What’s new with .NET 7.0?

ASP.NET Core

- Rate Limiting
- Output Caching Middleware
- Built-in HTTP/3 Support
- Request Decompression
- gRPC - JSON Transcoding

Blazor

- Custom Elements
- Improvements on JavaScript Interop

C#11

- Required Members
- Generic Attributes
- Raw String Literals
- List Patterns

abp.io

What’s new with .NET 7.0?

Entity Framework Core

- JSON Columns
- Improvements on Bulk Updates & Deletes
- Performance Improvements on SaveChanges & SaveChangesAsync

MAUI

- Map Control
- Performance Improvements on Mobile Rendering & Desktop Enhancements

ASP.NET Core

- Rate Limiting
- Output Caching
- Built-in HTTP/3 Support
- Request Decompression
- gRPC - JSON Transcoding

Blazor

- Custom Elements
- Improved JavaScript Interop on WASM

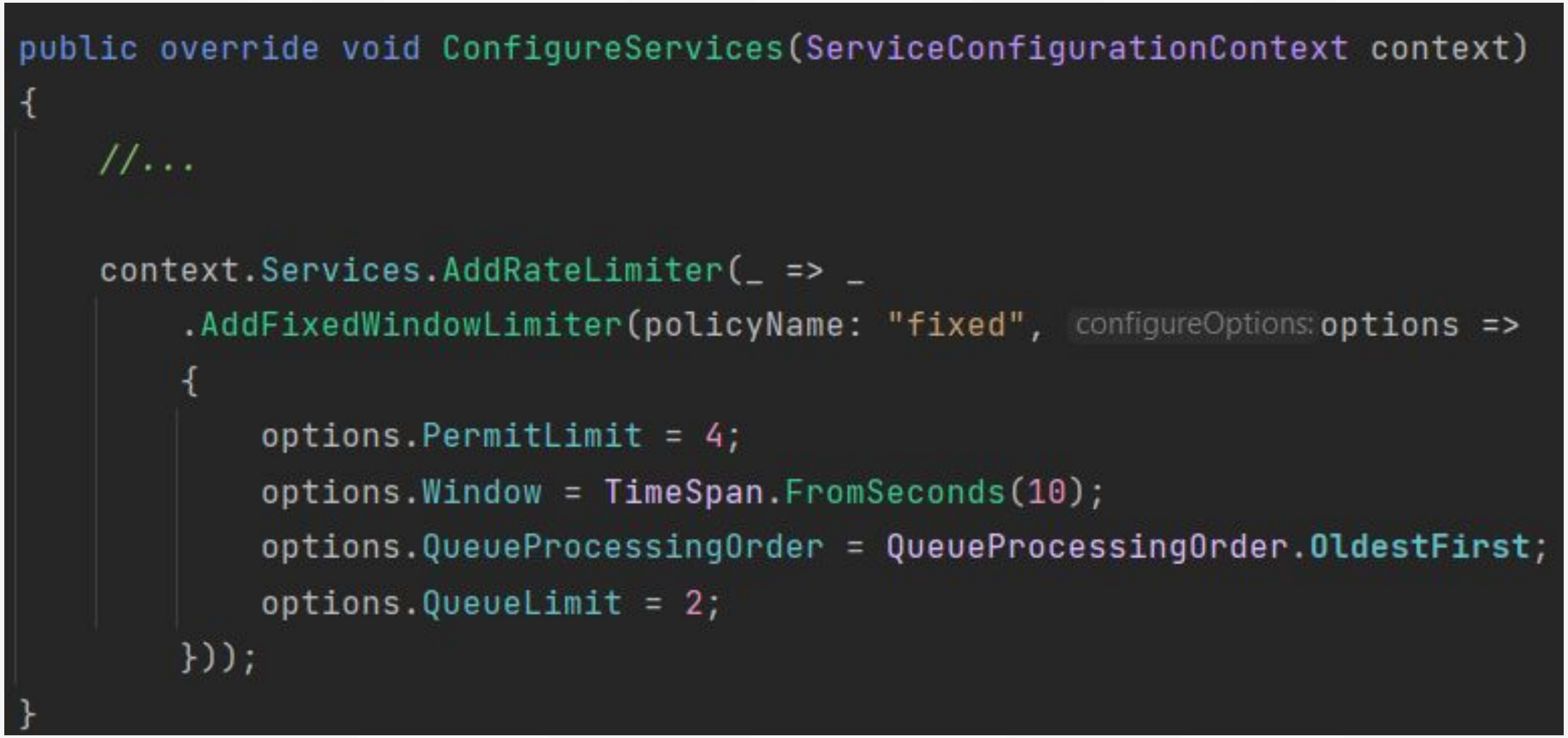
Rate Limiting

Rate Limiting is a way to prevent applications to get frequent requests and bottlenecks on the systems. Also, helpful to stop some common attacks such as DDos attacks by limiting the requests for a certain time.

.NET 7 introduces **built-in** Rate Limiting support.

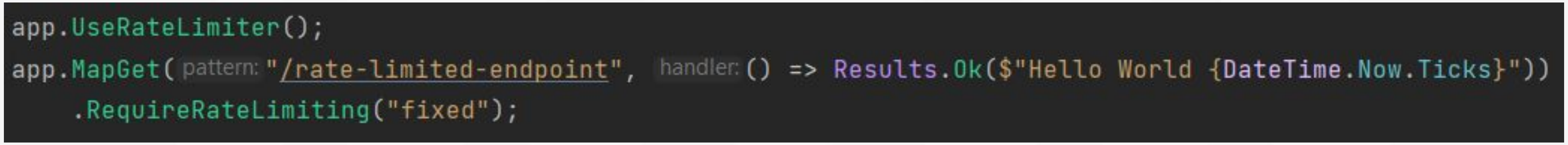
.NET applications can configure rate limiting policies and then attach these policies with their endpoints.

Rate Limiting

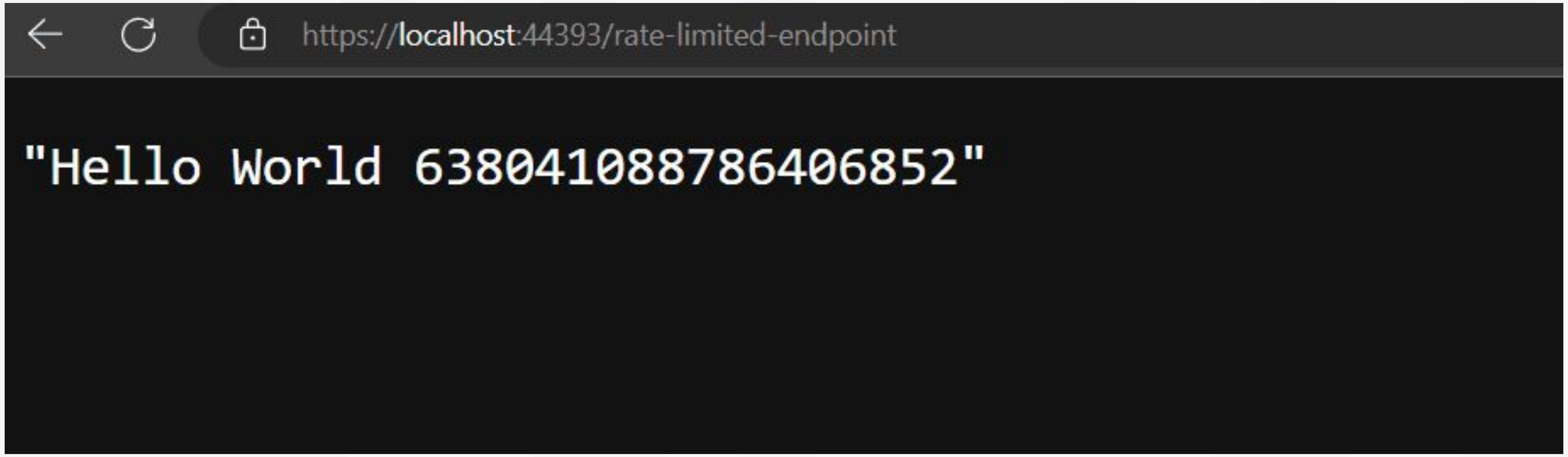


```
public override void ConfigureServices(ServiceConfigurationContext context)
{
    //...

    context.Services.AddRateLimiter(_ => _
        .AddFixedWindowLimiter(policyName: "fixed", configureOptions: options =>
        {
            options.PermitLimit = 4;
            options.Window = TimeSpan.FromSeconds(10);
            options.QueueProcessingOrder = QueueProcessingOrder.OldestFirst;
            options.QueueLimit = 2;
        }));
}
```

Rate Limiting



```
app.UseRateLimiter();
app.MapGet( pattern: "/rate-limited-endpoint",  handler: () => Results.Ok($"Hello World {DateTime.Now.Ticks}"))
    .RequireRateLimiting("fixed");
```



```
"Hello World 638041088786406852"
```

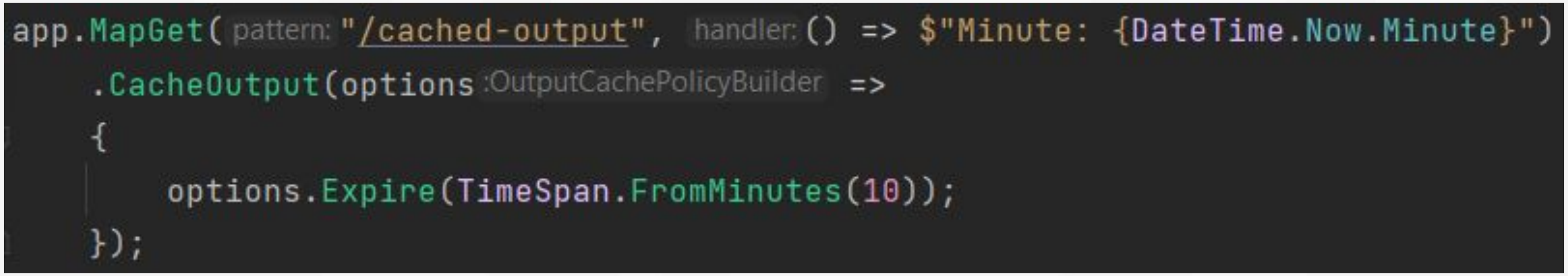
Output Caching Middleware

Output Caching is a new middleware that provides a caching mechanism and allow to store results from your web application and serve them from a cache rather than computing everytime. This improves performance and frees up resources for other tasks.

The output caching middleware can be used in all types of ASP.NET Core applications: Web APIs, MVC and Razor applications.

Output Caching Middleware

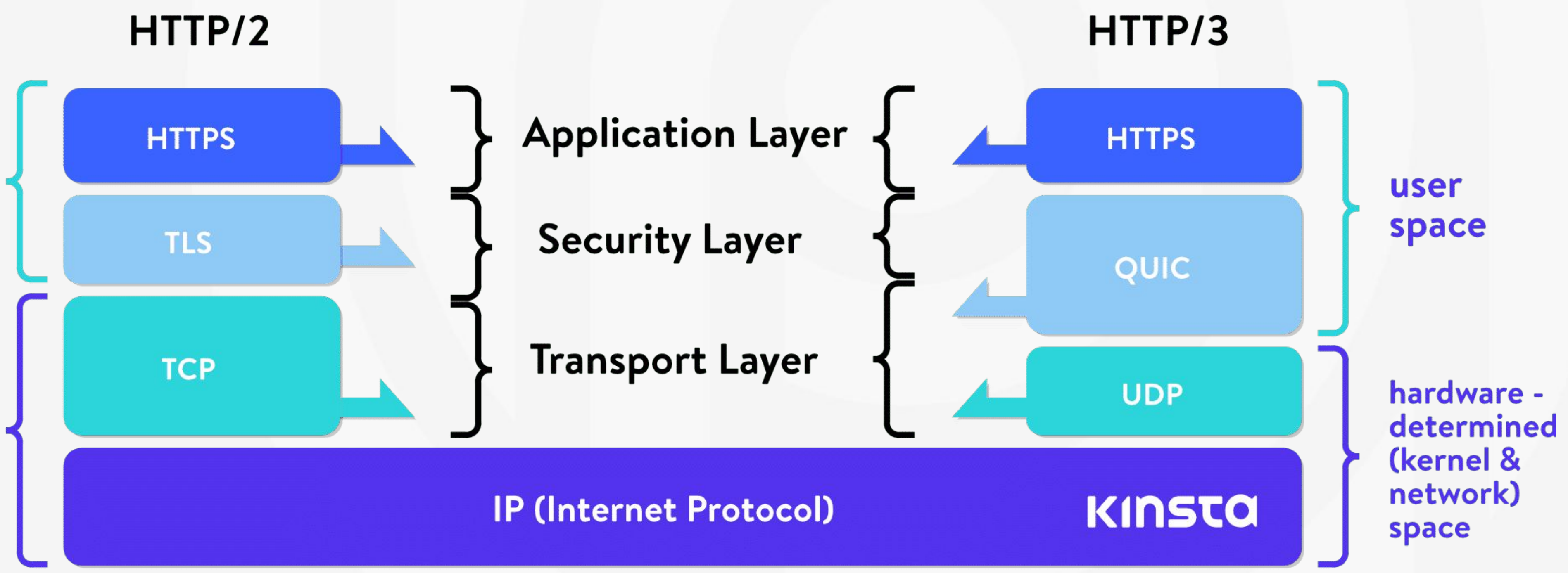
It’s pretty straightforward to use output caching in minimal APIs. You just need to create an endpoint and use the `CacheOutput` method with an expire time. Then when someone sends a request to your endpoint, it will be cached for specified time and not calculate the result every time.



```
app.MapGet( pattern: "/cached-output",  handler: () => $"Minute: {DateTime.Now.Minute}")
    .CacheOutput(options :OutputCachePolicyBuilder  =>
    {
        options.Expire(TimeSpan.FromMinutes(10));
    });
```

HTTP/3 Support

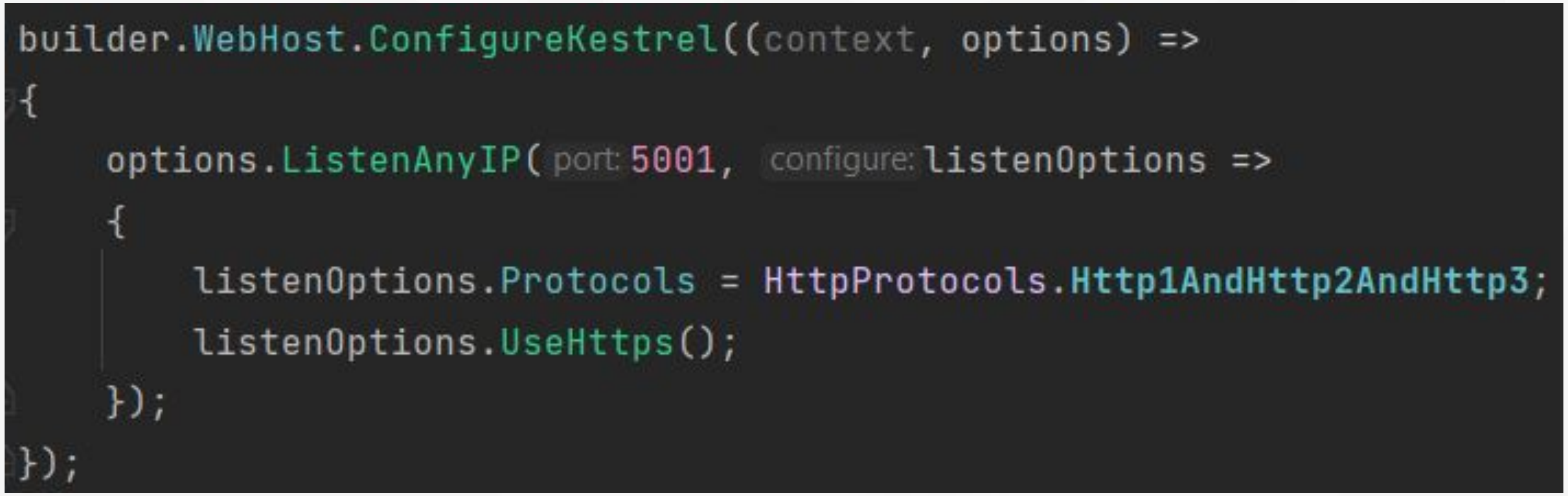
HTTP/3 is a new version of HTTP and supported by most modern browsers.



HTTP/3 Support

In .NET 6, HTTP/3 was introduced for an experimental purposes and with .NET 7 now it's fully supported.

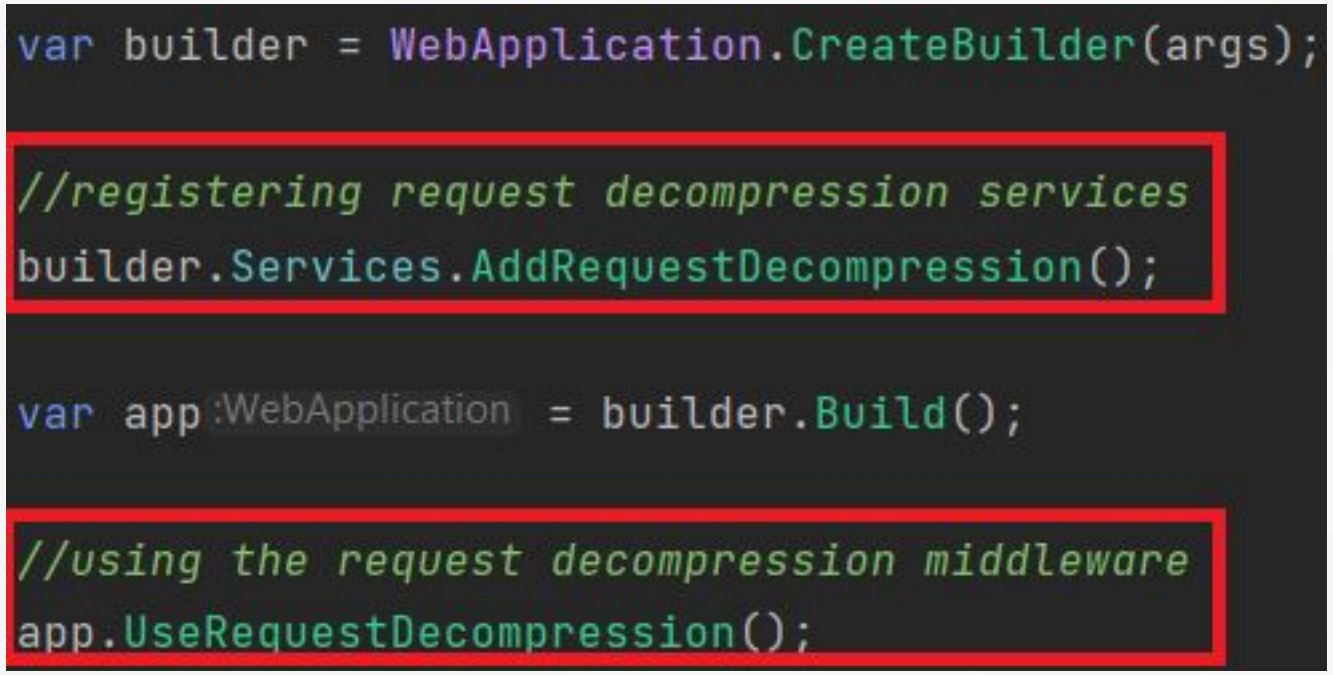
HTTP/3 is not enabled by default but it can be configured easily as follows:



```
builder.WebHost.ConfigureKestrel((context, options) =>
{
    options.ListenAnyIP( port: 5001,  configure: listenOptions =>
    {
        listenOptions.Protocols = HttpProtocols.Http1AndHttp2AndHttp3;
        listenOptions.UseHttps();
    });
});
```

Request Decompression

Request Decompression middleware accept request with **compressed content.** Uses the Content-Encoding HTTP Header to automatically identify and decompress requests which are compressed.



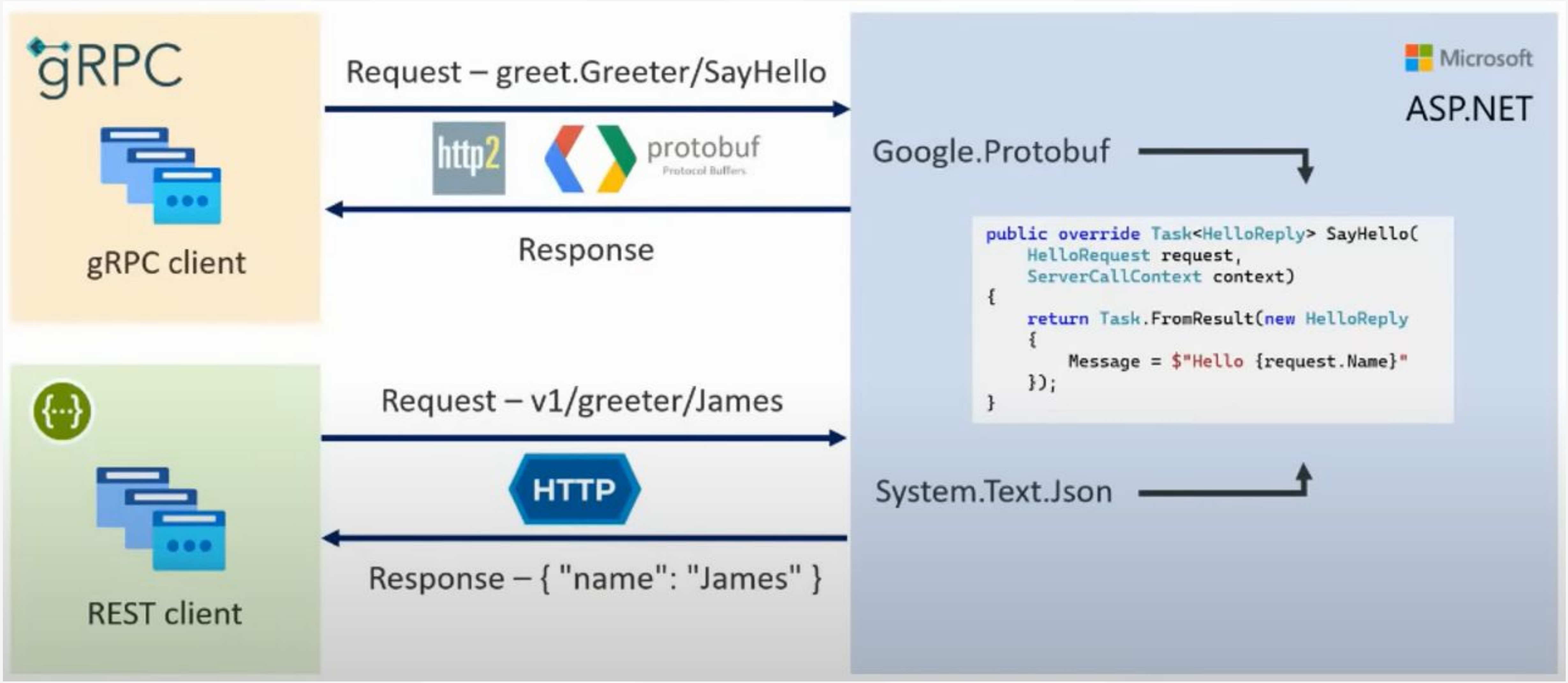
```
var builder = WebApplication.CreateBuilder(args);

//registering request decompression services
builder.Services.AddRequestDecompression();

var app :WebApplication = builder.Build();

//using the request decompression middleware
app.UseRequestDecompression();
```

gRPC - JSON Transcoding

gRPC is a high-performance RPC framework and uses HTTP/2 and Protobuf. Despite the benefits that gRPC brings, REST APIs have an important place in modern web applications.

gRPC - JSON Transcoding is an extension for ASP.NET Core that creates RESTful JSON APIs for gRPC services.

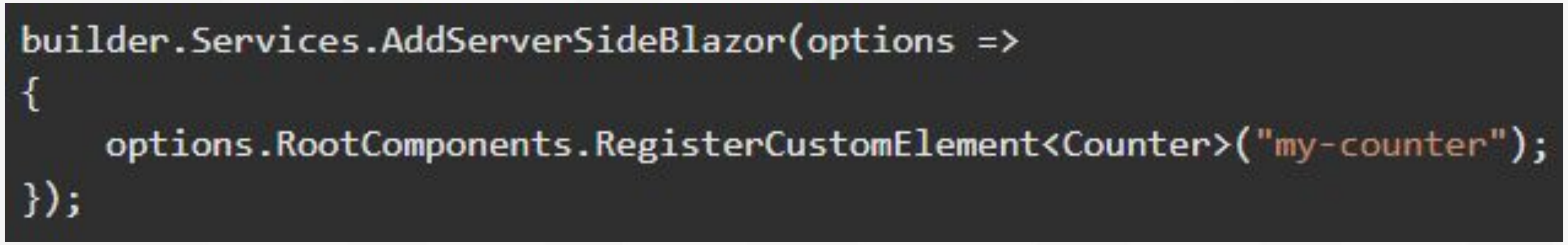
gRPC - JSON Transcoding



```
public override Task<HelloReply> SayHello(
    HelloRequest request,
    ServerCallContext context)
{
    return Task.FromResult(new HelloReply
    {
        Message = $"Hello {request.Name}"
    });
}
```

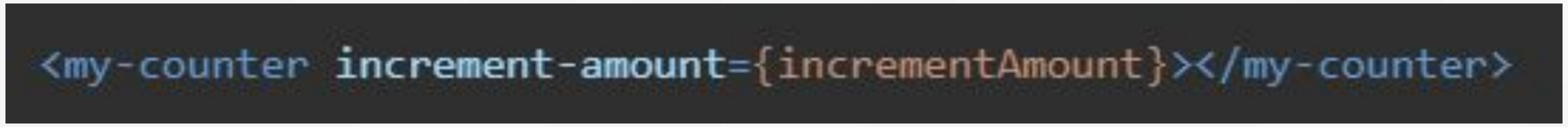
Blazor - Custom Elements

Registering Custom Elements



```
builder.Services.AddServerSideBlazor(options =>
{
    options.RootComponents.RegisterCustomElement<Counter>("my-counter");
});
```

Using the Custom Element



```
<my-counter increment-amount={incrementAmount}></my-counter>
```

Blazor - Improvements on JavaScript Interop

JavaScript **[JsImport] / [JsExport] interop** API released with .NET 7 and this allow us to interact with JavaScript in a Blazor WASM application easily.

- To import a JS function to call it from C# **->** **[JsImport]**
- To export a .NET method so that it can be called from JavaScript -> **[JsExport]** attributes should used.

C# 11

- Required Members
- Generic Attributes
- Raw String Literals
- List Patterns

Required Members

C#11 introduces the **required** keyword to ensure property initialization.

```
public class Person
{
    public required string FirstName { get; init; }
    public string? MiddleName { get; init; }
    public required string LastName { get; init; }
}
```

By using the **required** keyword, we can ensure the marked properties will be initialized. Otherwise, it will give compile-time errors.

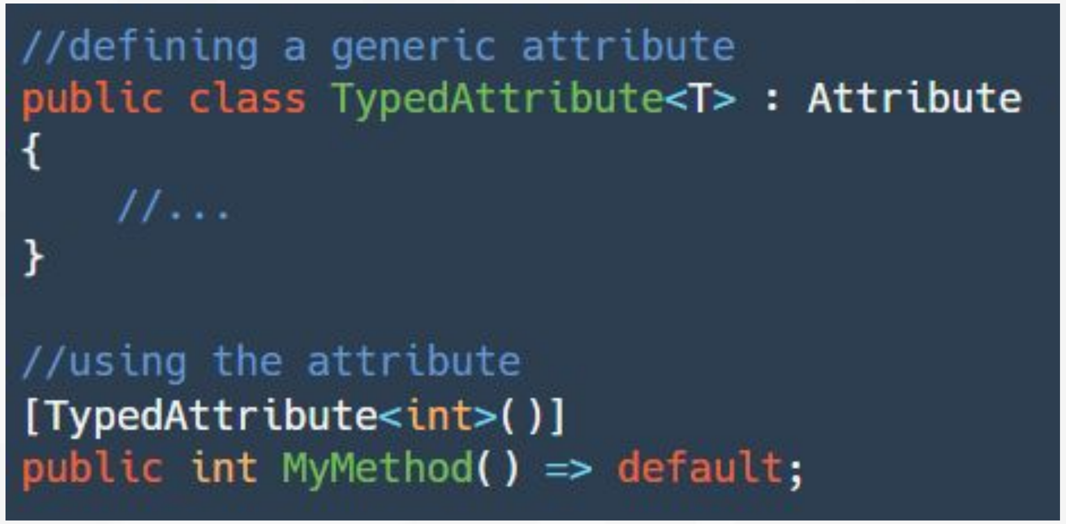
```
var person = new Person { FirstName = "Ada" }; // Error: no LastName!
```

abp.io

Generic Attributes

Before C#11, to create a typed attribute you would need to pass the **Type** object through the constructor and assign it to your property to use.

Now with C# 11, it’s possible to easily create generic attributes and use them:

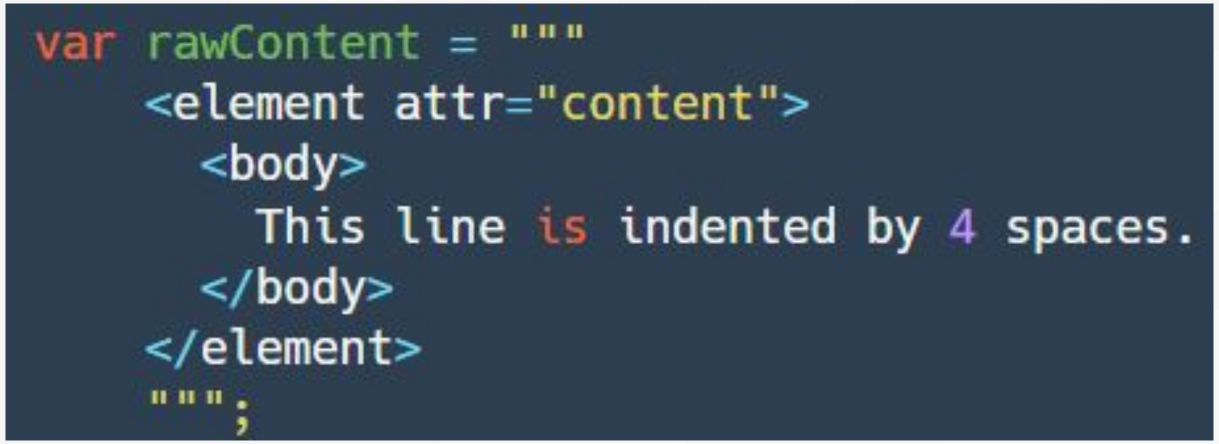


```
//defining a generic attribute
public class TypedAttribute<T> : Attribute
{
    //...
}

//using the attribute
[TypedAttribute<int>()]
public int MyMethod() => default;
```

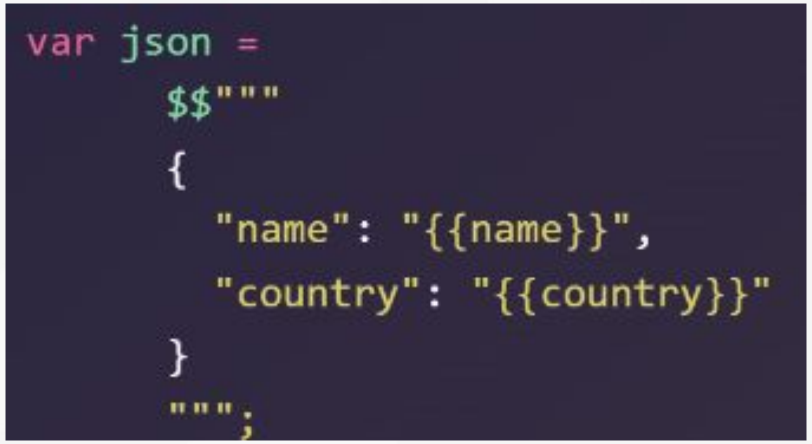
Raw String Literals

C# 11 introduces “**raw string literals”.** It allows containing arbitrary text without escaping.

By wrapping a string with three double quotes (”””...”””), we are free to put any string value into variables:



```
var rawContent = """
    <element attr="content">
      <body>
        This line is indented by 4 spaces.
      </body>
    </element>
    """;
```

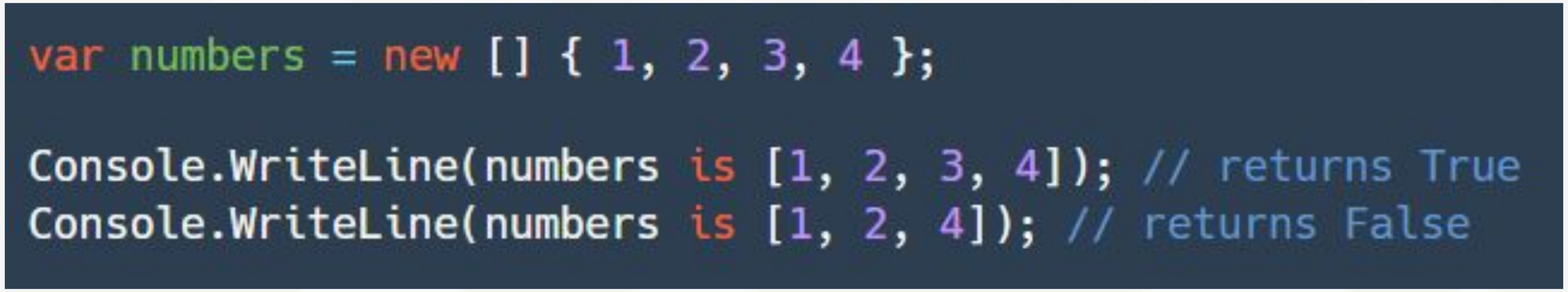


```
var json =
      $$"""
      {
        "name": "{{name}}",
        "country": "{{country}}"
      }
      """;
```

List Patterns

C# 11 introduced the “**List Pattern**”. It expands the pattern matching for lists and arrays.

You can match an array or a list against a sequence of patterns:

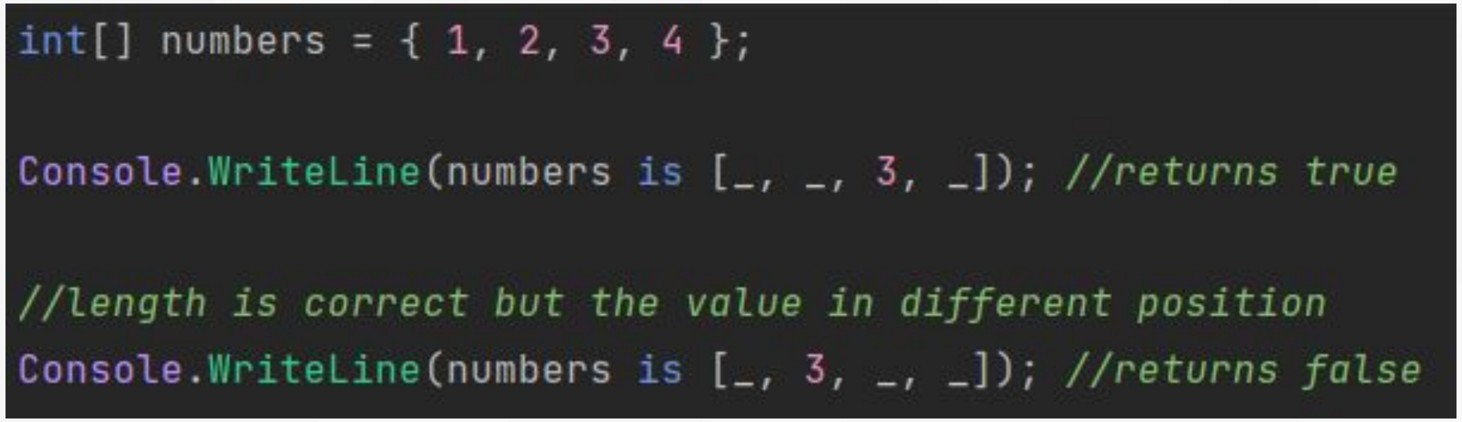


```
var numbers = new [] { 1, 2, 3, 4 };

Console.WriteLine(numbers is [1, 2, 3, 4]); // returns True
Console.WriteLine(numbers is [1, 2, 4]); // returns False
```

List Patterns

There are three different ways for list pattern matching:

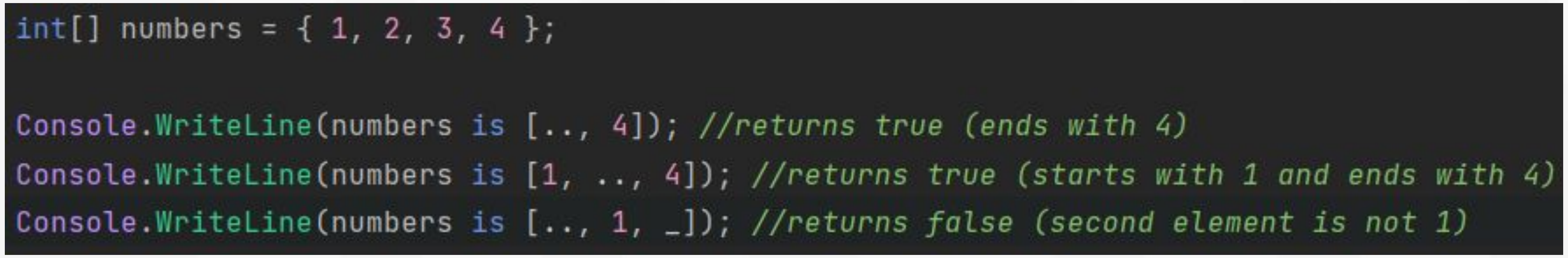
1-) Discard Pattern



```
int[] numbers = { 1, 2, 3, 4 };

Console.WriteLine(numbers is [_, _, 3, _]); //returns true

//length is correct but the value in different position
Console.WriteLine(numbers is [_, 3, _, _]); //returns false
```

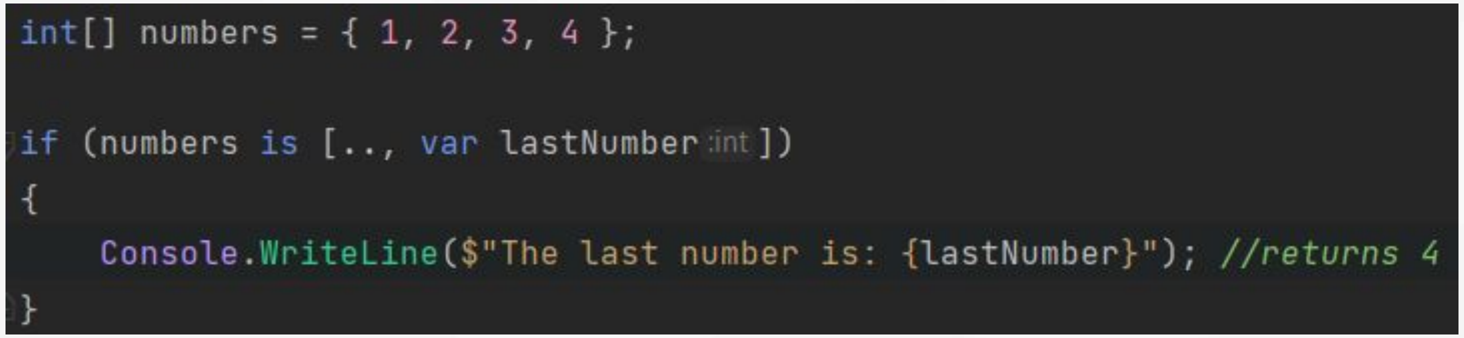
List Patterns

2-) Range Pattern



```
int[] numbers = { 1, 2, 3, 4 };

Console.WriteLine(numbers is [.., 4]); //returns true (ends with 4)
Console.WriteLine(numbers is [1, .., 4]); //returns true (starts with 1 and ends with 4)
Console.WriteLine(numbers is [.., 1, _]); //returns false (second element is not 1)
```

3-) var Pattern



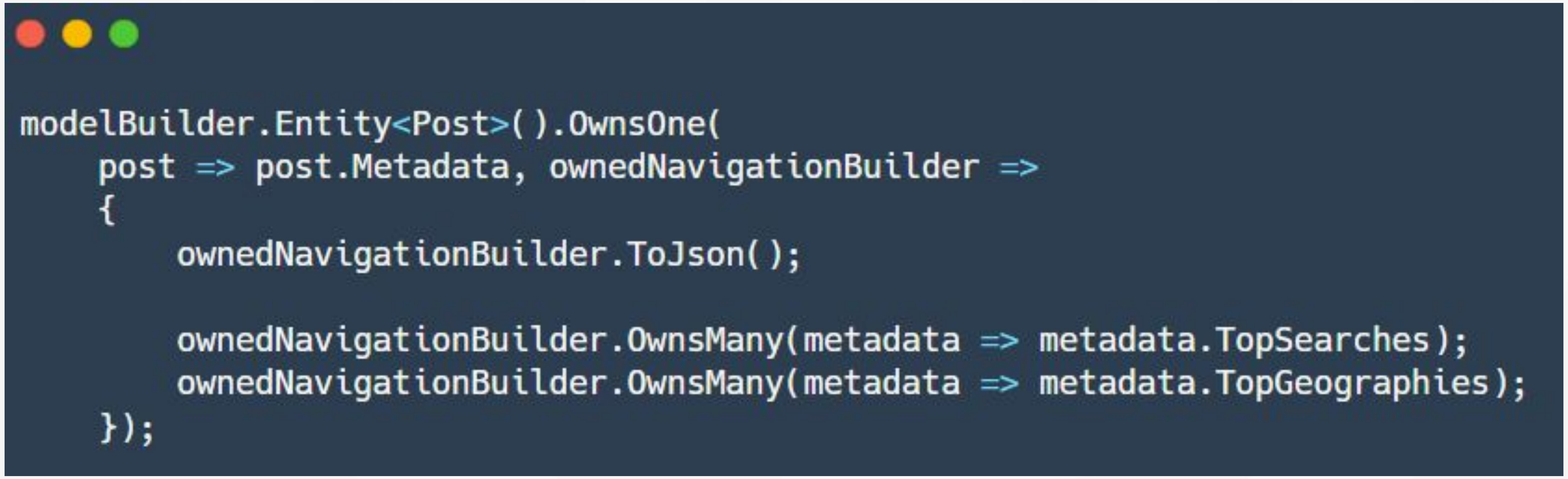
```
int[] numbers = { 1, 2, 3, 4 };

if (numbers is [.., var lastNumber:int ])
{
    Console.WriteLine($"The last number is: {lastNumber}"); //returns 4
}
```

Entity Framework Core

- JSON Columns
- Improvements on Bulk Updates & Deletes
- Performance Improvements on SaveChanges & SaveChangesAsync

JSON Columns

JSON Columns allows relational databases to use the advantage of document databases. EF7 supports JSON columns and this allows mapping of aggregates built from .NET types to JSON documents.



```
modelBuilder.Entity<Post>().OwnsOne(
    post => post.Metadata, ownedNavigationBuilder =>
    {
        ownedNavigationBuilder.ToJson();

        ownedNavigationBuilder.OwnsMany(metadata => metadata.TopSearches);
        ownedNavigationBuilder.OwnsMany(metadata => metadata.TopGeographies);
    });
```

JSON Columns

Also, it’s possible to query JSON by using the LINQ with this version as follows:

```
var postsWithViews = await context.Posts.Where(post => post.Metadata!.Views > 3000)
    .AsNoTracking()
    .Select(
        post => new
        {
            post.Author!.Name,
            post.Metadata!.Views,
            Searches = post.Metadata.TopSearches,
            Commits = post.Metadata.Updates
        })
    .ToListAsync();
```

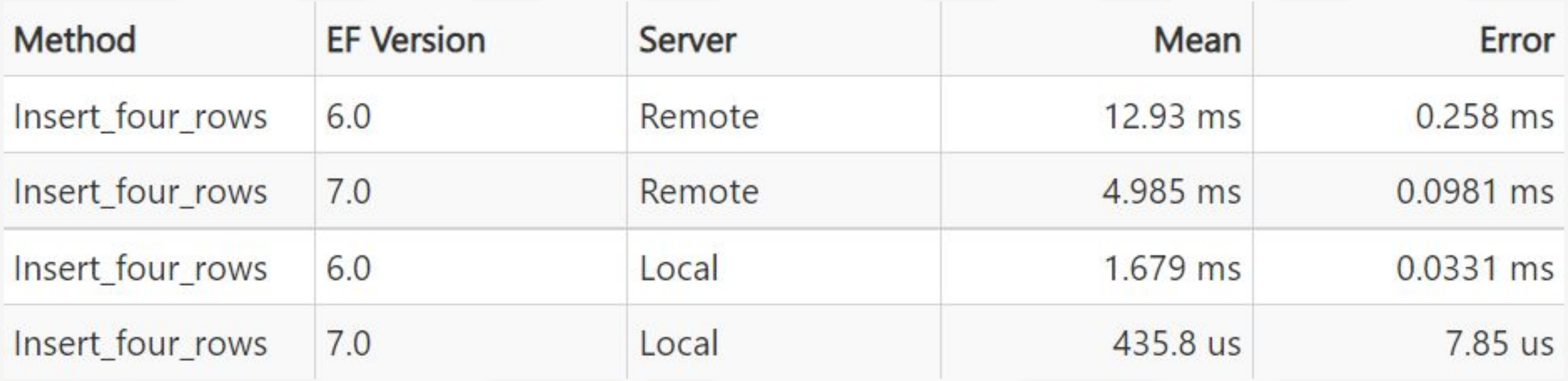
Improvements on Bulk Updates & Deletes

EF 7 introduces the new **ExecuteUpdateAsync** and **ExecuteDeleteAsync** methods. These methods are applied to a LINQ query and update/delete entities in the database immediately based on the results of that query without the need to track records.

With this feature, there is no need to load entities into memory and update them. Many entities can be updated with a single command without needing to load into memory.

Performance Improvements on SaveChanges & SaveChangesAsync

In EF7, the performance of **SaveChanges** and **SaveChangesAsync** has been significantly improved. According to the EF Team, saving changes is now **four times** faster than EF Core 6.



Method	EF Version	Server	Mean	Error
Insert_four_rows	6.0	Remote	12.93 ms	0.258 ms
Insert_four_rows	7.0	Remote	4.985 ms	0.0981 ms
Insert_four_rows	6.0	Local	1.679 ms	0.0331 ms
Insert_four_rows	7.0	Local	435.8 us	7.85 us

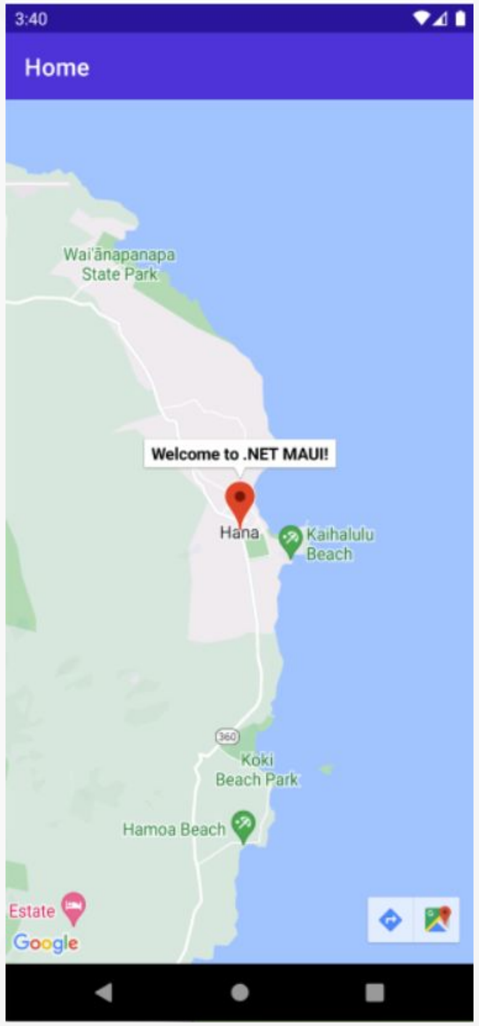
MAUI

- Map Control
- Performance Improvements on Mobile Rendering & Desktop Enhancements

Map Control

.NET MAUI 7 introduces **Map Control**. This provides us a native map control provided by each platform.

It supports pins, polygons, polylines, circles, geolocation and more…



Improvements on Mobile Rendering & Desktop Enhancements

.NET MAUI 7 came with an optimized rendering for mobile applications and is much faster than .NET MAUI 6.

In addition to that, there are some enhancements and improvements on the desktop:

- Window size and position
- Context menus
- Tooltips
- Gestures etc.

abp.io

Thanks for listening…