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

Elenco delle figure

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 tantissimo la vita ai developers, come il two-way data binding, la dependency injection, il routin, oltre ad altri strumenti utili per rendere più standard lo sviluppo nel frontend [1]. Questo framework largamente utilizzato, é stato riscritto nel 2013 diventando Angular 2 (e nelle versioni piú recenti rinominato semplicemente in Angular) senza mantenere retrocompatibilità e senza offrire un modo preciso per migrare alla nuova versione agli utilizzatori di AngularJS. Anche per questo React, un nuovo framework più leggero e modulare sviluppato dagli sviluppatori di Facebook, ha preso il posto di Angular come framework più utilizzato nel frontend.

Vue infine é il terzo dei principali framework che ha provato a prendere piede proponendo una versione intermedia tra il fortemente opinionato Angular e il piú flessibile React.

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.

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) e degli altri framework che facilitino i test(Enzyme, Karma + Jasmine, ...).

1.2 Problema

I continui cambiamenti nei molti framework utilizzati, la diversità degli strumenti stessi tra loro, che spesso realizzano in modo diverso tutti la stessa cosa, rendono sempre piú difficile per un junior developer iniziare a sviluppare, vista l'ampia curva di apprendimento e lo studio necessario, per poter essere spendibile a livello professionale.

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 tempo e non è scontato ad esempio che venga concesso di poterlo fare in orario lavorativo, pur essendo fondamentale.

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

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 propri linguaggi, C#, F# e VB. La figura 1.1 qui di seguito riassume le implementazioni del .NET Standard e le varie tipologie di applicazioni che si possono sviluppare con ciascuna.

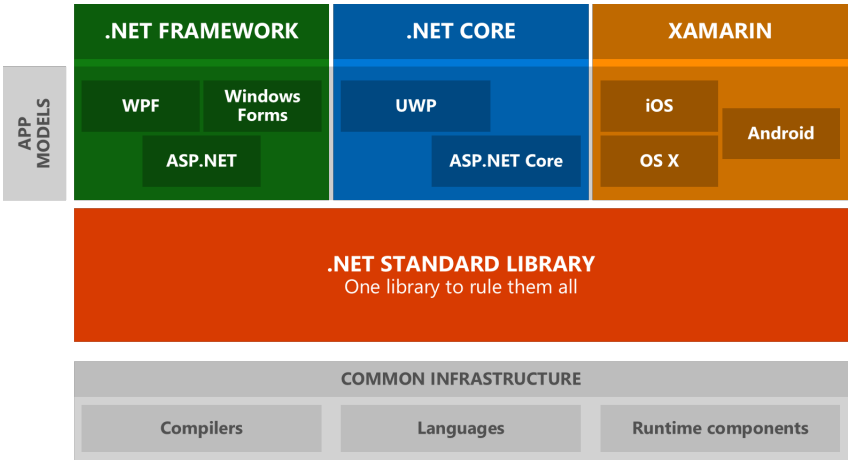

Figura 1.1: Implementazioni di .NET

Scrivendo ad esempio in C# è possibile sviluppare vari tipi di applicazioni come si puó vedere nella figura 1.2, ma se si decide di sviluppare codice per un applicazione client web ad oggi si è ancora costretti a scrivere utilizzando Javascript e un suo framework se si vuole essere veloci nello sviluppo e scrivere codice mantenibile specialmente in team piú grandi, come nel mondo enterprise.

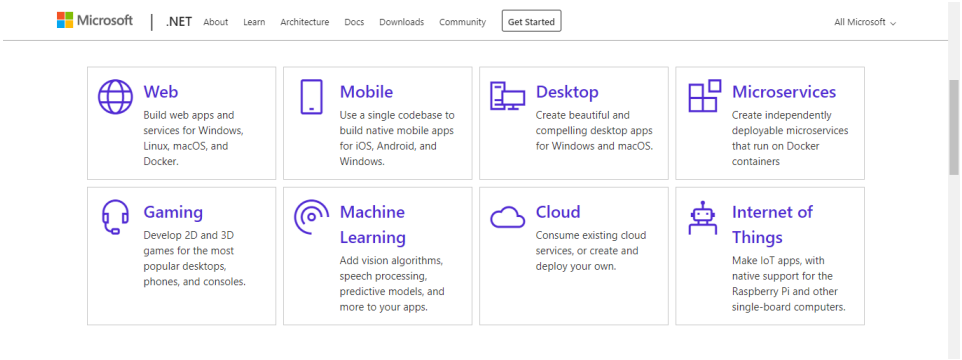

Figura 1.2: Possibilitá di .NET

Giá con Razor[3], Microsoft ha provato a permettere la generazione di codice HTML e CSS in modo dinamico utilizzando C#, ma è utilizzabile solo lato server e quindi ad esempio la cattura di un evento client side come il click di un utente su un bottone senza contattare il server non è gestibile utilizzando il solo C#.

Ecco cosa é quindi Blazor: la versione successiva di Razor(Web-Razor) che permette ai developer di gestire anche gli eventi client-side, direttamente in C# come se questo fosse effettivamente ció che viene eseguito lato client, mentre in reatá l'esecuzione lato client cambia a seconda del modello scelto, come poi vedremo piú nel dettaglio.

Capitolo 2

Modelli e Funzionamento

2.1 Blazor Server

Il primo dei modelli ufficialmente rilasciati e per il quale si puó ricevere supporto in produzione da settembre 2019[4], é proprio questo.

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.

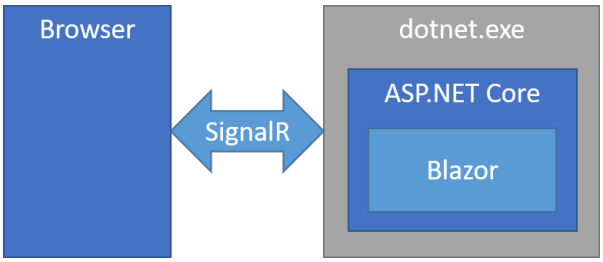

Figura 2.1: Blazor Server

Quando un utente scatena un evento, questo viene inviato attraverso la RTC al server, dove i vari componenti gestiscono l'evento. Quando l'evento é stato gestito, blazor compara l'output generato con quello precedente l'evento, e manda quindi le sole differenze al browser del client, per poi applicarle al DOM.[5]

Blazor Server quindi necessita di una connessione stabile e a bassa latenza per funzionare al meglio, e gli scenari offline non sono supportati.

É particolarmente indicato quando si vuole delegare il costo computazionale al server e non ai client connessi, dato che ció che il client esegue é il solo codice statico e le differenze di volta in volta inviate ma calcolate lato server. Ció rende molto veloce ed efficiente l'avvio dell'applicazione e in particolare il suo caricamento iniziale lato client, che rende il modello perfetto per funzionare su apparecchi a basso costo.

2.2 Blazor WebAssembly

Blazor WebAssembly é un modello attualmente in preview, che verrá ufficialmente rilasciato nella prima parte del 2020.

In questo modello il codice della SPA viene eseguito completamente lato client come solitamente avviene quando si utilizza un framework moderno per UI basato su JS, come i giá citati Angular,React,Vue.

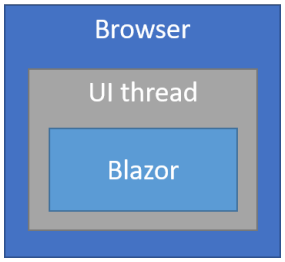

Figura 2.2: Blazor Webassembly

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 User Interface del Browser utilizzato, come visibile nella figura 2.2.

Quindi esegue nella stessa sandbox di qualsiasi altra applicazione javascript, e non puó fare niente di piú o di meno.

Ogni update alla UI e la gestione di ciascuno, avvengono utilizzando lo stesso processo nel browser.

Per questo modello, blazor.webassembly.js é il nome dello script Javascript che si occupa di scaricare il .NET runtime, l'applicazione e le dipendenze, come anche dell'inizializzazione dell'applicazione.

In particolare il nome Blazor WebAssembly é stato scelto perché il WebAssembly é il byte code del web che dal 2015 i piú diffusi browser al mondo si sono impegnati per sviluppare ed adottare.

2.3 Blazor PWA

2.4 Blazor Hybrid

2.5 Blazor Native

Capitolo 3

Scalabilitá, Pro e Contro

3.1 Scalabilitá

3.2 Pro

3.3 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://docs.microsoft.com/en-us/aspnet/core/razor-pages/?view=aspnetcore-3.0&tabs=visual-studio.

[4] Disponibile: https://devblogs.microsoft.com/aspnet/asp-net-core-and-blazor-updates-in-net-core-3-0.

[5] Disponibile: https://devblogs.microsoft.com/aspnet/blazor-server-in-net-core-3-0-scenarios-and-performance.

Ringraziamenti

Vorrei ringraziare il professor Lattanzi per avermi aiutato a scrivere questa tesi, i miei genitori per avermi permesso di studiare ci'o che ho voluto, e la mia ragazza Ivana per aver creduto in me e nelle mie capacitá anche quando non l'ho fatto io.