Blazor ieri, oggi e domani

Andrea Dottor

Sponsor

altitudo

beanTech
IT moves your business

OVERNET.
upgrade your digital skills

[stesi]
Powered by Innovation

Un po' di storia

- Presentato durante la NDC Oslo del 2017

- Rilasciato ufficialmente con .NET Core 3 il 23 settembre 2019, ma con solo **Blazor Server**
- Il 19 maggio 2020 con la versione 3.1.300 viene rilasciato **Blazor WebAssembly**

- Con .NET 8 viene considerato un framework "Full Stack Web UI" grazie a **Blazor Web App**

Blazor server

- Il client è collegato tramite SignalR (websocket)
- L'applicazione viene eseguita lato server

👍 download ridotto

👍 si ha accesso a tutte le risorse del/dal server (es: db, filesystem)

👎 l'applicazione deve essere sempre collegata con il server per poter essere utilizzata

Blazor WebAssembly

- Basato su WebAssembly
- L'applicazione viene eseguita nel browser

👍 rispetto a Blazor Server, viene ridotto il carico sul server

👍 possibilità di lavorare disconnessa (PWA)

👎 download dell'applicazione di diversi MB

Blazor Web App

- Aggiunge la possibilità di utilizzare Blazor per generare html statico lato server

👍 possibilità di scegliere il tipo di interattività che si vuole per ogni singolo componente e/o pagina*

👍 unifica la tecnologia di sviluppo tra back-end e front-end

👎 rispetto a RazorPages e MVC le performance sono inferiori

Blazor SSR - Static Server-Side Render

Il server restituisce l'HTML completo della pagina.

Vengono utilizzati file *.razor per generare l'html.

👍 SEO friendly

👍 Contenuto subito disponibile all'utente

👎 non c'è interazione o parti dinamiche

Ma il rendermode "auto"??

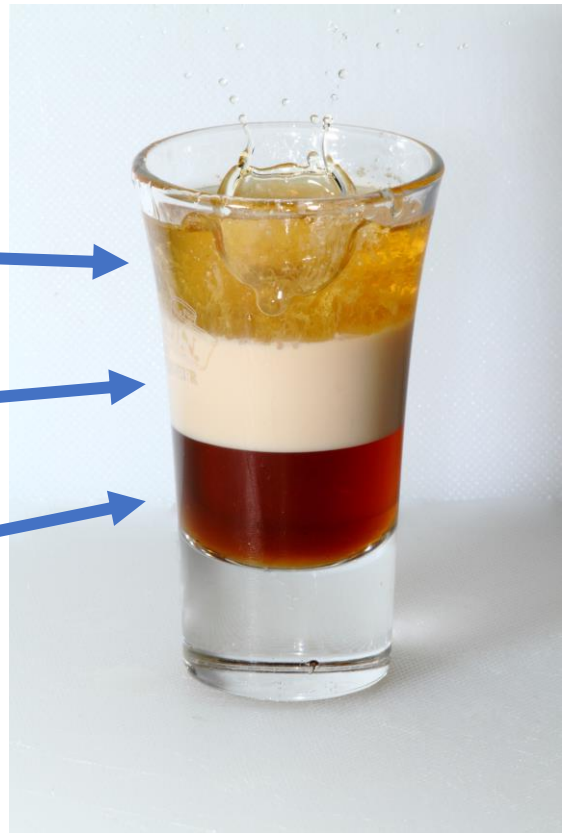
Bello sulla carta, ma nella pratica utile in poche occasioni.

Ha senso avere più rendermode nella stessa applicazione?

Static Server-side rendering

Blazor Server

Blazor WebAssembly



DEMO

Perché Blazor?

Riuso delle competenze

- Utilizzare C# anche per il front-end
  - Reflection
  - Eventi
  - …
- Limitato utilizzo (e conoscenza) di JavaScript

Image by congerdesign from Pixabay

Velocizza i tempi di sviluppo

- Condividere codice tra back-end e front-end
  - DTO, ViewModel, ma anche regole di validazione
  - Utility/algoritmi già sviluppati
- Accesso ai dati dai componenti senza il bisogno di realizzare API Rest e/o servizi gRPC
  - Utilizzando Blazor Server per le prime fasi dello sviluppo/rilasci
- Posso iniziare un'applicazione con InteractiveServer e impostarla su InteractiveWebAssembly in un secondo momento

Tipizzato, compilato

```
let num = 10; num(); // TypeError: num is not a function
```

```
if (x = 5) { console.log("x is equal to 5"); }
```

```
function add(x, y) { return x + y; }
add(1, 2, 3);
// TypeError: add() takes 2 arguments, but 3 were provided
```

```
let num = 10;
let value = num.property;
// TypeError: Cannot read property 'property' of number
```

```
let greeting = "Hello!"; console.log(gretting);
// ReferenceError: gretting is not defined
```

Source: https://www.linkedin.com/pulse/common-javascript-errors-types-causes-solutions-dinesh-rawat/

Dependency Injection

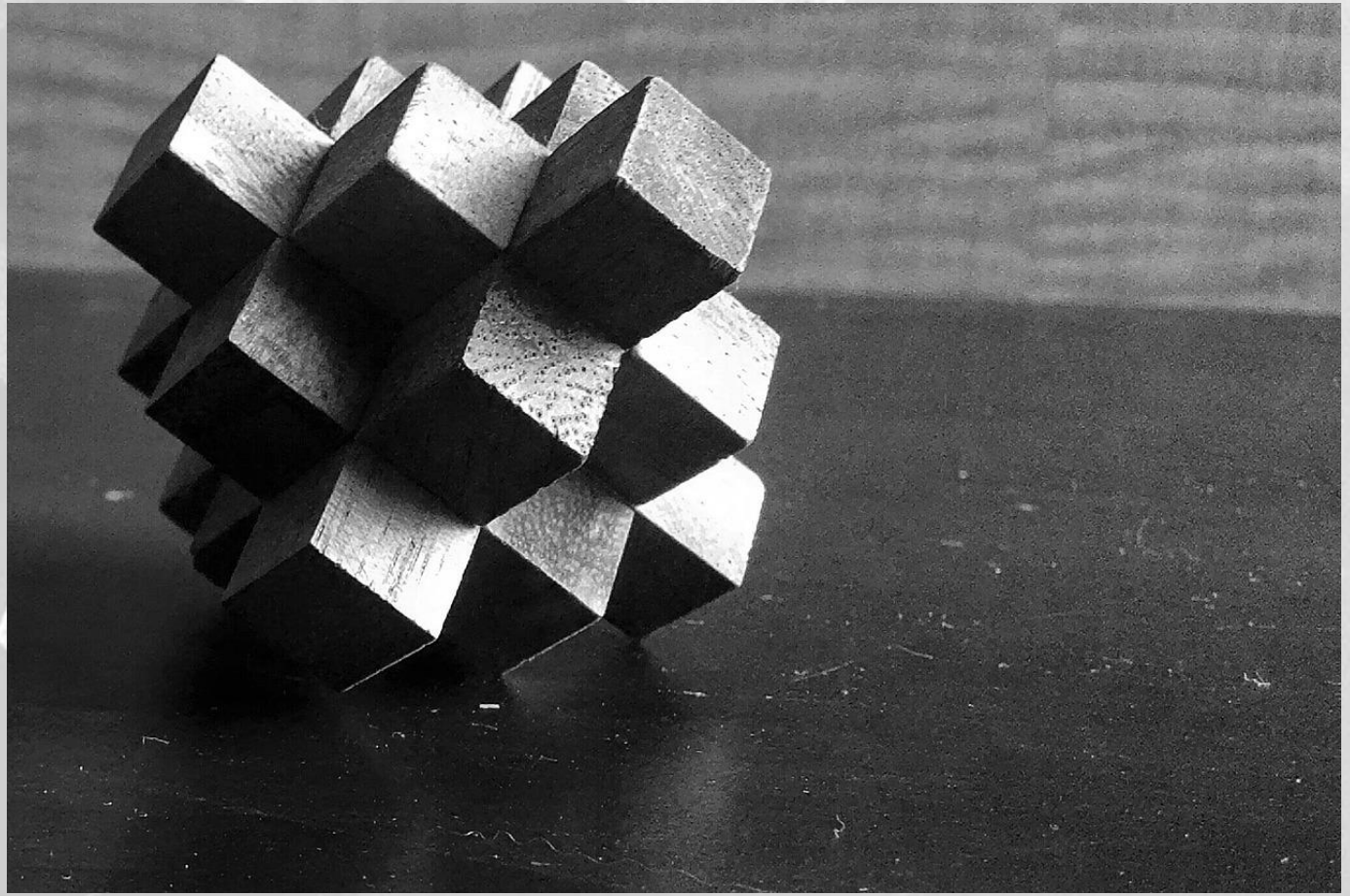


Image by TheLight from Pixabay

Testabile

Image by Gerd Altmann from Pixabay

DEMO

A chi non è utile Blazor?

Image by Gerd Altmann from Pixabay

.NET 9

Cosa dobbiamo aspettarci con .NET 9?

- Static asset delivery optimization
- Detect rendering location, interactivity, and assigned render mode at runtime
- Improved server-side reconnection experience
- Simplified authentication state serialization for Blazor Web Apps
- Add static server-side rendering (SSR) pages to a globally-interactive Blazor Web App
- Constructor injection
- …

Static asset delivery optimization

- MapStaticAssets è un nuovo middleware che aiuta a ottimizzare la distribuzione di risorse statiche in qualsiasi applicazione ASP.NET Core e Blazor.
  - Da sostituire a UseStaticFiles
- MapStaticAssets ha i seguenti vantaggi rispetto a UseStaticFiles:
  - Compressione di tutti gli assets durante la compilazione:
    - **gzip** in sviluppo e **gzip + brotli** in fase di publicazione.
  - **ETag** basati sul contenuto: Gli Etag per ogni risorsa sono prodotto dal Base64 dell'hash SHA-256 del contenuto. Questo assicura che il browser scarichi nuovamente un file solo se il suo contenuto è cambiato.

Static asset delivery optimization

File	Original	Compressed	% Reduction
bootstrap.min.css	163	17.5	89.26%
jquery.js	89.6	28	68.75%
bootstrap.min.js	78.5	20	74.52%
Total	331.1	65.5	80.20%

File	Original	Compressed	% Reduction
fluent.js	384	73	80.99%
fluent.css	94	11	88.30%
Total	478	84	82.43%

Detect rendering location, interactivity, and assigned render mode at runtime

Le proprietà ComponentBase.RendererInfo e ComponentBase.AssignedRenderMode consentono di conoscere i dettagli di location,interactivity e e il render-mode associato al componente.

Improved server-side reconnection experience

- Miglioramenti della modalità (di default) di riconnessione nelle app InteractiveServer
  - Quando l'utente naviga dal tasto Back con un circuito disconnesso, la riconnessione viene tentata immediatamente invece di attendere la durata del successivo intervallo di riconnessione.
  - Quando un tentativo di riconnessione raggiunge il server ma quest'ultimo ha già rilasciato il circuito, viene eseguito un refresh automatico della pagina.
  - La tempistica di riconnessione utilizza una strategia di backoff calcolata. Per impostazione predefinita, i primi tentativi di riconnessione avvengono in rapida successione, seguiti poi da altri tentativi ad intervalli prolungati.

Constructor injection

- I componenti Razor supporteranno la dependency injection tramite il costruttore.
  - (Ad oggi era permessa solo tramite proprietà decorate con l'attributo [Inject])
  - Sarà possibile utilizzare il primary constructor

```
public partial class MyCustomComponent(NavigationManager navigation)
{
    private void HandleClick()
    {
        navigation.NavigateTo("/counter");
    }
}
```


TO BE
CONTINUED...

Grazie!

Andrea Dottor

.NET IN PILLOLE il podcast

@dottor

andreadottor

andreadottor

andrea@dottor.net