

# ASP.NET CORE



Philip Japikse (@skimedic)  
skimedic@outlook.com  
CTO, Author, Teacher

Microsoft MVP, ASPInsider, PST, PSM II, PSD

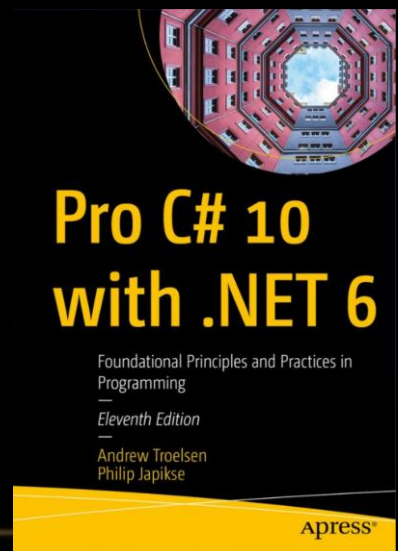


<https://github.com/skimedic/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimedic.com>

Phil>About()

- CTO/Chief Architect, Pintas & Mullins
- Author: Apress.com (<http://bit.ly/apressbooks>)
- Professional Scrum Trainer (PSF, PSD)
- Speaker: <http://www.skimedica.com/blog/page/Abstracts.aspx>
- Microsoft MVP, ASPInsider, PST, PSM II, PSD
- Founder, Agile Conferences, Inc.
  - <http://www.cincydeliver.org>
- President, Cincinnati .NET User's Group



<https://github.com/skimedica/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimedica.com>

# PLEASE MUTE YOUR CELL PHONES

```
attendees.Select(x=>x.Phone.IsOn)
```

```
{  
    x.Phone.EnableSilentMode();  
}
```

```
attendees.where(x => x.Phone.Rings())
```

```
{  
    x.Dispose();  
}
```

# SUPPORT LIFECYCLES

<https://github.com/skimedict/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimedict.com>

# .NET CORE SUPPORT LIFECYCLES

## ➤ Long Term Support (LTS)

- Upgraded with patches – typically security
- Supported for three years after GA release

## ➤ In Support:

- 8.0 LTS (support until 11/10/26)
- 9.0 STS (support until 05/12/26)
- 10.0 will be LTS

## ➤ ~~Current~~ Standard Term Support (STS)

- Minor releases
- Upgraded more rapidly
- Supported for 6 months after the next release (LTS or STS)
  - Releases happen every 12 months

## ➤ Out of Support:

- 5.0 support ended 05/10/22
- 6.0 support ended 11/12/24
- 7.0 support ended 05/14/24

<https://www.microsoft.com/net/core/support>

<https://github.com/skimedec/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimedec.com>

# MIGRATING FROM FRAMEWORK TO ASP.NET CORE

<https://github.com/skimedict/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimedict.com>

## ASP.NET (MVC/WEBAPI) -> ASP.NET CORE

- I Recommend:
  - File -> New project
  - Clipboard inheritance for controllers, views, and services
- Can try: <https://dotnet.microsoft.com/platform/upgrade-assistant>
  - Last I tried it, it didn't handle EF Core upgrade very well

## WCF -> ASP.NET CORE

- WCF is now supported through CoreWCF
  - Open-source port of WCF (.NET F/W) to .NET

# ASP.NET CORE FUNDAMENTALS

<https://github.com/skimedict/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimedict.com>

## ASP.NET CORE

- ASP.NET Core rebuilt on top of .NET Core
- Single framework for web, services, and microservices
  - WebApi + MVC + Razor Pages + Blazor = ASP.NET Core
- Cross-platform
  - Not tied to IIS or Windows
- Takes advantage of .NET Core performance
  - Includes a high performance web server (Kestrel) built on LibUV

# ASP.NET CORE FEATURES

- Pluggable Middleware
  - Routing, authentication, static files, etc.
- Full Dependency Injection integration
- Simplified and Improved Configuration System
- Tag Helpers
- View Components
- Plus more!

## NOTABLE UPDATES IN ASP.NET CORE 9

- Developer exception page improvements
  - Routing information, better formatting, improved readability
- Fixing 503's during app recycle
- Data protection key deletion
- Middleware support of Keyed DI
- Trust HTTPS dev cert on Linux

## NOTABLE UPDATES IN ASP.NET CORE 9 FOR SERVICES

- Minimal APIs - ProblemDetails on route groups
- OpenAPI package – Microsoft.AspNetCore.OpenApi
  - Nice progress, not quite there (no XML Comments support, no UI, versioning is still rough)
- Support for Pushed Authorization Requires (PAR)
- OIDC and Oauth parameter customization

# CREATING THE WEB HOST

<https://github.com/skimeddic/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimeddic.com>



# DEPENDENCY INJECTION

<https://github.com/skimeddic/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimeddic.com>

## ADDING CUSTOM DEPENDENCIES TO DI CONTAINER

- Configured in the Program.cs file
  - After creating the app builder and before building the app (ConfigureServices)
  - In .NET 8+, can be keyed
- Used to configure any services needed by the application
  - Transient – created each time they are requested
  - Scoped – created once per request
  - Singleton – created once
- Can plug in different IOC framework

# INJECTING SERVICES INTO CONTROLLERS (SERVICES)

- Dependencies are added in Program.cs (ConfigureServices)

```
builder.Services.AddScoped<ICarRepo, CarRepo>();
```

- Class constructors by the DI container

```
public RazorSyntaxController(ICarRepo carRepo)
```

- ActionMethods need [FromServices] (optional for API 7+)

```
public IActionResult RazorSyntax([FromServices] ICarRepo carRepo)
```

# INJECTING KEYED SERVICES INTO CONTROLLERS (8+)

## ➤ Dependencies are added in Program.cs (ConfigureServices)

```
builder.Services.AddKeyedScoped<IService, SimpleService>("key");
```

## ➤ Class constructors

```
public class SimpleService : IService  
{  
    public SimpleService(IServiceProvider serviceProvider)  
    {  
        // ...  
    }  
}
```

## ➤ ActionMethods\PageHandlers

```
public IActionResult Index([FromServices] IService provider)  
{  
    // ...  
}
```

# VALIDATE SERVICES ON RUN

➤ Add this to Program.cs (ConfigureServices)

➤ API

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

```
builder.Host.UseDefaultServiceProvider(o =>
{
    o.ValidateOnBuild = true;
    o.ValidateScopes = true;
});
```

# ENVIRONMENTAL AWARENESS APPLICATION CONFIGURATION

<https://github.com/skimeddic/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimeddic.com>

# ENVIRONMENTAL AWARENESS

- ASP.NET Core uses ASPNETCORE\_ENVIRONMENT variable
  - Development, Staging, and Production are built-in environments
- Environment is used throughout ASP.NET Core
  - Configuration
  - C# Execution paths
  - HTML/CSS/JS
- Simplifies deployment and testing

# APPLICATION CONFIGURATION

- Applications are configured using:
  - Default = Simple JSON
    - Other file types also supported
  - Command line arguments, Environment variables, In memory .NET objects, Encrypted user store, Custom providers
  
- Values are set in the order received - last in wins
  - appsettings.json
  - Environment determines which additional file to load
    - appsettings.<environment>.json

# THE OPTIONS PATTERN

<https://github.com/skimedec/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimedec.com>

## THE OPTIONS PATTERN

- Add custom classes to the services collection
- Classes are reconstituted from the application settings
- Injected using one of the `IOptions<T>` interfaces
  - Typically using `IOptionsSnapshot` or `IOptionsMonitor`

# USING THE OPTIONS PATTERN

- Add the option patter to the DI container

```
services.Configure<CustomSettings>(Configuration.GetSection("CustomSettings"));
```

- Inject the class into a controller constructor/action method:

```
public HomeController(IOptionsMonitor<CustomSettings> settings)  
    => _settings = settings.CurrentValue;  
public IActionResult Index([FromServices] IOptionsSnapshot<DealerInfo> dealerOptionsSnapshot)  
    => View(dealerOptionsSnapshot.Value);
```

# JSON OPTIONS

<https://github.com/skimeddic/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimeddic.com>

# CONTROLLING JSON INPUT/OUTPUT

- Default casing changed
  - Classic ASP.NET Pascal cases JSON
  - ASP.NET Core camel cases JSON
  - Revert to Pascal Casing
    - `JsonSerializerOptions.PropertyNamingPolicy = null;`
  - Accept all casing
    - `JsonSerializerOptions.PropertyNameCaseInsensitive = true;`
  - `JsonSerializerOptions.WriteIndented = true;`
  - `JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles;`

## JSONNAMINGPOLICY OUTPUT CASING OPTIONS (8+)

- Pascal Case (PascalCase)
  - Set the naming policy to null)
- CamelCase (camelCase)
- KebabCaseLower (kebab-case-lower)
- KebabCaseUpper (Kebab-Case-Upper)
- SnakeCaseLower (snake\_case\_lower)
- SnakeCaseUpper (Snake\_Case\_Upper)

# RETURNING JSON

```
[HttpGet]
public ActionResult<IEnumerable<string>> Get()
{
    return new string[] { "value1", "value2" };
}
[HttpGet("test1")]
public IActionResult Get1()
{
    return Ok(new string[] { "value1", "value2" });
}
[HttpGet("test2")]
public string[] Get2()
{
    return new string[] { "value1", "value2" };
}
[HttpGet("test3")]
public IActionResult Get3()
{
    return new JsonResult(new string[] { "value1", "value2" });
}
```

# CONTENT NEGOTIATION

<https://github.com/skimeddic/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimeddic.com>

## CONTENT NEGOTIATION (BUILT-IN)

- Content will be served to the client as JSON by default
- To support XML, add:

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

- Send request with Accept: "application/xml"
  - Object must be serializable to XML
- Multiple supported Accept headers will be processed first-come, first-served
- To throw when the accept header is of an unsupported type:

```
builder.Services.AddControllers(options =>
{
    // Enable 406 Not Acceptable status code
    options.ReturnHttpNotAcceptable = true;
})
```

<https://github.com>

# CUSTOM CONTENT NEGOTIATION (CSV)

```
using Microsoft.AspNetCore.Mvc.Formatters;
using MediaTypeHeaderValue = Microsoft.Net.Http.Headers.MediaTypeHeaderValue;
namespace AutoLot.Api.Formatters;
public class CustomCsvOutputFormatter : TextOutputFormatter
{
    public CustomCsvOutputFormatter()
    {
        SupportedMediaTypes.Add(MediaTypeHeaderValue.Parse("text/csv"));
        SupportedEncodings.Add(Encoding.UTF8);
    }
    protected override bool CanWriteType(Type type) => typeof(IEnumerable<object>).IsAssignableFrom(type) || type.IsClass;
    public override async Task WriteResponseBodyAsync(OutputFormatterWriteContext context, Encoding selectedEncoding)
    {
        var response = context.HttpContext.Response;
        var buffer = new StringBuilder();
        var type = context.Object.GetType();
        var enumerable = context.Object as IEnumerable<object> ?? new[] { context.Object };
        var props = type.GetGenericArguments().FirstOrDefault()?.GetProperties() ?? type.GetProperties();
        // Header
        buffer.AppendLine(string.Join(",", props.Select(p => p.Name)));
        // Rows
        foreach (var item in enumerable)
        {
            var values = props.Select(p => p.GetValue(item, null)?.ToString()?.Replace(",", " ") ?? "");
            buffer.AppendLine(string.Join(",", values));
        }
        await response.WriteAsync(buffer.ToString());
    }
}
```

<https://>

# DATA SHAPING

<https://github.com/skimeddic/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimeddic.com>

## DATA SHAPING (GET)

- Return only the selected fields listed in the request, plus the PK

```
api/DataShaping?fields=firstname,lastname,timestamp
```

- Use reflection and `ExpandObjects` to return the PK and the requested fields
- Reflection is expensive, and process is fragile, so consider the value

```
[HttpGet]
[Produces("application/json")]
public IActionResult GetFromQuery([FromQuery] string fields)
{
    var owners = driverRepo.GetAll(); // Assume this returns List<Owner>
    var shapedOwners = dataShaper.ShapeData(owners, fields);
    return Ok(shapedOwners);
}
```

## DATA SHAPING (POST/PUT)

- Send only specific fields in the request, plus the PK

```
/api/DataShaping/5?v=1.0&values={"FirstName":"Fred","LastName":"FlintStone"}
```

- Retrieve instance from data store
  - Option 1: Use Newtonsoft JsonConvert.PopulateObject

```
var car = _repo.Get(id);  
JsonConvert.PopulateObject(values, car);  
_repo.Update(car);
```

- Option 2: Use System.Text.Json and reflection
  - NOTE: Using dynamic or expando with objects isn't recommended
- Reflection is expensive, and the process is fragile, so consider the value

# CORS

<https://github.com/skimeddic/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimeddic.com>

# CROSS ORIGIN RESOURCE SHARING (CORS)

- ASP.NET Core supports CORS
  - Header
  - Method
  - Credentials
  - Origin
  
- Configured in the DI container with the web app builder (ConfigureServices)
- Applied to the pipeline in the web app (Configure)

# HYPERMEDIA AS THE ENGINE OF APPLICATION STATE (HATEOAS)

<https://github.com/skimedec/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimedec.com>

# HYPERMEDIA AS THE ENGINE OF APPLICATION STATE (HATEOAS)

- Provide hypermedia links in the API response
- Goal is to provide flexibility to the API
- Contrasts with “traditional” methods, like versioning and OpenAPI docs

# SIMPLE HATEOAS EXAMPLE

```
{
  "accountNumber": 10,
  "accountType": "Checking",
  "links": [
    {
      "href": "10/balance",
      "rel": "balance",
      "type": "GET"
    },
    {
      "href": "10/withdraw",
      "rel": "withdraw",
      "type": "POST"
    },
    {
      "href": "10/deposit",
      "rel": "deposit",
      "type": "POST"
    }
  ]
}
```

# MODEL BINDING AND VALIDATION

<https://github.com/skimeddic/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimeddic.com>

## MODEL BINDING

- Route data and query string values are used only for simple types.
- Implicit model binding of complex types uses action method/page handler parameter or `BIND_PROPERTY`
  - Explicit model binding is also available
- Errors in data type conversion are added to `ModelState`

## MODEL BINDING

- Sources for data (in order):
  - Form fields
  - The request body (API services)
  - Route data
  - Query string parameters
  - Uploaded files

## MODEL BINDING EXPLICIT SOURCES

- Can explicitly specify the source:
  - [FromQuery] - query string.
  - [FromRoute] - route data.
  - [FromForm] - posted form fields.
  - [FromBody] - request body.
  - [FromHeader] - HTTP headers.
  
- Can also explicitly call for model binding with code

## VALIDATION

- Data annotations offer built-in client and server-side validation
  - Custom validation attributes are also available
- Implicit validation occurs with model binding
  - Explicit validation is also available
- Errors in validation are added to ModelState

# CUSTOM VALIDATION

<https://github.com/skimeddic/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimeddic.com>

## CUSTOM VALIDATION ATTRIBUTES

- Derive from `ValidationAttribute` for server-side validation
  - Override the `IsValid` method
  - `ValidationContext` provides access to the entire model
- Implement `IClientModelValidator` for client-side validation
  - `AddValidation` methods adds data- attributes to rendered elements
  - Ties into JS validation (e.g. JQuery validations)

# LOGGING

<https://github.com/skimeddic/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimeddic.com>

## LOGGING IS BUILT INTO FRAMEWORK

- LoggerFactory is part of the framework
- Extensible using 3<sup>rd</sup> party loggers (e.g. SeriLog)

# SERVICE CONTROLLERS

<https://github.com/skimedict/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimedict.com>

# CONTROLLERBASE (MVC WEB APPLICATIONS AND API SERVICES)

- Contains helper methods for:
  - Returning response messages
    - No body: NoContent (204), Ok (200), BadRequest (400)
    - With body: OkObjectResult(car) (200)
  - Model Binding and Validation
  - Redirecting requests

# ACTIONS

- Return IActionResult (Task<IActionResult>)
- GET is the default verb for actions
  - Any other verbs must be specified (browsers support GET and POST)
  - All methods should specify HTTP Verb
    - Unmarked action methods support all other methods

# APICONTROLLER ATTRIBUTE

<https://github.com/skimedidc/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimedidc.com>

## API CONTROLLER ATTRIBUTE

- ApiController Attribute (applied at the project or controller level)
  - Enables REST-specific behavior for controllers
    - Automatic 400 responses on model validation errors
    - Binding source parameter inference
    - Multipart/form-data inference

## BINDING SOURCE PARAMETER INFERENCE

- FromBody - complex types (one per action)
  - [FromBody(EmptyBodyBehavior = EmptyBodyBehavior.Allow)]
- FromForm - IFormFile and IFormFileCollection
- FromRoute - any values matching the route
- FromQuery - all others

# MODIFYING APICONTROLLER ATTRIBUTE FUNCTIONALITY

- Suppressions:
  - SuppressModelStateInvalidFilter = true;
    - // Automatic model state binding errors
  - SuppressInferBindingSourcesForParameters = true;
    - //All binding inference
  - SuppressConsumesConstraintForFormFileParameters = true;
    - // Multipart/form-data content type inference

## PROBLEM DETAILS

- Don't create a problem details error object if set to true
  - `SuppressMapClientErrors = false;`
  - <https://tools.ietf.org/html/rfc7807>
- Add additional information to the problem details object
  - `ClientErrorMapping[StatusCodes.Status404NotFound].Link = "https://httpstatuses.com/404";`
  - `ClientErrorMapping[StatusCodes.Status404NotFound].Title = "Invalid location";`

# ROUTING

<https://github.com/skimeddic/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimeddic.com>

# ROUTING

- Routing is bi-directional
  - Determines which Controller/Action or PageHandler to execute
  - Generates URLs within the application
- HTTP Verb is part of the route (MVC, WebAPI)
- Get/Post requests are handled by the HTTP Verb specific page handlers
  - OnGet/OnPost
- In versioned API Services, the version is part of the route

# API SERVICES ROUTING

- Routes are independent of file structure or names
- The preferred routing mechanism is attribute routing
  - Attribute Routing
    - Controllers - Route Attribute that combines tokens and literal values
    - Actions - Route Attribute or augment controller route with HTTP Verb attributes
    - Route attributes that begin with "/" reset the route
  - Web Apps only call Post and Get requests
  - Services support all HTTP verbs

## HTTP FILES IN VS 2022

- The Visual Studio 2022 .http file editor provides a convenient way to test ASP.NET Core projects, especially API apps.
- The editor provides a UI that:
  - Creates and updates .http files.
  - Sends HTTP requests specified in .http files.
  - Displays the responses.
- <https://learn.microsoft.com/en-us/aspnet/core/test/http-files?view=aspnetcore-8.0>

# IHTTPCLIENTFACTORY

<https://github.com/skimedic/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimedic.com>

## IHTTPCLIENTFACTORY

- ASP.NET Core 2.1 includes a new IHttpConnectionFactory service that makes it easier to configure and consume instances of HttpClient in apps.
- The factory:
  - Makes registering of instances of HttpClient per named client more intuitive.
  - Implements a Polly handler that allows Polly policies to be used for Retry, CircuitBreakers, etc.
  - Handles pooling and lifetime management of HttpClient

# FILTERS

<https://github.com/skimedic/dotnetdemos/tree/main/ASP.NET%20Core/RESTfulServices/APIs/9.0>

All slides copyright Philip Japikse <http://www.skimedic.com>

# FILTERS

- *Filters* in ASP.NET MVC allow you to run code before or after a particular stage in the execution pipeline.
  - Filters can be configured globally, per-controller, or per-action.
- Many types available
  - Action
  - Exception
  - Authorization
  - Resource
  - Result

# AVAILABLE FILTERS

## ➤ Authorization filters

- Run first
- Short circuit if not authorized

## ➤ Resource filters

- Run before/after the rest of the pipeline

## ➤ Action filters

- Run before/after action methods

## ➤ Exception filters

- Run when an unhandled exception is thrown

## ➤ Result filters

- Run before/after action results

- Can be set at the app, controller, or action level

# EXCEPTION FILTERS

## ➤ Come into play on unhandled exceptions

```
public class CustomExceptionHandlerAttribute(
    IWebHostEnvironment hostEnvironment) : ExceptionFilterAttribute
{
    public override void OnException(ExceptionContext context)
    {
        string message = context.Exception.Message;
        context.Result = new BadRequestObjectResult(new {Message = message });
        //context.ExceptionHandled = true; //If this true, the exception is swallowed
    }
}
```

## ➤ Configured with AddControllers[WithViews] (for whole application filters)

```
builder.Services.AddControllers(config =>
{
    config.Filters.Add(new CustomExceptionHandlerAttribute(builder.Environment));
})
```

# ACTION FILTERS

- Come into play before and/or after actions are executed

```
public class SimpleAuthenticationActionFilter : IActionFilter
{
    public void OnActionExecuting(ActionExecutingContext context)
    {
        // do something before the action executes
    }
    public void OnActionExecuted(ActionExecutedContext context)
    {
        // do something after the action executes
    }
}
```

## Contact Me

[skimedic@outlook.com](mailto:skimedic@outlook.com)

[www.skimedic.com/blog](http://www.skimedic.com/blog)

[www.twitter.com/skimedic](http://www.twitter.com/skimedic)

<http://bit.ly/apressbooks>

Questions?



# Thank You!