.NET Conf 2022

DotNet Liguria

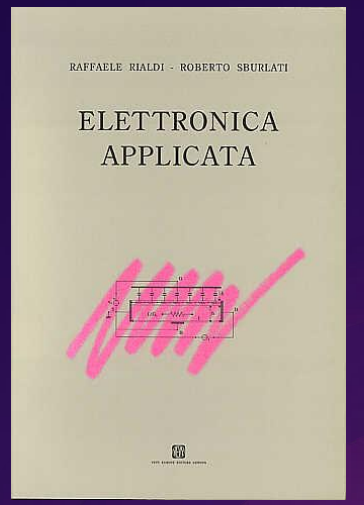
Da WebAssembly a Blazor

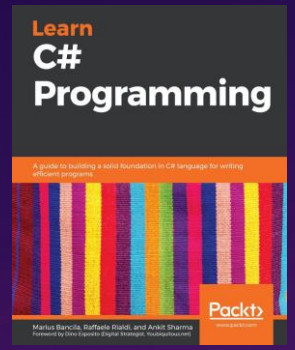
Ing. Raffaele Rialdi, Senior Software Architect

Chi sono

@raffaeler
aka "Raf"

- Laurea magistrale in Ingegneria Elettronica
- Insegnante all'Università di Ingegneria Informatica a Genova
- Senior Software Architect
- Consulenze di architettura e sviluppo software
  - Manufacturing, racing, healthcare, financial, ...
- Speaker, Trainer e Autore di libri (fisica elettronica e C#)
  - Italy, Romania, Bulgaria, Russia, USA, ...
- Orgoglioso membro della grande famiglia dei Microsoft MVP dal 2003





Blazor o WebAssembly?

- Blazor nasconde WebAssembly per favorire la semplicità
  - Impone lo sviluppo della UI basato sul paradigma Razor
  - Impone l'uso del browser come host del codice
  - Inadatto per UI con paradigma diverso da "business-oriented"
- In altre parole .... ☺

Blazor : 

= Wasm & .NET : 

Perché WebAssembly?

- WebAssembly è uno standard W3C che definisce due cose:
  - Una pseudo virtual-machine (sandobx)
    - Esegue codice binario, garantisce l'isolamento del codice
  - Il formato del codice binario eseguito (analogo al codice IL di .NET)
- Qualsiasi processo può essere host di WebAssembly
  - I browser sono host di WA e consentono di accedere al DOM
- Rust è il linguaggio principe di WebAssembly
  - Ma C, C++, .NET ed altri possono produrre codice WebAssembly.

Wasm e .NET

- In WebAssembly la dimensione del binario è fondamentale
  - .NET necessita di servizi di runtime che occupano spazio
  - Per ridurre la dimensione del CLR, al momento è usato Mono

- Il codice in WA deve interoperare con l'host
  - Nel browser è necessario interoperare con Javascript

```
[JSImport("renderCanvas", "main.js")]
internal static partial void RenderCanvas(...)
```

Chiama Javascript

```
[JSExport]
internal static async Task OnClick()
```

È chiamato da Javascript

Cosa bisogna installare?

- I wasm-tools sono gli stessi di Blazor e già finalizzati

```
dotnet workload install wasm-tools
```

- Il workload wasm-experimental contiene i template che non usano Blazor

```
dotnet workload install wasm-experimental
```

- I template sono disponibili in Visual Studio o dalla CLI
  - "dotnet new wasmbrowser" usa il browser come host
  - "dotnet new wasmconsole" usa la console e node.js come host

Wasm, dotnet e multi-threading

- Un package sperimentale abilita l'uso dei thread basato sui Web Workers
- Per provarlo bastano due step:
  1. Abilitare WasmEnableThreads nel csproj

```
<PropertyGroup>
  <WasmEnableThreads>true</WasmEnableThreads>
</PropertyGroup>
```

  2. Aggiungere questo pacchetto nuget al progetto:

```
dotnet add package --prerelease Microsoft.NET.WebAssembly.Threading
```

Prospettive

- **Molto** sperimentale: il "wasi-sdk" è in grado di compilare qualsiasi applicazione non visuale in formato wasm
- AOT: riduzione della dimensione dei wasm generati
- .NET vs Mono
  - prima o poi i due runtime saranno uniti e spariranno le differenze
- Container o wasm?
  - Wasm ha il grosso vantaggio di avere un 'bytecode' neutrale.

Domande?