



1506
**UNIVERSITÀ
DEGLI STUDI
DI URBINO
CARLO BO**

UNIVERSITÀ DEGLI STUDI DI URBINO CARLO BO

Dipartimento di Scienze Pure e Applicate
Corso di Laurea in Informatica Applicata

Tesi di Laurea

COS'È BLAZOR?

Relatore:
Chiar.mo Prof. Emanuele Lattanzi

Candidato:
Francesco Belacca

Anno Accademico 2018-2019

A tutti quelli che mi hanno detto almeno una volta di non mollare.

Indice

1	Introduzione	1
1.1	Contesto	1
1.2	Problema	2
1.3	Linguaggi General-Purpose	3
1.4	Blazor	4
2	Modelli e Funzionamento	5
2.1	Blazor Server	5
2.1.1	BlazorPong	6
2.2	Blazor WebAssembly	8
2.2.1	WebAssembly	8
2.2.2	Blazor PWA	9
2.3	Modelli in roadmap	11
2.3.1	Blazor Hybrid	11
2.3.2	Blazor Native	12
2.4	Funzionamento	12
3	Scalabilità, Pro e Contro	15
3.1	Scalabilità	15
3.2	Blazor Server	16
3.2.1	Pro	16
3.2.2	Contro	16
3.3	Blazor WebAssembly	16
3.3.1	Pro	16
3.3.2	Contro	16
4	Conclusioni	17
4.1	Conclusioni	17
4.2	Futuro del progetto	17
	Bibliografia	18

Elenco delle figure

1.1	Implementazioni del .NET Standard	3
1.2	Possibilità di .NET	3
2.1	Blazor Server	5
2.2	BlazorPong	6
2.3	Blazor WebAssembly	8
2.4	Confronto tra WebAssembly e Javascript	9
2.5	Progressive Web Apps	10
2.6	Blazor Hybrid Application	11
2.7	File Razor[13]	12
2.8	Output[13]	13
2.9	Primitive di Blazor[13]	13

Capitolo 1

Introduzione

1.1 Contesto

A seguito della crescita esponenziale del web in questo secolo e dell'abituarsi di tutti coloro che ne usufruiscono ad un livello grafico sempre migliore e ad una esperienza mano a mano più interattiva e vicina all'utente medio, i siti web e le tecnologie utilizzate si sono adattati per permettere uno sviluppo sempre più rapido di codice più facilmente testabile e mantenibile.

Di conseguenza nel frontend si sono susseguiti una serie di framework e di strumenti, a partire da JQuery[2] nel 2006, che per primo si è occupato di risolvere il problema della compatibilità tra browsers, permettendo ai developers di scrivere una volta, e poter eseguire su tutti i browsers.

AngularJS nel 2010 è stato il primo MVC framework ad offrire in un unico pacchetto un insieme di features che hanno facilitato molto la vita agli sviluppatori, come il two-way data binding, la dependency injection, il routing integrato, ed altri strumenti utili per rendere più standard lo sviluppo nel frontend [1]. AngularJS pur essendo stato largamente utilizzato, è stato cambiato radicalmente nel 2013 e rinominato in Angular2 (e nelle versioni più recenti semplicemente Angular) senza mantenere retrocompatibilità e senza offrire un modo preciso per migrare alla nuova versione ai precedenti utilizzatori di AngularJS. Anche per questo motivo, React, un nuovo framework più leggero e modulare mantenuto da Facebook e la cui prima versione risale proprio al 2013, ha mano a mano preso il posto di Angular come framework più utilizzato nel frontend.

Vue infine è il terzo dei principali framework moderni per la creazione di interfacce utente che è stato fortemente adottato, proponendo una versione intermedia tra il fortemente opinionato Angular e il più flessibile React, lanciato nel 2014 [3].

Oltre a questi, ciascuno con la propria semantica, organizzazione logica dei folder, spesso una CLI dedicata, ad un developer frontend viene solitamente richiesto di conoscere HTML, CSS e chiaramente Javascript essendo

ció su cui si basano poi i vari framework, jQuery per retrocompatibilità e Bootstrap(versione 3 e 4) per creare componenti più gradevoli e più in fretta.

Oltre a Javascript, se si vuole scrivere degli unit test facilmente mantenibili, bisogna conoscere TypeScript(specialmente se si utilizza Angular, che rende il suo utilizzo obbligatorio). TypeScript in particolare è un linguaggio di programmazione che estende la sintassi di Javascript, rendendolo tipizzato(e non interpretato), e viene quindi transpilato in javascript per poter essere utilizzato dal browser.

1.2 Problema

I continui cambiamenti nei molti framework utilizzati, la diversità degli strumenti tra loro, che spesso realizzano in modo diverso la stessa cosa, rendono sempre più difficile per un junior developer iniziare a sviluppare, vista l'ampia curva di apprendimento e quindi il tempo e lo studio necessario, per poter essere pronto a lavorare come un professionista.

Oltretutto un web developer ad oggi finisce per essere costretto a scegliere se diventare uno sviluppatore frontend o backend, dato che rimanere al passo e aggiornarsi già in solo uno di questi due campi richiede molto tempo ed impegno, spesso oltre l'orario di lavoro.

É quindi chiaro che per tutti i web developer, e specialmente per una figura mista e più esperta, spesso identificata con il titolo "Full-Stack Developer", ci sia una continua ricerca del modo per rendere le proprie competenze quanto più trasversali possibile, anche in termini di tecnologie utilizzate.

1.3 Linguaggi General-Purpose

Microsoft, nel backend e in ambito applicativo, ha reso nel tempo il framework .NET e le sue implementazioni(.NET Core, .NET Framework e Mono) utilizzabili nei vari linguaggi per sviluppati da Microsoft stessa ovvero C#, F# e VB. La figura 1.1 di seguito riassume le implementazioni del .NET Standard e le varie tipologie di applicazioni che si possono sviluppare con ciascuna.

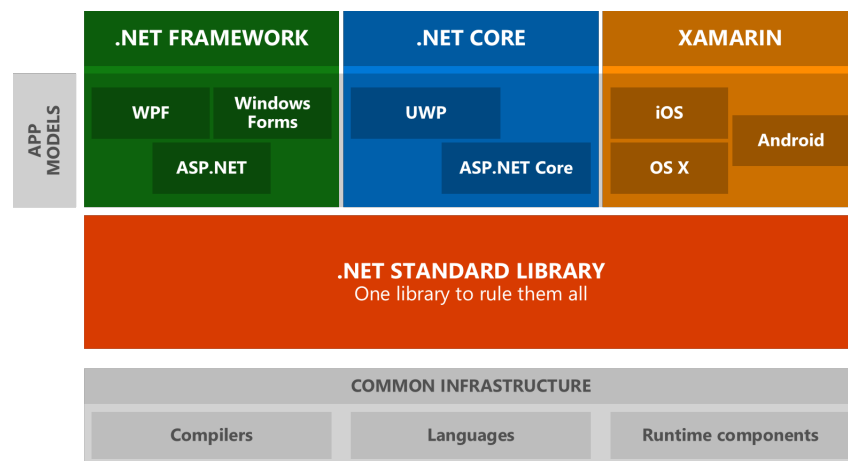


Figura 1.1: Implementazioni del .NET Standard

Scrivendo ad esempio in C# è possibile sviluppare vari tipi di applicazioni come si può vedere nella figura 1.2, ma se si decide di sviluppare codice per un applicazione web che gestisca eventi del Browser(come click, drag, ...) ad oggi si è di fatto costretti a scrivere utilizzando Javascript. Eventualmente utilizzando un framework se si vuole essere più veloci nello sviluppo e scrivere codice mantenibile specialmente se a livello enterprise.

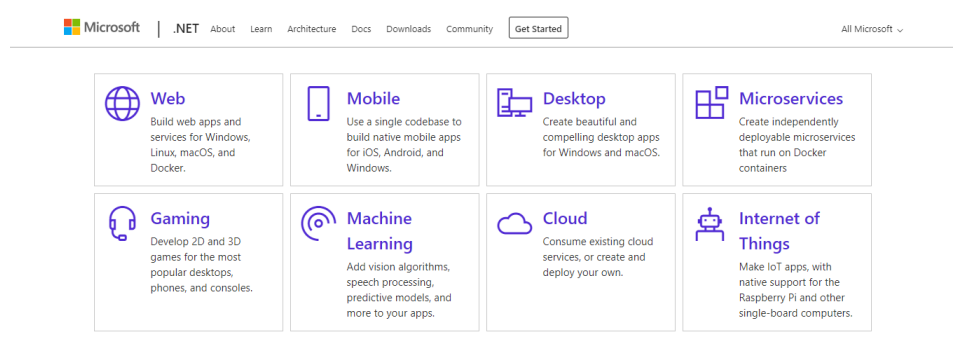


Figura 1.2: Possibilità di .NET

Già con Razor[4], Microsoft ha permesso la generazione di codice HTML e CSS in modo dinamico utilizzando C#, ma questa tecnologia per il server-side rendering è utilizzabile solo lato server e quindi la cattura di un evento client side come il click di un utente su un bottone, senza passare per il server sul quale viene eseguito l'hosting dell'applicazione, non è stato possibile utilizzando esclusivamente questa tecnologia.

1.4 Blazor

Ecco cosa è quindi Blazor: la versione più avanzata di Razor(Browser+Razor[?]), o meglio ancora un Framework per la creazione di User Interfaces(da qui in poi UI) di Single Page Application(da qui in poi SPA) che permette ai developer di gestire anche gli eventi client-side, direttamente in C# o nel linguaggio scelto tra quelli supportati. Questo framework come se il linguaggio scelto fosse effettivamente ciò che viene eseguito dal client, mentre in realtà ciò che viene eseguito lato client cambia a seconda del modello scelto, come poi vedremo più nel dettaglio.

Blazor utilizza per i vari modelli, delle tecnologie diverse(ad esempio il WebAssembly) ma che rispettano sempre lo standard del web.

Non è un plugin da installare, e non dipende da un browser specifico. Non è una tecnologia che sfrutta la transpilazione verso javascript, contrariamente a TypeScript.

Capitolo 2

Modelli e Funzionamento

2.1 Blazor Server

Il primo dei modelli ufficialmente rilasciati e per il quale si può ricevere supporto in produzione da settembre 2019[5], è Blazor Server.

Un'applicazione Blazor Server ospita i componenti Blazor lato Server e gestisce le interazioni dell'utente con la UI attraverso una connessione in tempo reale sfruttando SignalR, come visibile nella figura 2.1.

SignalR è una libreria di software open-source sviluppata da Microsoft, che facilita l'interazione tra Client e Server, specialmente quando si necessita di mandare messaggi dal server ad uno o più client con un'alta frequenza [6]. In particolare mette a disposizione delle API che permettono al codice che viene eseguito sul server di mandare notifiche real-time ad uno o più client connessi contemporaneamente, e questa libreria viene sfruttata da questo modello per trasmettere dal Server a ciascun Client connesso, che modifiche devono essere applicate al DOM perchè la UI di quel Client rispetti sempre la UI che il server mantiene in memoria per la specifica sessione di ciascun Client.

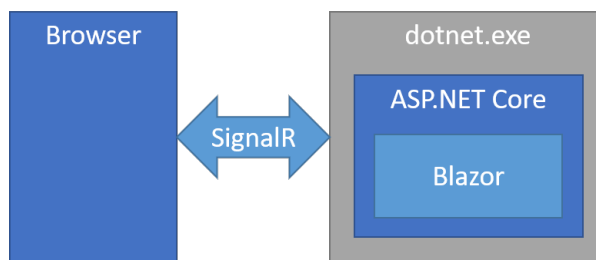


Figura 2.1: Blazor Server

Ciò significa che quando un utente scatena un evento, questo viene inviato attraverso la real time connection al server, dove il rispettivo componente di competenza gestisce l'evento. Quando l'evento è stato gestito, blazor compara

l'output appena generato con quello precedente all'evento, e manda quindi le sole differenze al browser del client, per poi applicarle al DOM.[7]

Blazor Server perciò necessita di una connessione stabile e a bassa latenza per funzionare al meglio, e gli scenari offline non sono supportati. Ciò significa anche che la posizione del server sul quale è ospitata l'applicazione non può essere troppo distante dal client che si sta connettendo per garantire un funzionamento senza lag.

Questo modello è particolarmente indicato quando si vuole delegare il costo computazionale al server e non ai client connessi, dato che ciò che il client deve computare è il solo codice statico scaricato inizialmente, e le differenze di volta in volta ricevute dal server, dove avviene invece la gestione di ogni evento e il calcolo del differenziale tra ciò che viene visualizzato prima di un evento e ciò che deve cambiare nella User-Interface dopo la sua gestione. Ciò rende molto veloce ed efficiente il download e l'avvio iniziale dell'applicazione lato client, il che lo rende il modello perfetto per funzionare su apparecchi a basso costo, che tuttavia devono sempre essere connessi ad internet.

2.1.1 BlazorPong

Un'esempio di applicazione scritta con questo modello, è BlazorPong, appositamente implementata per questo lavoro di tesi e la cui demo è disponibile sul relativo repository in GitHub[11].

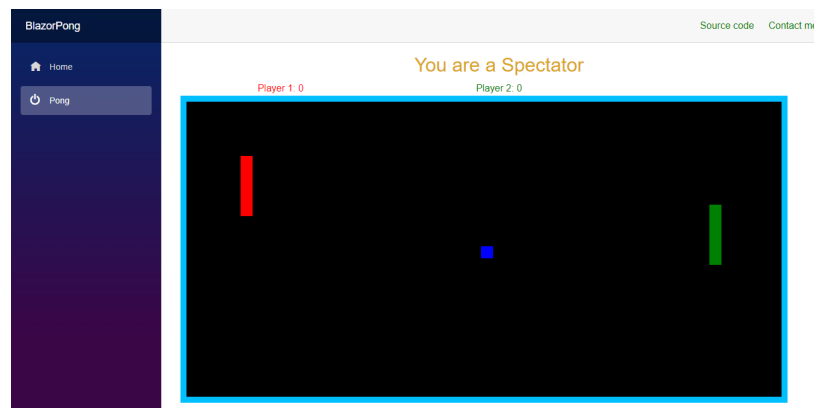


Figura 2.2: BlazorPong

Questa applicazione, visibile nella figura 2.2 permette a due giocatori che si collegano al sito contemporaneamente di giocare, e ai successivi utenti che si collegano di visionare la partita in corso come spettatori in tempo reale. In questa applicazione è stato utilizzato il modello Blazor Server side per vari motivi:

1. Il service worker che si occupa di aggiornare la posizione della pallina su tutti i client connessi, deve essere eseguito in un unico thread utilizzato

per tutti i client connessi al server, ad ognuno dei quali invece devono arrivare gli aggiornamenti.

2. Il calcolo delle differenze da applicare a ciascun DOM di ciascun utente connesso, per quanto avvenga continuamente(60fps circa), conviene avvenga direttamente lato server poichè l'applicazione è molto semplice e infatti anche un host di livello gratuito come quello che si è scelto di utilizzare riesce a far giocare senza problemi due persone con diversi spettatori(i test che sono stati fatti sono tutti stati eseguiti dall'interno dell'europa, poichè il server utilizzato si trova in Francia).
3. Essendo la gestione degli eventi server side, il gioco può essere visualizzato in modalità spettatore senza problemi anche su cellulari non performanti.

Il funzionamento è piuttosto semplice: una volta che l'applicazione è stata avviata e che ciascun player si è connesso al sito ed ha cliccato play, quindi quando entrambi gli utenti sono pronti a giocare, inizia la partita. Lato server viene gestito lo spostamento costante della pallina ed eventuali collisioni con muri verticali, orizzontali o con il blocco di uno dei player. Rispettivamente gli eventi gestiti comunque dall'unico background worker condiviso che è eseguito sul Server, sono quindi:

1. Collisione con un muro verticale con conseguente punto per il player che si trova dal lato opposto di quello in cui avviene la collisione;
2. Collisione con un muro orizzontale con conseguente inversione della velocità di spostamento della pallina sull'asse y;
3. Collisione con uno dei blocchi dei giocatori con conseguente inversione della velocità di spostamento della pallina sugli assi x ed y;
4. L'arrivo a 3 punti di uno dei due giocatori o la disconnessione di uno dei due, gestito con l'invio di un messaggio di fine partita contenente il player vincitore a tutti gli utenti connessi.

Ciò che invece avviene grazie all'utente, è lo spostamento del proprio player attraverso l'evento di drag del proprio blocco. Ad ogni evento scatenato dall'utente, la connessione SignalR invia l'evento al server, che lo processa e restituisce a tutti gli utenti connessi(compreso quello che ha scatenato l'evento) la differenza di CSS necessaria a far combaciare l'immagine visualizzata con la nuova posizione reale del blocco del player che si è mosso.

2.2 Blazor WebAssembly

Blazor WebAssembly è un modello attualmente in preview, che verrà ufficialmente rilasciato verso maggio del 2020.

In questo modello il codice della SPA viene scaricato ed eseguito nel browser del client che si collega alla SPA, come solitamente avviene quando si utilizza un framework moderno per UI come i già citati Angular, React e Vue.

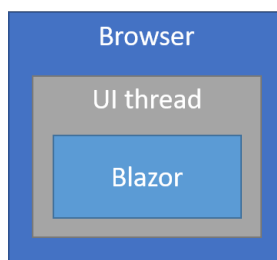


Figura 2.3: Blazor WebAssembly

Vengono quindi scaricati dal client l'applicazione Blazor, le sue dipendenze, ed il runtime del .NET scelto come target per l'applicazione. L'applicazione viene quindi eseguita direttamente nel thread della UI del Browser utilizzato, come visibile nella figura 2.2.

L'ambito di esecuzione è la stessa sandbox di qualsiasi altra applicazione scritta con javascript, ossia il browser che si sta utilizzando. Ciò è importante perchè implica(ed è così) che una web app scritta utilizzando Blazor non può fare niente di più o di meno di una web app standard. Ogni update alla UI e la relativa gestione, avvengono utilizzando lo stesso processo nel browser. Per questo modello, `blazor.WebAssembly.js` è il nome dello script Javascript che si occupa di scaricare il .NET runtime, l'applicazione e le dipendenze, come anche dell'inizializzazione dell'applicazione.

2.2.1 WebAssembly

In particolare il nome Blazor WebAssembly è stato scelto perchè utilizzato client il codice viene eseguito grazie al file `mono.wasm`, ossia la compilazione del Runtime del framework Mono in WebAssembly, e Mono è una delle implementazioni esistenti del framework di base .NET Standard. Il WebAssembly, che è un open standard che definisce un formato portatile di codice binario per programmi eseguibili e il rispettivo linguaggio assembly, come anche delle interfacce per facilitare le interazioni del codice con il proprio host. È anche detto il byte code del web, dato che dal 2017 i browser più diffusi al mondo si sono impegnati per svilupparlo ed adottarlo.[12]

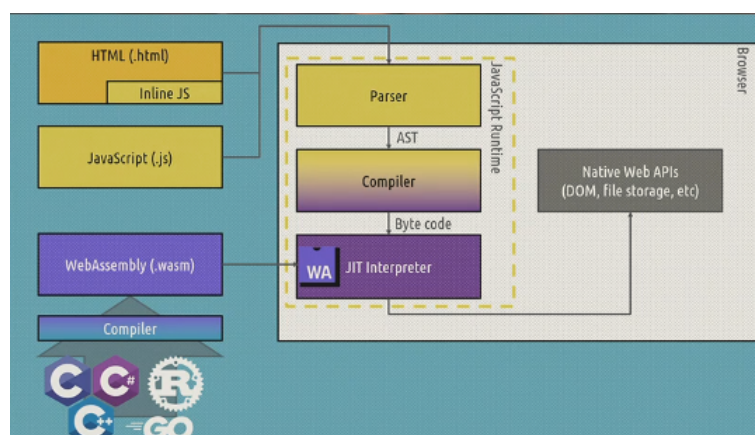


Figura 2.4: Confronto tra WebAssembly e Javascript

Come si può vedere nella figura 2.3, uno dei motivi per cui WASM è più veloce di Javascript è che non deve essere processato e compilato prima di poter essere interpretato, ma solamente scompattato ed eseguito.

La compilazione avviene in modalità AOT(Ahead of Time) e non più JIT(Just in Time), il che si traduce in un sensibile miglioramento delle prestazioni del codice dinamico. Il C# quindi non è l'unico linguaggio che potrà essere utilizzato come partenza per essere compilato in WASM, tuttavia questa tecnologia è ciò che rende possibile il funzionamento di Blazor.

In questo modello quindi, come visibile dai Chrome Developer Tools, vengono scaricate le DLL dell'applicazione direttamente nel browser dell'utente.

2.2.2 Blazor PWA

Lo step successivo per avvicinarsi al client allontanandosi dal modello server è poi Blazor PWA.

È così chiamato perché in questo modello Blazor, permette di sviluppare l'interfaccia utente di una Progressive Web App. Nella figura 2.3 viene riassunto cosa sia una Progressive Web App.

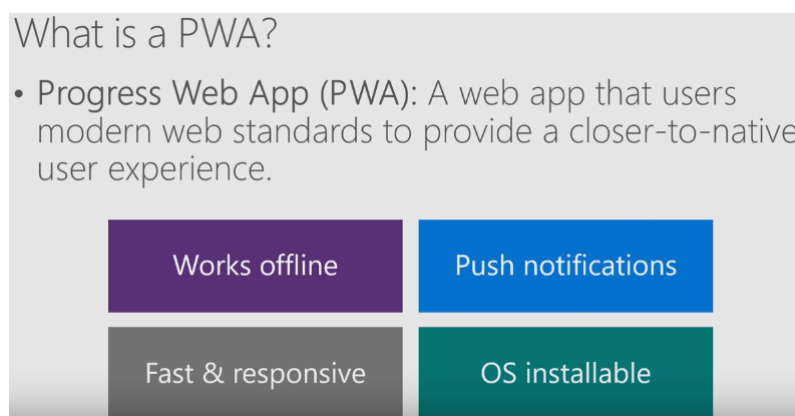


Figura 2.5: Progressive Web Apps

In particolare queste sono applicazioni web che hanno la capacità di funzionare anche offline, e che spesso possono essere scaricate in modo persistente sulla macchina dell'utente che le esegue. Offrono chiaramente una maggiore velocità di esecuzione e la possibilità di sfruttare alcune API native. Ad esempio possono essere utilizzate quando la necessità è quella di utilizzare le notifiche push native del SO che sta utilizzando il client (e.g. Windows).

Per realizzarne una in Blazor al momento, bisogna partire dal modello Blazor WebAssembly aggiungendo un manifesto che descriva le capacità dell'applicazione, i permessi richiesti e l'icona da utilizzare una volta installata, oltre chiaramente a dover implementare l'applicazione in modo che possa lavorare anche offline, basandosi su un service worker.[8]

2.3 Modelli in roadmap

2.3.1 Blazor Hybrid

Questo modello di Blazor ed anche il successivo, servono a sviluppare applicazioni native. Nel modello Hybrid, l'applicazione sviluppata non è quindi più considerabile una web app ma rimane ibrida perchè pur essendo un app nativa, utilizza tecnologie web per effettuare il rendering della UI.

Esempi di Hybrid Apps possono essere applicazioni mobile native, che hanno accesso alle API esposte ad esempio da Android, ma che utilizzano delle WebViews per la gestione dell'interazione dell'utente con la UI.

Un'altro esempio molto interessante sono le applicazioni che sfruttano Electron, che permette di compilare un'applicazione web in un'applicazione desktop cross platform. Infatti utilizzando Electron si può creare un'applicazione nativa, con l'interfaccia utente scritta utilizzando blazor, facendo in modo che in fase di esecuzione il processo host sia .NET Core(avendo quindi pieno accesso alle capacità native e ad esempio al file system) pur rimanendo cross platform e potendo quindi eseguire su Windows, Linux e Mac. Di seguito nell'immagine 2.4 si può vedere una Web Application compilata nativamente con Electron(con target Windows), e quindi eseguita come applicazione desktop:

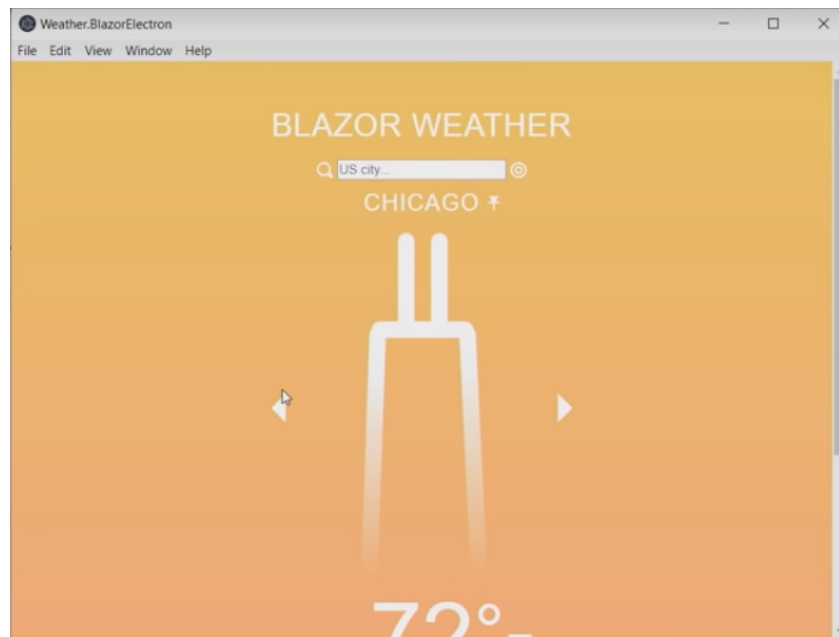


Figura 2.6: Blazor Hybrid Application

Il codice open source dell'applicazione qui sopra, si trovare al seguente link:
<https://github.com/danroth27/BlazorWeather/tree/master/BlazorWeather.Electron>

2.3.2 Blazor Native

Infine esiste il modello Native che è possibile grazie al fatto che Blazor è stato architettato per poter renderizzare controlli della UI che non siano obbligatoriamente strumenti web, e può quindi integrarsi con controlli nativi. Il rendering layer è infatti intercambiabile, pur essendo quello di default dedicato all'HTML.

Un esempio di applicazione sviluppata utilizzando Blazor per il rendering di controlli nativi nella UI, si può vedere durante la presentazione di Steve Sanderson all'evento NDC Oslo di quest'anno, pur non essendo stato ancora rilasciato il codice di un esempio ufficiale.[10] In questa applicazione, si è scelto di sostituire il default rendering layer per utilizzarne uno custom, utilizzando componenti di Flutter, il toolkit di Google per costruire interfacce utente native CrossPlatform. Questo modello viene qui citato per completezza, ma al momento non è nemmeno presente nella documentazione ufficiale ed è solo stato citato da Daniel Roth durante la presentazione dei futuri modelli di Blazor lato client.[9]

2.4 Funzionamento

Partendo dal codice, come già anticipato quando si scrive si utilizza un mix di HTML per lo scheletro, CSS per lo stile del documento e C# preceduto dal relativo carattere di escape(@) per la parte dinamica del codice.

Writing Blazor Code

```
<div>
  <FancySpan IsBold="@ (currentCount % 2 == 0)">So exciting</FancySpan>
  <p>Current count: @currentCount</p>
</div>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
    }
}
```

Figura 2.7: File Razor[13]

Si può scrivere utilizzando questa sintassi solamente nei file che hanno estensione ".razor", il cui esempio si può trovare in figura 2.7. Quando l'applicazione viene compilata, i file .razor vengono utilizzati come input per generare

dei file C#(quindi con estensione .cs) equivalenti al codice che si è descritto.[13]
Un esempio di output di file razor compilato, generato a partire da quello in figura 2.7 è visibile in figura 2.8.

What does the compiler output

```
__builder.OpenElement(1, "div");
__builder.AddMarkupContent(2, "\r\n  ");
__builder.OpenComponent<code_snippets.Shared.FancySpan>(3);
__builder.AddAttribute(4, "IsBold", currentCount % 2 == 0);
__builder.AddAttribute(5, "ChildContent", (Microsoft.AspNetCore.Components.RenderFragment)((__builder2) => {
    __builder2.AddContent(6, "So exciting");
}
));
__builder.CloseComponent();
__builder.AddMarkupContent(7, "\r\n  ");
__builder.OpenElement(8, "p");
__builder.AddContent(9, "Current count: ");
__builder.AddContent(10, currentCount);
__builder.CloseElement();
__builder.AddMarkupContent(11, "\r\n");
__builder.CloseElement();
```

Figura 2.8: Output[13]

Saranno poi questi file ad essere effettivamente utilizzati da Roslyn(chè è il nome del compilatore open-source del linguaggio C#) e a finire nella DLL generata come output dal progetto nel quale si trova il file .razor di partenza. Questo passaggio non è solo necessario per fare in modo che si possano generare le DLL compilate relative ai file di partenza, ma è anche il momento in cui viene ottimizzato ciò che ha scritto il developer per ogni parte relativa alla UI in ogni file .razor del progetto, riducendolo alle sole primitive di Blazor, che sono visibili nella figura 2.9.

Primitives of components

-  Elements
-  Components
-  Attributes
-  Content
-  High level constructs like events and @bind map to attributes (mostly).

Figura 2.9: Primitive di Blazor[13]

In ordine quindi possiamo descrivere le primitive:

1. Un elemento è ad esempio l'elemento html "div";
2. Un component è un altro componente(file .razor) che viene utilizzato nel file che si sta compilando;
3. Un attributo è appunto un attributo di un elemento html, o un parametro passato in ingresso ad un component;
4. Un contenuto è del testo, costante o basato su un parametro, inserito all'interno di un elemento html;
5. L'ultima primitiva di un componente Blazor sono sostanzialmente gli eventi che vengono gestiti dal componente che si sta compilando e le direttive di bind che quindi collegano e mantengono sincronizzate due variabili.

Capitolo 3

Scalabilità, Pro e Contro

3.1 Scalabilità

Il problema della Scalabilità, ossia della capacità dell'applicazione di resistere all'aumentare del numero di utenti concorrenti che ne usufruiscono, dipende dal modello scelto, come anche dalla tipologia di applicazine sviluppata. Non è quindi risolvibile banalmente e non esiste una scelta universalmente giusta.

Il problema della scalabilità comunque è fondamentalmente diverso tra Blazor Server, e tutti gli altri modelli. Questo principalmente per 3 motivi:

1. Il primo è che Blazor Server delega completamente al Server che ospita l'applicazione il carico computazionale necessario per gestire ogni singolo evento della UI di ogni utente connesso, compreso il salvataggio in-memory dello stato di ciascuna UI nel tempo durante l'utilizzo, visto che ad ogni Client connesso (o meglio allo specifico DOM) arrivano solamente le minime differenze necessarie rispettper generare la ui corretta rispetto al frame precedente, ogni volta che deve cambiare qualcosa. Ciò implica che la potenza del server debba tener conto di eventuali picchi di utenze, e debba avere a disposizione sufficiente RAM per poter mantenere in memoria lo stato dell'applicazione di ciascun utente concorrente connesso.
2. Il secondo è che Blazor Server, necessita di una connessione sempre attiva con ogni utente collegato.
3. Il terzo è che questo modello sfrutta SignalR, che per funzionare al meglio utilizza il protocollo di trasporto WebSocket, quindi la macchina server sul quale viene ospitata l'applicazione Blazor è consigliato che lo supporti.

Gli altri modelli invece sfruttano l'hardware di ogni client connesso, come solitamente avviene per le SPA odierne. Sono quindi in linea con il comportamento degli altri framework già citati e come Angular React o Vue.js,

in particolare Blazor WebAssembly, che quindi prenderemo in considerazione confrontandolo con Blazor Server.

3.2 Blazor Server

3.2.1 Pro

3.2.2 Contro

3.3 Blazor WebAssembly

3.3.1 Pro

3.3.2 Contro

Capitolo 4

Conclusioni

4.1 Conclusioni

4.2 Futuro del progetto

Bibliografia

- [1] M. Wanyoike, Disponibile: <https://blog.logrocket.com/history-of-frontend-frameworks>.
- [2] Disponibile: <https://en.wikipedia.org/wiki/JQuery>.
- [3] Disponibile: <https://it.wikipedia.org/wiki/Vue.js>
- [4] R. Anderson, R. Nowak, Disponibile: <https://docs.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-3.0&tabs=visual-studio>
- [5] D. Roth, Disponibile: <https://devblogs.microsoft.com/aspnet/asp-net-core-and-blazor-updates-in-net-core-3-0>.
- [6] Disponibile: <https://dotnet.microsoft.com/apps/aspnet/signalr>
- [7] D. Roth, Disponibile: <https://devblogs.microsoft.com/aspnet/blazor-server-in-net-core-3-0-scenarios-and-performance>.
- [8] D. Roth, Disponibile: <https://youtu.be/qF6ixMjCzHA?t=485>
- [9] D. Roth, Disponibile: <https://youtu.be/qF6ixMjCzHA?t=872>
- [10] S. Sanderson Disponibile: <https://www.aka.ms/blutter>
- [11] Disponibile: <https://github.com/MACEL94/BlazorPong>
- [12] Disponibile: <https://webassembly.org/roadmap/>
- [13] S. Sanderson, R. Nowak
Disponibile: <https://www.youtube.com/watch?v=dCgqTDki-VM>
- [14] Disponibile: <https://github.com/aspnet/Blazor/wiki/FAQ#q-where-does-the-name-blazor-come-from>

Vorrei ringraziare il professor Lattanzi per avermi aiutato a scrivere questa tesi, ed i miei genitori per avermi dato la possibilità di studiare ciò che volevo e lavorare quando l'ho scelto senza condizionarmi.

Infine un ringraziamento speciale va alla mia ragazza Ivana per aver creduto in me e nelle mie capacità anche quando non l'ho fatto io.