HTML Mobile App e Web App: il web ha bisogno delle VIEW

Come avrai notato sono un ossessionato delle **prestazioni di rendering** e **della fluidità *lato-client,*** soprattutto per il fatto che il numero di coloro che navigano da dispositivi mobile, molto meno potenti per caratteristiche hardware rispetto a quelli desktop, è cresciuto incredibilmente, e lo sta ancora facendo.

Diventa pertanto fondamentale realizzare dei prodotti che possano essere utilizzati in maniera piacevole da parte dell'utente.

Forse avrai già sentito parlare del ***pattern MVC*** per la realizzazione di **Single Page App** e sicuramente già saprai che esistono oltre **50 javascript framework** di cui ti puoi servire per sviluppare progetti *MVC-based.*

Se vuoi saperne di più sui patterns MVC ti consiglio di leggere questo mio articolo:

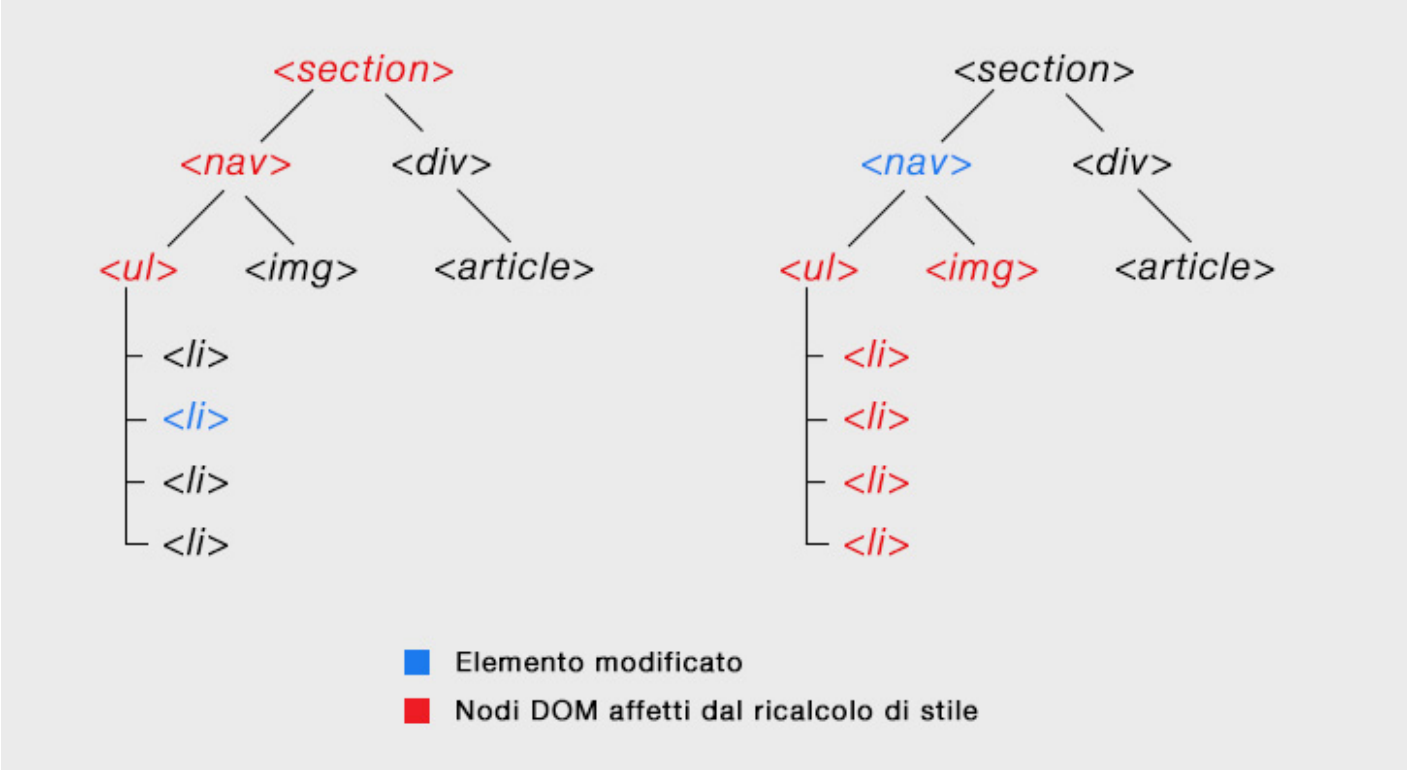
- Creare applicazioni web: il pattern Model-View-Controller ed i principali MVC javascript frameworks

Una delle cose che accomuna tutti quei frameworks è la possibilità di **suddividere l'interfaccia** in ***views*** **separate,** e renderne la **struttura altamente scalabile** e riutilizzabile, oltre che velocizzare il flusso di lavoro attraverso un'adeguata logica.

Ok, ma cosa c'entrano **fluidità** ed **esperienza-utente** con la suddivisione in **views** separate?

Come abbiamo visto nel secondo punto di un precedente articolo, il processo di **rendering** di qualsiasi **browser moderno,** è condizionato anche dal numero di elementi **DOM nidificati** tra di loro.

Riassumendo, più è complessa e profonda la struttura ad albero DOM del documento, più lente saranno le prestazioni dell'interfaccia.



Perché questo?

Ogni volta che un elemento DOM subisce delle modifiche, il browser si preoccupa di **ricalcolare tutti gli stili non solo di quell'elemento,** ma anche di tutti quelli che lo contengono o che sono in esso contenuti.

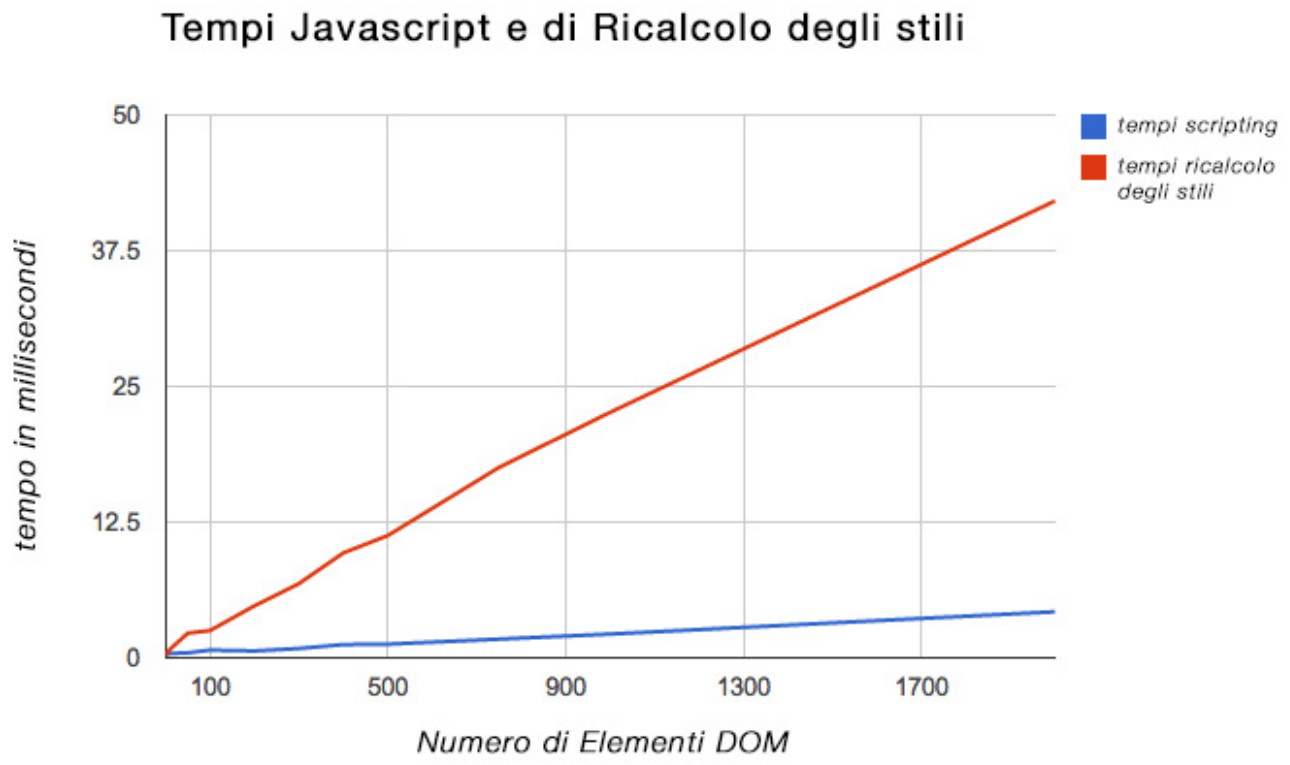
Ovviamente, più l'elemento si trova in profondità e maggiore saranno i nodi dei livelli superiori soggetti al **ricalcolo di stile;** allo stesso tempo, più **livelli DOM** saranno in esso presenti, maggiori saranno i **nodi dei livelli** contenuti affetti dal ricalcolo.

Inoltre, il **processo di ricalcolo** per ogni elemento si compone di **3 fasi principali** e, a seconda del tipo di modifica effettuata, può accadere che il browser ripercorra solo uno di quegli step oppure tutte le fasi per intero, con ripercussioni incredibilmente negative sulle prestazioni (tutto ciò può facilmente causare un ***overload della memoria*** e conseguente crash dell'applicazione, soprattutto nel caso di dispositivi mobile).

Per conoscere meglio le fasi del processo di rendering e quali attributi di stile sono associati ad ognuna di esse, ti consiglio di leggere questo articolo:

Performance CSS3: consigli per ottimizzare le animazioni

Ecco un grafico che mostra come i **tempi di ricalcolo degli stili** aumentano in maniera proporzionale al numero degli elementi DOM interessati dalle modifiche di stile:



Per fortuna ci sono le view!

Come abbiamo detto, uno dei vantaggi principali offerti dai **framework javascript MVC** è proprio la possibilità di organizzare **l'interfaccia** di qualunque applicazione mobile in ***views.***

Suddividere le parti delle nostre **HTML Mobile App** o **Desktop Web App,** ci permette di tenere sotto controllo due aspetti fondamentali:

1. limitare la **profondità** dei **livelli DOM**
2. limitare il **numero** di **livelli DOM**

I problemi dello sviluppo “senza logica”

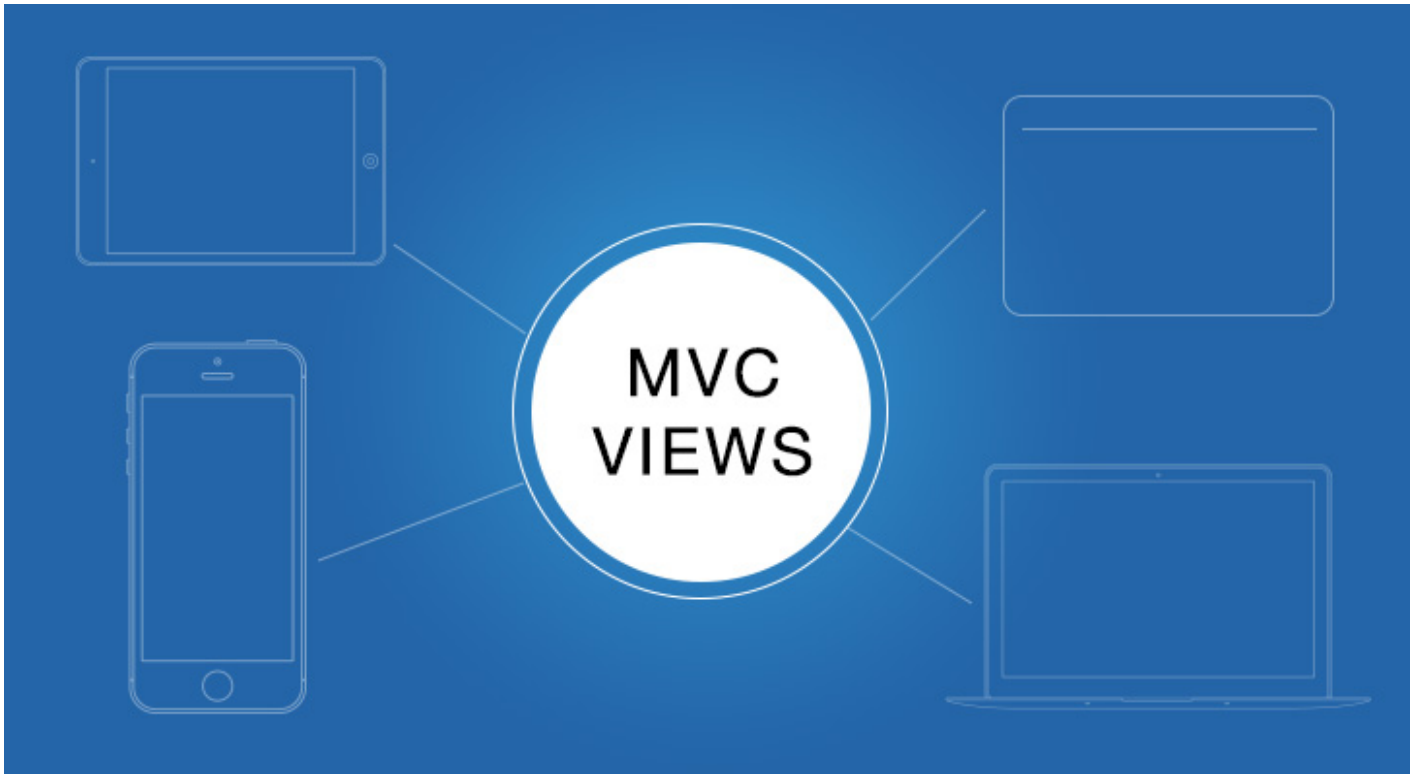
In una normale **applicazione javascript** realizzata senza nessuna **logica MVC** solitamente abbiamo due possibilità per creare il **markup dell'interfaccia:**

- Il primo è quello di scrivere subito il **markup di tutti i componenti** nel **documento iniziale,** per poi nascondere o visualizzare gli elementi che ci interessano a seconda delle esigenze
- il secondo è quello di aggiungere il **markup via javascript,** senza seguire **nessuna logica,** lasciando il più delle volte l'elemento **DOM** di ogni controllo creato all'interno del documento, anche quando esso non viene più utilizzato.

Entrambi i casi rappresentano una pessima metodologia di sviluppo: avremo sempre un altissimo numero di elementi DOM presenti contemporaneamente all'interno del documento, oltre al fatto che per utilizzare uno stesso elemento in diverse zone, magari contemporaneamente, bisogna per forza scrivere più volte il **markup HTML** che lo rappresenta, e richiamare il **codice javascript necessario** per abilitarne le funzionalità.

Ne puoi dedurre che anche la gestione del codice da parte dello sviluppatore diventa pressoché complicata!

I vantaggi dello sviluppo organizzato in views



I frameworks MVC sono pensati in maniera tale che **ogni view all'inizio venga solamente dichiarata,** per poi essere aggiunta al DOM del documento solo quando è utilizzata effettivamente.

Tale view **resterà poi "in ascolto" in modo autonomo** su eventuali cambiamenti apportati ai dati o ad altri elementi DOM, preoccupandosi da sola di **rimuovere** dal documento **il markup** ad essa relativo quando il suo utilizzo non è più richiesto dall'interfaccia (ovviamente si ha la possibilità di rimuoverla anche dall'esterno).

Ecco il codice di una view in backbone.js che resta in ascolto per eventuali modifiche:

Così avremo la possibilità di **limitare il numero di elementi DOM presenti contemporaneamente nel documento** (ti consiglio di dare ancora una sbirciata all'asse delle ascisse del grafico).

Inoltre, siccome una stessa *view* dichiarata può essere utilizzata in più parti dell'interfaccia, è pratica comune fare in modo che il markup HTML di ogni view venga aggiunto direttamente nel tag *, r**iducendo notevolmente la profondità del DOM del documento** (ci sono molti casi in cui dobbiamo per forza di cose inserire una view all'interno di un'altra, ma il livello di profondità del DOM rimane generalmente poco elevato).*

Due vantaggi quindi molto importanti, soprattutto considerato il fatto che per poter usare una ***view*** in più parti dell'interfaccia sarà sufficiente dichiararla una volta soltanto, per poi richiamarne una nuova istanza ogni volta che ne abbiamo bisogno.

Conclusioni

Nonostante il grandissimo supporto di cui possiamo disporre grazie a **frameworks** esterni, non esiste ancora un metodo per dichiarare l'utilizzo delle View in modo semantico.

E non sto parlando dal punto di vista dell'indicizzazione **SEO** o della leggibilità da parte dell'utente, tanto di come poter comunicare a Chrome, Safari, Firefox, Internet Explorer, Opera ecc., che una precisa parte dell'interfaccia rappresenta una **view,** in modo che il browser possa prevenire il calcolo degli stili quando altre **views** subiscono delle modifiche.

Tuttavia l'utilizzo delle view nella realizzazione di **interfacce web,** soprattutto per quanto riguarda **HTML5 Mobile App** e **Single Page App**, è un aspetto che non possiamo sottovalutare.

Ne è una dimostrazione il fatto che **tutti i linguaggi nativi** *iOS, Android, Windows* e perfino l'ormai (quasi del tutto) abbandonato Flash, sono costruiti secondo la logica dell'MVC e dell'utilizzo di views.

Per fortuna, grazie a tutti i numerosi **frameworks** MVC **presenti on-line** abbiamo un'ottima soluzione a questo problema, e possiamo utilizzare le view anche su prodotti **web-based.**

Utilizzando il **pattern MVC** e le **web views** insieme a molte altre tecniche ho creato per i miei clienti oltre 10 applicazioni mobile, tra cui due progetti personali: Tint Weather App e Dieta SI o NO?.

Se anche tu sei un **web designer** e vuoi realizzare la tua applicazione mobile da distribuire nei vari stores, forse sarai interessato al mio e-book: HTML Mobile Accelerato.

Grazie ad esso verrai a conoscenza di tutte quelle tecniche che renderanno la tua mobile app fluida e reattiva come quelle native.

E tu? Hai mai utilizzato framweworks MVC per la programmazione lato-client di Single Page Apps? Cosa ne pensi?