



# Asynchronous programming in .NET

by Gerardo Lijs

Twitter @GerardoLijs

C# MeetUp Barcelona

April 2019

# Content

What is Asynchronous Programming

CPU-bound vs IO-bound

Obsolete patterns

Tasks and async/await

Practical use cases

- Responsive user interfaces (WPF, WinForms, etc)
- Scalability and performance in web applications

# CPU-bound operations

## Synchronous method

```
int GetPrimesCount(int start, int count)
{
    return ParallelEnumerable.Range(start, count).Count(n =>
        Enumerable.Range(2, (int)Math.Sqrt(n) - 1).All(i => n % i > 0));
}
```

## Asynchronous method

```
Task<int> GetPrimesCountAsync(int start, int count)
{
    return Task.Run(() =>
        ParallelEnumerable.Range(start, count).Count(n =>
            Enumerable.Range(2, (int)Math.Sqrt(n) - 1).All(i => n % i > 0)));
}
```

- Use Task.Run to execute expensive CPU-bound code in the ThreadPool
- Define a Task returning method
- In general you will not need other methods of TaskFactory class

# IO-bound operations

- Hard drive
- Network
- Database
- API calls
- Hardware read/write (serial port, camera, sensor, etc)

Unless you work in very specific industries, 95% or more of the code you write will never be CPU-bound. Most of the time you will need asynchronous to deal with IO-bound operations and the times you will use CPU-bound operations you will probably use a library that provides asynchronous methods for you to interact with.



# Responsive user interfaces

## Demo of a WPF desktop application

- Targets .NET Core 3 Preview4 but works exactly the same with .NET 4.7.2 or earlier
- The behaviour is the same in WinForms

You will need <https://dotnet.microsoft.com/download/dotnet-core/3.0>

and VS 2019 enabled for .NET Core

<https://visualstudiomagazine.com/articles/2019/03/08/vs-2019-core-tip.aspx>

# Responsive user interfaces

The screenshot shows a Windows desktop with two windows. On the left is a window titled 'MainWindow (Not Responding)' with a red close button. It contains two input fields: 'Start number' with the value '1' and 'End number' with the value '50000000'. Below these is a button labeled 'Calculate prime numbers'. On the right is the Windows Task Manager window, which is open to the 'Performance' tab. The 'Processes' tab is also visible. The 'Performance' tab shows a table of system resources: CPU at 100%, Memory at 33%, Disk at 0%, Network at 0%, GPU at 8%, and GPU usage is also shown. The 'Processes' tab shows a list of running applications. The 'AsyncDesktopDemo (32 bit)' application is highlighted in orange and has a status of 'Not responding'. Other applications listed include 'Microsoft Visual Studio 2017 (32 bit) (4)', 'Windows Explorer (5)', and 'Task Manager'. The 'Background processes (81)' section is also visible. At the bottom of the Task Manager window, there is a 'Fewer details' button and an 'End task' button.

Name	Status	CPU	Memory	Disk	Network	GPU
<b>Apps (4)</b>						
> AsyncDesktopDemo (32 bit)	Not responding	96.8%	12.0 MB	0 MB/s	0 Mbps	0%
> Microsoft Visual Studio 2017 (32 bit) (4)		0.2%	368.7 MB	0 MB/s	0 Mbps	0%
> Windows Explorer (5)		0%	85.9 MB	0 MB/s	0 Mbps	0%
> Task Manager		0.2%	23.1 MB	0 MB/s	0 Mbps	0%
<b>Background processes (81)</b>						

This is bad! Application is not responding

# Responsive user interfaces

The screenshot displays two windows side-by-side. On the left is a window titled 'MainWindow' with a 'Calculate prime numbers' button. On the right is the Windows Task Manager 'Performance' tab, showing 100% CPU usage. The application window remains responsive despite the high system load.

**MainWindow**

Start number  
1

End number  
50000000

Calculate prime numbers

**Task Manager - Performance**

Name	Status	100% CPU	31% Memory	2% Disk	0% Network	2% GPU	GPU
<b>Apps (5)</b>							
> AsyncDesktopDemo (32 bit)		99.4%	12.1 MB	0 MB/s	0 Mbps	0%	
> Microsoft Visual Studio 2017 (32 bit) (11)		0%	334.8 MB	0 MB/s	0 Mbps	0%	
> Snipping Tool		0%	9.6 MB	0 MB/s	0 Mbps	0%	
> Windows Explorer (5)		0%	87.1 MB	0 MB/s	0 Mbps	0%	
> Task Manager		0.2%	23.2 MB	0 MB/s	0 Mbps	0%	

Fewer details

End task

This is good! Application still responsive with 100% CPU use

# Responsive user interfaces

## Sync code - Before

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    int.TryParse(StartNumberTextBox.Text, out int start);
    int.TryParse(EndNumberTextBox.Text, out int end);

    ResultTextBlock.Text = "";
    int result = GetPrimesCount(start, end);
    ResultTextBlock.Text = $"{result} prime numbers between {start} and {end}";
}

private int GetPrimesCount(int start, int count)
{
    return ParallelEnumerable.Range(start, count).Count(n => Enumerable.Range(2, (int)Math.Sqrt(n) - 1).All(i => n % i > 0));
}
```

## Async code - After

```
private async void Button_Click(object sender, RoutedEventArgs e)
{
    int.TryParse(StartNumberTextBox.Text, out int start);
    int.TryParse(EndNumberTextBox.Text, out int end);

    ResultTextBlock.Text = "";
    int result = await GetPrimesCount(start, end);
    ResultTextBlock.Text = $"{result} prime numbers between {start} and {end}";
}

private Task<int> GetPrimesCount(int start, int count)
{
    return Task.Run(() =>
        ParallelEnumerable.Range(start, count).Count(n => Enumerable.Range(2, (int)Math.Sqrt(n) - 1).All(i => n % i > 0)));
}
```



# Responsive user interfaces

## Summary

- Use async with IO-bound operations whenever possible
- Don't block!
- Use async for expensive CPU-bound operations
- Use progress report and progress dialogs for better user experience
- Allow user to cancel operations when possible
- Use benchmarking tools such as BenchmarkDotNet  
(<https://github.com/dotnet/BenchmarkDotNet>)

## Source code

<https://github.com/gerardo-lijs/CSharpMeetup-Mar2019>

# Responsive user interfaces

## Demo of a Web API running IO-bound operation

- Targets .NET Core 3 Preview4 but works exactly the same with .NET Core 2
- The behaviour is the same in Web API using .NET 4.7.2 or earlier

Source code

<https://github.com/gerardo-lijs/CSharpMeetup-Mar2019>

# Scalability and performance in web applications

Simple API method that downloads a web page using synchronous code

```
[Route("api/[controller]")]
[ApiController]
public class IOBoundController : ControllerBase
{
    private string GetHtml()
    {
        var client = new System.Net.WebClient();
        string response = client.DownloadString("https://www.dotnetfoundation.org");
        return response;
    }

    // GET api/iobound
    // .\bombardier.exe "http://localhost:5000/api/iobound" -n 20 -t 100s
    [HttpGet]
    public ActionResult<int> Get()
    {
        var webContent = GetHtml();
        return webContent.Length;
    }
}
```

# Scalability and performance in web applications

Same method refactored to use asynchronous code

```
[Route("api/[controller]")]
[ApiController]
public class IOBoundController : ControllerBase
{
    private async Task<string> GetHtml()
    {
        var client = new System.Net.Http.HttpClient();
        string response = await client.GetStringAsync("https://www.dotnetfoundation.org");
        return response;
    }

    // GET api/iobound
    // .\bombardier.exe "http://localhost:5001/api/iobound" -n 20 -t 100s
    [HttpGet]
    public async Task<ActionResult<int>> Get()
    {
        var webContent = await GetHtml();
        return webContent.Length;
    }
}
```



# Scalability and performance in web applications

```
Windows PowerShell
PS C:\Users\Gerardo\go\bin> .\bombardier.exe "http://localhost:5000/api/iobound" -n 20 -t 100s
Bombarding http://localhost:5000/api/iobound with 20 request(s) using 125 connection(s)
20 / 20 [=====] 100.00% 1/s 15s
Done!
Statistics      Avg      Stdev      Max
Reqs/sec        1.55      18.57     376.04
Latency         15.62s     36.19ms    15.67s
HTTP codes:
  1xx - 0, 2xx - 20, 3xx - 0, 4xx - 0, 5xx - 0
  others - 0
Throughput:      301.22/s
PS C:\Users\Gerardo\go\bin> .\bombardier.exe "http://localhost:5000/api/iobound" -n 20 -t 100s
Bombarding http://localhost:5000/api/iobound with 20 request(s) using 125 connection(s)
20 / 20 [=====] 100.00% 1/s 10s
Done!
Statistics      Avg      Stdev      Max
Reqs/sec         3.88      48.47    1003.11
Latency          9.57s     306.64ms    9.88s
HTTP codes:
  1xx - 0, 2xx - 20, 3xx - 0, 4xx - 0, 5xx - 0
  others - 0
Throughput:      477.56/s
PS C:\Users\Gerardo\go\bin>

Windows PowerShell
PS C:\Users\Gerardo\go\bin> .\bombardier.exe "http://localhost:5001/api/iobound" -n 20 -t 100s
Bombarding http://localhost:5001/api/iobound with 20 request(s) using 125 connection(s)
20 / 20 [=====] 100.00% 8/s 2s
Done!
Statistics      Avg      Stdev      Max
Reqs/sec        13.53      50.10     334.22
Latency          1.75s      37.69ms    1.81s
HTTP codes:
  1xx - 0, 2xx - 20, 3xx - 0, 4xx - 0, 5xx - 0
  others - 0
Throughput:       2.55KB/s
PS C:\Users\Gerardo\go\bin> .\bombardier.exe "http://localhost:5001/api/iobound" -n 20 -t 100s
Bombarding http://localhost:5001/api/iobound with 20 request(s) using 125 connection(s)
20 / 20 [=====] 100.00% 16/s 1s
Done!
Statistics      Avg      Stdev      Max
Reqs/sec        19.96      45.00     200.53
Latency          0.96s      64.10ms    1.04s
HTTP codes:
  1xx - 0, 2xx - 20, 3xx - 0, 4xx - 0, 5xx - 0
  others - 0
Throughput:       4.44KB/s
PS C:\Users\Gerardo\go\bin>
```

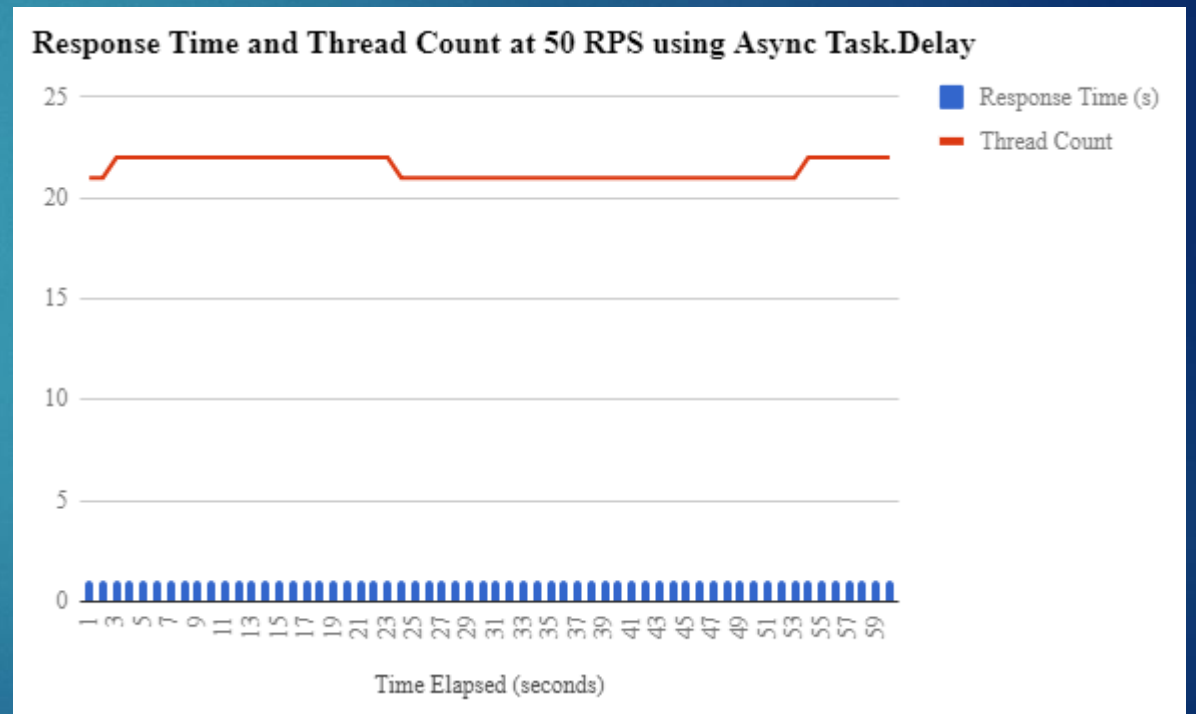
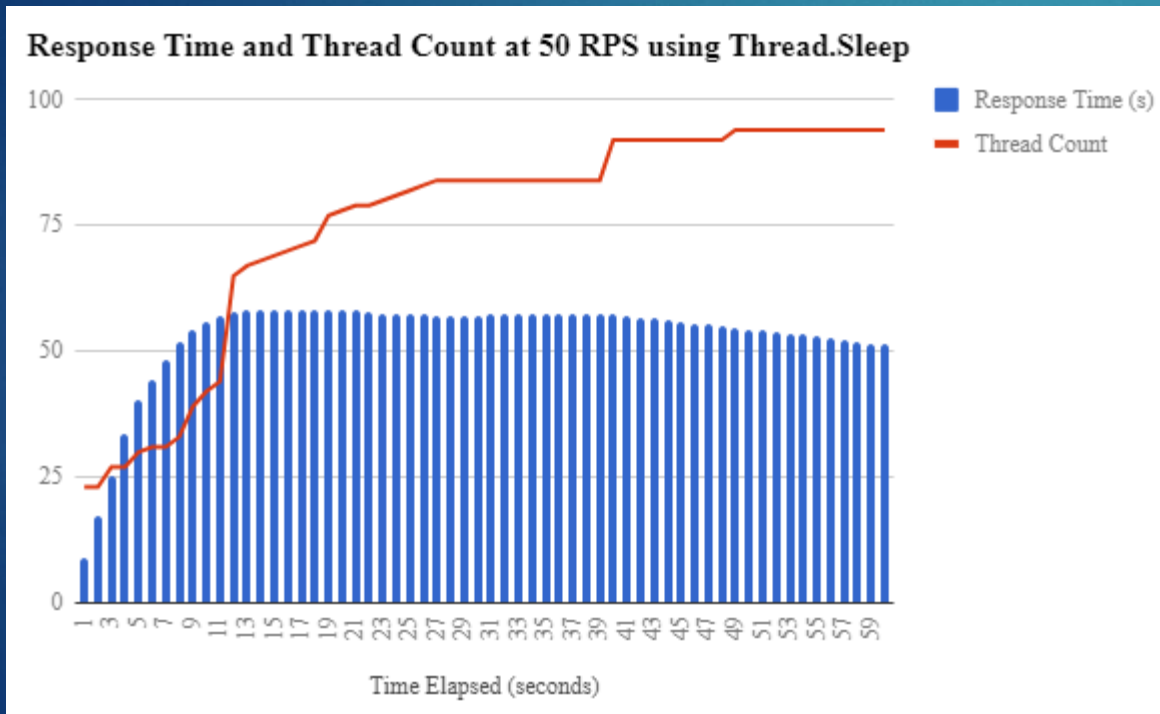
This is bad!

This is good! Using async/await

# Scalability and performance in web applications

## Asynchronous benefits

Improves throughput because of reduced number of threads usage, less memory use and less CPU use



# Responsive user interfaces

## Demo of a Web API running CPU-bound operation

- Targets .NET Core 3 Preview4 but works exactly the same with .NET Core 2
- The behaviour is the same in Web API using .NET 4.7.2 or earlier

Source code

<https://github.com/gerardo-lijs/CSharpMeetup-Mar2019>

# Scalability and performance in web applications

Simple API method running an expensive CPU-bound operation

```
[Route("api/[controller]")]
[ApiController]
public class CPUBoundController : ControllerBase
{
    private int GetPrimesCount(int start, int count)
    {
        return ParallelEnumerable.Range(start, count).Count(n => Enumerable.Range(2, (int)Math.Sqrt(n) - 1).All(i => n % i > 0));
    }

    // GET api/cpubound
    // .\bombardier.exe "http://localhost:5000/api/cpubound?start=1&end=1000000" -n 20 -t 100s
    [HttpGet]
    public ActionResult<int> Get([FromQuery] int start, [FromQuery] int end)
    {
        return GetPrimesCount(start, end);
    }
}
```



# Scalability and performance in web applications

Same method refactored to use asynchronous code

```
[Route("api/[controller]")]
[ApiController]
public class CPUBoundController : ControllerBase
{
    private Task<int> GetPrimesCountAsync(int start, int count)
    {
        return Task.Run(() =>
            ParallelEnumerable.Range(start, count).Count(n => Enumerable.Range(2, (int)Math.Sqrt(n) - 1).All(i => n % i > 0)));
    }

    // GET api/cpubound
    // .\bombardier.exe "http://localhost:5001/api/cpubound?start=1&end=1000000" -n 20 -t 100s
    [HttpGet]
    public async Task<ActionResult<int>> Get([FromQuery] int start, [FromQuery] int end)
    {
        return await GetPrimesCountAsync(start, end);
    }
}
```

# Scalability and performance in web applications

```
Windows PowerShell
PS C:\Users\Gerardo\go\bin> .\bombardier.exe "http://localhost:5000/api/values?start=1&end=10000000" -n 30 -t 100s
Bombarding http://localhost:5000/api/values?start=1&end=10000000 with 30 request(s) using 125 connection(s)
30 / 30 [=====] 100.00% 0/s 1m23s
Done!
Statistics      Avg      Stdev      Max
Reqs/sec       0.32      8.75     418.45
Latency        1.24m     11.82s    1.39m
HTTP codes:
  1xx - 0, 2xx - 30, 3xx - 0, 4xx - 0, 5xx - 0
  others - 0
Throughput:     92.44/s
PS C:\Users\Gerardo\go\bin>
```

Synchronous  
CPU-bound

```
Windows PowerShell
PS C:\Users\Gerardo\go\bin> .\bombardier.exe "http://localhost:5001/api/values?start=1&end=10000000" -n 30 -t 100s
Bombarding http://localhost:5001/api/values?start=1&end=10000000 with 30 request(s) using 125 connection(s)
30 / 30 [=====] 100.00% 0/s 1m26s
Done!
Statistics      Avg      Stdev      Max
Reqs/sec       0.15      3.10     116.98
Latency        1.32m     10.72s    1.44m
HTTP codes:
  1xx - 0, 2xx - 30, 3xx - 0, 4xx - 0, 5xx - 0
  others - 0
Throughput:     89.21/s
PS C:\Users\Gerardo\go\bin>
```

Asynchronous  
CPU-bound

Using async/await makes it worse in this case!

# Scalability and performance in web applications

## Summary

- Use async with IO-bound operations whenever possible
- Don't block!
- Don't use async for expensive CPU-bound operations
- Use benchmarking tools such as Bombardier  
(<https://github.com/codesenberg/bombardier>)

I still need more performance, what can I do?

- Scale up
- Scale out whole application using Docker/Kubernetes
- Scale out only expensive CPU-bound operations to Azure Functions

# Obsolete patterns

## BackgroundWorker

```
backgroundWorker1.DoWork += new DoWorkEventHandler(backgroundWorker1_DoWork);  
backgroundWorker1.RunWorkerCompleted += new RunWorkerCompletedEventHandler(backgroundWorker1_RunWorkerCompleted);  
backgroundWorker1.ProgressChanged += new ProgressChangedEventHandler(backgroundWorker1_ProgressChanged);
```

## Event-Based Asynchronous Pattern (EAP)

```
public void DownloadDataAsync(Uri address);  
public event DownloadDataCompletedEventHandler DownloadDataCompleted;  
public bool IsBusy { get; } // Indicates if still running
```

## Asynchronous Programming Model (APM)

```
public IAsyncResult BeginRead(byte[] buffer, int offset, int size, AsyncCallback callback, object state);  
public int EndRead(IAsyncResult asyncResult);
```



# Task Cancellation

- Use CancellationToken in your method and check manually

```
async Task Foo(CancellationTokn cancellationToken)
{
    for (int i = 0; i < 10; i++)
    {
        Console.WriteLine(i);
        await Task.Delay(1000);
        cancellationToken.ThrowIfCancellationRequested();
    }
}
```

# Task Cancellation

- Use Extension method

```
public static Task<TResult> WithCancellation<TResult>(this Task<TResult> task, CancellationToken cancellationToken)
{
    var tcs = new TaskCompletionSource<TResult>();
    var reg = cancellationToken.Register(() => tcs.TrySetCanceled());
    task.ContinueWith(ant =>
    {
        reg.Dispose();
        if (ant.IsCanceled)
            tcs.TrySetCanceled();
        else if (ant.IsFaulted)
            tcs.TrySetException(ant.Exception.InnerException);
        else
            tcs.TrySetResult(ant.Result);
    });
    return tcs.Task;
}
```

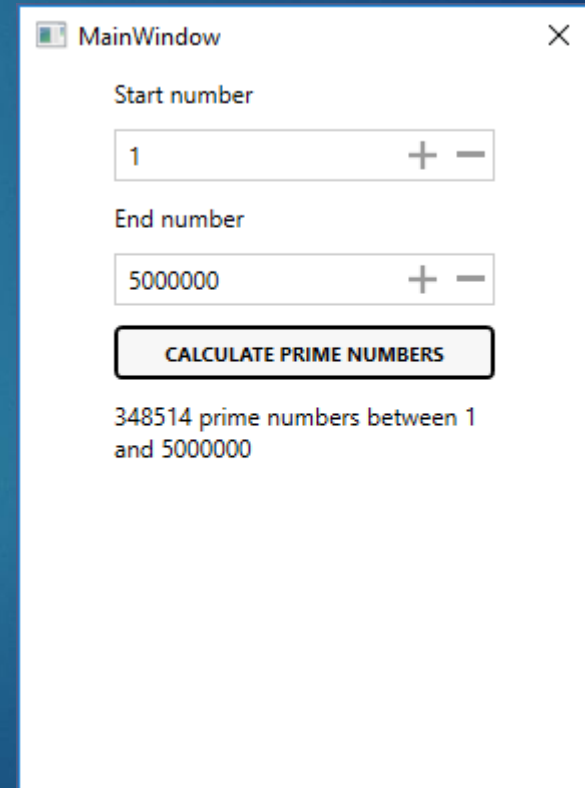
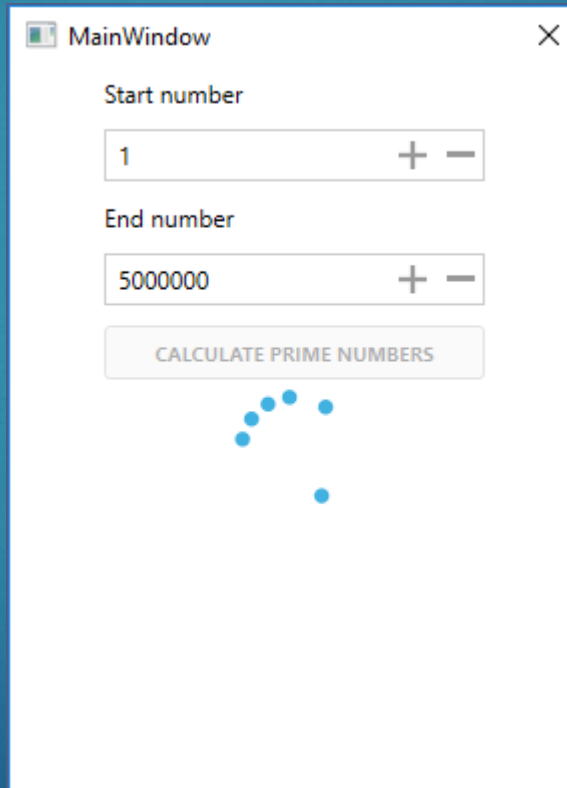
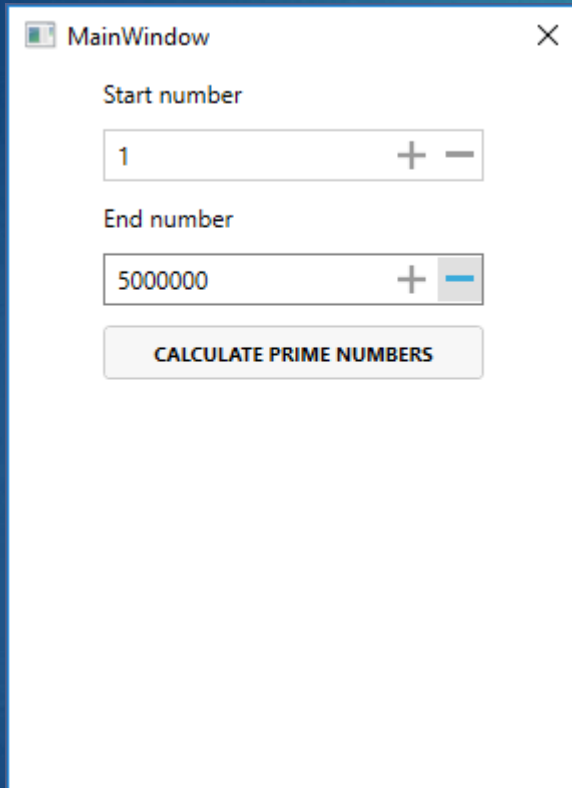
# Run a Task with a Timeout

- Use Extension method

```
public async static Task<TResult> WithTimeout<TResult>(this Task<TResult> task, TimeSpan timeout)
{
    Task winner = await (Task.WhenAny(task, Task.Delay(timeout)));
    if (winner != task) throw new TimeoutException();
    return await task; // Unwrap result/re-throw
}
```

# Progress reporting

- Use `IProgress<T>` interface
- Use indeterminate `ProgressRing`





# Running asynchronous method synchronously

Use helper method borrowed from Microsoft

Call like this

```
public static class AsyncHelper
{
    private static readonly TaskFactory _taskFactory = new
        TaskFactory(CancellationToken.None,
            TaskCreationOptions.None,
            TaskContinuationOptions.None,
            TaskScheduler.Default);

    public static TResult RunSync<TResult>(Func<Task<TResult>> func)
        => _taskFactory
            .StartNew(func)
            .Unwrap()
            .GetAwaiter()
            .GetResult();

    public static void RunSync(Func<Task> func)
        => _taskFactory
            .StartNew(func)
            .Unwrap()
            .GetAwaiter()
            .GetResult();
}
```

```
var result = AsyncHelper.RunSync(() => DoAsyncStuff());
```

## Warning

Try to avoid this! Most of the times (probably always) there are better ways and you don't need to run async code synchronously

# Async class constructors

They are not available by default in the framework

Option 1 - Refactor with InitializeAsync method

Previous class

```
public class ExampleClass
{
    public ExampleClass()
    {
        // Current initialization code here
    }
}
```

Refactored class

```
public class ExampleClass
{
    public ExampleClass()
    {
        // No more code here
    }

    public async Task InitializeAsync()
    {
        // Previous initialization code moved here
        // Plus you can call async methods with await now
        // Problem -> Consumers need to know they have to
        // always run this method after creating class
        // It's better to use Factory Pattern in my opinion
    }
}
```

Call like this

```
var classInstance = new ExampleClass();
await classInstance.InitializeAsync();
```

# Async class constructors

They are not available by default in the framework

Option 2 - Refactor with Factory Pattern

Previous class

```
public class ExampleClass
{
    public ExampleClass()
    {
        // Current initialization code here
    }
}
```

Refactored class

```
public class ExampleClass
{
    // We explicitly create a private parameterless constructor
    // so that we are forced to use the static async method
    private ExampleClass() { }

    private async Task InitializeAsync()
    {
        // Previous initialization code moved here
        // Plus you can call async methods with await now
    }

    public static async Task<ExampleClass> CreateAsync()
    {
        var ret = new ExampleClass();
        await ret.InitializeAsync();
        return ret;
    }
}
```

Call like this

```
var classInstance = await ExampleClass.CreateAsync();
```

# Parallel programming

## Demo of a WPF application using TPL and Task.WhenAll

- Uses ReactiveUI and MahApps but they are not a requirement
- Uses OpenCvSharp library as example of expensive operations
- Targets .NET Core 3 Preview4 but works exactly the same with .NET Core 2
- The behaviour is the same in Web API using .NET 4.7.2 or earlier

Source code

<https://github.com/gerardo-lijs/CSharpMeetup-Mar2019>



Slides, resources and source code

<https://github.com/gerardo-lijs/CSharpMeetup-Mar2019>

<https://github.com/gerardo-lijs/Asynchronous-Programming>

Gerardo Lijs

Software Developer At Microptic SL

Email [gerardo.lijs@gmail.com](mailto:gerardo.lijs@gmail.com)

Twitter [@GerardoLijs](https://twitter.com/GerardoLijs)

Thank You!