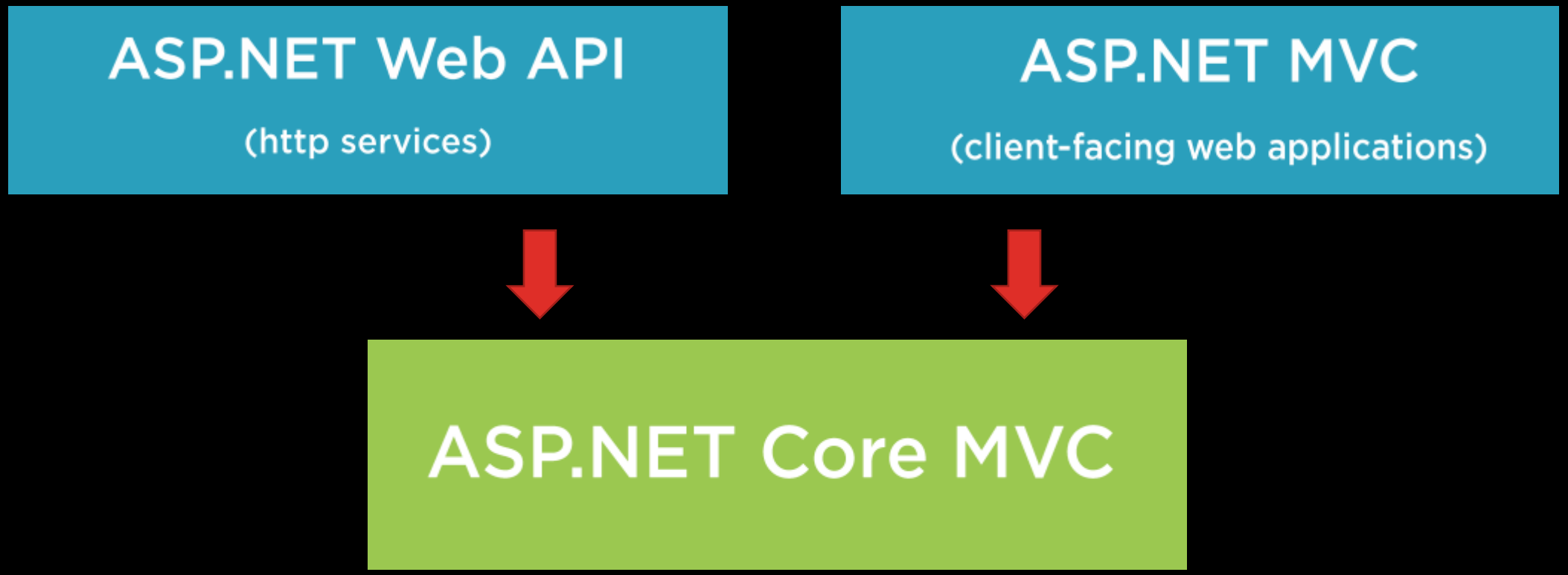
AGENDA DAY 2

- ASP.NET Core: The Big Picture
- Starting a New ASP.NET Core Project
- Routing
- Configuration
- Entity Framework Core
- UnitOfWork
- Log and Cache
- .NET core deployment

ASP.NET CORE - MIDDLEWARE



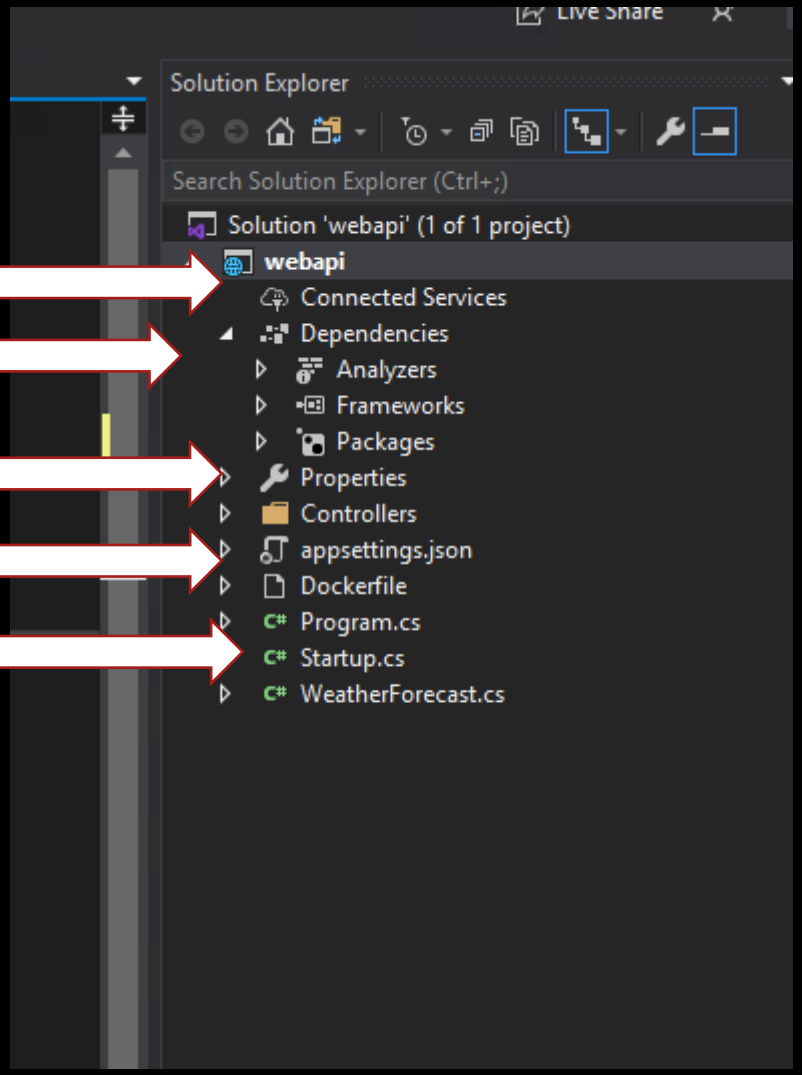
SOLUTION TREE

Legacy web service/ WCF

NuGet/ project ref / npm /dll's / other

Debug profile running settings

Setup



THE MAIN METHOD VERSIONS 3.1-5

```
namespace webapi
{
    0 references
    public class Program
    {
        0 references
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        1 reference
        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}
```

STARTUP VERSIONS 3.1-5

For Dependency Injection

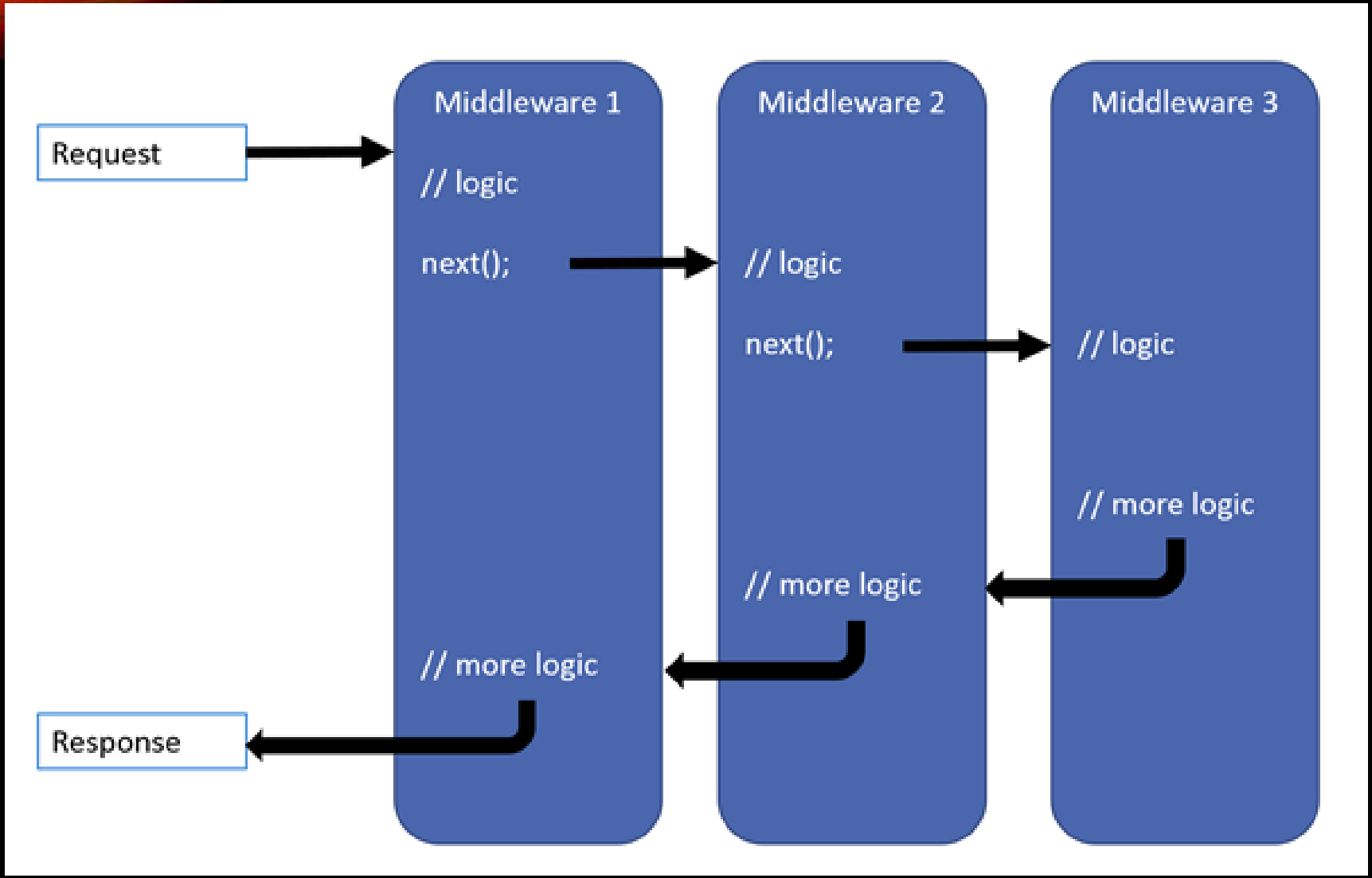
For HTTP request pipeline

```
public class Startup
{
    0 references
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    1 reference
    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this meth
    0 references
    public void ConfigureServices(IServiceCollection service
    {
        services.AddControllers();
    }

    // This method gets called by the runtime. Use this meth
    0 references
    public void Configure(IApplicationBuilder app, IWebHostE
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
```

THE MAIN METHOD & STARTUP VERSION 6

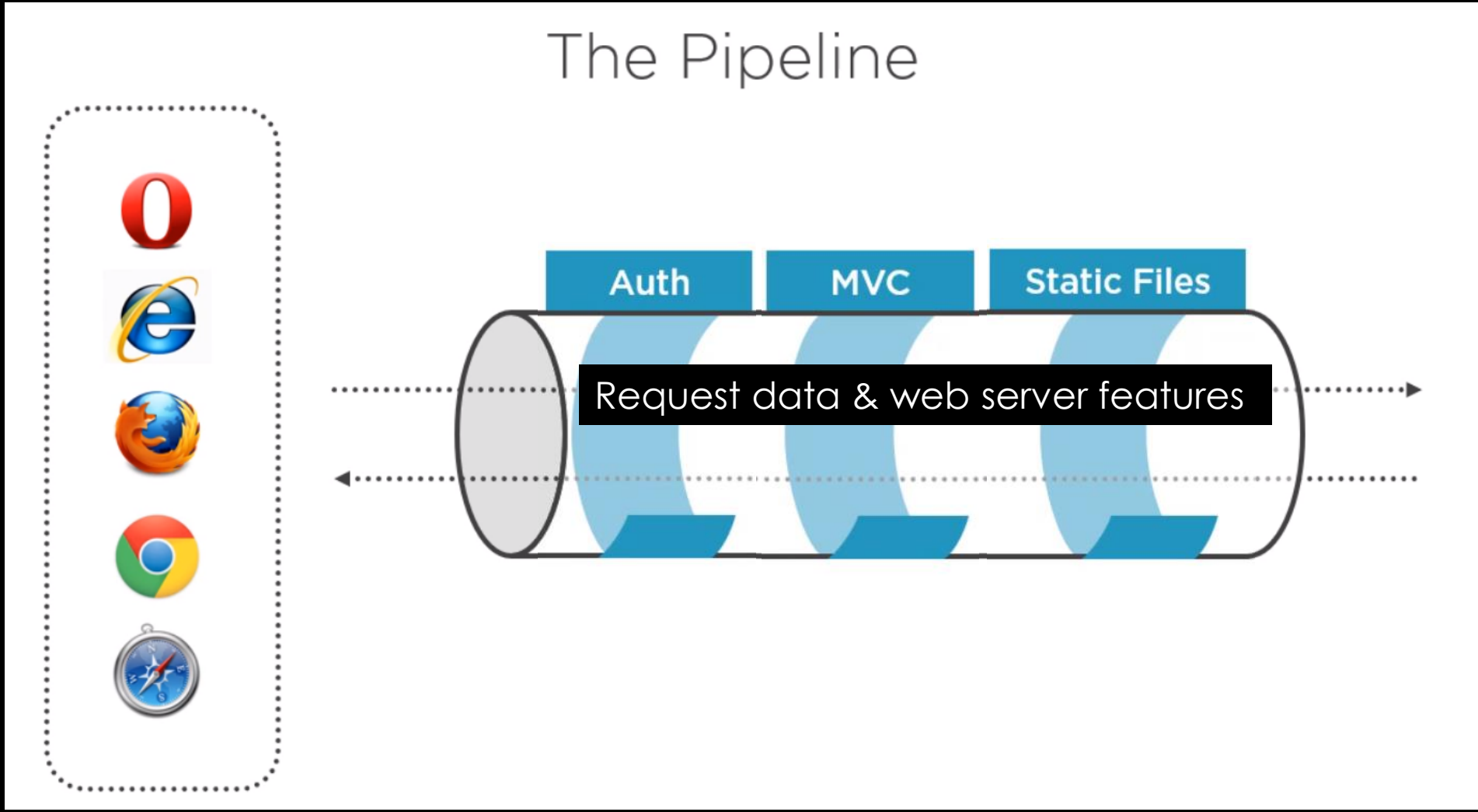
```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

MIDDLEWARE



https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-6.0

MIDDLEWARE


The Pipeline
Auth
MVC
Static Files
Request data & web server features

MIDDLEWARE

RUN()

his method only receives only context parameter and doesn’t know about the next middleware.

These delegates are usually known as terminal delegates because they terminate or end the middleware pipeline.

MIDDLEWARE

RUN()

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

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

app.Run();
```

MAP()

Map extensions are used for branching the pipeline.

Map extensions branch the request pipeline based on matching the given request path.

If the request path starts with the given path, the branch is executed.

MIDDLEWARE

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Map("/map1", HandleMapTest1);

app.Map("/map2", HandleMapTest2);

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello from non-Map delegate. <p>");
});

app.Run();

static void HandleMapTest1(IApplicationBuilder app)
{
    app.Run(async context =>
    {
        await context.Response.WriteAsync("Map Test 1");
    });
}

static void HandleMapTest2(IApplicationBuilder app)
{
    app.Run(async context =>
    {
        await context.Response.WriteAsync("Map Test 2");
    });
}
```

Request	Response
ocalhost:1234	Hello from non-Map delegate.
ocalhost:1234/map1	Map Test 1
ocalhost:1234/map2	Map Test 2
ocalhost:1234/map3	Hello from non-Map delegate.

MIDDLEWARE

USE()

The whole idea behind middleware is to link one after another.

Let us take a look at the Use() method, which helps us to chain the delegates one after the other.

This method will accept two parameters, context and next. Let us create a inline middleware using the Use()

MIDDLEWARE

USE()

```
app.Use(async (context, next) => {
    await context.Response.WriteAsync($ "Before Request {Environment.NewLine}");
    await next();
    await context.Response.WriteAsync($ "After Request {Environment.NewLine}");
});
app.Run(async context => {
    await context.Response.WriteAsync($ "Hello Readers!{Environment.NewLine}");
});
```

```
Before Request
Hello Readers!
After Request
```

ADDING MIDDLEWARE

```
// This method gets called by the runtime. Use this r
public void Configure(IApplicationBuilder app, IWebHo
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseStaticFiles();
    app.UseAuthentication();

    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Conference}/{actior

    });
}
```

wwwroot folder

ROUTING

MVC

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
});
}
```

Attribute controllers

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

https://learn.microsoft.com/en-us/aspnet/core/fundamentals/routing?view=aspnetcore-6.0

ROUTING

Routing basics

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/", () => "Hello World!");

app.Run();
```

```
app.MapGet("/hello/{name:alpha}", (string name) => $"Hello {name}!");
```

ROUTE CONSTRAINTS

constraint	Example	Example Matches	Notes
int	{id:int}	123456789- ,123456789	Matches any integer
bool	{active:bool}	true, FALSE	Matches true or false. Case-insensitive
datetime	{dob:datetime}	2016-12-31, 2016-12-31 7:32pm	Matches a valid DateTime value in the invariant culture. See preceding warning.
decimal	{price:decimal}	1,000.01- ,49.99	Matches a valid decimal value in the invariant culture. See preceding warning.
double	{weight:double}	1.234, -1,001.01e8	Matches a valid double value in the invariant culture. See preceding warning.
float	{weight:float}	1.234, -1,001.01e8	Matches a valid float value in the invariant culture. See preceding warning.

ROUTE CONSTRAINTS

constraint	Example	Example Matches	Notes
guid	{id:guid}	CD2C1638-1638-72D5-1638-DEADBEEF1638	Matches a valid Guid value
long	{ticks:long}	123456789- ,123456789	Matches a valid long value
minlength(value)	{username:minlength(4)}	Rick	String must be at least 4 characters
maxlength(value)	{filename:maxlength(8)}	MyFile	String must be no more than 8 characters
length(length)	{filename:length(12)}	somefile.txt	String must be exactly 12 characters long
length(min,max)	{filename:length(8,16)}	somefile.txt	String must be at least 8 and no more than 16 characters long
min(value)	{age:min(18)}	19	Integer value must be at least 18
max(value)	{age:max(120)}	91	Integer value must be no more than 120
range(min,max)	{age:range(18,120)}	91	Integer value must be at least 18 but no more than 120
alpha	{name:alpha}	Rick	String must consist of one or more alphabetical characters, a-z and case-insensitive.
regex(expression)	{ssn:regex(^\\d{{3}}-\\d{{2}}-\\d{{4}}$)}	123-45-6789	String must match the regular expression. See tips about defining a regular expression.
required	{name:required}	Rick	Used to enforce that a non-parameter value is present during URL generation

ROUTE CONSTRAINTS

constraint	Example	Example Matches	Notes
alpha	{name:alpha}	Rick	String must consist of one or more alphabetical characters, a-z and case-insensitive.
regex(expression)	{ssn:regex(^\\d{{3}}-\\d{{2}}-\\d{{4}}$)}	123-45-6789	String must match the regular expression. See tips about defining a regular expression.
required	{name:required}	Rick	Used to enforce that a non-parameter value is present during URL generation

ROUTING
ATTRIBUTE-ROUTING

- [HttpDelete]
- [HttpGet]
- [HttpHead]
- [HttpOptions]
- [HttpPatch]
- [HttpPost]
- [HttpPut]

ROUTING
ATTRIBUTE-ROUTING

```
[Route("users/{id:int:min(1)}")]
public User GetUserById(int id) { }
```

```
[HttpGet("{id:regex(\\d-\\d)}")]
public string Get(string id)
{
        return "value";
}
```

```
[HttpGet("{id:length(2,4)}")]
public string Get(string id)
{
        return "value";
}
```

```
[HttpGet("{message:regex(^\\d{{2}}-\\d{{2}}-\\d{{2}}$)}")]
0 references
public IEnumerable<WeatherForecast> Get()
{
```

ROUTING

MVC

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
});
}
```

Attribute controllers

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

CONFIGURATION

Microsoft.Extensions.Configuration

DEFINING CONFIGURATION IN JSON FILES

- Is a default
- Is very essay
- The usual way

```
"ConnectionStrings": {
  "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=TennisBookings;Trusted_Connect
},

"Logging": {
  "LogLevel": {
    "Default": "Warning"
  }
},

"AllowedHosts": "*",
```

```
  "Features": {
    "HomePage": {
      "EnableGreeting": true,
      "EnableWeatherForecast": true,
      "ForecastSectionTitle": "What's the weather doing?"
    },
    "WeatherForecasting": {
      "EnableWeatherForecast": true
    }
  },

  "ExternalServices": {
    "WeatherApiUrl": "http://localhost:62855"
```

HOW TO USE DIRECTLY

```
public class IndexModel : PageModel
{
    private readonly IGreetingService _greetingService;
    private readonly IConfiguration _configuration;

    0 references | 0 exceptions
    public IndexModel(IGreetingService greetingService, IConfiguration configuration)
    {
        _greetingService = greetingService;
        _configuration = configuration;
    }
```

```
_configuration.GetValue<bool>("Features:HomePage:EnableGreeting")
```

LOGICAL CONFIGURATION STRUCTURE

MyStringKey ="This isa string value"

```
_configuration.GetValue<string>("MyStringKey")
```

MyB001eanKey = true

```
_configuration.GetValue<bool>("MyB001eanKey")
```

MyIntegerKey = 100

```
_configuration.GetValue<int>("MyIntegerKey")
```

CONFIGURATION HIERARCHY

```
"Features": {
  "HomePage": {
    "EnableGreeting": true,
    "EnableWeatherForecast": true,
    "ForecastSectionTitle": "What's the weather doing?"
  },
  "WeatherForecasting": {
    "EnableWeatherForecast": true
  }
},
```

```
_configuration.GetValue<bool>("Features:HomePage:EnableGreeting")
```

CONFIGURATION HIERARCHY BY SECTION

```
"Features": {
  "HomePage": {
    "EnableGreeting": true,
    "EnableWeatherForecast": true,
    "ForecastSectionTitle": "What's the weather doing?"
  },
  "WeatherForecasting": {
    "EnableWeatherForecast": true
  }
},
```

```
var homePageFeatures = _configuration.GetSection("Features:HomePage");

if (homePageFeatures.GetValue<bool>("EnableGreeting"))
```

DEFINE CONNECTION STRING

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<TennisBookingDbContext>(options =>
        options.UseSqlServer(
            Configuration.GetConnectionString("DefaultConnection")));
```

appsettings.json

```
"ConnectionStrings": {
  "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=TennisBookings;Trusted_Connect
},
"Logging": {
  "LogLevel": {
    "Default": "Warning"
  }
},
```

PROBLEMS WITH GETTING VALUES

- Repetitive code
- Fragile naming
- Can lead to bugs

STRONGLY TYPED - FIRST OPTION BIND CLASS

Create a class

```
private class Features
{
    0 references | 0 exceptions
    public bool EnableRandomGreeting { get; set; }
    0 references | 0 exceptions
    public bool EnableWeatherForecast { get; set; }
    0 references | 0 exceptions
    public string ForecastSectionTitle { get; set; }
}
```

Bind to class

```
var features = new Features();
_configuration.Bind("Features:HomePage", features);
```

STRONGLY TYPED - SECOND OPTION APPLYING THE OPTIONS PATTERN

Injecting options with IOptions<T>

On class Startup

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddOptions();

    services.Configure<Features>(Configuration.GetSection("Features:HomePage"));
```

On class

```
0 references | 0 exceptions
public IndexModel(IOptions<Features> featureSetting, IWeather
```

STRONGLY TYPED - THIRD OPTION
APPLYING THE OPTIONS PATTERN WITH SINGLETON

On class Startup

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddOptions();

    services.Configure<Features>(Configuration.GetSection("Features:HomePage"));
    services.AddSingleton(resolver =>
                    resolver.GetRequiredService<IOptions<Features>>().Value);
```

On class

```
public Ind del(Features featureSetting, IWeatherForecaster w
```

IOPTIONS<T>

- Does not support options reloading
- Registered as a singleton in D.I. container
- Values bound when first used
- Can be injected into all service lifetimes
- Does not support named options

OTHER OPTIONS
IOPTIONS SNAPSHOT<T>

- Supports reloading of configuration
- Registered as scoped in D.I. container
- Values may reload per request
- Can not be injected into singleton services
- Supports named options

OTHER OPTIONS
IOPTIONS MONITOR<T>

- Supports reloading of configuration
- Registered as a singleton in D.I. container
- Values are reloaded immediately
- Can be injected into all service lifetimes
- Supports named options

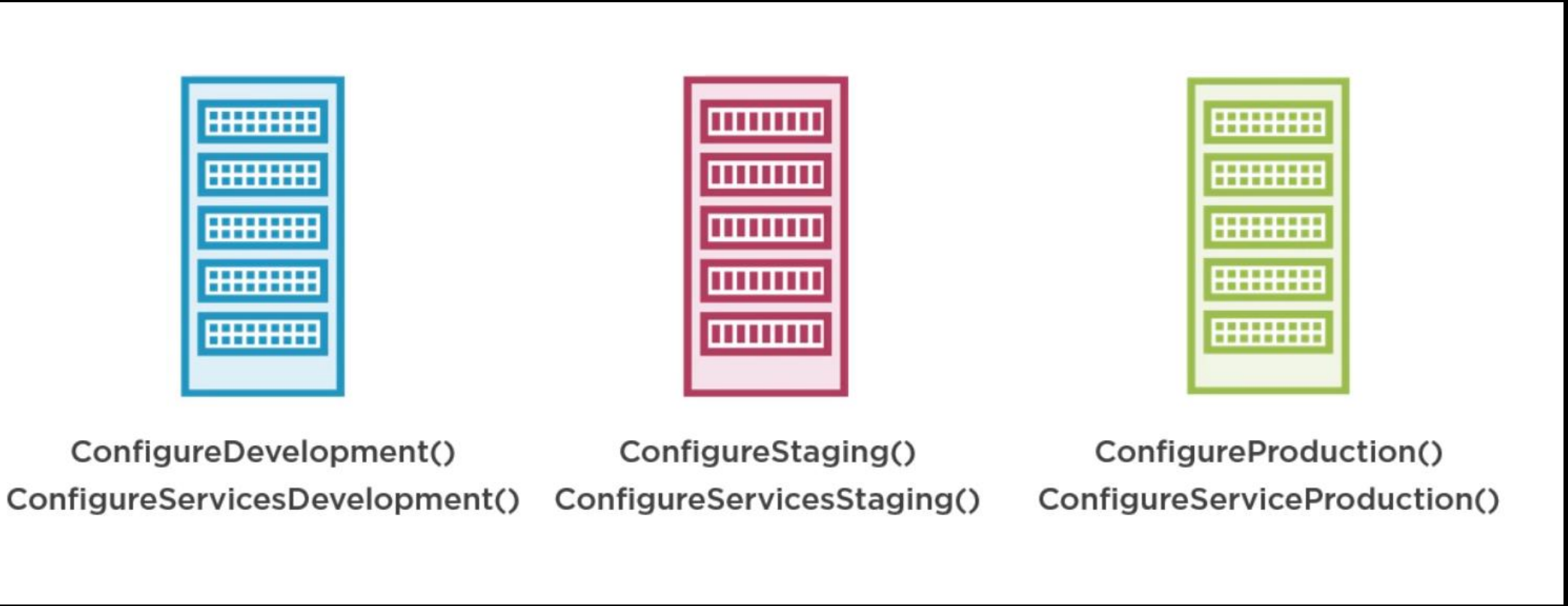
CHOOSING AN OPTIONS INTERFACE

	Use in singletons	Supports reloading	Named options
IOptions	✓	✗	✗
IOptionsSnapshot	✗	✓	✓
IOptionsMonitor	✓	✓	✓

DEMO

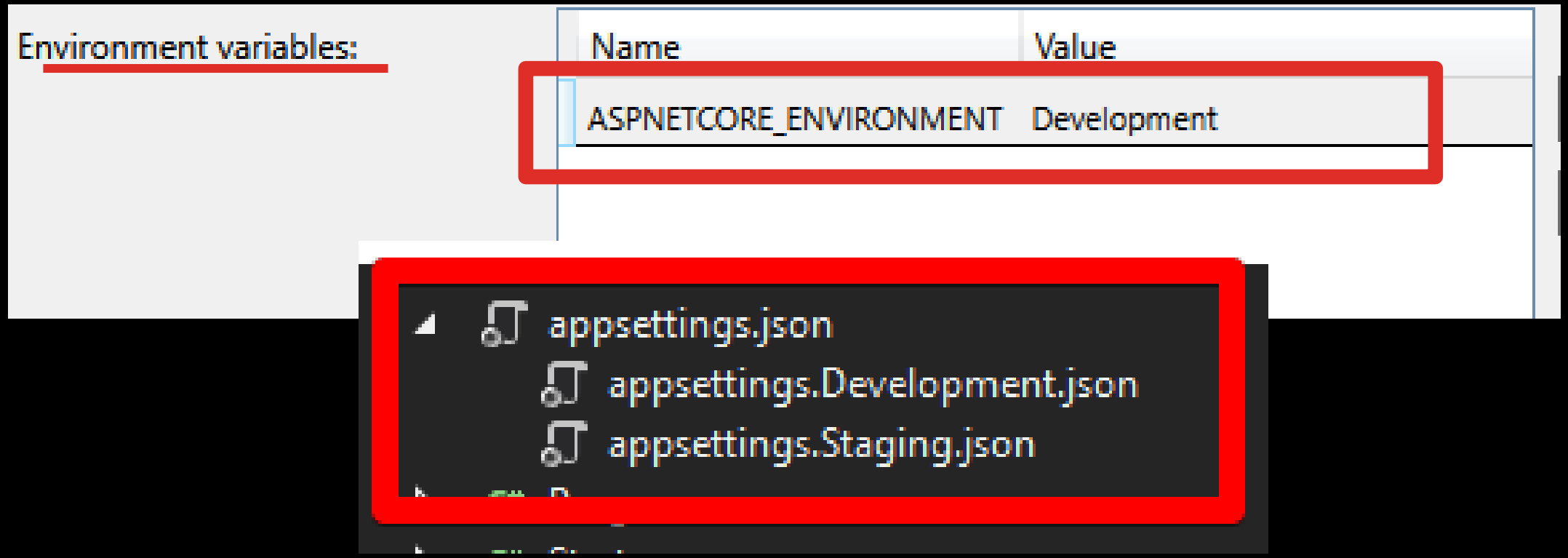
ENVIRONMENTS SETTINGS

- Launch Profiles
- Environments

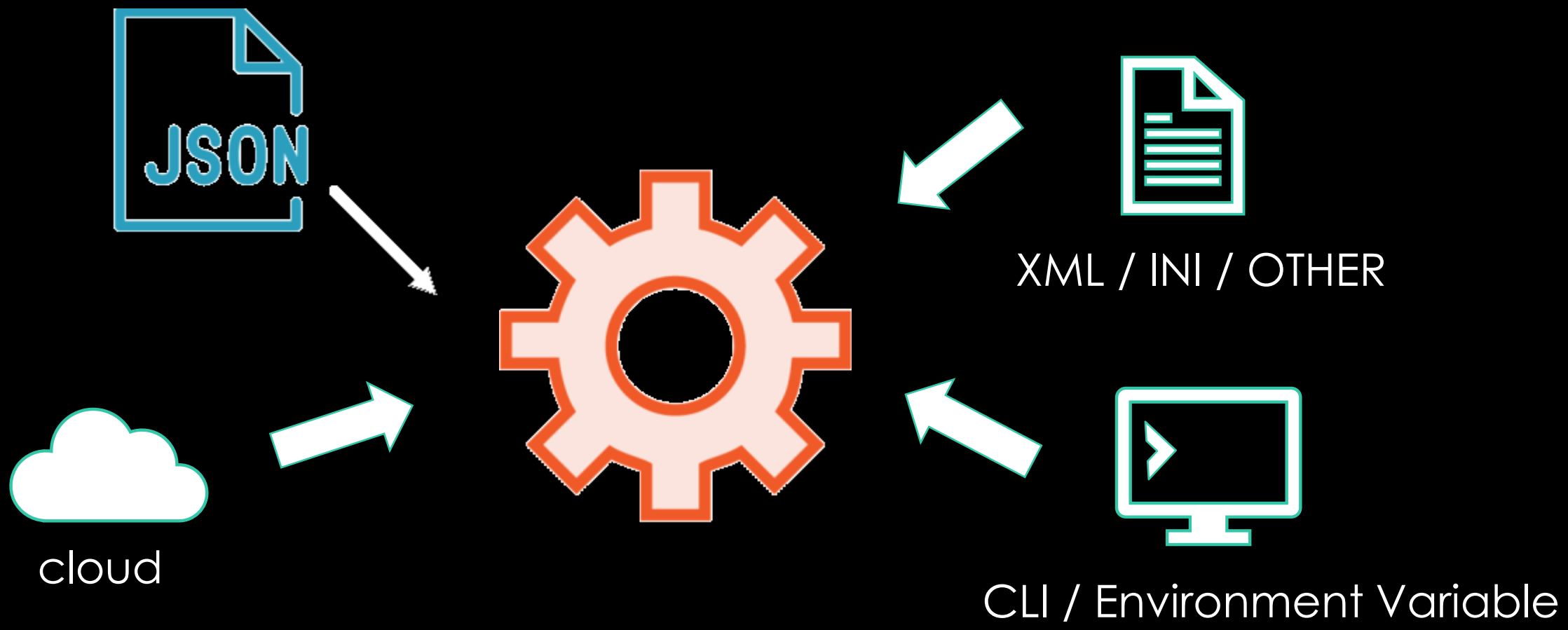


ON VISUAL STODIO

environment values : ASPNETCORE_ENVIRONMENT



CONFIGURATION PROVIDERS



XML FILE

- section0:key0
- section0:key1
- section1:key0
- section1:key1

```
public static IHostBuilder CreateHostBuilder(string[] args) =
    Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
        {
            webBuilder.ConfigureAppConfiguration((hostingContext, config) =>
            {
                config.AddXmlFile(
                    "config.xml", optiona
            });

            webBuilder.UseStartup<Startu
        });
```

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <section0>
    <key0>value</key0>
    <key1>value</key1>
  </section0>
  <section1>
```

INI FILE

```ini
[section0]
key0=value
key1=value

[section1]
subsection:key=value

[section2:subsection0]
key=value

[section2:subsection1]
key=value
```

- section0:key0
- section0:key1
- section1:subsection:key
- section2:subsection0:key
- section2:subsection1:key

ENVIRONMENT VARIABLE

HomePage:ShowGallery ➔ HomePage__showGallery=true

```
s.Web>set Features__Greeting__GreetingColour=#00FF00
```

SECURING SENSITIVE DATA IN CONFIGURATION

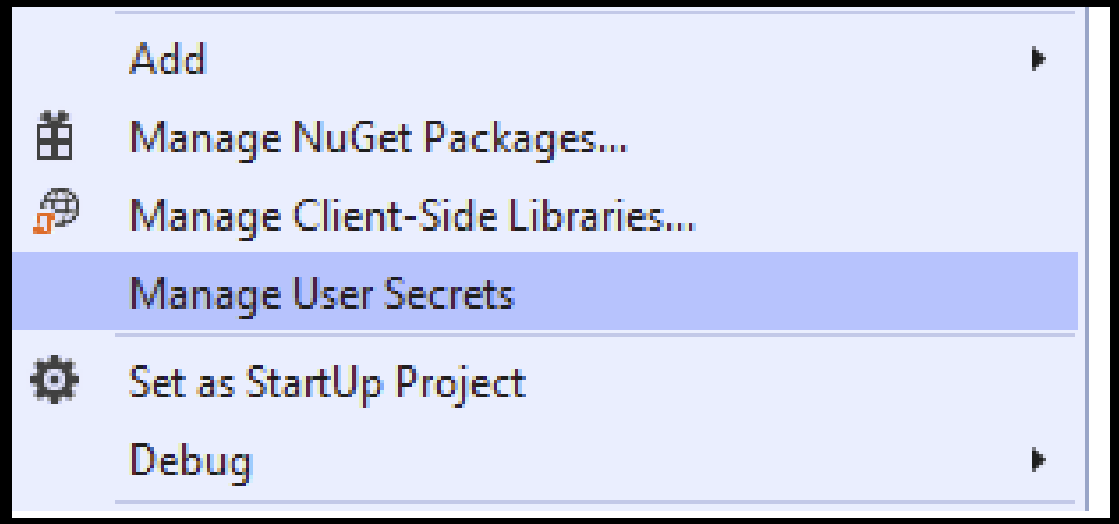
```
<PropertyGroup>
  <TargetFramework>netcoreapp3.1</TargetFramework>
  <UserSecretsId>18b97cbe-01f2-415a-8560-1cb369d8cb5a</UserSecretsId>
  <DockerDefaultTargetOS>Linux</DockerDefaultTargetOS>
  <DockerfileContext>.</DockerfileContext>
</PropertyGroup>
```

USER SECRETS USED

CLI

```
ion3>dotnet user-secrets set "ExternalServices:WeatherApiUrl" "shalomSec"
```

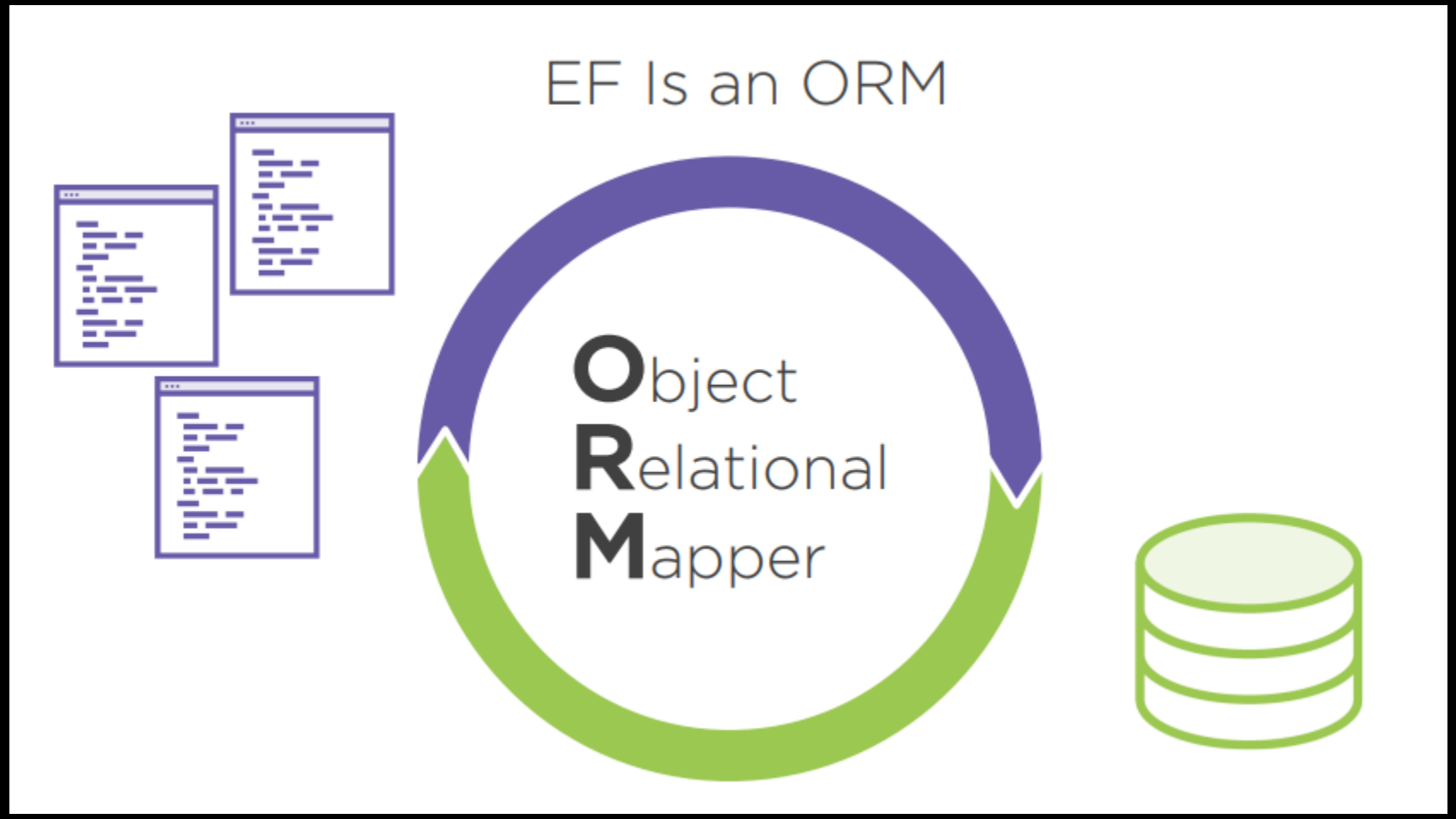
Visual Studio Click right on project



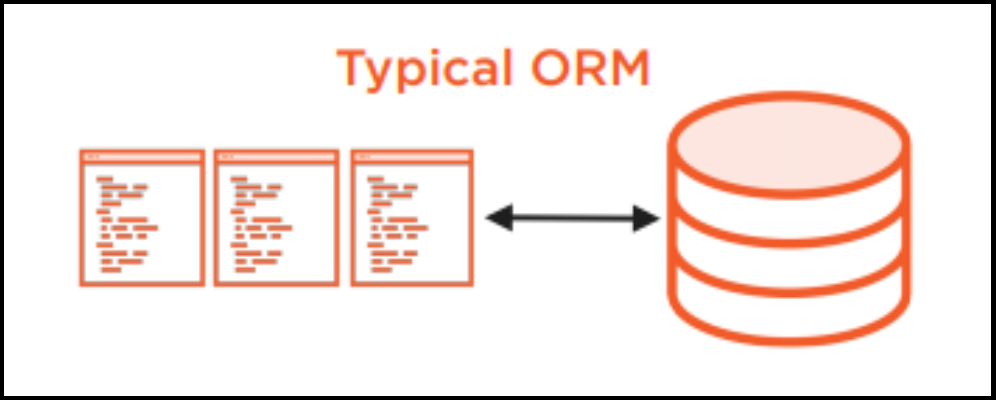
תרגיל 3

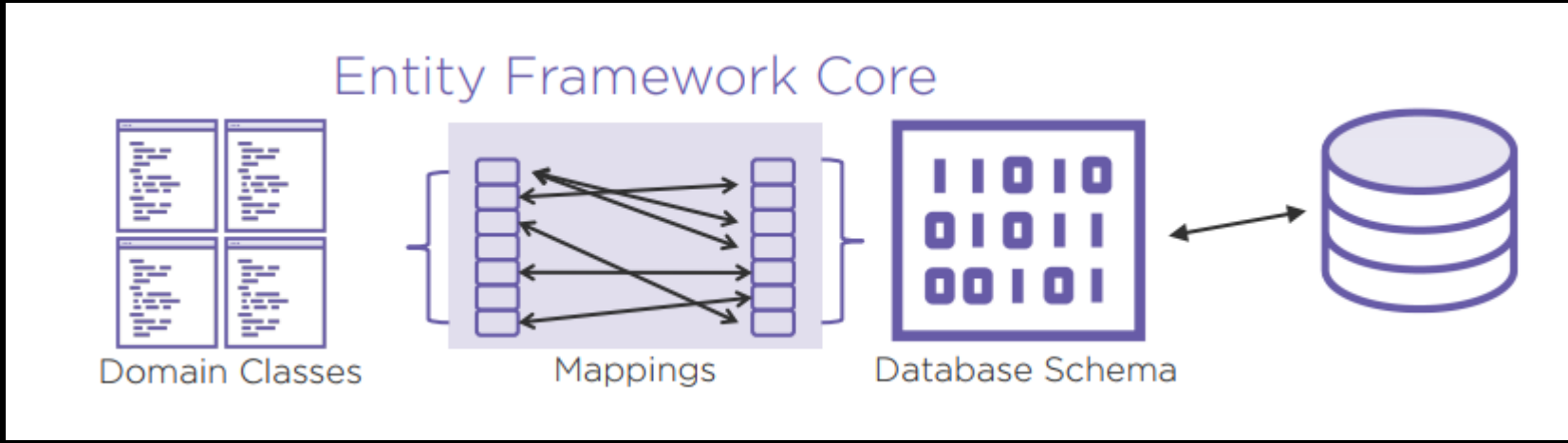
- קובץ תרגיל מצורף

ENTITY FRAMEWORK CORE



EF MAPS DIFFERENTLY THAN MOST ORMS





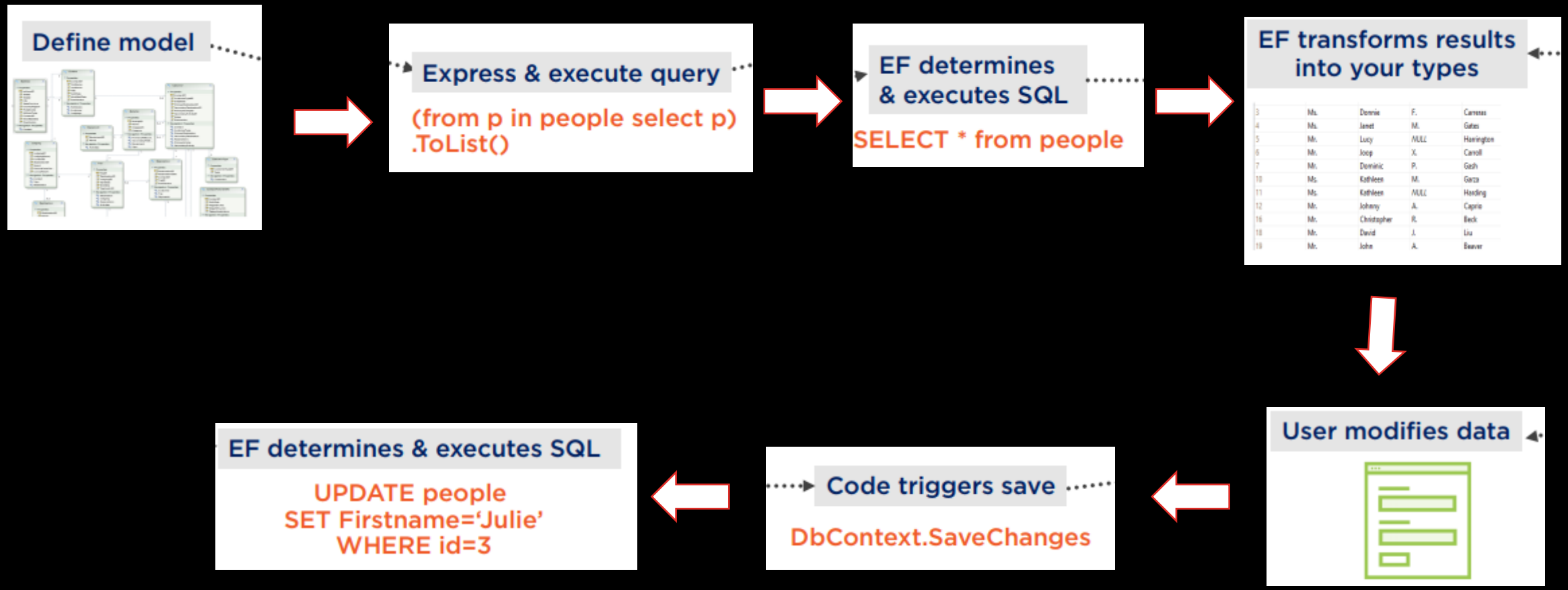
WHY THIS ORM, EF CORE?

- Developer productivity
- First class member of Microsoft .NET stack
- Consistent query syntax with LINQ to Entities
- Focus on domain, not on DB, connections, commands, etc

CURRENTLY AVAILABLE PROVIDERS FOR EF CORE

- SQL Server (Microsoft)
- SQLite (Microsoft, Devart)
- InMemory (Microsoft)
- SQL Server Compact (Erik Eilskov Jensen (MVP))
- MySQL (Oracle, Pomelo, Devart)
- Oracle (Devart)
- PostgreSQL (Npgsql/Shay Rojansky (MVP), Devart)
- IBM Data Server DB2 (IBM, Devart)
- MyCat (Pomelo)
- Firebird (Rafael Almeida)

BASIC WORKFLOW



INSTALL ON PROJECT

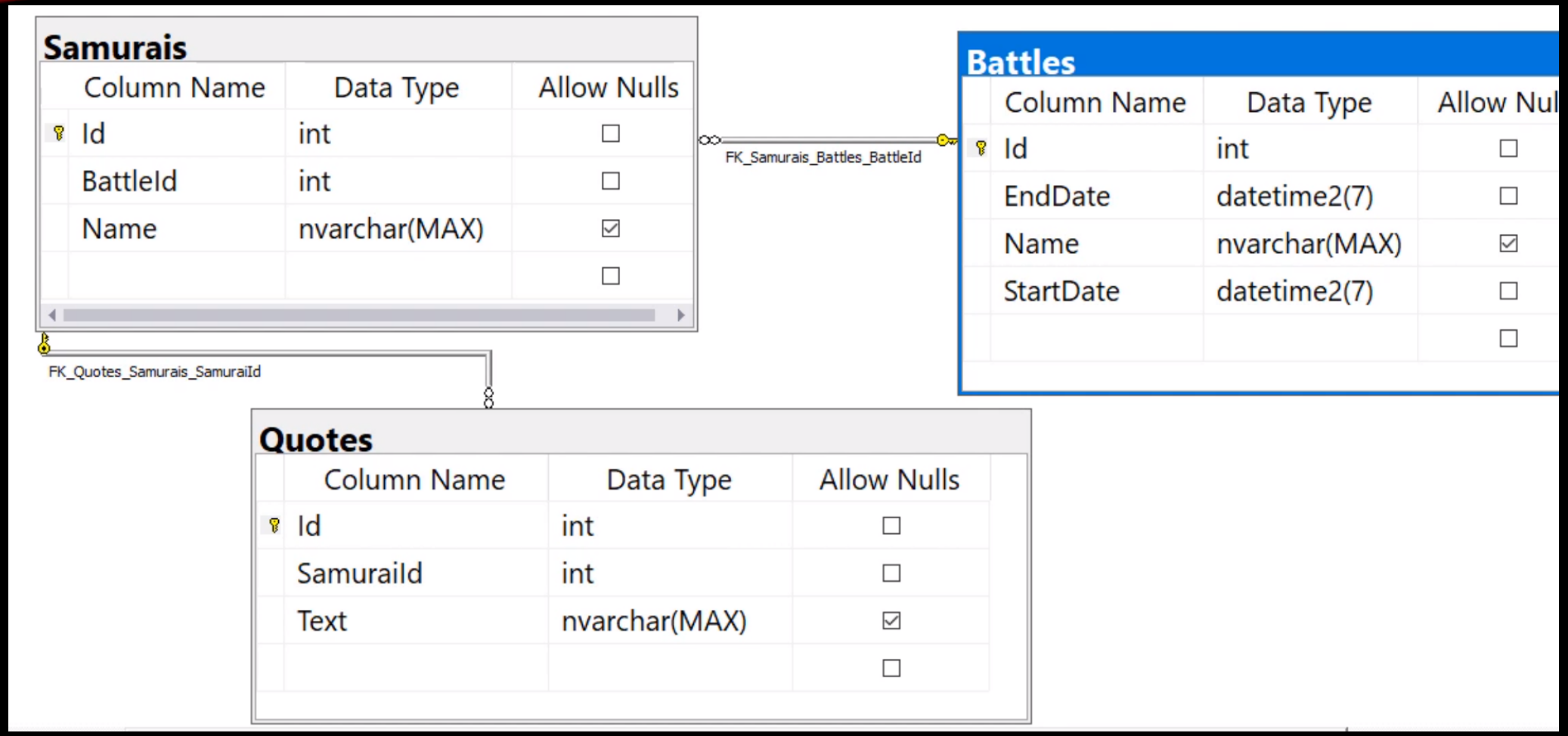
- Microsoft.EntityFrameworkCore
- To Manage
  - Microsoft.EntityFrameworkCore.Tools
  - Microsoft.EntityFrameworkCore.Design
  - Microsoft.EntityFrameworkCore.SqlServer

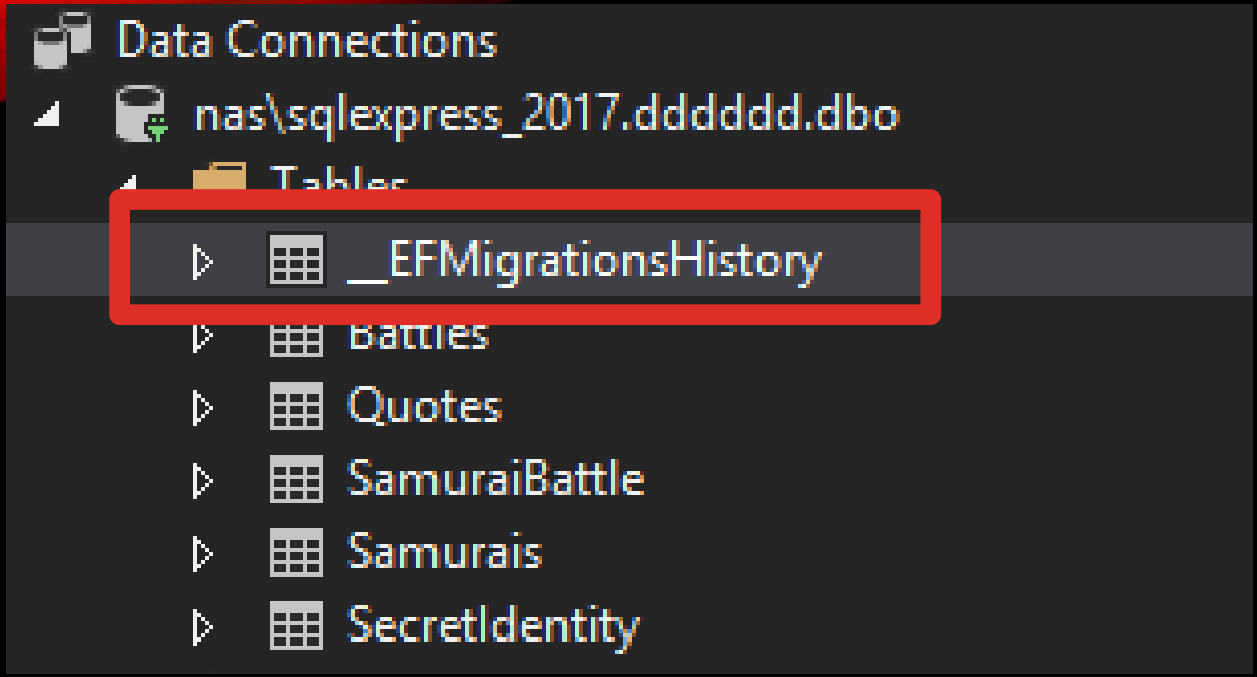
DATABASE FIRST

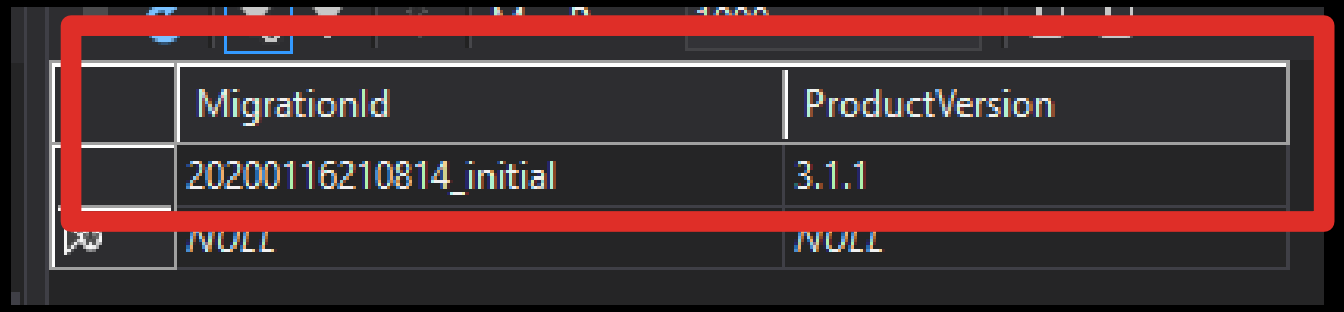
1) First time: Add-Migration initial

2) Commit on DB: Dev: update-database –v / prod: script-migration

3) To update: Add-Migration [name_mig]

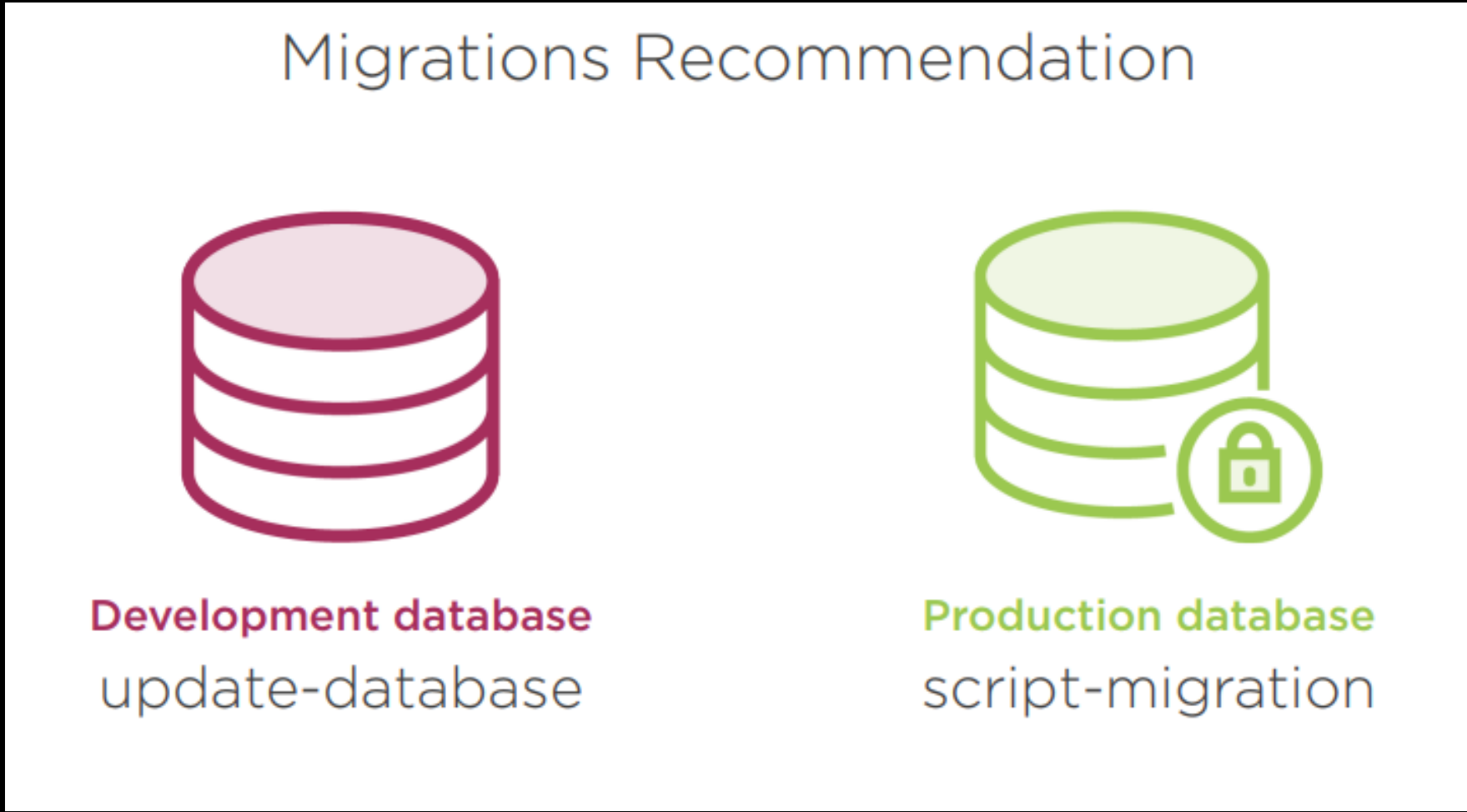

Samurais
Column Name
Data Type
Allow Nulls
Id
int
BattleId
int
Name
nvarchar(MAX)
FK_Samurais_Battles_BattleId
Battles
Column Name
Data Type
Allow Nul
Id
int
EndDate
datetime2(7)
Name
nvarchar(MAX)
StartDate
datetime2(7)
FK_Quotes_Samurais_SamuraiId
Quotes
Column Name
Data Type
Allow Nulls
Id
int
SamuraiId
int
Text
nvarchar(MAX)





MigrationId	ProductVersion
20200116210814_initial	3.1.1
NULL	NULL

MIGRATIONS RECOMMENDATION



REVERSE ENGINEERING AN EXISTING DATABASE

- Create DbContext & classes from database
- Updating model is not currently supported
- Transition to migrations is not pretty … look
- for helpful link in resources
- PowerShell command:

scaffold-dbcontext

- Scaffold-DbContext "Data Source=localhost;Initial Catalog=dddddd;Integrated Security=True" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -DataAnnotations -Context MaccabiPushVoipDbContext

NEW QUERYING ON EF CORE

Like

EF.Functions.Like(property, %abc%)

```
_context.Samurais.Where(s=>
  EF.Functions.Like(s.Name,"%abc%")
)
```

SQL LIKE(%abc%)

EXECUTESQLINTERPOLATED

OLD : ~~_DbSet.SqlQuery~~

New: _DbSet.FromSqlInterpolated

- context.Database.ExecuteSqlInterpolated(

  $"SELECT * FROM [dbo].[SearchBlogs]({userSuppliedSearchTerm})“)

```
var e = db
    .Database
    .ExecuteSqlInterpolated($"UPDATE peoples SET Name={name} WHERE Id={id}");
```

FROMSQLRAW

- context.Database.ExecuteSqlRaw ("SELECT * FROM [dbo].[SearchBlogs]({0})", userSuppliedSearchTerm)

```
var e = db
    .Database
    .ExecuteSqlRaw("UPDATE peoples SET Name={0} WHERE Id={1}", "New name", id);
```

LOGGING

```
var builder = WebApplication.CreateBuilder(args);
builder.Logging.ClearProviders();
builder.Logging.AddConsole();
```

LOGGING

```
public class AboutModel : PageModel
{
    private readonly ILogger _logger;

    public AboutModel(ILogger<AboutModel> logger)
    {
        _logger = logger;
    }

    public void OnGet()
    {
        _logger.LogInformation("About page visited at {DT}",
            DateTime.UtcNow.ToLongTimeString());
    }
}
```

CONFIGURE LOGGING

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  }
}
```

```
{
  "Logging": {
    "LogLevel": { // All providers, LogLevel applies to all the enabled providers.
      "Default": "Error", // Default logging, Error and higher.
      "Microsoft": "Warning" // All Microsoft* categories, Warning and higher.
    },
    "Debug": { // Debug provider.
      "LogLevel": {
        "Default": "Information", // Overrides preceding LogLevel:Default setting.
        "Microsoft.Hosting": "Trace" // Debug:Microsoft.Hosting category.
      }
    },
    "EventSource": { // EventSource provider
      "LogLevel": {
        "Default": "Warning" // All categories of EventSource provider.
      }
    }
  }
}
```

LOG IN PROGRAM.CS

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();
app.Logger.LogInformation("Adding Routes");
app.MapGet("/", () => "Hello World!");
app.Logger.LogInformation("Starting the app");
app.Run();
```

LOG LEVEL

ogLevel	Value	Method	Description
Trace	0	LogTrace	Contain the most detailed messages. These messages may contain sensitive app data. These messages are disabled by default and should **not** be enabled in production.
Debug	1	LogDebug	For debugging and development. Use with caution in production due to the high volume.
Information	2	LogInformation	Tracks the general flow of the app. May have long-term value.
Warning	3	LogWarning	For abnormal or unexpected events. Typically includes errors or conditions that don't cause the app to fail.
Error	4	LogError	For errors and exceptions that cannot be handled. These messages indicate a failure in the current operation or request, not an app-wide failure.
Critical	5	LogCritical	For failures that require immediate attention. Examples: data loss scenarios, out of disk space.
None	6		Specifies that a logging category shouldn't write messages.

LOG LEVEL

```
[HttpGet("{id}")]
public async Task<ActionResult<TodoItemDTO>> GetTodoItem(long id)
{
    _logger.LogInformation(MyLogEvents.GetItem, "Getting item {Id}", id);

    var todoItem = await _context.TodoItems.FindAsync(id);

    if (todoItem == null)
    {
        _logger.LogWarning(MyLogEvents.GetItemNotFound, "Get({Id}) NOT FOUND", id);
        return NotFound();
    }

    return ItemToDTO(todoItem);
}
```

ERROR HANDLING WITH NLOG

- Add Packges
  - NLog.Web.AspNetCore
  - NLog.Config
- Add on Program.cs

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
        {
            webBuilder.UseStartup<Startup>();
        }).ConfigureLogging(logging =>
        {
            logging.ClearProviders();
            logging.SetMinimumLevel(Microsoft.Extensions.Logging.LogLevel.Trace);
        }).UseNLog();
```

https://github.com/NLog/NLog/wiki/Getting-started-with-ASP.NET-Core-3

.NET CORE DEPLOYMENT

Framework-dependent (FDD)	Self-contained (SCD)

Framework-dependent Deployment

Advantages

No need to configure target operating systems up-front

Deployment size is small

.NET Core itself is shared

Disadvantages

Runs only if .NET Core is pre-installed

Possible compatibility problems

Self-contained Deployment

Advantages

Sole control about the .NET Core version your app uses

100% certainty that the app will run on target system

Side-by-side execution without problems

Disadvantages

Need to configure operating systems to deploy to up-front

Large deployment size

Disk space

Framework-dependent executables (FDE)

- Is a self-contained deployment
- Uses resources from installed .NET Core
- Generated an OS-native executable

Publishing a Self-contained Application

- .NET Core and runtime will be included
- No need to pre-install
- OS-native start
- RID must be known