

<epam>

# Module "Web"

## Submodule "Web API"

Part 2

# AGENDA

---

- 1** Routing in Web API
- 2** Data models. Model binding and validation
- 3** JSON and XML Serialization in Web API
- 4** Exception Handling in Web API. Global Error Handling in Web API

# Routing in Web API

# Routing in Web API

The **Routing** is the Process by which ASP.NET Core Web API inspects the incoming URLs and maps them to Controller Actions (executable endpoints).

**Endpoints** are the app's units of executable request-handling code. Endpoints are defined in the app and configured when the app starts. The endpoint matching process can extract values from the request's URL and provide those values for request processing. Using endpoint information from the app, routing is also able to generate URLs that map to endpoints.

## **Apps can configure routing using:**

- Controllers
- Razor Pages
- SignalR
- gRPC Services
- Endpoint-enabled middleware such as Health Checks.
- Delegates and lambdas registered with routing.

# Routing in Web API: How Routing Works

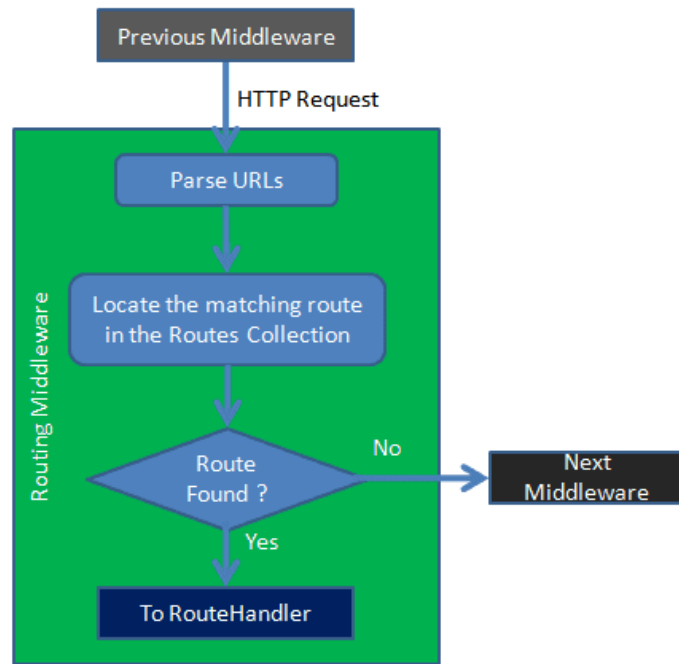
```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
```

```
{  
    if (env.IsDevelopment())  
    {  
        app.UseDeveloperExceptionPage();  
    }  
}
```

```
app.UseRouting();
```

```
app.UseEndpoints(endpoints =>  
{  
    endpoints.MapGet("/", async context =>  
    {  
        await context.Response.WriteAsync("Hello World!");  
    });  
});  
}
```

## How Routing Works in ASP.NET Core



# Routing in Web API: EndpointRoutingMiddleware and EndpointMiddleware

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
```

```
{  
    if (env.IsDevelopment())  
    {  
        app.UseDeveloperExceptionPage();  
    }  
}
```

```
app.UseRouting();  
app.Use(async (context, next) =>
```

```
{  
    // get the end point  
    Endpoint endpoint = context.GetEndpoint();
```

```
    if (endpoint != null)
```

```
    {  
        // get the route pattern that is associated with the endpoint  
        var routePattern = (endpoint as Microsoft.AspNetCore.Routing.RouteEndpoint)?.  
            RoutePattern?.RawText;
```

```
        Debug.WriteLine($"Endpoint Name: {endpoint.DisplayName}");  
        Debug.WriteLine($"Route Pattern: {routePattern}");
```

```
        // if the end point is defined, we pass the processing on  
        await next();  
    }  
}
```

```
else
```

```
{  
    Debug.WriteLine("Endpoint: null");  
    // if no endpoint is defined, end processing  
    await context.Response.WriteAsync("Endpoint is not defined");  
}
```

```
});  
app.UseEndpoints(endpoints =>  
{  
    endpoints.MapGet("/index", async context =>  
    {  
        await context.Response.WriteAsync("Hello Index!");  
    });  
    endpoints.MapGet("/", async context =>  
    {  
        await context.Response.WriteAsync("Hello World!");  
    });  
});  
}
```

# Routing in Web API: Endpoint

---

The **MapGet** method is used to define an endpoint. An endpoint is something that can be:

- Selected, by matching the URL and HTTP method.
- Executed, by running the delegate.

Additional methods can be used to connect ASP.NET Core framework features to the routing system:

- [MapRazorPages](#) for Razor Pages
- [MapControllers](#) for controllers
- [MapHub<THub>](#) for SignalR
- [MapGrpcService<TService>](#) for gRPC

# Routing in Web API: Endpoint

---

**DEMO**



# Routing in Web API: RouterMiddleware

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
```

```
{
```

```
// define a route handler
```

```
var myRouteHandler = new RouteHandler(Handle);
```

```
// create a route using a handler
```

```
var routeBuilder = new RouteBuilder(app, myRouteHandler);
```

```
// definition of the route - it must match the request {controller} / {action}
```

```
routeBuilder.MapRoute("default", "{controller}/{action}");
```

```
// build a route
```

```
app.UseRouter(routeBuilder.Build());
```

```
app.Run(async (context) =>
```

```
{
```

```
    await context.Response.WriteAsync("Hello World!");
```

```
});
```

```
}
```

```
// the route handler
```

```
private async Task Handle(HttpContext context)
```

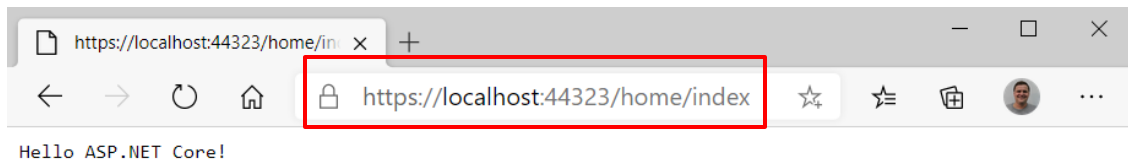
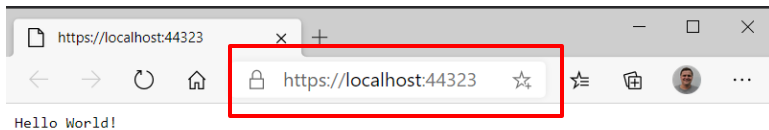
```
{
```

```
    await context.Response.WriteAsync("Hello ASP.NET Core!");
```

```
}
```

# Routing in Web API: URL matching

- Is the process by which routing matches an incoming request to an endpoint.
- Is based on data in the URL path and headers.
- Can be extended to consider any data in the request.



# Routing in Web API: Defining routes

Method	Description
MapRoute(IRouteBuilder, String, String)	Adds a route to the IRouteBuilder with the specified name and template.
MapRoute(IRouteBuilder, String, String, Object)	Adds a route to the IRouteBuilder with the specified name, template, and default values.
<u>MapRoute(IRouteBuilder, String, String, Object, Object)</u>	Adds a route to the IRouteBuilder with the specified name, template, default values, and constraints.
MapRoute(IRouteBuilder, String, String, Object, Object, Object)	Adds a route to the IRouteBuilder with the specified name, template, default values, and data tokens.

# Routing in Web API

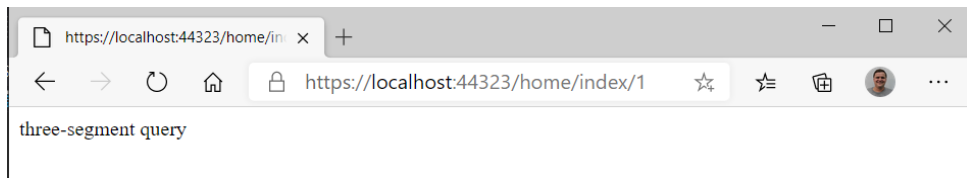
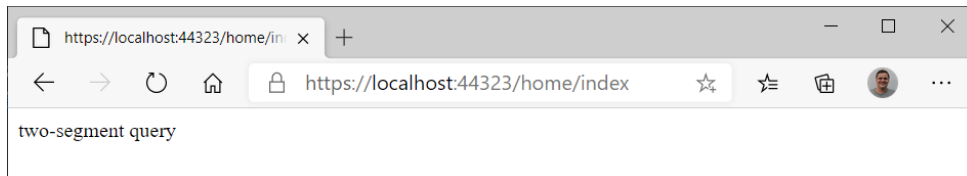
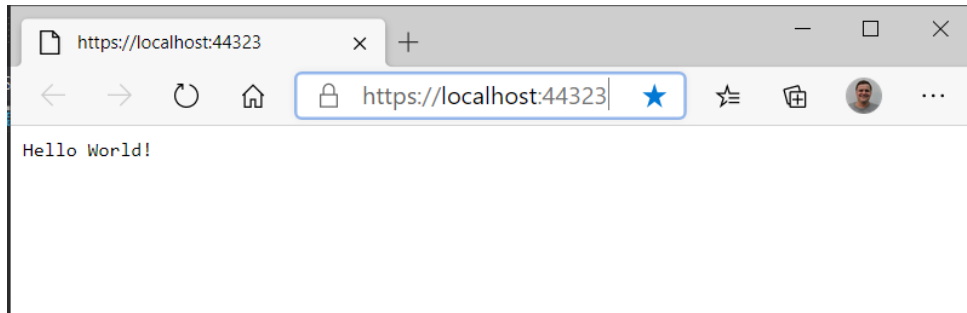
```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    var routeBuilder = new RouteBuilder(app);

    routeBuilder.MapRoute("{controller}/{action}",
        async context =>
        {
            context.Response.ContentType = "text/html; charset=utf-8";
            await context.Response.WriteAsync("two-segment query");
        });

    routeBuilder.MapRoute("{controller}/{action}/{id}",
        async context =>
        {
            context.Response.ContentType = "text/html; charset=utf-8";
            await context.Response.WriteAsync("three-segment query");
        });

    app.UseRouter(routeBuilder.Build());

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello World!");
    });
}
```



# Routing in Web API: URL matching

```
routeBuilder.MapRoute("default", "{controller}/{action}");
```

**http://localhost:xxxx/Home/Index**

controller	action
Home	Index

**http://localhost:xxxx/Book/Order**

controller	action
Book	Order

# Routing in Web API: Static segments

---

```
routeBuilder.MapRoute("default", "store/{action}");
```

=> <http://localhost:xxxx/Store/Order>

```
routeBuilder.MapRoute("default", "api/{controller}/{action}/{id?}");
```

=> <http://localhost:56130/api/Home/Index/1>

# Routing in Web API

---

```
routeBuilder.MapRoute("default", "{controller}/{action?}/{id?}");
```

localhost:56130/Home/Index/

localhost:56130/Home/Index/2

{controller}/{action?}/{id}

{controller}/{action?}/{id?}

# Routing in Web API: Default values

```
routeBuilder.MapRoute("default", "{controller}/{action}/{id?}",  
    new { controller = "home", action = "index" });
```

Query	Parameters
http://localhost:56130/	controller=Home action=Index
http://localhost:56130/Book	controller=Book action=Index
http://localhost:56130/Book/Show	controller=Book action=Show
http://localhost:56130/Book/Show/2	controller=Book action=Show id=2

```
routeBuilder.MapRoute("default", "{controller=Home}/{action=Index}/{id?}");
```



# Routing in Web API

```
routeBuilder.MapRoute("default", "{controller=Home}/{action=Index}/{id?}/{*catchall}");
```

http://localhost:56130/Home/Index/1/name/book/order

controller	Home
action	Index
id	1
catchall	name/book/order

# Routing in Web API: Using prefixes

---

```
routeBuilder.MapRoute("default", "Ru{controller=Home}/{action=Index}-en/{id?}");
```

http://localhost:56130/RuHome/Index-en/1

# Routing in Web API: Multiple parameters in a segment

```
routeBuilder.MapRoute("default", "{controller=Home}/{action=Index}/{name}-{year}");
```

http://localhost:56130/Store/Order/lumia-2015

controller	Store
action	Order
name	lumia
year	2015

# Routing in Web API: Routes for different types of requests

- **MapGet**
- **MapDelete**
- **MapPost**
- **MapPut**
- **MapVerb**

.MapGet(string template, RequestDelegate handler)

.MapGet(string template, Func < HttpRequest, HttpResponse, RouteData, Task > handler)

```
routeBuilder.MapGet("{controller}/{action}", async (context) =>
{
    await context.Response.WriteAsync("Hello From MapGet!");
});
```

MapVerb(string verb, string template, RequestDelegate handler)

MapVerb(string verb, string template, Func < HttpRequest, HttpResponse, RouteData, Task > handler)

# Routing in Web API: Installing middleware for processing requests

- **MapMiddlewareGet**
- **MapMiddlewareDelete**
- **MapMiddlewarePost**
- **MapMiddlewarePut**
- **MapMiddlewareRoute**
- **MapMiddlewareVerb**

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    var routeBuilder = new RouteBuilder(app);

    routeBuilder.MapMiddlewareGet("{controller}/{action}", app =>
    {
        app.Run(async context =>
        {
            await context.Response.WriteAsync("Hello from MapMiddlewareGet");
        });
    });
    app.UseRouter(routeBuilder.Build());
    app.Run(async context =>
    {
        await context.Response.WriteAsync("Hello World!");
    });
}
```

# Routing in Web API

Properties RouteData class:

- Values
- DataTokens
- Routers

```
private async Task Handle(HttpContext context)
{
    await context.Response.WriteAsync("Hello ASP.NET Core!");
}
```

```
RouteData routeData = context.GetRouteData();
```

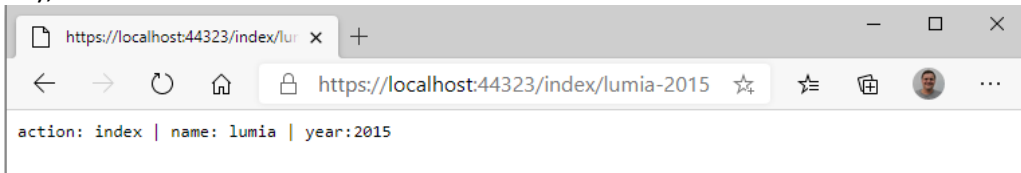
```
string controller = context.GetRouteValue("controller").ToString();
```

# Routing in Web API

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    var myRouteHandler = new RouteHandler(Handle);
    var routeBuilder = new RouteBuilder(app, myRouteHandler);
    routeBuilder.MapRoute("default", "{action=Index}/{name}-{year}");
    app.UseRouter(routeBuilder.Build());

    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello World!");
    });
}

private async Task Handle(HttpContext context)
{
    var routeValues = context.GetRouteData().Values;
    var action = routeValues["action"].ToString();
    var name = routeValues["name"].ToString();
    var year = routeValues["year"].ToString();
    await context.Response.WriteAsync($"action: {action} | name: {name} | year:{year}");
}
```



# Routing in Web API: Adding routes

```
var myRouteHandler = new RouteHandler(Handle);  
var routeBuilder = new RouteBuilder(app, myRouteHandler);  
routeBuilder.MapRoute("default", "{action=Index}/{name}-{year}");  
routeBuilder.MapRoute("default2", "{controller}/{action}/{id?}");  
app.UseRouter(routeBuilder.Build());
```

<http://localhost:5634/index/lumia-2015>

action	index
name	lumia
year	2015

controller	index
action	lumia-2015



# Routing in Web API: Route constraint

```
routeBuilder.MapRoute("default",  
    "{controller}/{action}/{id?}",  
    new { action = "Index" }, // default parameters  
    new { controller = "^H.*" } // Constraint  
);
```

```
routeBuilder.MapRoute("default",  
    "{controller}/{action}/{id?}",  
    new { action = "Index" },  
    new { controller = "^H.*", id = @"\d{2}" }  
);
```

## Name Space: Microsoft.AspNetCore.Routing.Constraints

```
routeBuilder.MapRoute("default",  
    "{controller}/{action}/{id?}",  
    null,  
    new { controller = new RegexRouteConstraint("^H.*"), id = new RegexRouteConstraint(@"\d{2}") }  
);
```

# Routing in Web API: Microsoft.AspNetCore.Routing.Constraints

## CLASSES

<a href="#">AlphaRouteConstraint</a>	Constrains a route parameter to contain only lowercase or uppercase letters A through Z in the English alphabet.
<a href="#">BoolRouteConstraint</a>	Constrains a route parameter to represent only Boolean values.
<a href="#">CompositeRouteConstraint</a>	Constrains a route by several child constraints.
<a href="#">DateTimeRouteConstraint</a>	Constrains a route parameter to represent only <a href="#">DateTime</a> values.
<a href="#">DecimalRouteConstraint</a>	Constrains a route parameter to represent only decimal values.
<a href="#">DoubleRouteConstraint</a>	Constrains a route parameter to represent only 64-bit floating-point values.
<a href="#">FileNameRouteConstraint</a>	Constrains a route parameter to represent only file name values. Does not validate that the route value contains valid file system characters, or that the value represents an actual file on disk.
<a href="#">FloatRouteConstraint</a>	Constrains a route parameter to represent only 32-bit floating-point values.
<a href="#">GuidRouteConstraint</a>	Constrains a route parameter to represent only <a href="#">Guid</a> values. Matches values specified in any of the five formats "N", "D", "B", "P", or "X", supported by <a href="#">Guid.ToString(string)</a> and <a href="#">Guid.ToString(String, IFormatProvider)</a> methods.
<a href="#">HttpMethodRouteConstraint</a>	Constrains the HTTP method of request or a route.
<a href="#">IntRouteConstraint</a>	Constrains a route parameter to represent only 32-bit integer values.
<a href="#">LengthRouteConstraint</a>	Constrains a route parameter to be a string of a given length or within a given range of lengths.
<a href="#">LongRouteConstraint</a>	Constrains a route parameter to represent only 64-bit integer values.
<a href="#">MaxLengthRouteConstraint</a>	Constrains a route parameter to be a string with a maximum length.
<a href="#">MaxRouteConstraint</a>	Constrains a route parameter to be an integer with a maximum value.
<a href="#">MinLengthRouteConstraint</a>	Constrains a route parameter to be a string with a minimum length.
<a href="#">MinRouteConstraint</a>	Constrains a route parameter to be a long with a minimum value.
<a href="#">NonFileNameRouteConstraint</a>	Constrains a route parameter to represent only non-file-name values. Does not validate that the route value contains valid file system characters, or that the value represents an actual file on disk.
<a href="#">OptionalRouteConstraint</a>	Defines a constraint on an optional parameter. If the parameter is present, then it is constrained by <a href="#">InnerConstraint</a> .
<a href="#">RangeRouteConstraint</a>	Constrains a route parameter to be an integer within a given range of values.
<a href="#">RegexInlineRouteConstraint</a>	Represents a regex constraint which can be used as an <a href="#">InlineConstraint</a> .
<a href="#">RegexRouteConstraint</a>	
<a href="#">RequiredRouteConstraint</a>	Constrains a route parameter that must have a value.
<a href="#">StringRouteConstraint</a>	Constrains a route parameter to contain only a specified string.

<https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.routing.constraints?view=aspnetcore-3.1>

# Routing in Web API: CompositeRouteConstraint

```
routeBuilder.MapRoute(  
    name: "default",  
    template: "{controller}/{action}/{id?}",  
    defaults: new { controller = "Home", action = "Index" },  
    constraints: new  
    {  
        action = new CompositeRouteConstraint(new IRouteConstraint[]  
        {  
            new AlphaRouteConstraint(),  
            new MinLengthRouteConstraint(5)  
        })  
    });
```

# Routing in Web API: inline constraints

```
routeBuilder.MapRoute("default", "{controller:regex(^H.*)}/{action}/{id?}");
```

int	{id:int}
bool	{active:bool}
datetime	{date:datetime}
decimal	{price:decimal}
double	{weight:double}
float	{height:float}
guid	{id:guid}
long	{id:long}
minlength(value)	{name:minlength(3)}
maxlength(value)	{name:maxlength(20)}

length(value)	{name:length(10)}
length(min, max)	{name:length(3, 20)}
min(value)	{age:min(3)}
max(value)	{age:max(20)}
range(min, max)	{age:range(18, 99)}
alpha	{name:alpha}
regex(expression)	{phone:regex(^\\d{{3}}-\\d{{3}}-\\d{{4}}\$)}
required	{name:required}

# Routing in Web API: inline constraints

---

```
routeBuilder.MapRoute("default", "{controller:length(4)}/{action:alpha}/{id:range(4,100)}");
```

```
routeBuilder.MapRoute(  
    name: "default",  
    template: "{controller}/{action:alpha:minlength(5)}/{id?}",  
    defaults: new { controller = "Home", action = "Index" });
```

# Routing in Web API: IRouteConstraint

---

```
public interface IRouteConstraint
{
    bool Match(HttpContext httpContext,
        IRouter route,
        string routeKey,
        RouteValueDictionary values,
        RouteDirection routeDirection);
}
```

# Routing in Web API

```
public class CustomConstraint : IRouteConstraint
{
    private string uri;
    public CustomConstraint(string uri)
    {
        this.uri = uri;
    }
}
```

/Home/Index/12

```
public bool Match(HttpContext httpContext, IRouter route, string routeKey,
    RouteValueDictionary values, RouteDirection routeDirection)
{
    return !(uri == httpContext.Request.Path);
}
```

```
routeBuilder.MapRoute("default",
    "{controller}/{action}/{id?}",
    null,
    new { myConstraint = new CustomConstraint("/Home/Index/12") }
);
```

# Routing in Web API

```
public class PositionConstraint : IRouteConstraint
{
    string[] positions = new[] { "admin", "director", "accountant" };
    public bool Match(HttpContext httpContext, IRouter route, string routeKey,
        RouteValueDictionary values, RouteDirection routeDirection)
    {
        return positions.Contains(values[routeKey]?.ToString().ToLowerInvariant());
    }
}

routeBuilder.MapRoute(
    "default",
    "{controller}/{action}/{id}",
    new { controller = "Home", action = "Index" },
    new { id = new PositionConstraint() });
```



# Data models. Model binding and validation

# Data models

---

A **model** is a set of classes that represent the data that the app manages.

A **Plain Old CLR Objects (POCO)** is a class, which doesn't depend on any framework-specific base class. It is like any other normal .NET class. Due to this, they are called Plain Old CLR Objects. These POCO entities (also known as persistence-ignorant objects) support most of the same LINQ queries as Entity Object derived entities. These classes (POCO classes) implements only the domain business logic of the Application.

Some developers use Data Transfer Objects (DTOs) with the classes to pass the data between the layers because POCOs are also used to pass the data between the layers, but they become heavy. Hence they use DTOs, which are also the classes.

The main difference between DTO and POCO is that DTOs do not contain any methods. They only contain public members. Thus, sending the data, using a DTO is easy because they are lightweight objects.

# Data models

## Anemic Domain Model

```
public class Phone
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Company { get; set; }
    public int Price { get; set; }
}
```

## Rich Domain Model / Fat Model / Thick Model

```
public class Phone
{
    private decimal _discount = 0;
    public Phone(decimal discount)
    {
        this._discount = discount;
    }
    public int Id { get; set; }
    public string Name { get; set; }
    public string Company { get; set; }
    public decimal Price { get; set; }

    public decimal GetPriceWithDiscount()
    {
        return this.Price - (this.Price * this._discount);
    }
}
```

# Parameter Binding

Action methods in Web API controller can have one or more parameters of different types. It can be either **primitive** type or **complex** type. Web API binds action method parameters either with URL's query string or with request body depending on the parameter type. By default, if parameter type is of .NET **primitive** type such as int, bool, double, string, GUID, DateTime, decimal or any other type that can be converted from string type then it sets the value of a parameter from the query string. And if the parameter type is **complex** type then Web API tries to get the value from request body by default.

HTTP Method	Query String	Request Body
GET	Primitive Type, Complex Type	NA
POST	Primitive Type	Complex Type
PUT	Primitive Type	Complex Type
PATCH	Primitive Type	Complex Type
DELETE	Primitive Type, Complex Type	NA

# Parameter Binding

POST http://localhost:49705/api/student/12

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary

Key	Value	Description
name	Vlad	
age	27	

```
0 references
public class StudentController : ApiController
{
    0 references
    public void Post(int id, Student student)
    {
        ≤ 50,864ms elapsed
    }
}
```

student {WebApiProject.Models.Student}

- Age "27"
- Name "Vlad"

```
1 reference
public class Student
{
    0 references
    public string Name { get; set; }

    0 references
    public string Age { get; set; }
}
```

# [FromUri] and [FromBody]

Use **[FromUri]** attribute to force Web API to get the value of complex type from the query string and **[FromBody]** attribute to get the value of primitive type from the request body, opposite to the default rules.

```
0 references
public class StudentController : ApiController
{
    0 references
    public Student Get([FromUri]Student stud)
    {
        return stud;
    }

    0 references
    public void Post([FromBody]int id, [FromUri]Student student)
    {
    }
}
```

GET http://localhost:49705/api/student?name=Vlad&age=27

Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Code

Type No Auth

POST http://localhost:49705/api/student?name=Vlad&age=27

Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Code

☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> id	12			

# Model validation

---

Code	Status
200	Ok
201	Created
204	NoContent
400	BadRequest
401	Unathorized
403	Forbidden
404	NotFound

# Data models. Model binding and validation

```
public class User
{
    public int Id { get; set; }
    [Required(ErrorMessage = "Enter your username")]
    public string Name { get; set; }
    [Range(1, 100, ErrorMessage = "Age must be between 1 and 100")]
    [Required(ErrorMessage = "Specify the age of the user")]
    public int Age { get; set; }
}
```



# Model validation

```
// POST api/users
[HttpPost]
public async Task<ActionResult<User>> Post(User user)
{
    // handling special cases of validation
    if (user.Age == 99)
        ModelState.AddModelError("Age", "Age must not be 99");

    if (user.Name == "admin")
    {
        ModelState.AddModelError("Name", "Invalid username - admin");
    }

    // if there are errors, return 400 error
    if (!ModelState.IsValid)
        return BadRequest(ModelState);

    // if there are no errors, save to the database
    db.Users.Add(user);
    await db.SaveChangesAsync();
    return Ok(user);
}
```

# Model validation

// Create User

```
async function CreateUser(userName, userAge) {
```

```
    const response = await fetch("api/users", {
        method: "POST",
        headers: { "Accept": "application/json", "Content-Type": "application/json" },
        body: JSON.stringify({
            name: userName,
            age: parseInt(userAge, 10)
        })
    });
```

```
    if (response.ok === true) {
        const user = await response.json();
        reset();
        document.querySelector("tbody").append(row(user));
    }
```

```
    else {
        const errorData = await response.json();
        console.log("errors", errorData);
        if (errorData) {
```

```
            // errors due to attribute validation
            if (errorData.errors) {
                if (errorData.errors["Name"]) {
                    addError(errorData.errors["Name"]);
                }
            }
        }
```

```
        if (errorData.errors["Age"]) {
            addError(errorData.errors["Age"]);
        }
    }
```

```
    // custom errors defined in the controller
    // add errors to the Name property
    if (errorData["Name"]) {
        addError(errorData["Name"]);
    }
```

```
    // add errors of the Age property
    if (errorData["Age"]) {
        addError(errorData["Age"]);
    }
```

```
    document.getElementById("errors").style.display = "block";
```

```
    }
}
```

# Data models. Model binding and validation

The screenshot shows a web browser window with the title 'User list' and the URL 'https://localhost:44395'. The page displays a form for adding or editing a user. The form has two input fields: 'Name' with the value 'admin' and 'Age' with the value '99'. Below the inputs are 'Save' and 'Reset' buttons. A red error message box is visible at the top of the form area, containing the text 'Invalid username - admin' and 'Age must not be 99'. Below the form is a table with the following data:

Id	Name	age		
1	Tom	26	Change	Delete
2	Alice	31	Change	Delete
4	Oleksii2	42	Change	Delete
6	ivan	22	Change	Delete

# JSON and XML Serialization in Web API

# Content negotiation

```
[HttpGet]
public IEnumerable<User> Get()
{
    return db.Users.ToList();
}
```

```
[HttpGet("{id}")]
public IActionResult Get(int id)
{
    User user = db.Users.FirstOrDefault(x => x.Id == id);
    if (user == null)
        return NotFound();
    return new ObjectResult(user);
}
```

```
[HttpGet]
public IActionResult Get()
{
    return new ObjectResult(db.Users.ToList());
}
```

```
[HttpDelete("{id}")]
public IActionResult Delete(int id)
{
    User user = db.Users.FirstOrDefault(x => x.Id == id);
    if (user == null)
    {
        return NotFound();
    }
    db.Users.Remove(user);
    db.SaveChanges();
    return Ok(user);
}
```

# Content negotiation

---

Accept: text/html,application/xhtml+xml,application/xml;q = 0.9,image/webp,\*/\*;q = 0.8

# Content negotiation

## System.Runtime.Serialization.DataContractSerializer

**NuGet:** Microsoft.AspNetCore.Mvc.Formatters.Xml

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc().AddXmlDataContractSerializerFormatters();

    // methods....
}
or
services.AddMvc(options =>
{
    options.OutputFormatters.Add(new XmlDataContractSerializerOutputFormatter());
});
```

# Content negotiation

## System.Xml.Serialization.XmlSerializer

```
public void ConfigureServices(IServiceCollection services)
```

```
{
```

```
    services.AddMvc().AddXmlSerializerFormatters();
```

```
    // alternative way
```

```
    // services.AddMvc(options =>
```

```
    // {
```

```
    //     options.OutputFormatters.Add(new XmlSerializerOutputFormatter());
```

```
    // });
```

```
    // methods...
```

```
}
```

```
[HttpGet]
```

```
public IActionResult Get()
```

```
{
```

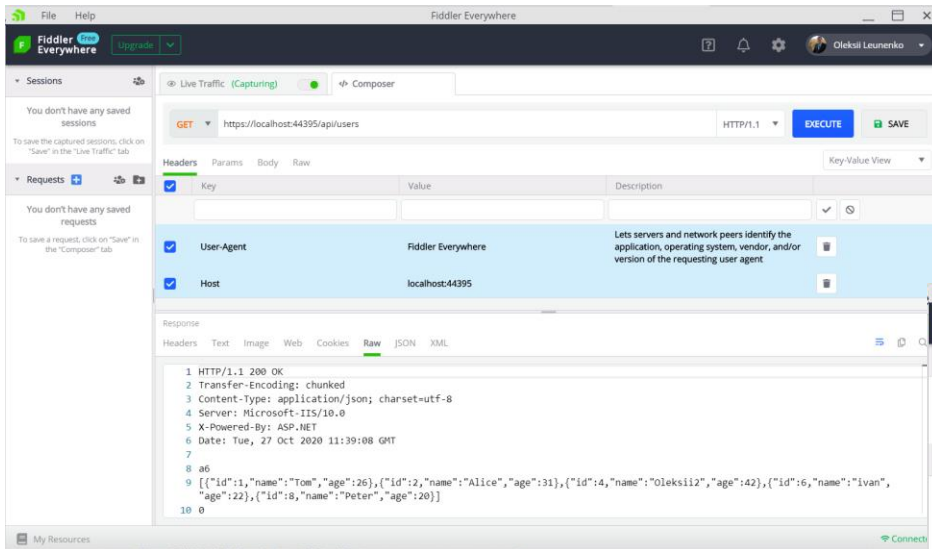
```
    return new ObjectResult(db.Users.ToList());
```

```
}
```

Accept: application/xml

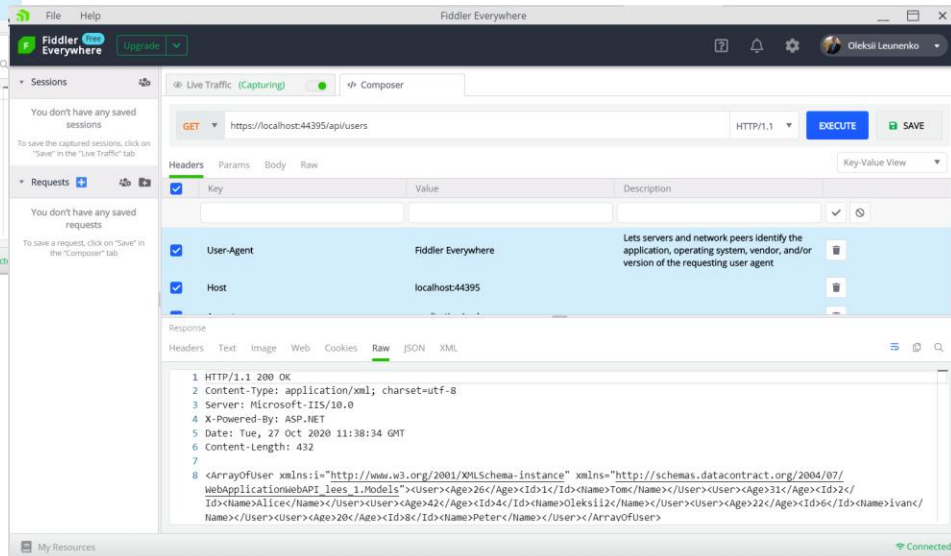


# Content negotiation



Fiddler Everywhere interface showing a captured HTTP request to `https://localhost:44395/api/users`. The response is in raw format, showing a 200 OK status and a JSON body with an array of user objects.

```
1 HTTP/1.1 200 OK
2 Transfer-Encoding: chunked
3 Content-Type: application/json; charset=utf-8
4 Server: Microsoft-IIS/10.0
5 X-Powered-By: ASP.NET
6 Date: Tue, 27 Oct 2020 11:39:08 GMT
7
8 a0
9 [{"id":1,"name":"Tom","age":26},{"id":2,"name":"Alice","age":31},{"id":4,"name":"Oleksii2","age":42},{"id":6,"name":"Ivan",
  "age":22},{"id":8,"name":"Peter","age":20}]
10 0
```



Fiddler Everywhere interface showing a captured HTTP request to `https://localhost:44395/api/users`. The response is in raw format, showing a 200 OK status and an XML body with an array of user objects.

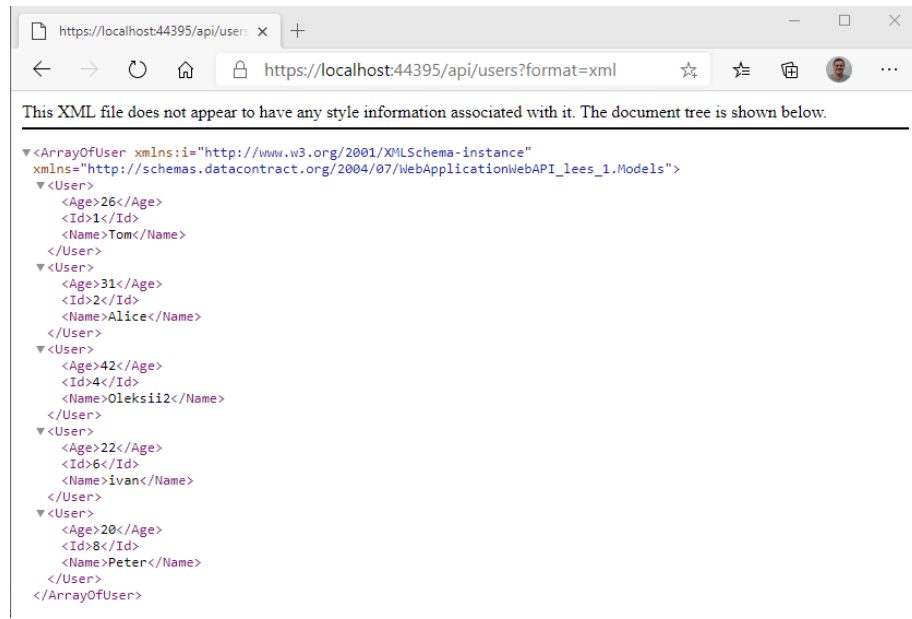
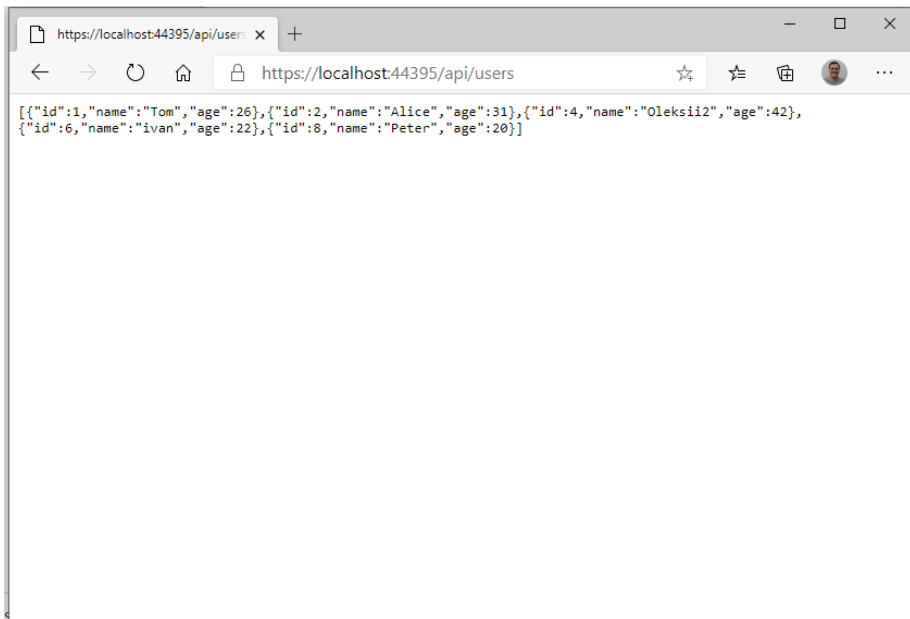
```
1 HTTP/1.1 200 OK
2 Content-Type: application/xml; charset=utf-8
3 Server: Microsoft-IIS/10.0
4 X-Powered-By: ASP.NET
5 Date: Tue, 27 Oct 2020 11:38:34 GMT
6 Content-Length: 432
7
8 <ArrayOfUser xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/
  WebApplicationWebAPI_1.Models"><user><age>26</age><id>1</id><name>Tom</name></user><user><age>31</age><id>2</
  id><name>Alice</name></user><user><age>42</age><id>4</id><name>Oleksii2</name></user><user><age>22</age><id>6</id><name>Ivan</
  Name></user></ArrayOfUser>
```

# Content negotiation

```
services.AddMvc()  
    .AddXmlDataContractSerializerFormatters()  
    .AddMvcOptions(opts => {  
        opts.FormatterMappings.SetMediaTypeMappingForFormat("xml", new MediaTypeHeaderValue("application/xml"));  
    });
```

```
[HttpGet]  
[FormatFilter]  
public IActionResult Get()  
{  
    return new ObjectResult(db.Users.ToList());  
}
```

# Content negotiation



# Web API Request/Response Data Formats

Web API converts request data into CLR object and also serializes CLR object into response data based on **Accept** and **Content-Type** headers. Web API includes built-in support for **JSON, XML, BSON, and form-urlencoded data**. It means it automatically converts request/response data into these formats OOB out-of-the box.

GET  Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Code

Key	Value	Description	Bulk Edit	Presets
<input checked="" type="checkbox"/> Accept	application/json			
New key	Value	Description		

Body Cookies Headers (10) Test Results Status: 200 OK Time: 24 ms

Pretty Raw Preview JSON

```
1 {
2   "Name": "Vlad",
3   "Age": "27"
4 }
```

GET  Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Code

Key	Value	Description	Bulk Edit	Presets
<input checked="" type="checkbox"/> Accept	application/xml			
New key	Value	Description		

Body Cookies Headers (10) Test Results Status: 200 OK Time: 20 ms

Pretty Raw Preview XML

```
1 <Student xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org
2   /2004/07/WebApiProject.Models">
3   <Age>27</Age>
4   <Name>Vlad</Name>
5 </Student>
```

# Configure JSON Serialization

JSON formatter can be configured in WebApiConfig class. The JsonMediaTypeFormatter class includes various properties and methods using which you can customize JSON serialization. For example, Web API writes JSON property names with **PascalCase** by default. To write JSON property names with **camelCase**, set the CamelCasePropertyNamesContractResolver on the serializer settings as shown below.

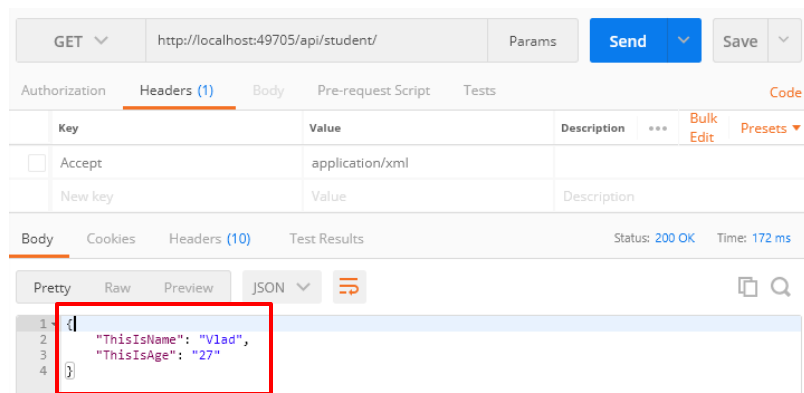
## With default JSON Configuration

```
2 references
public class Student
{
    0 references
    public string ThisIsName { get; set; }

    0 references
    public string ThisIsAge { get; set; }
}
```

```
0 references
public Student Get()
{
    var stud = new Student
    {
        ThisIsName = "Vlad",
        ThisIsAge = "27"
    };

    return stud;
}
```



# Configure JSON Serialization

## With camelCase JSON Configuration

```
1 reference
public static class WebApiConfig
{
    1 reference
    public static void Register(HttpConfiguration config)
    {
        // Web API configuration and services

        // Configure json formatter
        var jsonFormatter = config.Formatters.JsonFormatter;
        jsonFormatter.SerializerSettings.ContractResolver =
            new CamelCasePropertyNamesContractResolver();

        // Web API routes
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```



GET http://localhost:49705/api/student/ Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Code

Key	Value	Description	...	Bulk Edit	Presets
<input type="checkbox"/> Accept	application/xml				
New key	Value	Description			

Body Cookies Headers (10) Test Results Status: 200 OK Time: 172 ms

Pretty Raw Preview JSON

```
{
  "thisIsName": "Vlad",
  "thisIsAge": "27"
}
```

# **Exception Handling in Web API.**

## **Global Error Handling in Web API**

# Exception Handling in Web API

---

Exceptions are the errors that happen at runtime. Exception handling is the technique to handle this runtime error in our application code. If any error is thrown in web API that is caught, it is translated into an HTTP response with status code 500- "Internal Server Error".

API calls are most often called by back-end code or javascript code and in both cases, you never want to simply display the response from the API. Instead we check the status code and parse the response to determine if our action was successful, displaying data to the user as necessary. An error page is not helpful in these situations. It bloats the response with HTML and makes client code difficult because JSON (or XML) is expected, not HTML.



# Exception Handling in Web API

---

The exception handling features help us deal with the unforeseen errors which could appear in our code.

To handle exceptions we can use:

- Error Handling with Try-Catch Block
- Handling Errors Globally with the Built-In Middleware
- Handling Errors Globally with the Custom Middleware

# Exception Handling in Web API

## Error Handling with Try-Catch Block

```
[HttpGet("{city}")]
public WeatherForecast Get(string city)
{
    if (!string.Equals(city?.TrimEnd(), "Redmond", StringComparison.OrdinalIgnoreCase))
    {
        throw new ArgumentException(
            $"We don't offer a weather forecast for {city}.", nameof(city));
    }

    return Get().First();
}
```

Enable the Developer Exception Page only when the app is running in the Development environment. You don't want to share detailed exception information publicly when the app runs in production.

```
[HttpGet("{city}")]
public IActionResult Get(string city)
{
    try
    {
        if (!string.Equals(city?.TrimEnd(), "Redmond", StringComparison.OrdinalIgnoreCase))
        {
            throw new ArgumentException(
                $"We don't offer a weather forecast for {city}.", nameof(city));
        }
        return Ok(Get().First());
    }
    catch (Exception ex)
    {
        return StatusCode(500, ex.Message );
    }
}
```

# Global Error Handling in Web API

## Handling Errors Globally with the Built-In Middleware

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
```

```
{
```

```
    if (env.IsDevelopment())
    {
        // app.UseDeveloperExceptionPage(); // RFC 7807
        app.UseExceptionHandler("/error-local-development");
    }
    else
    {
        app.UseExceptionHandler("/error");
    }
}
```

```
app.UseHttpsRedirection();
```

```
app.UseRouting();
```

```
app.UseAuthorization();
```

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

```
}
```

```
[ApiController]
```

```
public class ErrorController : ControllerBase
```

```
{
```

```
    [Route("/error")]
```

```
    public IActionResult Error() => Problem();
```

```
}
```

# Global Error Handling in Web API

```
[ApiController]
public class ErrorController : ControllerBase
{
    [Route("/error-local-development")]
    public IActionResult ErrorLocalDevelopment(
        [FromServices] IWebHostEnvironment webHostEnvironment)
    {
        if (webHostEnvironment.EnvironmentName != "Development")
        {
            throw new InvalidOperationException(
                "This shouldn't be invoked in non-development environments.");
        }

        var context = HttpContext.Features.Get<ExceptionHandlerFeature>();

        return Problem(
            detail: context.Error.StackTrace,
            title: context.Error.Message);
    }

    [Route("/error")]
    public IActionResult Error() => Problem();
}
```

# Global Error Handling in Web API

## Use exceptions to modify the response

1. Create a well-known exception type named `HttpResponseException`:

```
public class HttpResponseException : Exception
{
    public int Status { get; set; } = 500;
    public object Value { get; set; }
}
```

```
public class HttpResponseExceptionFilter : IActionFilter, IOrderedFilter
{
    public int Order { get; } = int.MaxValue - 10;
    public void OnActionExecuting(ActionExecutingContext context) {}
    public void OnActionExecuted(ActionExecutedContext context)
    {
        if (context.Exception is HttpResponseException exception)
        {
            context.Result = new ObjectResult(exception.Value)
            {
                StatusCode = exception.Status,
            };
            context.ExceptionHandled = true;
        }
    }
}
```

2. Create an action filter named `HttpResponseExceptionFilter`:

3. In `Startup.ConfigureServices`, add the action filter to the filters collection:

```
services.AddControllers(options =>
options.Filters.Add(new HttpResponseExceptionFilter()));
```

# Global Error Handling in Web API

## Validation failure error response

```
services.AddControllers()
    .ConfigureApiBehaviorOptions(options =>
    {
        options.InvalidModelStateResponseFactory = context =>
        {
            var result = new BadRequestObjectResult(context.ModelState);

            // TODO: add `using System.Net.Mime;` to resolve MediaTypeNames
            result.ContentTypes.Add(MediaTypeNames.Application.Json);
            result.ContentTypes.Add(MediaTypeNames.Application.Xml);

            return result;
        }
    });
```

## Client error response

An error result is defined as a result with an HTTP status code of 400 or higher. For web API controllers, MVC transforms an error result to a result with `ProblemDetails`.

The error response can be configured in one of the following ways:

1. Implement `ProblemDetailsFactory`
2. Use `ApiBehaviorOptions.ClientErrorMapping`

# Global Error Handling in Web API

---

## Implement ProblemDetailsFactory

```
public void ConfigureServices(IServiceCollection serviceCollection)
{
    services.AddControllers();
    services.AddTransient<ProblemDetailsFactory, CustomProblemDetailsFactory>();
}
```

# Global Error Handling in Web API

## Use ApiBehaviorOptions.ClientErrorMapping

```
services.AddControllers()  
    .ConfigureApiBehaviorOptions(options =>  
    {  
        options.SuppressConsumesConstraintForFormFileParameters = true;  
        options.SuppressInferBindingSourcesForParameters = true;  
        options.SuppressModelStateInvalidFilter = true;  
        options.SuppressMapClientErrors = true;  
        options.ClientErrorMapping[StatusCodes.Status404NotFound].Link =  
            "https://httpstatuses.com/404";  
    });
```



# .NET Online UA Training Course Feedback

---

I hope that you will find this material useful.

If you find errors or inaccuracies in this material or know how to improve it, please report on to the electronic address:

Oleksii\_Leunenko@epam.com

With the note [.NET Online UA Training Course Feedback]

Thank you.

# Q&A



DRIVEN



CANDID



CREATIVE



ORIGINAL



INTELLIGENT



EXPERT

UA .NET Online LAB