Backend Principle

IoC + Hosting

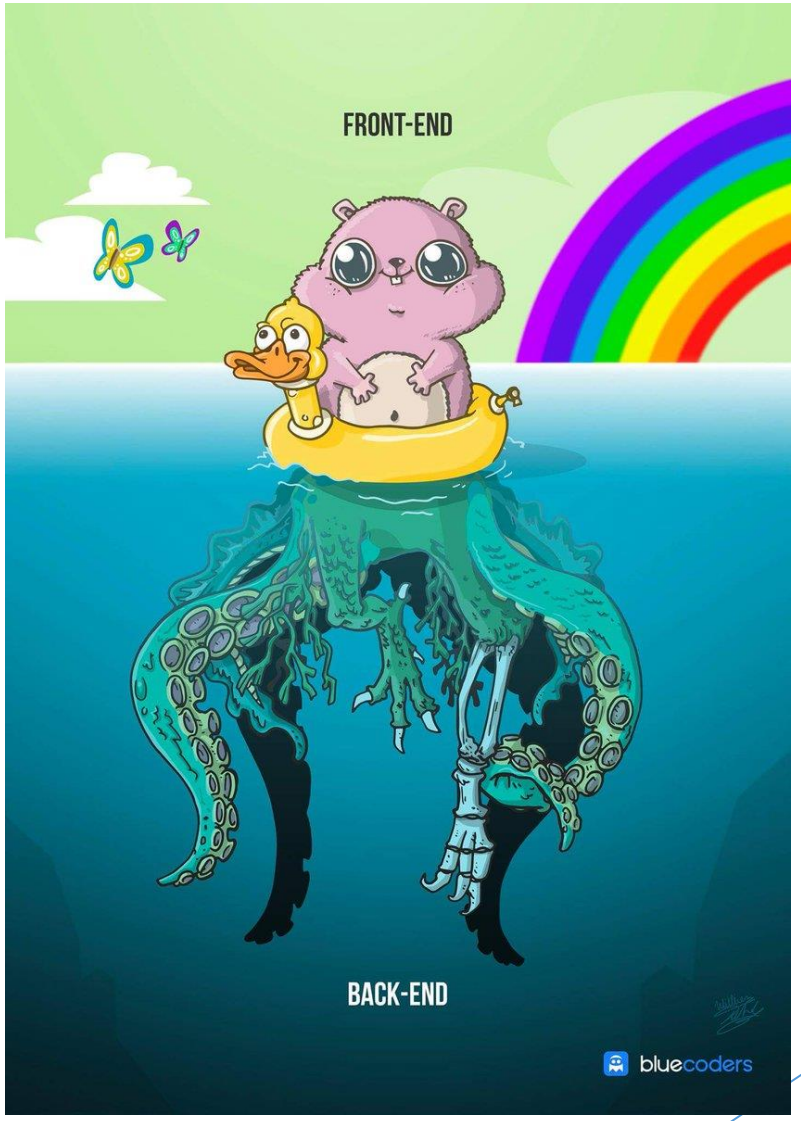
Di cosa abbiamo bisogno...

- Visual Studio 2019
- .Net Core 3.1+
- Saper usare **bene** le interfacce

Backend ?

Di cosa parliamo quando diciamo backend?

NO



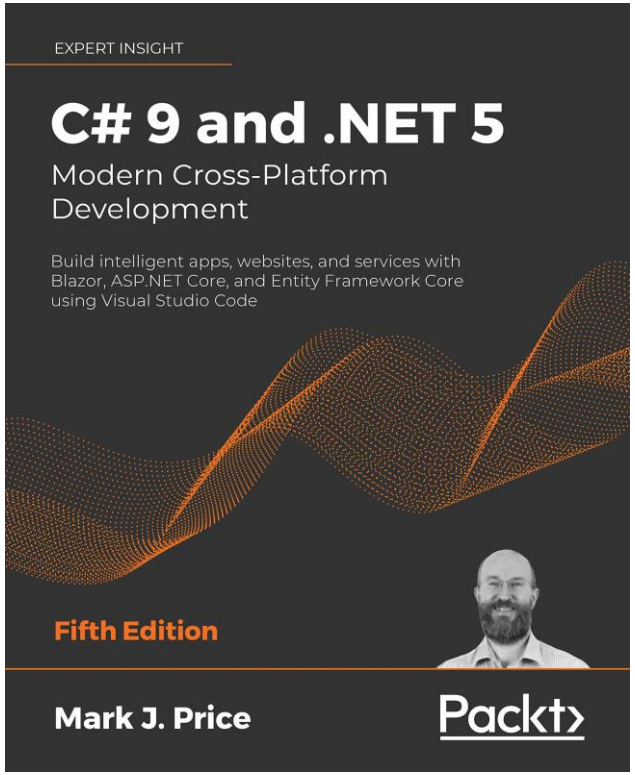
YES

Backend ?

- Con Backend intendiamo tutti quei sotware che non hanno interazione diretta con l'utente ma che nascono per dare supporto agli applicativi di Frontend
- Come il Frontend anche il Backend ha la sua suite di tecnologie specifiche e di linguaggi appositi

Cosa vedremo in questo percorso

- Self Hosting
  - IHostBuilder
  - IConfiguration
  - BackgroundService
- Quartz
- Inversion Of Control (IoC)
  - IServiceCollection
- Self Hosting + IoC + Quartz = livello



GRUPPO EURIS

Self Hosting

Self-Hosting

- Con l'avvento di .NET Core non esistono più "tipologie" di progetti nativi come in .Net Framework:
  - WCF
  - ASP .NET Web Forms
  - ASP .NET MVC
  - Service
  - ….
- Ogni tipologia di applicativo è *Console App* che si occupa di *hostare* un applicativo Web o Generico

Self-Hosting

- Un host è un oggetto che incapsula le risorse di un'app, ad esempio:
  - Inserimento di dipendenze (DI)
  - Registrazione
  - Configurazione
  - IHostedService Implementazioni

- All'avvio, un host chiama su ogni implementazione di registrata nella raccolta di servizi ospitati del IHostedService.StartAsync IHostedService contenitore del servizio. In un'app Web, una delle implementazioni di IHostedService è un servizio web che avvia un'implementazione del server HTTP.

- Il motivo principale per cui tutte le risorse interdipendenti dell'app sono incluse in un unico oggetto è la gestione del ciclo di vita, vale a dire il controllo sull'avvio dell'app e sull'arresto normale.

Self-Hosting

- Cosa differenzia i vari progetti?
  - Il self-hosting!
- L'host è in genere configurato, compilato ed eseguito da codice nella classe Program.

  Il metodo Main:
  - Chiama un metodo CreateHostBuilder per creare e configurare un oggetto generatore.
  - Chiamate **Build** e metodi **Run**

C#

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

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureServices((hostContext, services) =>
            {
                services.AddHostedService<Worker>();
            });
}
```

C#

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateWebHostBuilder(args).Build().Run();
    }

    public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>();
}
```

Configurazione

La configurazione in ASP.NET Core viene eseguita usando uno o più provider di configurazione.

I provider di configurazione leggono i dati di configurazione da coppie chiave-valore usando un'ampia gamma di origini di configurazione:

- Impostazioni file, ad esempio *appsettings.json*
- Variabili di ambiente
- File della directory
- …

Configurazione

È possibile utilizzarlo:

- In fase di build dell'host:

```
C#

public class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureAppConfiguration((hostingContext, config) =>
            {
                config.AddEnvironmentVariables(prefix: "MyCustomPrefix_");
            })
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseStartup<Startup>();
            });
}
```

- Tramite l'uso dell'interfaccia IConfiguration:

```
C#

public class TestModel : PageModel
{
    // requires using Microsoft.Extensions.Configuration;
    private readonly IConfiguration Configuration;

    public TestModel(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public ContentResult OnGet()
    {
        var myKeyValue = Configuration["MyKey"];
        var title = Configuration["Position:Title"];
        var name = Configuration["Position:Name"];
        var defaultLogLevel = Configuration["Logging:LogLevel:Default"];


        return Content($"MyKey value: {myKeyValue} \n" +
                       $"Title: {title} \n" +
                       $"Name: {name} \n" +
                       $"Default Log Level: {defaultLogLevel}");
    }
}
```

Configurazione – accesso chiave valore

JSON

```json
{
    "Position": {
        "Title": "Editor",
        "Name": "Joe Smith"
    },
    "MyKey": "My appsettings.json Value",
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft": "Warning",
            "Microsoft.Hosting.Lifetime": "Information"
        }
    },
    "AllowedHosts": "*"
}
```

C#

```csharp
public class TestModel : PageModel
{
    // requires using Microsoft.Extensions.Configuration;
    private readonly IConfiguration Configuration;

    public TestModel(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public ContentResult OnGet()
    {
        var myKeyValue = Configuration["MyKey"];
        var title = Configuration["Position:Title"];
        var name = Configuration["Position:Name"];
        var defaultLogLevel = Configuration["Logging:LogLevel:Default"];


        return Content($"MyKey value: {myKeyValue} \n" +
                       $"Title: {title} \n" +
                       $"Name: {name} \n" +
                       $"Default Log Level: {defaultLogLevel}");
    }
}
```

Configurazione - Bind

JSON

```
  "Position": {
    "Title": "Editor",
    "Name": "Joe Smith"
  }
```

C#

```
public class PositionOptions
{
    public const string Position = "Position";

    public string Title { get; set; }
    public string Name { get; set; }
}
```

```
private readonly IConfiguration Configuration;

public Test22Model(IConfiguration configuration)
{
    Configuration = configuration;
}

public ContentResult OnGet()
{
    var positionOptions = new PositionOptions();
    Configuration.GetSection(PositionOptions.Position).Bind(positionOptions);

    return Content($"Title: {positionOptions.Title} \n" +
                   $"Name: {positionOptions.Name}");
}
```

- Tutte le proprietà devono essere pubbliche e in lettura e scrittura.
- La classe deve essere non astratta e con un costruttore pubblico senza parametri.

Background Service

Background Services – Introduzione

- I background service (o Worker) in windows sono degli applicativi eseguibili che svolgono dei compiti specifici e sono progettati per lavorare in background, senza l'intervento da parte degli utenti.
- La loro controparte Linux è chiamata daemon.

Background Services – Introduzione

- Su .NET Framework i servizi windows potevano essere creati tramite VS utilizzando l'apposito tipo progetto, **NON** era possibile creare deamon su Linux
- Con .NET Core i Worker vengono creati tramite BackgroundService che sono compatibili sia con Linux che con sistemi Windows. Questi ultimi possono essere facilmente trasformati in Windows Service.

Background Services

```
CreateHostBuilder(args).Build().Run();

static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureServices((hostContext, services) =>
        {
            services.AddHostedService<Worker>();
        });
```

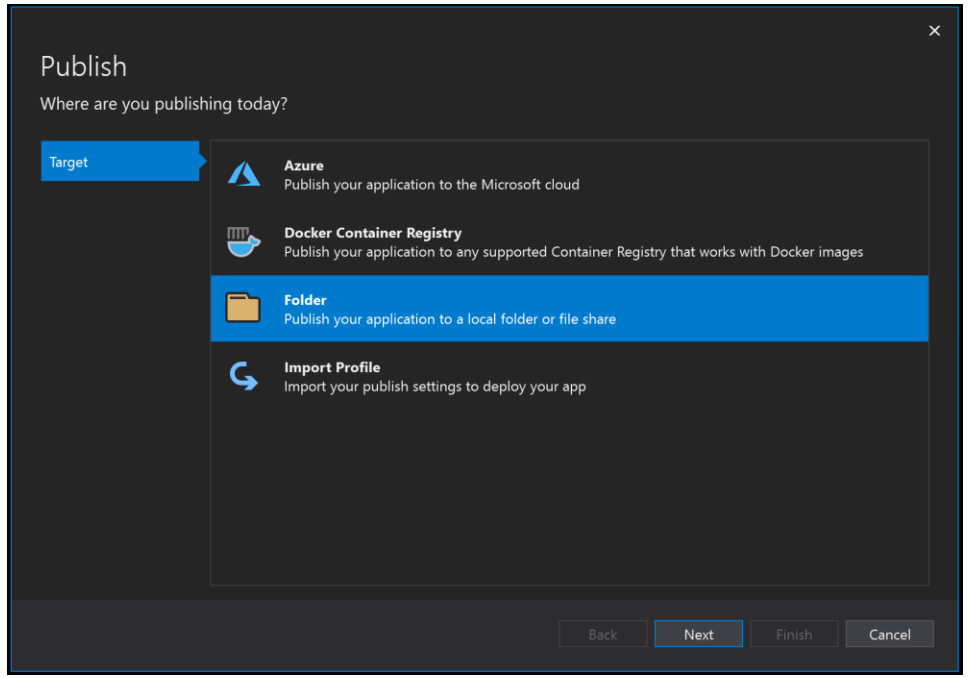
```
public class Worker : BackgroundService
{
    private readonly ILogger<Worker> _logger;

    0 references
    public Worker(ILogger<Worker> logger) => _logger = logger;

    0 references
    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            _logger.LogInformation("Worker running at: {time}", DateTimeOffset.Now);
            await Task.Delay(1000, stoppingToken);
        }
    }
}
```

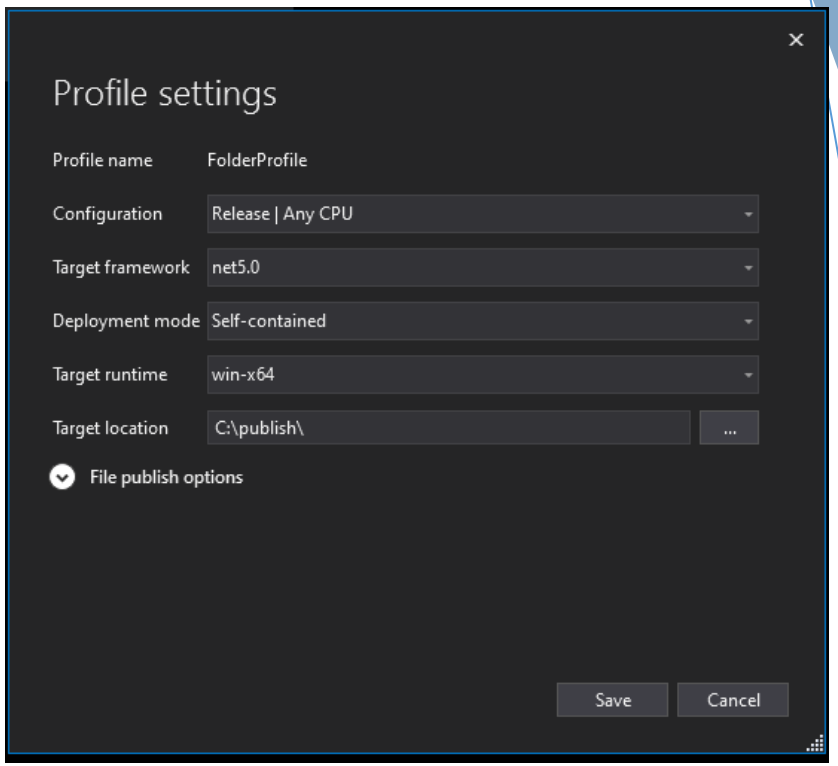
```
C:\Users\biondo\source\repos\WorkerService1\bin\Debug\net5.0\WorkerService1.exe
info: Worker[0]
      Worker running at: 10/19/2021 14:35:09 +02:00
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\biondo\source\repos\WorkerService1
info: Worker[0]
      Worker running at: 10/19/2021 14:35:10 +02:00
info: Worker[0]
      Worker running at: 10/19/2021 14:35:11 +02:00
info: Worker[0]
      Worker running at: 10/19/2021 14:35:13 +02:00
```

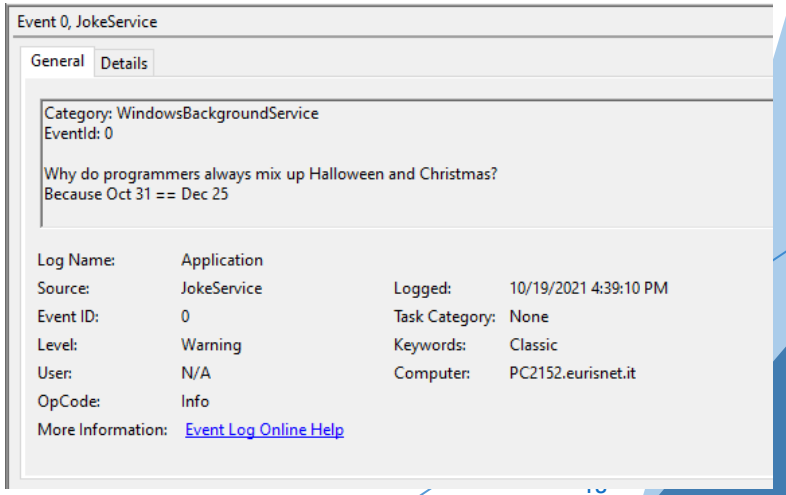

GRUPPO EURIS

Windows Services

Mani al codice Qui!








```
PS C:\> sc.exe create "Joker Service1" binpath=C:\publish\JokerService.exe
[SC] CreateService SUCCESS
PS C:\> sc.exe delete "Joker Service1"
[SC] DeleteService SUCCESS
PS C:\>
```

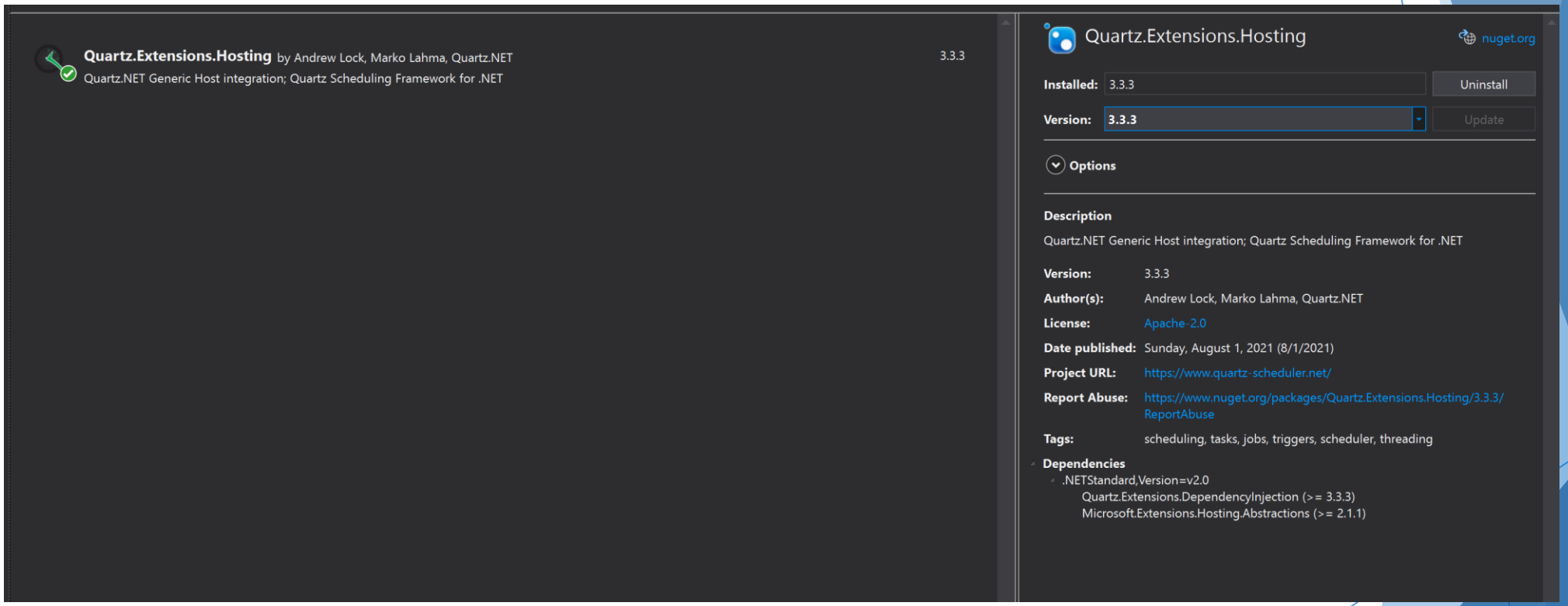
Attenzione: bisogna aprire **PS** o **cmd** come Administrator altrimenti l'installazione fallirà

GRUPPO EURIS

Quartz

- Quartz è una libreria per .NET che permette di creare schedulazioni temporali e di legare a queste degli eventi che possiamo gestire lato codice

Self Hosting + Quartz



GRUPPO EURIS

Self Hosting + Quartz

```
using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        services.AddQuartz(q =>
        {
            q.UseMicrosoftDependencyInjectionScopedJobFactory();

            q.AddJob<JokeJob>(opts => opts.WithIdentity("JokeJob"))
            .AddTrigger(opts => opts
                .ForJob("JokeJob")
                .WithCronSchedule("0/5 * * * * ?"));
        });
        services.AddQuartzHostedService(q => q.WaitForJobsToComplete = true);

    })
    .Build();

await host.RunAsync();
```

```
[DisallowConcurrentExecution]
2 references | 0 changes | 0 authors, 0 changes
public class JokeJob : IJob
{
    private JokeClient _jokeClient;

    0 references | 0 changes | 0 authors, 0 changes
    public JokeJob()
    {
        //it's really ugly!!
        _jokeClient = new JokeClient(new System.Net.Http.HttpClient());
    }

    0 references | 0 changes | 0 authors, 0 changes
    async Task IJob.Execute(IJobExecutionContext context)
    {
        var result = await _jokeClient.GetJokeAsync();
        Console.WriteLine(result);
    }
}
```

Mani al codice Qui!

Inversion Of Control
(IoC)

IoC– Definizione formale

*«**IoC (Inversion of Control)** si intende un design pattern, secondo il quale si tende a tener **disaccoppiati** i singoli componenti di un sistema, in cui le eventuali **dipendenze** non vengono scritte all'interno del componente stesso, ma gli vengono iniettate dall'esterno ... gli oggetti quindi non istanziano e richiamano gli oggetti dal quale il loro lavoro dipende, ma queste funzionalità vengono fornite da un ambiente esterno tramite dei contratti definiti da entrambe le entità.»*

IoC- Descrizione informale

Il pattern IoC segue il cosiddetto principio di Holliwood:

“don’t call us, we’ll call you”

L’idea principale è quella di far sì che un unico punto centrale del software si faccia carico di gestire la relazione tra i componenti e le loro dipendenze e che si occupi di iniettarle e «richiamarle» solo quando necessario.

IoC

Essendo l'argomento complesso lo suddivideremo in due parti:

- **Parte teorica:** approfondiremo del modulo Best Practice (in particolare DI) tornando sull'argomento
- **Parte pratica:** studieremo un esempio su come applicare l'IoC

IoC

Una classe può creare un'istanza **MyDependency** della classe per utilizzare il relativo metodo **WriteMessage** . Nell'esempio seguente la **MyDependency** classe è una dipendenza della classe **IndexModel** :

C#

```
public class IndexModel : PageModel
{
    private readonly MyDependency _dependency = new MyDependency();

    public void OnGet()
    {
        _dependency.WriteMessage("IndexModel.OnGet created this message.");
    }
}
```

IoC

La classe crea e dipende direttamente dalla **MyDependency** classe . Le dipendenze del codice, come nell'esempio precedente, sono problematiche e devono essere evitate per i motivi seguenti:

- Per sostituire MyDependency con un'implementazione diversa, IndexModel è necessario modificare la classe .

- Se MyDependency sono disponibili dipendenze, devono essere configurate anche dalla IndexModel classe . In un progetto di grandi dimensioni con più classi che dipendono da MyDependency, il codice di configurazione diventa sparso in tutta l'app.

- *È difficile eseguire unit test di questa implementazione. L'app dovrebbe usare una classe MyDependency fittizia o stub, ma ciò non è possibile con questo approccio.*

IoC

L'inserimento delle dipendenze consente di risolvere questi problemi tramite:

- L'uso di un'interfaccia o di una classe di base per astrarre l'implementazione delle dipendenze.
- **La registrazione della dipendenza in un contenitore di servizi (IServiceCollection).**
- L'inserimento del servizio nel costruttore della classe in cui viene usato. Il framework si assume la responsabilità della creazione di un'istanza della dipendenza e della sua eliminazione quando non è più necessaria.

IoC

C#

```
public interface IMyDependency
{
    void WriteMessage(string message);
}
```

C#

```
public class MyDependency : IMyDependency
{
    public void WriteMessage(string message)
    {
        Console.WriteLine($"MyDependency.WriteMessage Message: {message}");
    }
}
```

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddScoped<IMyDependency, MyDependency>();
```

C#

```
public class Index2Model : PageModel
{
    private readonly IMyDependency _myDependency;

    public Index2Model(IMyDependency myDependency)
    {
        _myDependency = myDependency;
    }

    public void OnGet()
    {
        _myDependency.WriteMessage("Index2Model.OnGet");
    }
}
```

Self Hosting + Quartz + IoC

```
using IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        services.AddScoped<IHelloWorld, HelloWorld>();
        services.AddHttpClient<IJokeClient, JokeClient>();

        services.AddQuartz(q =>
        {
            q.UseMicrosoftDependencyInjectionScopedJobFactory();

            q.AddJob<JokeJob>(opts => opts.WithIdentity("JokeJob"))
            .AddTrigger(opts => opts
                .ForJob("JokeJob")
                .WithCronSchedule("0/5 * * * * ?"));

            q.AddJob<HelloJob>(opts => opts.WithIdentity("HelloJob"))
            .AddTrigger(opts => opts
                .ForJob("HelloJob")
                .WithCronSchedule("0/30 * * * * ?"));
        });
        services.AddQuartzHostedService(q => q.WaitForJobsToComplete = true);

    })
    .Build();

await host.RunAsync();
```

```
[DisallowConcurrentExecution]
2 references | 0 changes | 0 authors, 0 changes
public class HelloJob : IJob
{
    private readonly IHelloWorld _helloWorld;

    0 references | 0 changes | 0 authors, 0 changes
    public HelloJob(IHelloWorld helloWorld)
    {
        _helloWorld = helloWorld;
    }

    0 references | 0 changes | 0 authors, 0 changes
    Task IJob.Execute(IJobExecutionContext context)
    {
        var result = _helloWorld.Hello();
        Console.WriteLine(result);
        return Task.CompletedTask;
    }
}
```

Mani al codice Qui!

Backend Principle – Qualche esempio pratico

Mano al codice!!

Realizzare un servizio che:

Con una schedulazione di 30 secondi vada a leggere dalla folder

C:\bmi-calculator\input tutti i file .csv presenti con il formato:

Nome Paziente ; Peso ; Altezza

Sulla base dei dati letti per paziente devono calcolare l'indice di massa corporeo con la formula:

BMI = peso / altezza * altezza

e stabilire se il paziente è

Classificazione	BMI
Sottopeso	< 18.5
Normale	18.5 – 24.9
Sovrappeso	25 – 29
Obeso	>= 30

Backend Principle – Qualche esempio pratico

Mano al codice!!

- I dati in output dovranno essere prodotti sulla folder

  C:\bmi-calculator\output e dovranno essere in formato .csv con formato:

Noma Paziente ; BMI Calcolato ; Classificazione

Thank you!