What's new in .NET 8 & C# 12

Filip Ekberg

Hello,

I'm Filip Ekberg

filip@ekberg.dev

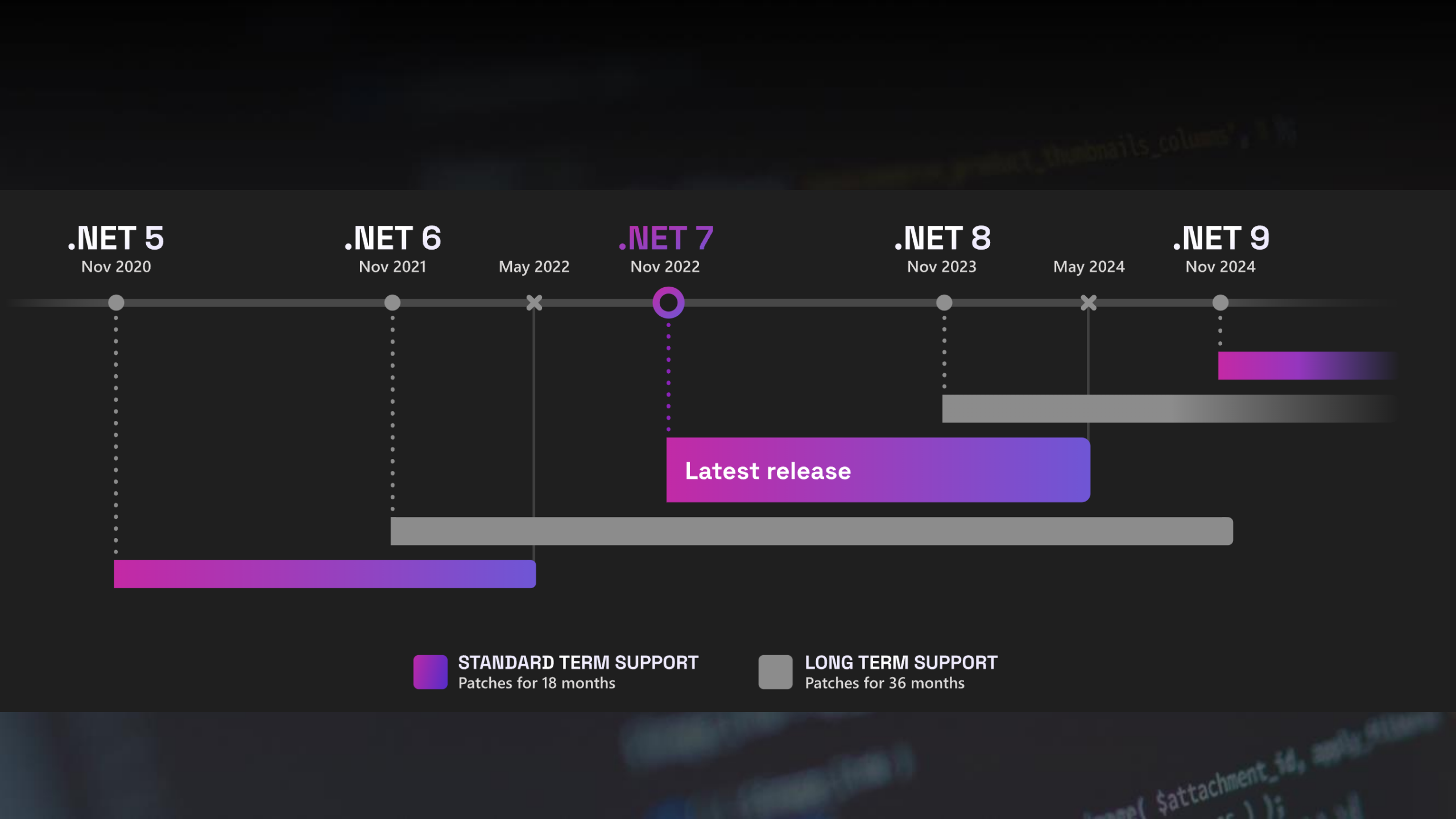
C# smorgasbord free @ filipekberg.se

PLURALSIGHT

fekberg

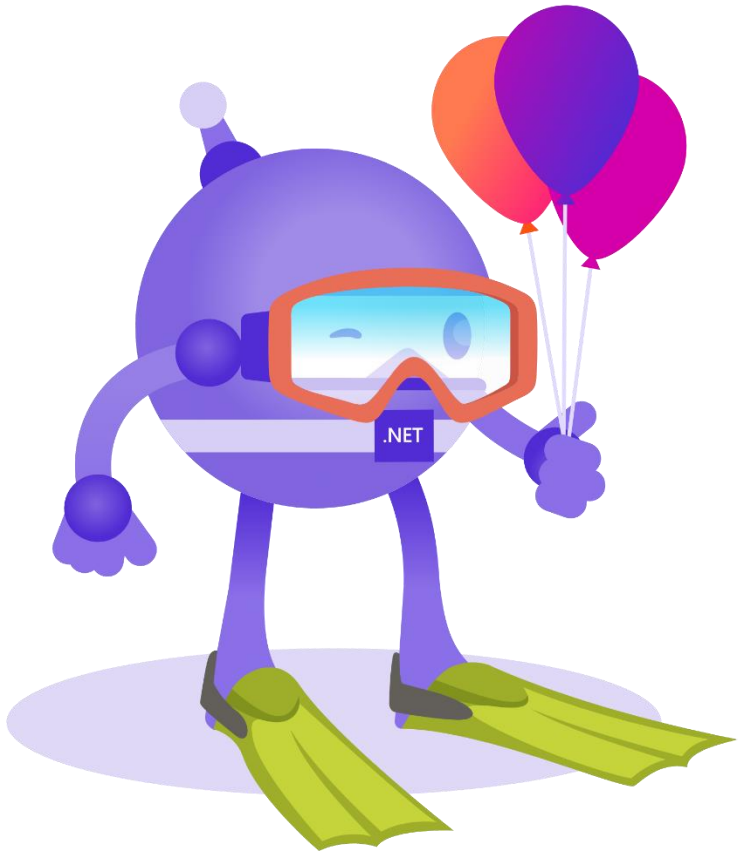

Using Statements for Static Members
Implicitly typed local variables
Extension methods
Getter & Setter separate accessibility
Null-conditional operators
Exception Filters
Anonymous methods
Named and optional parameters
Lambdas
Static classes
Auto-properties
stackalloc initializers
String interpolation
Caller info attributes
Nullable types
Digit Separators
private protected
stackalloc with Span
Dynamic binding
Default Expressions
digit separators
non-trailing named arguments
out improvements
Generics
Expression trees
Dictionary Initializers
Local Functions
Tuples and Deconstruction
Ref Assemblies
Async & Await
Infer Tuple Names
ref returns
Await inside Catch & Finally blocks
Query expressions
Iterators
nameof operator
Generic co- and contravariance
Partial types
Improved generic constraints
Pattern Matching
Object and collection initializers
conditional ref operator
Async Main
Tuple equality
Binary Literals
Expression-bodied members
Auto-Property Initializers
Anonymous types
ref readonly & in parameter

.NET 8


.NET 5
Nov 2020
.NET 6
Nov 2021
May 2022
.NET 7
Nov 2022
.NET 8
Nov 2023
May 2024
.NET 9
Nov 2024
Latest release
STANDARD TERM SUPPORT
Patches for 18 months
LONG TERM SUPPORT
Patches for 36 months

“Build anything”

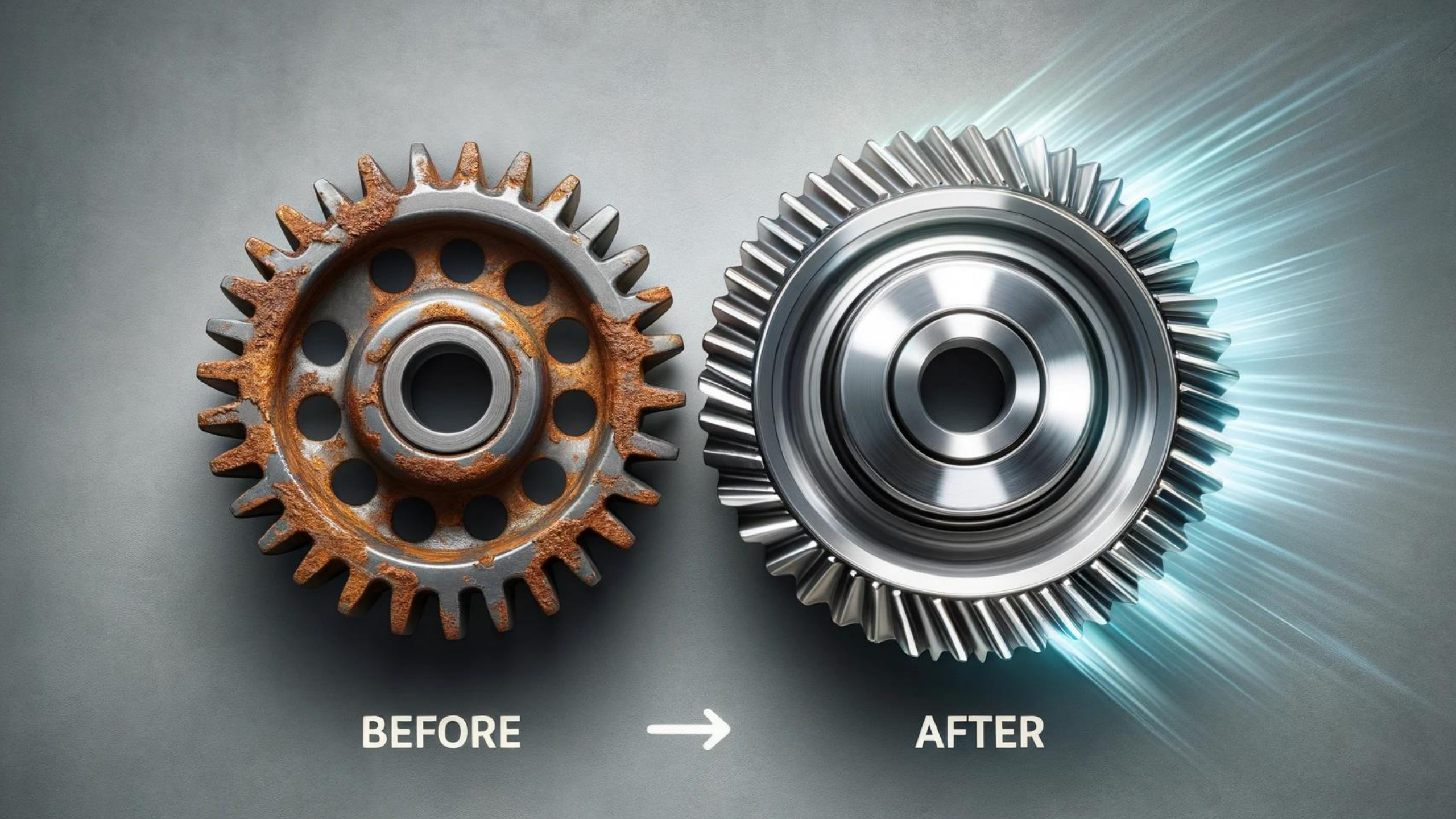
.NET 8



- Performance, performance, performance!
- Time abstraction
- System.Text.Json
- ASP.NET Core
- .NET MAUI
- Blazor

.NET + C#

- .NET Core 3.0 for C# 8
- .NET 5.0 for C# 9
- .NET 6.0 for C# 10
- .NET 7.0 for C# 11
- .NET 8.0 for C# 12


BEFORE
AFTER

Spans + Ranges = Performance

Range

"Ranges and indices provide a succinct syntax for accessing single elements or ranges in a sequence."

```
[2..]    [..^1]    [^2..]
```

Zero Allocations

```
byte[] payload = [0x1, 0xf1, 0xaa, 0xf2];

Span<byte> payloadSpan = payload;

Span<byte> subset = payloadSpan[^2..];
```

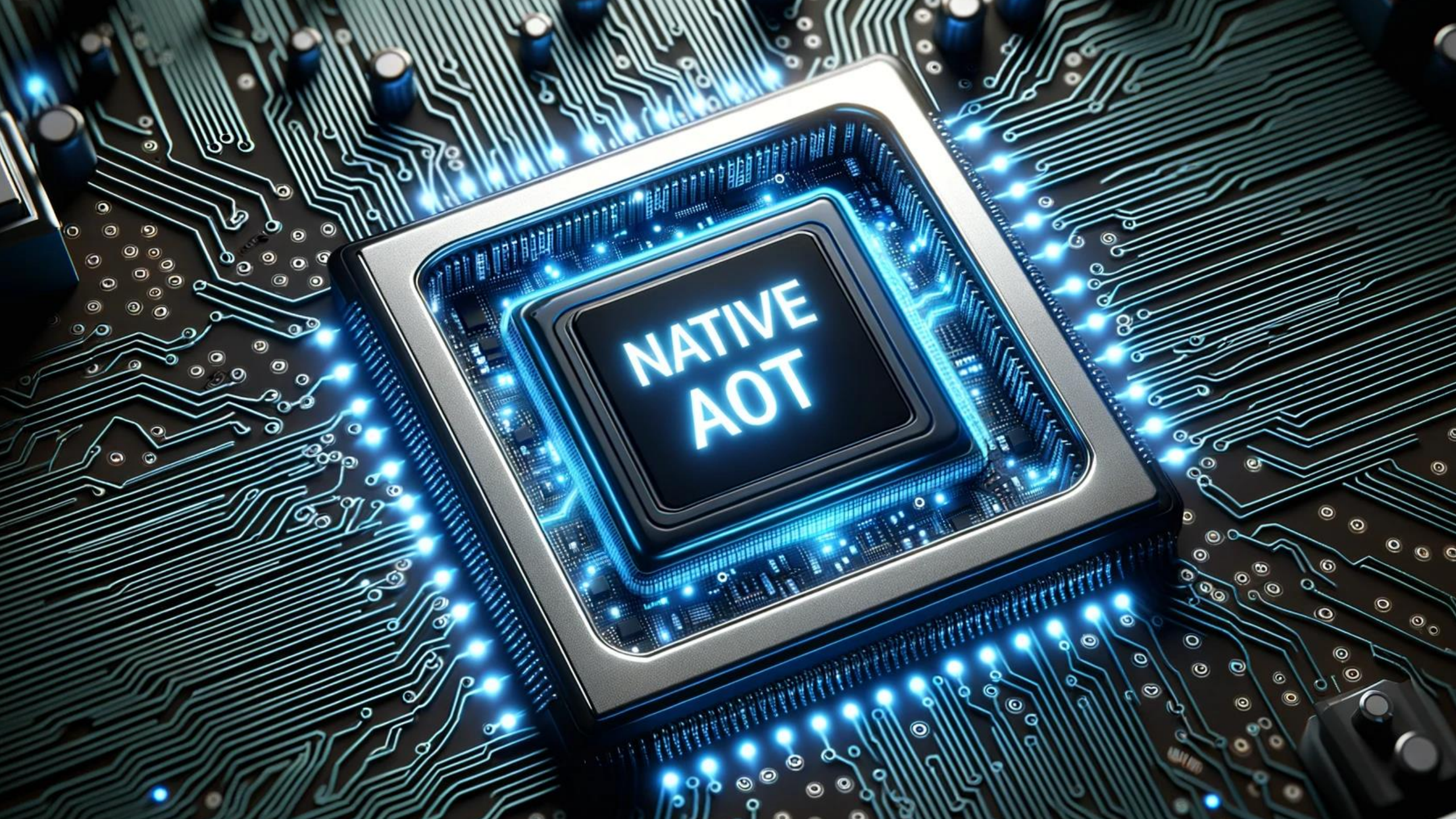
Zero Allocations

```
ReadOnlySpan<byte> payload = "Filip Ekberg"u8;

int space = payload.IndexOf(" "u8);

ReadOnlySpan<byte> first = payload[..space];

ReadOnlySpan<byte> second = payload[++space..];
```


NATIVE
AOT

Performance-focused types

```
System.Collections.Frozen.
FrozenDictionary.ToFrozenDictionary(source);
```

Serialization

Time abstraction

How'd you test this?

```
class OrderService
{
    public bool HasPaymentExpired(Order order)
    {
        return (DateTimeOffset.UtcNow - order.PaymentReservedOn).TotalDays > 30;
    }
}
```

Randomness

```
var randomItems = Random.Shared.GetItems([1, 2, 3, 4, 5, 6, 7], 3);
```

ASP.NET Core

Short circuit routes

```
app.MapGet("/short-circuit", () => "I'm executed immediately!")
   .ShortCircuit();
```

Keyed Services

```
builder.Services.AddKeyedTransient<ICache, DistributedCache>("distributed");
builder.Services.AddKeyedTransient<ICache, InMemoryCache>("memory");

[FromKeyedServices("memory")]ICache cache
```

Complex Binding in Minimal APIs

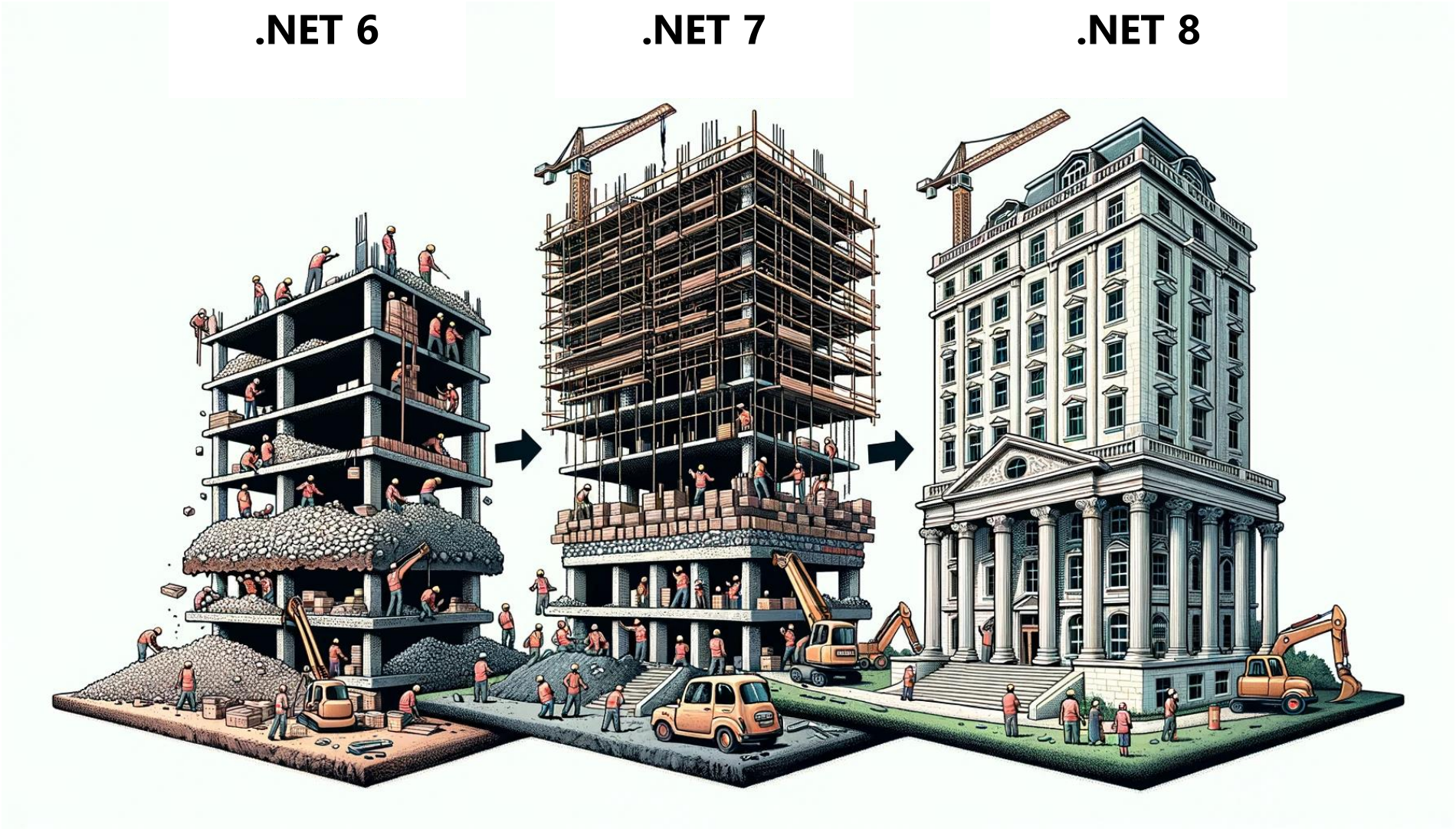
```
app.MapPost("/upload", async (
    IFormFile data,
    HttpContext context,
    IAntiforgery antiforgery) =>
{
    await antiforgery.ValidateRequestAsync(context);
});
```

Blazor

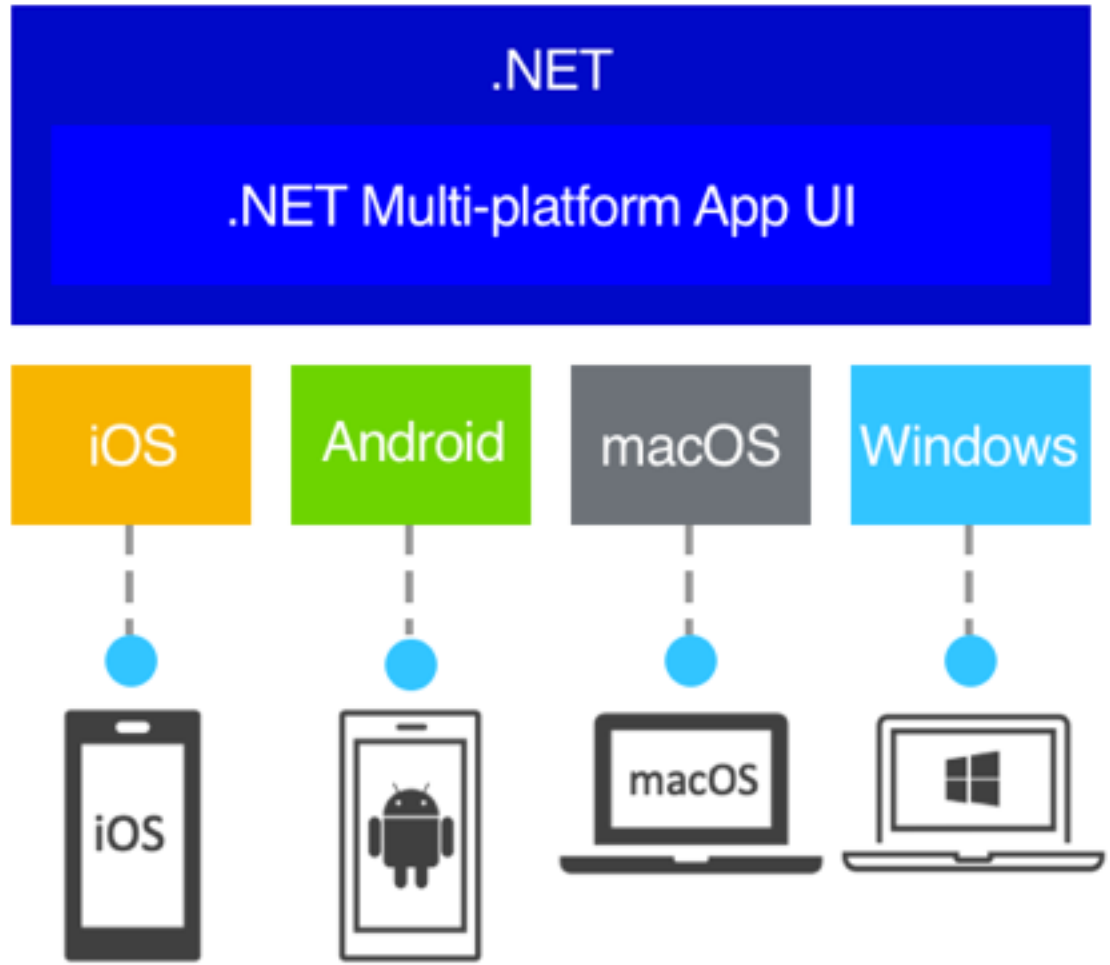
”With the release of .NET 8, Blazor is a full-stack web UI framework for developing apps that render content at either the component or page”

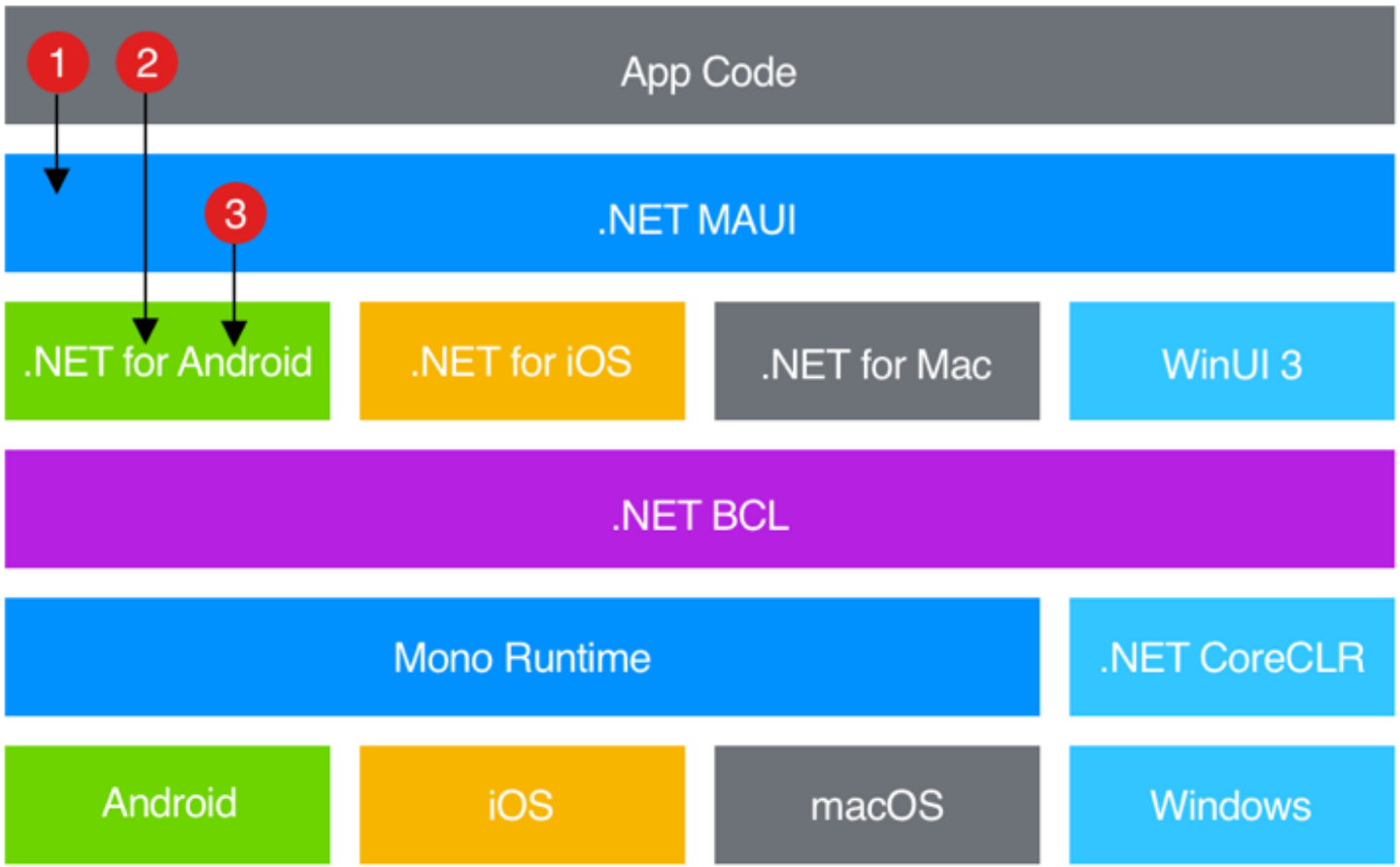
.NET MAUI

.NET MAUI




.NET MAUI
.NET 7
.NET 8


.NET
.NET Multi-platform App UI
iOS
Android
macOS
Windows
iOS
macOS



https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui


Using Statements for Static Members
Implicitly typed local variables
Extension methods
Getter & Setter separate accessibility
Null-conditional operators
Exception Filters
Anonymous methods
Named and optional parameters
Lambdas
Static classes
Auto-properties
stackalloc initializers
String interpolation
Caller info attributes
Nullable types
Digit Separators
private protected
stackalloc with Span
Dynamic binding
Default Expressions
digit separators
non-trailing named arguments
out improvements
Generics
Expression trees
Dictionary Initializers
Local Functions
Tuples and Deconstruction
Ref Assemblies
Async & Await
Infer Tuple Names
ref returns
Await inside Catch & Finally blocks
Query expressions
Iterators
nameof operator
Generic co- and contravariance
Partial types
Improved generic constraints
Pattern Matching
Object and collection initializers
conditional ref operator
Async Main
Tuple equality
Binary Literals
Expression-bodied members
Auto-Property Initializers
Anonymous types
ref readonly & in parameter

C# 12

C# 12

- Primary Constructors
- Collection expressions & spread operator
- Interceptors
- Optional parameters in lambdas
- Alias any type
- Inline arrays
- ref readonly parameters
- Experimental attribute

Collection expressions

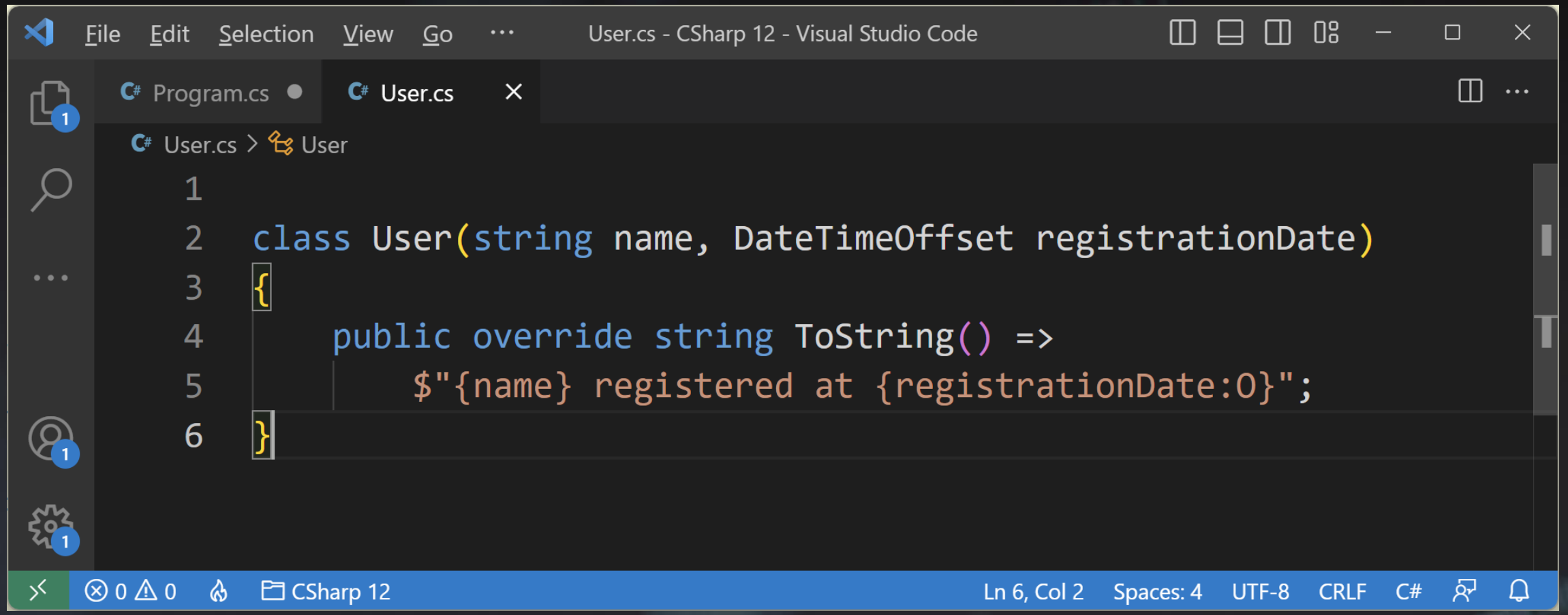
```
Span<byte> payload  = [0x1, 0xf1, 0xaa, 0xf2];
Span<byte> checksum = [0xff, 0xab];

Span<byte> result = [..payload, ..checksum];
```

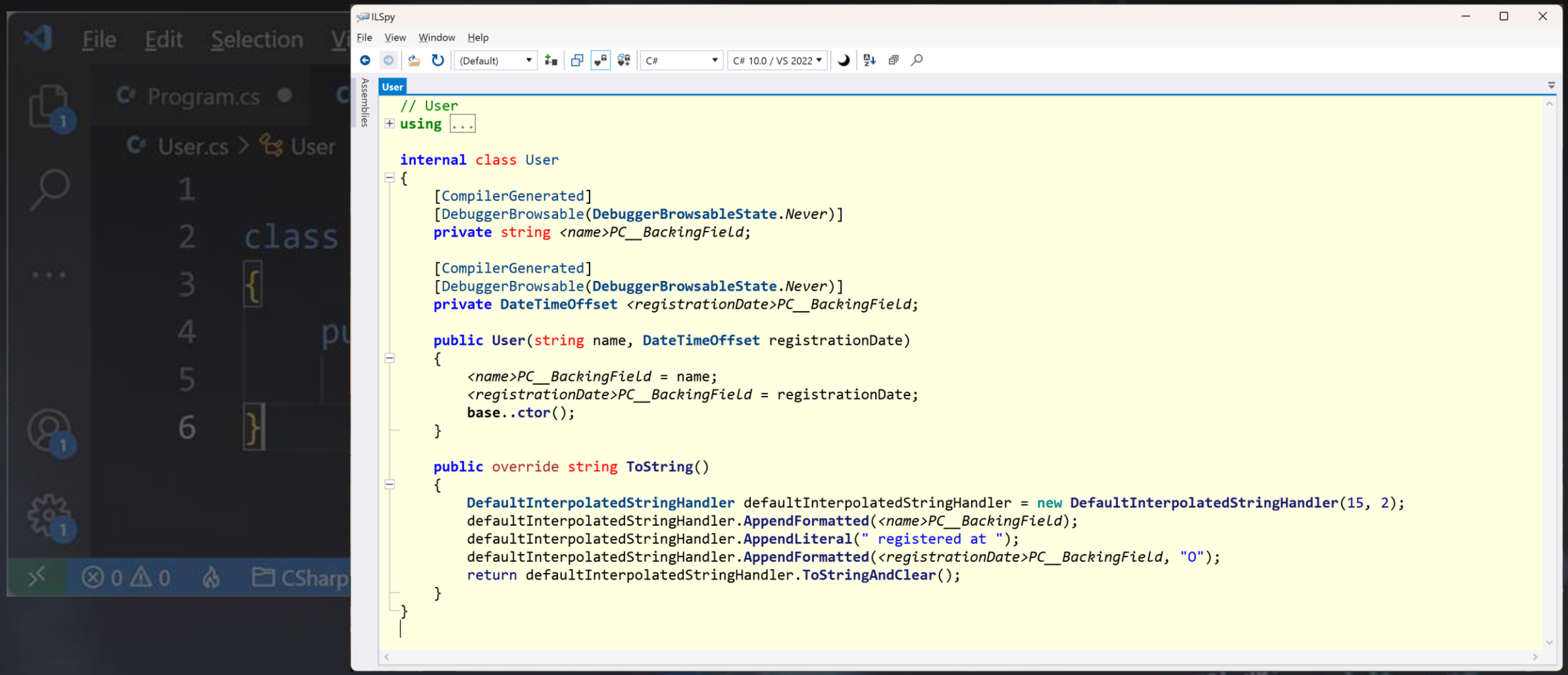
Primary constructors

```
class OrderController(IUserRepository repository)
{
    public (Guid, decimal) GetTotalFor(Guid orderId)
    {
        return (orderId, repository.Sum(orderId));
    }
}
```

Primary constructors



Primary constructors



Optional parameters in lambda expressions

A.K.A. Default lambda parameters

```
var process = (string name = "Default") => {
    Console.WriteLine($"Processing: {name}");
};
```

Optional parameters in lambda expressions

A.K.A. Default lambda parameters

```
var process = (string name = "Default") => {
    Console.WriteLine($"Processing: {name}");
};

process("Filip Ekberg");

process();
```

Optional parameters in lambda expressions

A.K.A. Default lambda parameters

```
var process = (string name = "Default") => {
    Console.WriteLine($"Processing: {name}");
};

process("Filip Ekberg");

process();
```

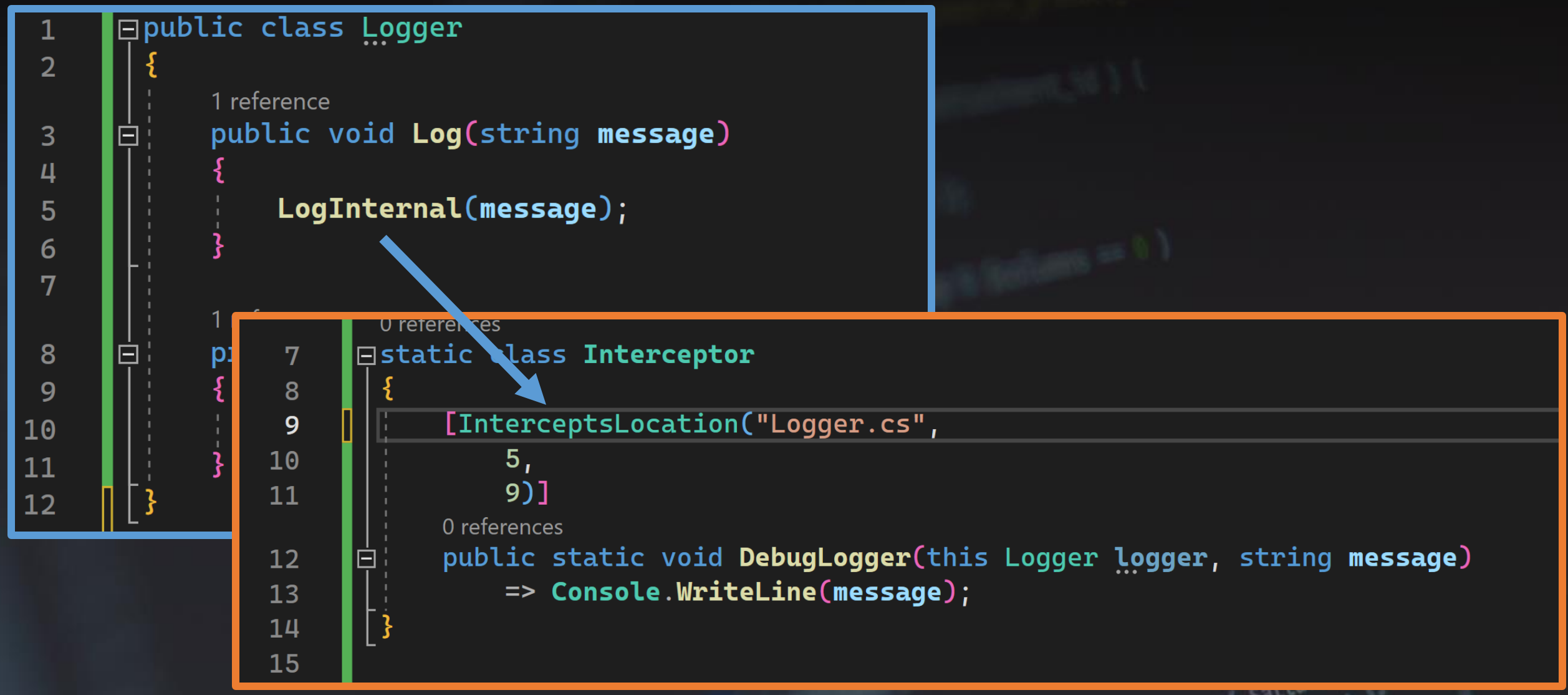
```
Processing: Filip Ekberg
Processing: Default
```

Alias any type

```
using Point = (int x, int y);

void Refresh(Point p)
{
    Console.WriteLine(p.x);
    Console.WriteLine(p.y);
}
```

Interceptors



Interceptors in ASP.NET Core

Improved performance using Interceptors

The Request Delegate Generator uses the new C# 12 interceptors compiler feature to support intercepting calls to minimal API's Map action methods with statically generated variants at runtime. The use of interceptors results in increased startup performance for apps compiled with `PublishAot`.

learn.microsoft.com/en-us/aspnet/core/release-notes/aspnetcore-8.0

What's next?

github.com/dotnet/roslyn/blob/main/docs/Language%20Feature%20Status.md

Filip Ekberg

Pluralsight Author

Following

2807 Followers

Filip is an enthusiastic developer that strives to learn something new every day. With over a decade of experience in .NET, Filip actively spreads his knowledge and ideas around the globe, be it speaking at conferences or online. Filip has worked in a range of different technologies such as WPF,...

Show more...

- fekberg.com
- Twitter
- Facebook
- LinkedIn

CONTENT AUTHORED

24

All time

TOPICS AUTHORED

C#

TOTAL RATINGS

4,531

AVG CONTENT RATING

4.5

Content authored

Building a Real-world C# 10 Application NEW!

Course · Intermediate · 3 hr 39m · Jun 28, 2023 · ★★★★½ (16)

Globalization and Internationalization in .NET 6 NEW!

Course · Intermediate · 2 hr 36m · Apr 21, 2023 · ★★★★★ (10)

Date and Time in .NET 6

Course · Intermediate · 2 hr 4m · Feb 24, 2023 · ★★★★★ (20)

Asynchronous Programming in C# 10

Course · Intermediate · 5 hr 1m · Oct 13, 2022 · ★★★★★ (86)

Data Access in C# 10 Fundamentals

Course · Intermediate · 4 hr 18m · Aug 15, 2022 · ★★★★½ (46)

C# 10 Advanced Language Features

Course · Intermediate · 6 hr 50m · Mar 31, 2022 · ★★★★★ (132)

Thanks!

I'm Filip Ekberg

@fekberg

filip@ekberg.dev

C# smorgasbord free @ filipekberg.se

PLURALSIGHT

fekberg