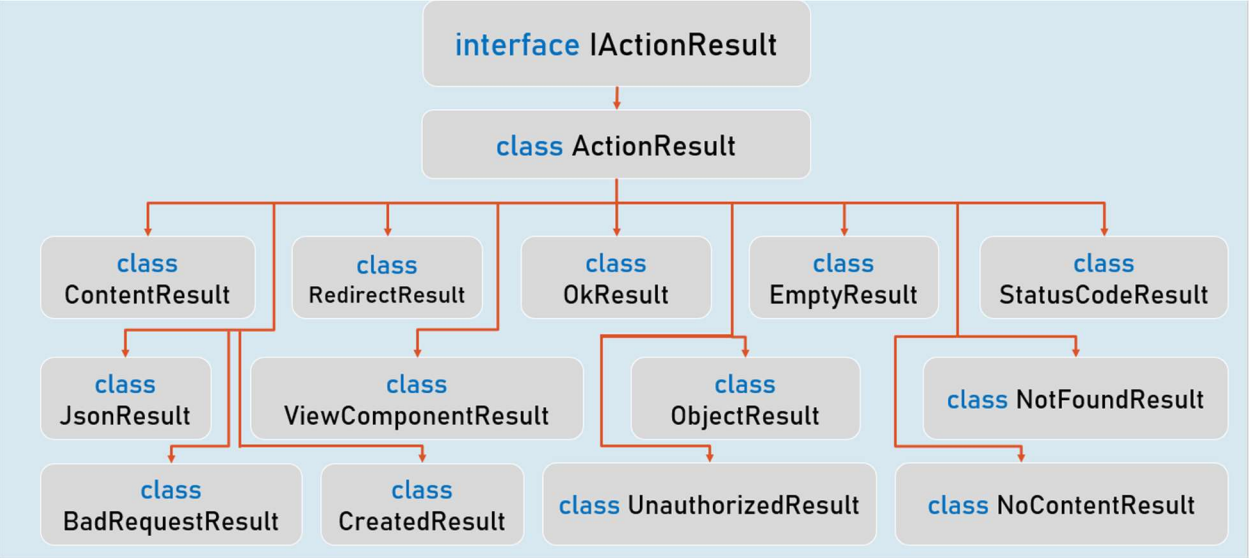
IActionResult

```
1. public interface IActionResult
2. {
3.     Task ExecuteResultAsync(ActionContext context); //converts an object into
   response
4. }
```

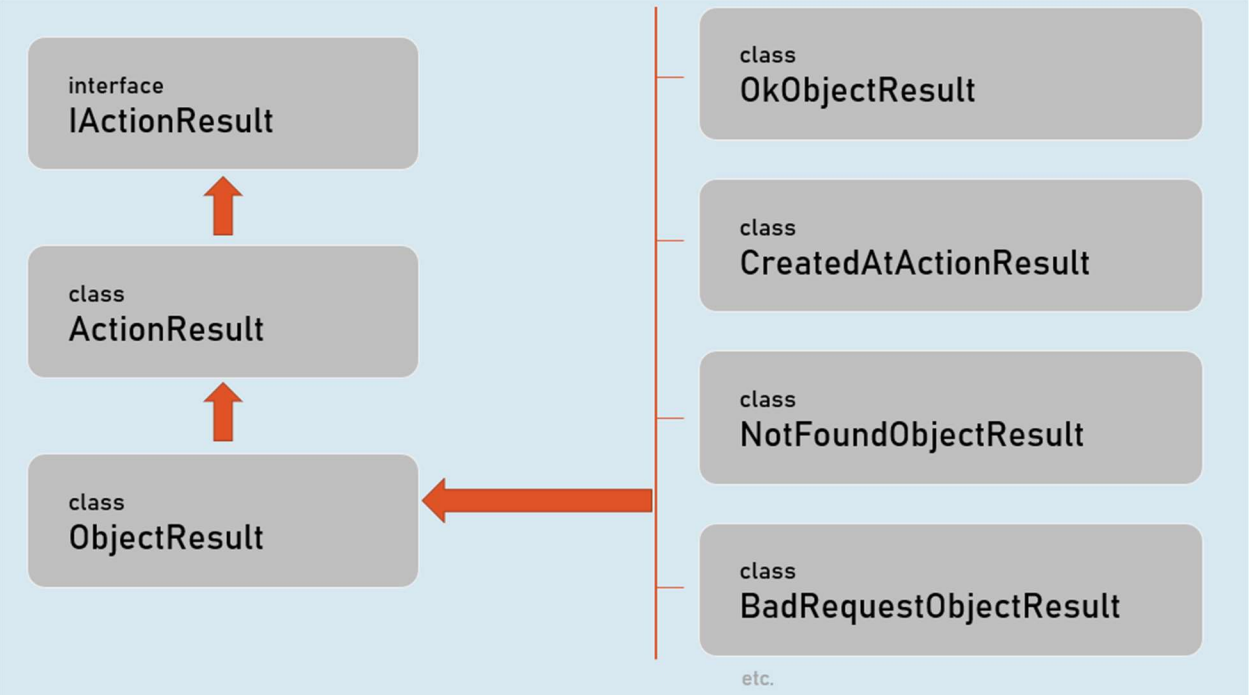
ActionResult<T>

```
1. public sealed class ActionResult<T>
2. {
3.     IActionResult Convert(); //converts the object into ObjectResult
4. }
```

IActionResult



ObjectResult



Introduction to Swagger

Swagger is a set of open-source tools that help developers to generate interactive UI to document, test RESTful services.

Swagger is a set of tools to implement Open API.

1. Swasbuckle.AspNetCore

Framework that makes it easy to use swagger in asp.net core.

2. Swagger

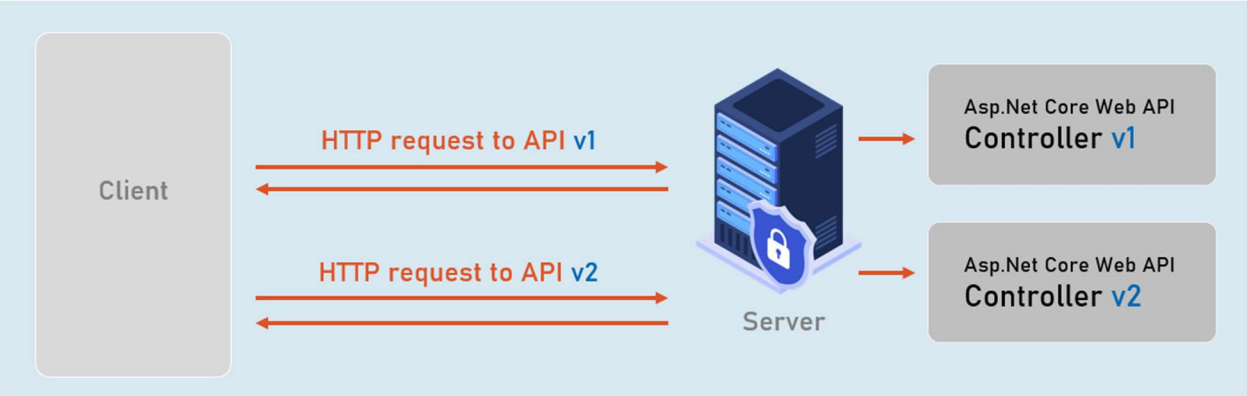
Set of tools to generate UI to document & test RESTful services.

3. Open API

Specification that defines how to write API specifications in JSON).

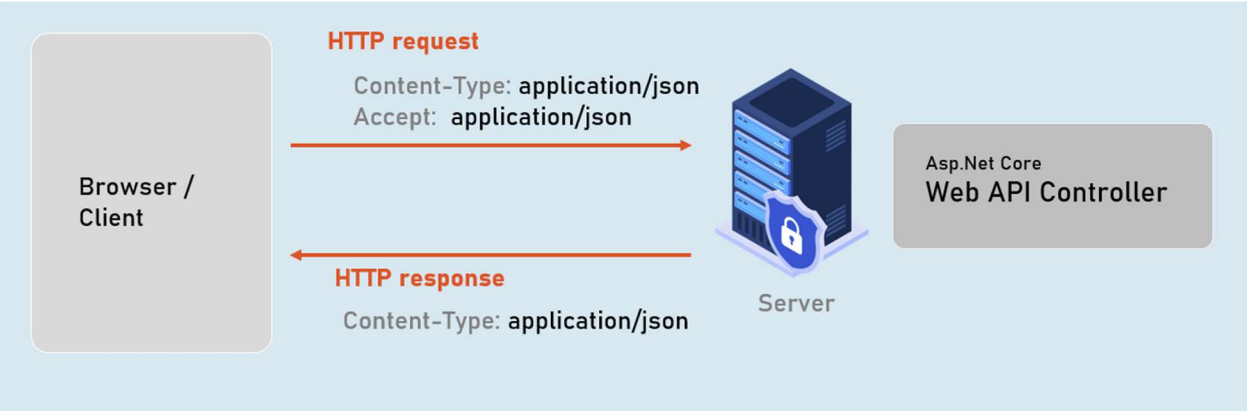
API Versions

API Versioning is the practice of transparently managing changes to your API, where the client requests a specific version of API; and the server executes the same version of the API code.



Content Negotiation

Content negotiation is the process of selecting the appropriate format or language of the content to be exchanged between the client (browser) and Web API.



Enabling Swagger in Core:

To enable Swagger in ASP.NET Core, you need to follow these steps:

1. Install the "Swashbuckle.AspNetCore" NuGet package.

2. In the Startup.cs file, add the following code within the ConfigureServices method:

```
builder.Services.AddSwaggerGen(c =>
{
  c.SwaggerDoc("v1", new OpenApiInfo { Title = "Your API Name",
Version = "v1" });
});
```

3. In the Configure method, add the following code:

```
app.UseSwagger();
app.UseSwaggerUI(c =>
{
  c.SwaggerEndpoint("/swagger/v1/swagger.json", "Your API Name v1");
});
```

4. Run your application and navigate to the Swagger UI at "/swagger" to view and interact with the API documentation.

Versioning API using Swagger in Core:

1. Install the "Microsoft.AspNetCore.Mvc.Versioning" NuGet package.

2. Configure the API versioning in the Startup.cs file within the ConfigureServices method:

```
builder.Services.AddApiVersioning(options =>
{
  options.DefaultApiVersion = new ApiVersion(1, 0);
  options.AssumeDefaultVersionWhenUnspecified = true;
});
```

3. Modify the Swagger configuration in ConfigureServices to generate versioned Swagger documents:

```
builder.Services.AddSwaggerGen(c =>
{
```

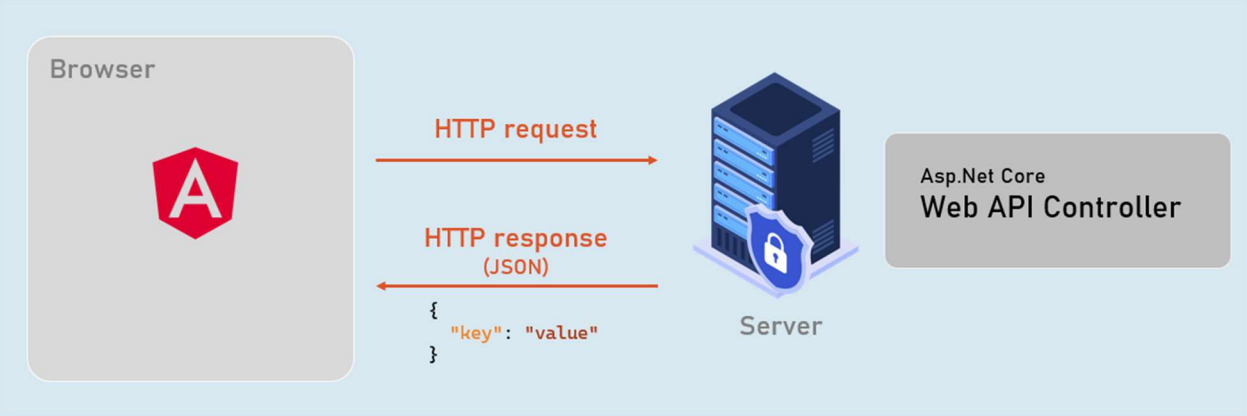
```
  c.SwaggerDoc("v1", new OpenApiInfo { Title = "Your API Name",
Version = "v1" });
  c.SwaggerDoc("v2", new OpenApiInfo { Title = "Your API Name",
Version = "v2" });
});
```

4. Configure Swagger UI in the Configure method:

```
app.UseSwagger();
app.UseSwaggerUI(c =>
{
  c.SwaggerEndpoint("/swagger/v1/swagger.json", "Your API Name v1");
  c.SwaggerEndpoint("/swagger/v2/swagger.json", "Your API Name v2");
});
```

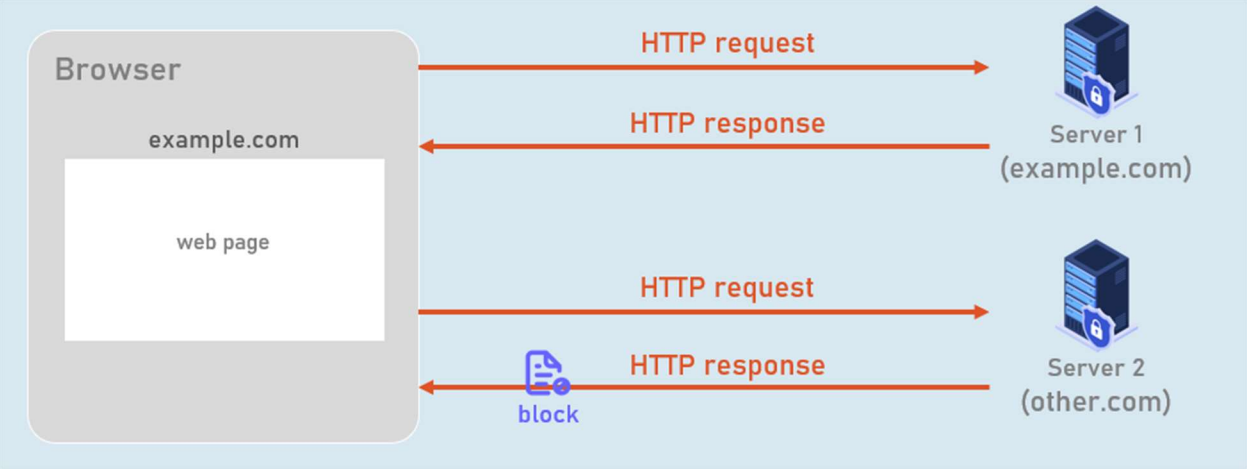
Angular

Angular is a TypeScript-based, free and open-source web application framework that is used to create dynamic, single-page web applications that run on the browser.

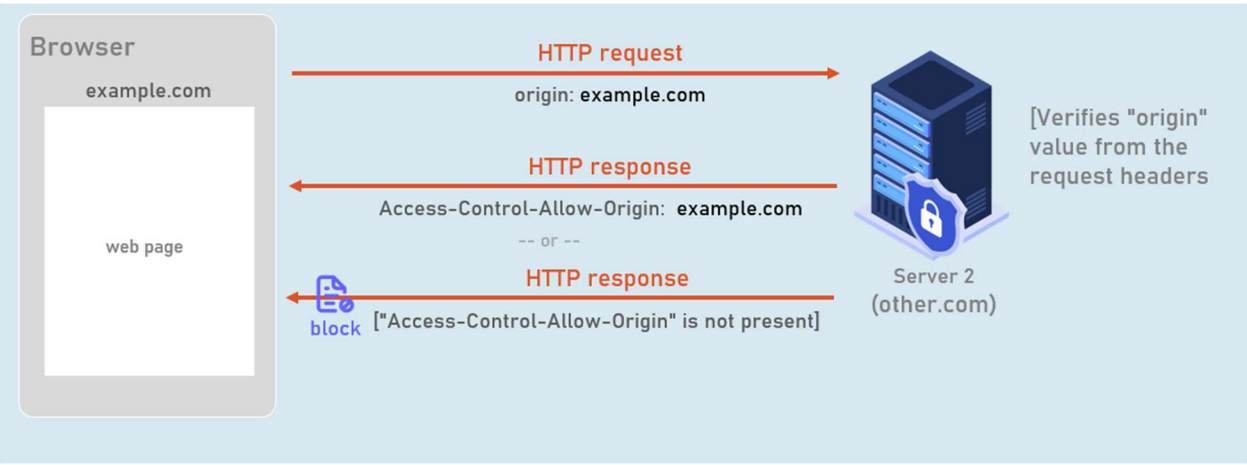


CORS

CORS (Cross-Origin Resource Sharing) is a security feature implemented by web browsers to allow or deny a web page from making requests to a different domain than the one that served the web page.



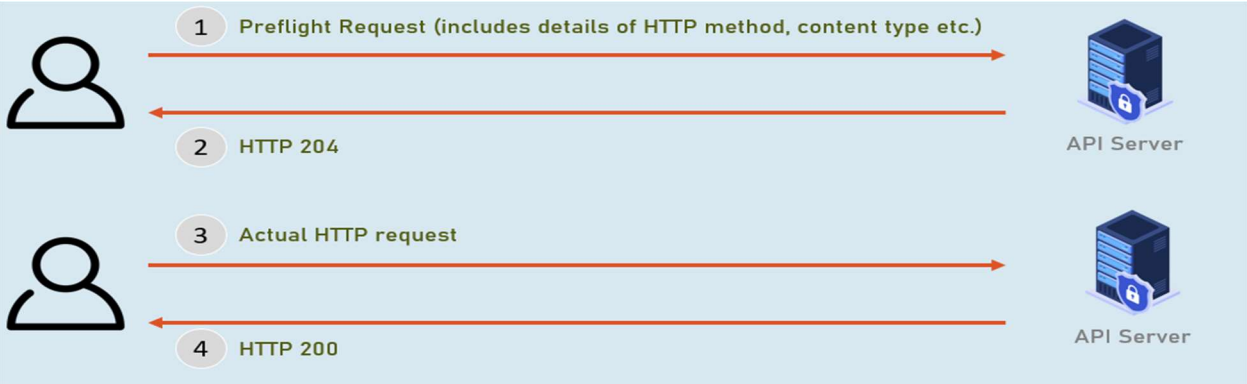
How CORS works?



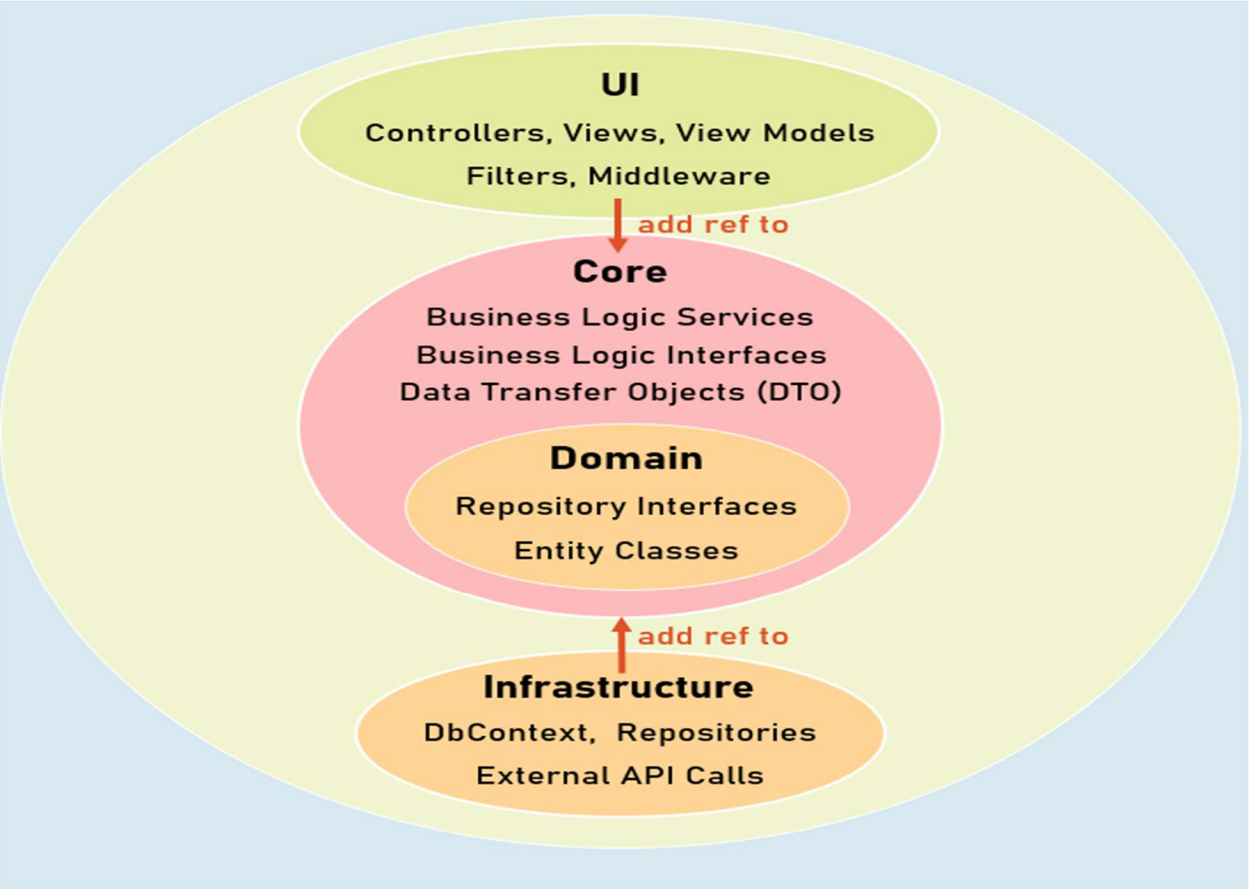
Preflight Request

It is an HTTP OPTIONS request that is sent by the browser to the Web API server before the actual request is made.

The preflight request is used to determine whether the Web API server is willing to accept the actual request.



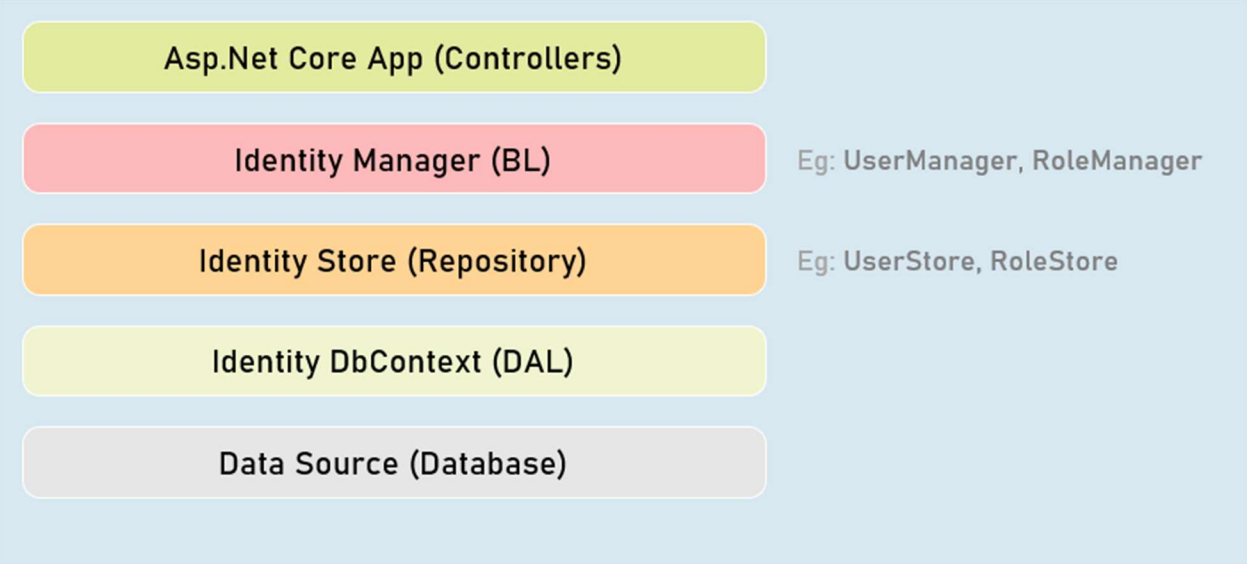
Clean Architecture



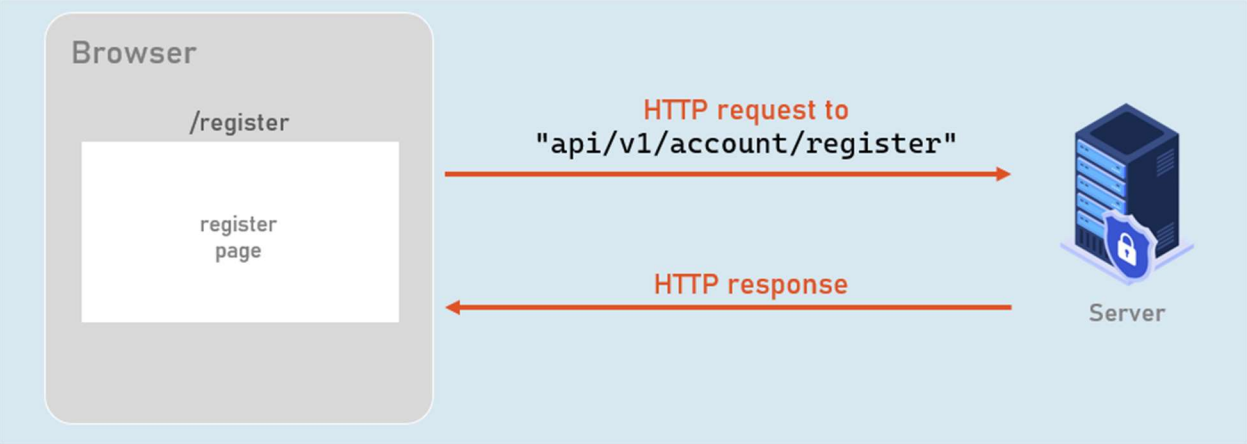
Identity with Web API

It is an API that manages users, passwords, profile data, roles, tokens, email confirmation, external logins etc.

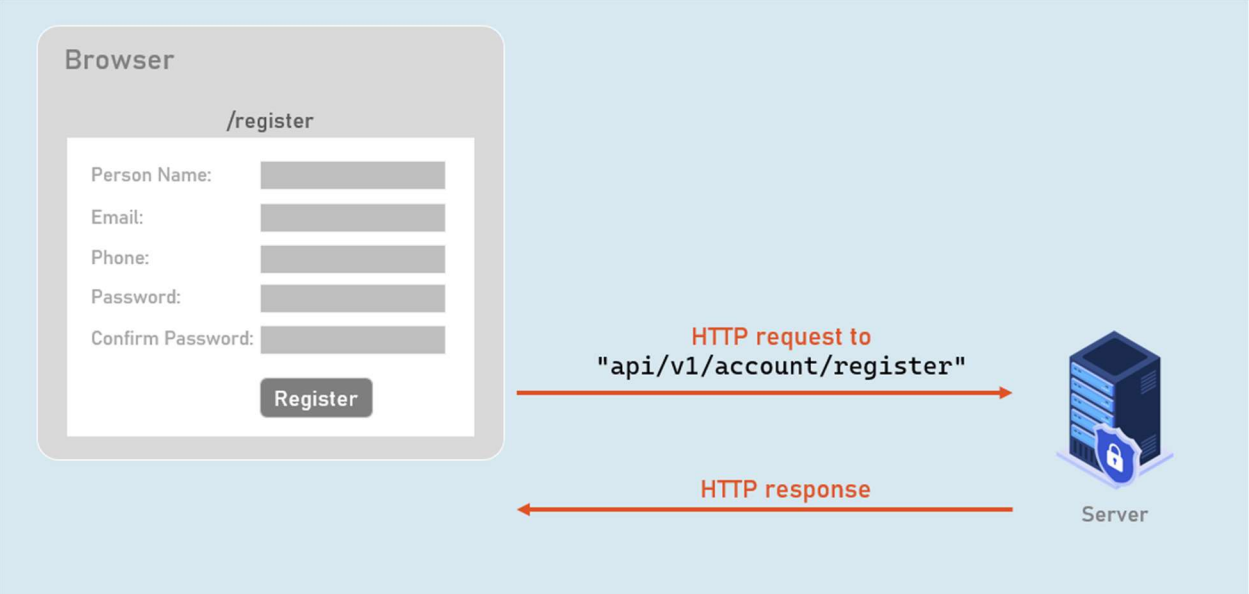
It is by default built on top of EntityFrameworkCore; you can also create custom data stores.



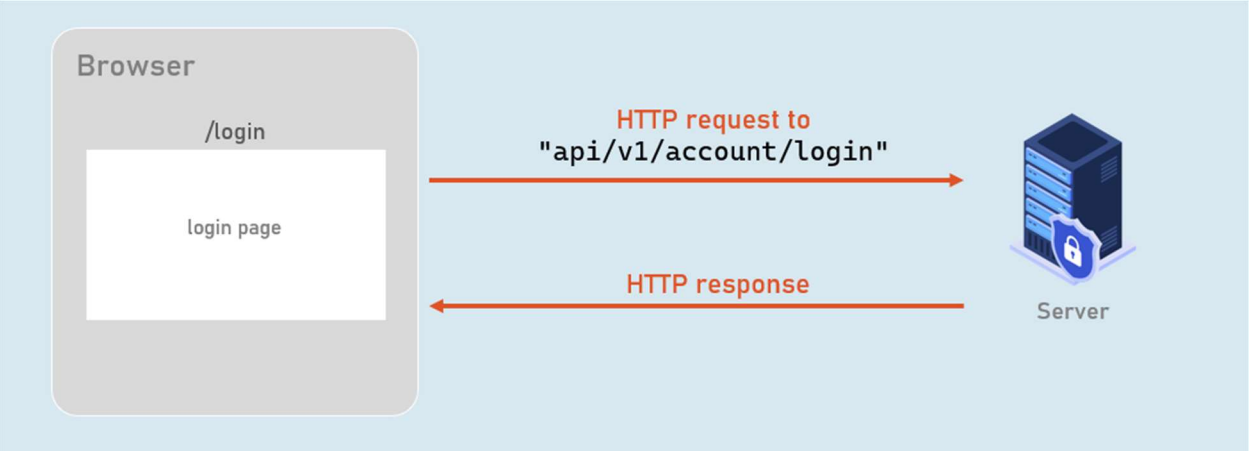
Register Endpoint



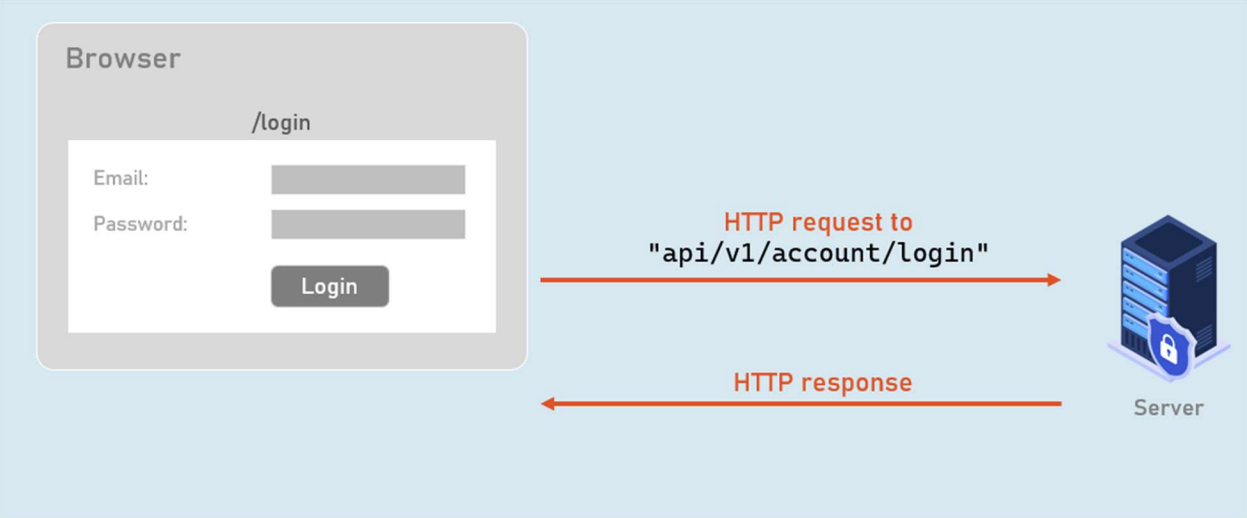
Register UI



Login Endpoint

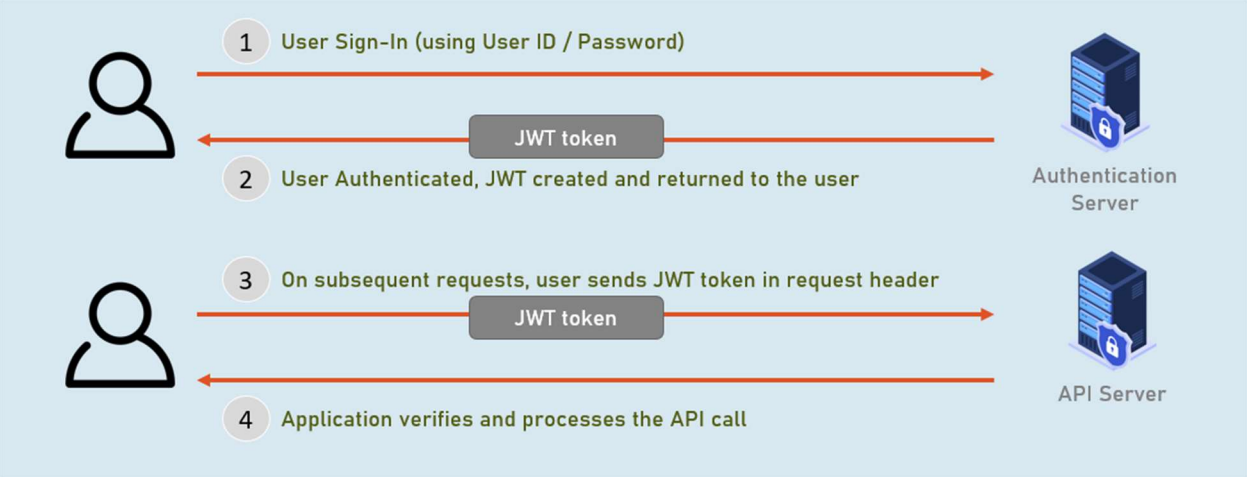


Login UI



Introduction to JWT

A JSON Web Token (JWT) is a compact and self-contained object for securely transmitting information between parties as a JSON object.



Contents of JWT

1. Header (base 64 string)

Defines the type of token and the signing algorithm used.

Eg: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9

2. Payload (base 64 string)

Contains user claims (user details such as name, email or user type).

Eg: { "userId": "b08f86af-35da-48f2-8fab-cef3904660bd"}

Eg: eyJ1c2VySWQiOiJiMDhmOZhZi0zNWRhLTQ4ZjItOOTA0NjYwYmQifQ

3. Signature (base 64 string)

It is used to verify to ensure that the message wasn't changed along the way.

It is usually signed by using a secret key (HMAC algorithm).

-xN_h82PHVTA9vdoHrcZxH-x5

JWT Algorithm

Inputs:

header

```
1. {
2. "typ": "JWT",
3. "alg": "HS256"
4. }
```

payload

```
1. {
2. "userId": "b08f86af-35da-48f2-8fab-cef3904660bd"
3. }
```

secret

MySecret

Algorithm:

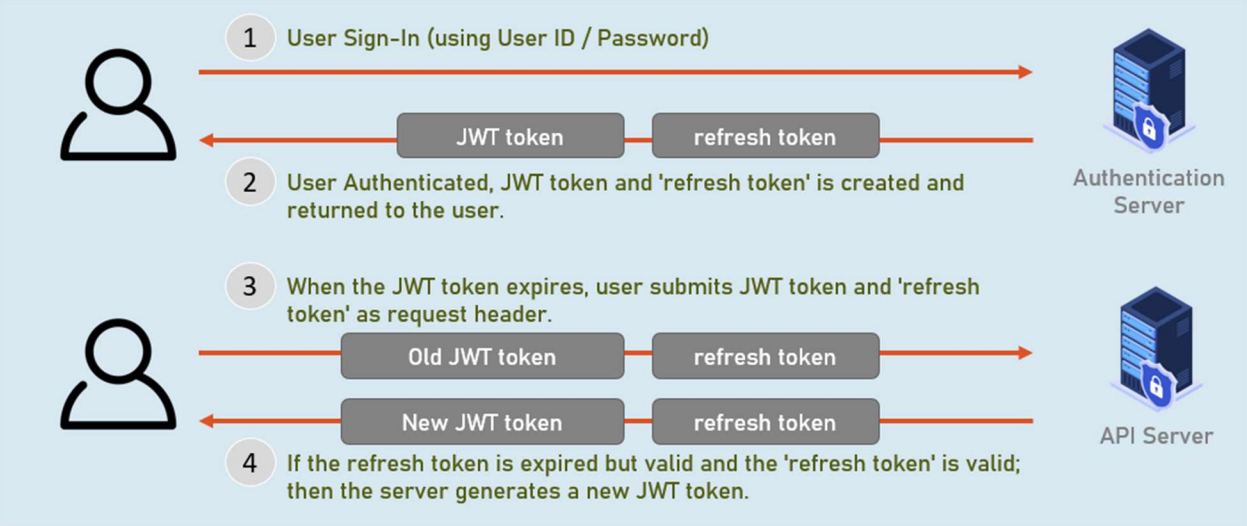
```
1. data = base64Encode( header ) + "." + base64Encode( payload )
2. hashedData = hash( data, secret )
3. signature = base64encode( hashedData )
4. jwtToken = data + "." + signature
```

Example JWT token:

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VySWQiOiJiMDhmODZhZi0zNWRhLTQ4ZjItOOTA0NjYwYmQifQ.-xN_h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzcM

Refresh Tokens - JWT

A refresh token is a token (base-64 string of a random number) that is used to obtain a new JWT token every time, when it is expired.

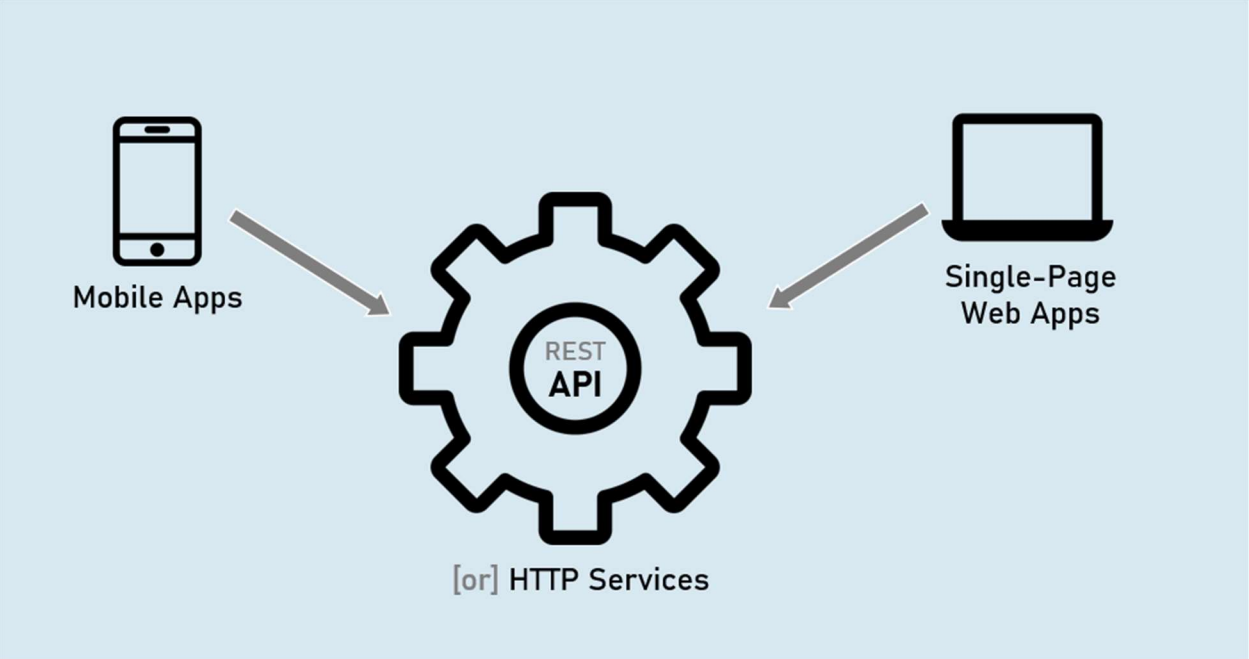


Overview of Minimal API

- It is a Microsoft's API that is used to create HTTP services (or HTTP APIs) with minimal dependencies on packages.

- Alternative to Web API Controllers. Mainly used to create HTTP services or Microservices.

REST API (Representational State Transfer)



MVC Controller (Microsoft.AspNetCore.Mvc.Controller)

- Full support for model binding and model validation.
- Full support for views.
- Full support for filters & filter pipeline.

API Controller (Microsoft.AspNetCore.Mvc.ApiControllerAttribute)

- Full support for model binding and model validation.
- No support for views.
- Full support for filters & filter pipeline.

Minimal API (IApplicationBuilder.Map* Methods)

- Limited support for custom model binding and custom model validation (needs to improve).
- No support for views.
- No support for filters & filter pipeline; but supports "Endpoint Filters" alternatively.

Routing in Minimal API

1. MapGet()

Creates an endpoint that receives HTTP GET request.

```
1. app.MapGet("/route", async (HttpContext context) => {
2.     await context.Response.WriteAsync("your response");
3. });
```

2. MapPost()

Creates an endpoint that receives HTTP DELETE request.

```
1. app.MapDelete("/route", async (HttpContext context) => {
2.     await context.Response.WriteAsync("your response");
3. });
```

3. MapPut()

Creates an endpoint that receives HTTP PUT request.

```
1. app.MapPut("/route", async (HttpContext context) => {
2.     await context.Response.WriteAsync("your response");
3. });
```

4. MapDelete()

Creates an endpoint that receives HTTP DELETE request.

```
1. app.MapDelete("/route", async (HttpContext context) => {
2.     await context.Response.WriteAsync("your response");
3. });
```

Route Parameters in Minimal API

Route parameters can be created as you were creating them in UseEndpoints() or in MVC controllers.

```
1. app.MapGet("/route/{parameter}", async (HttpContext context) => {
2.     await context.Response.WriteAsync("your response");
3. });
```

Route Constraints in Minimal API

Route constraints can be used as you were using them in UseEndpoints() or in MVC controllers.

```
1. app.MapGet("/route/{parameter:constraint}", async (HttpContext context) => {
2.     await context.Response.WriteAsync("your response");
3. });
```

Eg: int, bool, datetime, decimal, double, float, guid, long, Minlength, maxlength, length, min, max, range, alpha, regex, required

Map Groups in Minimal API

A map group (or route group) is a set of endpoints with a common prefix.

A map group is a collection of endpoints created with Map* methods such as MapGet(), MapPost() etc.

MapGet()

Creates an endpoint that receives HTTP GET request.

```
1. var mapGroup = app.MapGroup("/route-prefix");
2.
3. mapGroup.MapGet(…);
4. mapGroup.MapPost(..);
```

IResult

The Microsoft.AspNetCore.Http.IResult is the base interface that is implemented by different result types such as Ok, Json, BadRequest etc., which can be returned by endpoints in minimal API.

1. Results.Ok( )
2. Results.Json( )
3. Results.Text( )
4. Results.File( )
5. Results.BadRequest( )
6. Results.NotFound( )
7. Results.Unauthorized( )
8. Results.ValidationProblem( )

IResult Implementations

1. Results.Ok

Response Content-type: application/json [or] text/plain

Response Status Code: 200

```
return Results.Ok(response_object); //can be a string or model
```

2. Results.Json

Response Content-type: application/json

Response Status Code: 200

```
return Results.Json(response_object); //should be a model
```

3. Results.Text

Response Content-type: text/plain

Response Status Code: 200

```
return Results.Text(response_string); //should be a string
```

4. Results.File

Response Content-type: application/octet-stream

Response Status Code: 200

```
return Results.File(stream_object); //should be
'System.IO.Stream' type
```

5. Results.BadRequest

Response Content-type: N/A

Response Status Code: 400

```
return Results.BadRequest(response_object); //can be a string or
model
```

6. Results.NotFound

Response Content-type: N/A

Response Status Code: 404

```
return Results.NotFound(response_object); //can be a string or
model
```

7. Results.Unauthorized

Response Content-type: N/A

Response Status Code: 401

```
return Results.Unauthorized(response_object); //can be a string
or model
```

8. Results.ValidationProblem

Response Content-type: application/json

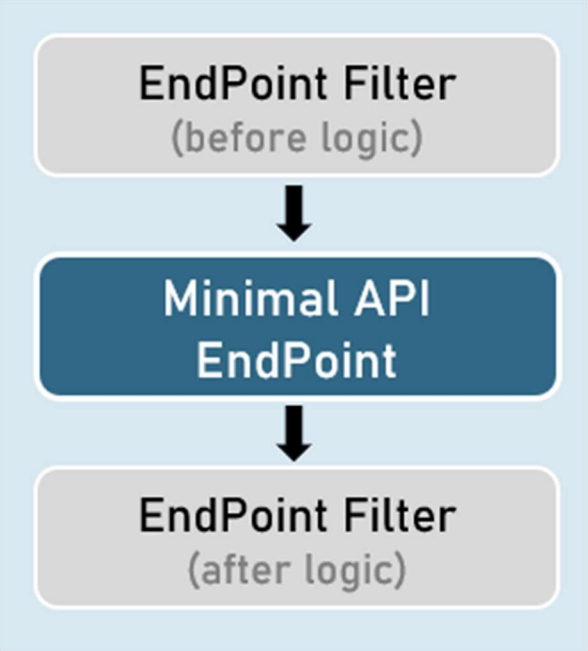
Response Status Code: 400

```
return Results.ValidationProblem(response_object);
//automatically creates JSON with validation errors
```

Endpoint Filter

EndPoint Filters execute much like 'action filters' i.e., 'before' and 'after' the execution of minimal API endpoint.

They are mainly used to validate parameters of the endpoint.



Creating an Endpoint Filter

Endpoint filters can be registered by providing a Delegate that takes a EndpointFilterInvocationContext and returns a EndpointFilterDelegate.

```
1. app.MapGet("/route", () => {
2.     //your endpoint code here
3. })
4. .AddEndpointFilter(async (context, next) => {
5.     //before logic
6.     var result = await next(context); //calls subsequent filter or endpoint
7.
8.     //after logic
9.     return result;
10. });
```

IEndpointFilter

Creating an Endpoint Filter by implementing IEndpointFilter interface

The InvokeAsync() method takes a EndpointFilterInvocationContext and returns a EndpointFilterDelegate.

```
1. class CustomEndpointFilter : IEndpointFilter
2. {
3.    public async ValueTask<object?> InvokeAsync(EndpointFilterInvocationContext
   context, EndpointFilterDelegate next)
4.    {
5.       //before logic
6.       var result = await next(context); //calls subsequent filter or endpoint
7.
8.       //after logic
9.       return result;
10.   }
11. }
```