UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Dipartimento di Informatica e Comunicazione

Tesi Di Laurea Triennale

Dal Web al Mobile - L'evoluzione delle Tecnologie web per lo sviluppo di applicazioni multipiattaforma

---

Autore:
Marco Predari

Relatore:
Dott. Paolo Ceravolo
Correlatore:
Dott. Christian Nastasi

Anno Accademico 2013/2014

"L'intelligenza è l'abilità di riuscire ad evitare il lavoro,
portando comunque a compimento i propri compiti."
Linus Torvalds

Abstract

Dal Web al Mobile - L'evoluzione delle Tecnologie web per lo sviluppo di applicazioni multipiattaforma

I dispositivi mobili, al giorno d'oggi, rivestono un ruolo sempre più importante tanto nelle aziende quando nella nostra vita privata, permettendoci di compiere operazioni e svolgere dei compiti che, fino a qualche anno fa, erano eseguibili solo attraverso un normale PC. Con il passare del tempo, l'evoluzione tecnologica che ha accompagnato lo sviluppo dei normali PC, ha coinvolto i dispositivi così detti "mobili". Evoluzione che li ha trasformati da semplici organizer "da tasca" a veri e propri terminali ricchi di funzionalità, discreta potenza di calcolo ma soprattutto di connettività. Quest'ultima caratteristica li ha resi estremamente versatili soprattutto con l'arrivo di applicazioni di tipo aziendale e di produttività personale.

Dai dati sull'uso dei differenti device utilizzati per accedere a internet, risulta che il 66,4% del tempo totale speso online è generato dalla fruizione di internet da mobile e, più in dettaglio, il 55,7% del totale dalla fruizione tramite mobile applications.
[1]

Il compito dello sviluppatore, è quello di realizzare applicazioni non solo funzionanti nel senso stretto del termine ma funzionali e usabili su questa tipologia di dispositivi. Inoltre se si vuole far conoscere il prodotto ad un numero sempre più elevato di utenti, ci si deve anche preoccupare che l'applicazione sia disponibile su diverse o tutte le piattaforme, in quanto il mercato delle applicazioni dei dispositivi mobili è abbastanza frammentato.

Esistono diversi modi per sviluppare una applicazione mobile, Nativa, Web o ibrida: ogni approccio ha vantaggi e svantaggi, e la scelta può limitare le opzioni degli strumenti di sviluppo in un secondo momento.

App native Sviluppare un'app utilizzando l'interfaccia e il linguaggio di programmazione per un determinato dispositivo e sistema operativo. Ciò può fornire le prestazioni migliori ma richiede una versione differente (costosa) per ogni sistema operativo.

App Web Si tratta di sviluppare un sito web che abbia le sembianze di una applicazione vera e propria che possa essere raggiunta tramite un semplice browser web, anche se la frammentazione dell'attuale mercato fa si che i browser comunemente disponibili sui vari dispositivi non supportino in modo uniforme gli attuali standard del web. Con questo approccio inoltre non si ha la possibilità di accedere alle funzionalità del dispositivo come ad esempio fotocamera, contatti, accelerometro, etc ....

App ibride Un compromesso tra nativa e Web. Lo sviluppo avviene utilizzando i linguaggi di programmazione Web standard di settore, come HTML5 e JavaScript quindi viene creato un pacchetto di installazione nativa (es. apk file) per la distribuzione tramite app store. Esistono anche altri metodi ma questo è uno di quelli più usati. E' cosi possibile ridurre i costi con il riutilizzo del codice.

L'obiettivo di questo lavoro e quello di trovare un metodo di sviluppo veloce per avere la stessa applicazione disponibile nelle diverse piattaforme. Ovviamente ci sono più percorsi che ci possono portare allo stesso risultato, la scelta della strada da intraprendere dipende da molti fattori che si vedrà essere di vario genere.

Questa tesi si propone l'obiettivo di analizzare le tecnologie sul mercato attuale e proporre un modello di sviluppo rapido di applicazioni multi-piattaforma per dispositivi mobili, attraverso l'utilizzo di tecnologie web, mostrando vantaggi e svantaggi che si possono riscontrare rispetto agli altri metodi di sviluppo.

Ringraziamenti

A mia madre Giusy a mio padre Angelo a mio fratello Dario. Ringrazio la mia famiglia per avermi sostenuto e incoraggiato a seguire le mie passioni nel mio percorso formativo, senza dei quali nulla sarebbe stato possibile.

Ringrazio i compagni universitari più vicini e gli amici del collegio Modena, per l'ottimo tempo trascorso assieme durante gli studi.

Ringrazio la ditta BigThink SRL per l'opportunità di tirocinio data per la mia tesi.

Infine ringrazio pippo, pluto, paperino e topolino per l'aiuto nei test del codice.

Indice dei Contenuti

Abstract ii

Ringraziamenti iv

Elenco delle Figure vii

Elenco delle Tabelle viii

Elenco degli Esempi ix

1 Stato dell'arte 1
- 1.1 Sviluppo Nativo . . . 1
- 1.2 Mobile Web Applications . . . 2
- 1.3 Confronto tra i due approcci . . . 4
- 1.4 Applicazioni Ibride . . . 5
- 1.5 Lo sviluppo di Applicazioni e i framework . . . 7
- 1.6 Considerazioni . . . 10
  - 1.6.1 Qual'è l'approccio migliore . . . 11
  - 1.6.2 L'importanza della scelta del framework . . . 11
  - 1.6.3 E' possibile uno sviluppo multipiattaforma senza l'utilizzo di framework? 13

2 Metodi e Modelli per lo sviluppo di applicazioni Mobile 14
- 2.1 Design Patterns . . . 15
  - 2.1.1 Frontend e Backend . . . 15
  - 2.1.2 Pattern Client-Server . . . 16
  - 2.1.3 MVC Pattern . . . 17
  - 2.1.4 MVVM Pattern . . . 18
- 2.2 API . . . 19
  - 2.2.1 API Rest . . . 19
  - 2.2.2 Implementazioni Esistenti . . . 22
  - 2.2.3 CORS . . . 23
- 2.3 Template System . . . 24

3 Case Study: L'approccio ibrido con tecnologie web 27
- 3.1 Le Tecnologie Web utilizzate . . . 28
  - 3.1.1 CSS Preprocessor . . . 28
  - 3.1.2 AngularJS . . . 29

3.1.3 Ionic . . . 34
3.2 Il Wrapper Framework . . . 35
3.2.1 Cordova/Phonegap . . . 35
3.2.1.1 Storia . . . 36
3.2.1.2 Differenze . . . 36
3.2.1.3 Cordova Core . . . 37
3.2.2 ngCordova . . . 37
3.3 Strumenti di sviluppo . . . 37
3.3.1 Package Manager . . . 38
3.3.2 Task Runner . . . 39
3.4 SDK . . . 40
3.5 Pattern Javascript e best practices . . . 40
3.5.1 Utilizzo delle Promises . . . 40
3.5.2 Stato dell'applicazione . . . 41
3.5.3 Routing . . . 42
3.5.4 OAUTH . . . 43

4 Conclusioni **44**
4.1 Ibrido vs Nativo . . . 44
4.2 I cambiamenti sul processo di sviluppo . . . 45
4.3 Ulteriori otimizzazioni dell'approcio ibrido . . . 47
4.4 Nuove tecnologie web emergenti . . . 49
4.4.1 FirefoxOS . . . 49
4.4.2 WebRTC . . . 50

A Appendice Argomenti **51**
A.1 HTML5 . . . 51
A.2 CSS . . . 51
A.3 Javascript . . . 51
A.4 Dependency Injection . . . 51
A.5 Inversion Of Control . . . 51
A.6 Future / Promises . . . 51

Bibliografia **52**

Elenco delle Figure

1.1 Javascript API 6
1.2 Lista dei principali framework Javascript per la definizione della UI 8
1.3 Lista dei principali framework contenitori con i quali è possibile utilizzare tecnologie web 9
1.4 Elenco dei principali Runtime framework 10

2.1 MVC Pattern 17
2.2 MVVM Pattern 18

3.1 Css Preprocessors 28
3.2 AngularJS framework logo 29
3.3 Custom Directive 30
3.4 Data Bindings 31
3.5 AngularJS Singleton 33
3.6 Ionic Framework Logo 34
3.7 Cordova Framework Logo 35
3.8 Struttura a Livelli di Cordova 36
3.9 ngCordova Logo 37
3.10 ngCordova Plugin 38
3.11 Loghi di Grunt e Gulp 39
3.12 Stati di una applicazione 42

4.1 Schema generale 46
4.2 Schemi di sviluppo a confronto 48
4.3 FirefoxOS logo 49
4.4 Architettura di FirefoxOS 49

Elenco delle Tabelle

1.1 Elenco dei vari linguaggi di programmazione utilizzati nei sistemi operativi per dispositivi mobili 1

4.1 HTML5 vs Nativo, pro e contro dei due approcci 45

Elenco degli Esempi

2.1 Esempio di URL che non rispetta il vincolo di REST 22
2.2 Identificazione di una risorsa all'interno di una architettura REST 22
2.3 esempio di richiesta HTTP utilizzando CORS 23
2.4 esempio di risposta dal server che utilizza CORS 24
2.5 Un esempio di modello in json: la sua corrispondeza con il template system mustache e il risultato che si ottiene 25
3.1 Un esempio delle direttive standard di AngularJS 29
3.2 Associazione tra un elemento del DOM e un controller 31
3.3 Creazione di una applicazione in AngularJS con le relative dipendenze 32
3.4 Un esempio di creazione di un modulo e la sua inclusione all'interno di un altro . 33
3.5 Una tipica configurazione del file .bowerrc 39
3.6 Una tipica configurazione del file bower.json 39
3.7 Un esempio di uso delle promises in ng-cordva 41

Capitolo 1

Stato dell'arte

L'obiettivo di questo capitolo introduttivo e quello di mostrare quali sono gli approcci esistenti per lo sviluppo di applicazioni su dispositivi mobili ed analizzarne i vantaggi e gli svantaggi. In particolare per lo sviluppo di applicazioni multipiattaforma si adotterà il modello di sviluppo ibrido, e di conseguenza verranno mostrate le varie tecnologie che ho ritenuto necessarie per la sua l'implementazione.

1.1 Sviluppo Nativo

Dopo un'attenta analisi dei requisiti, una delle strade percorribili quando si intende sviluppare un'applicazione mobile è quella di orientarsi verso lo sviluppo nativo, ovvero utilizzare il linguaggio di programmazione specifico per una determinata piattaforma.

Lo sviluppo nativo implica competenze specifiche e capacità di sviluppo sui più diffusi sistemi operativi: iOS, Android, Windows Phone, la tabella seguente 1.1 indica i linguaggi utilizzati nelle rispettive piattaforme:

Sistema Operativo	**Linguaggi/o**
iOS	ObjectiveC / Swift
Android	Java
Windows Phone	.NET
BlackBerry	C++ / Qt
FirefoxOS	Javascript + CSS + HTML5

Tabella 1.1: Elenco dei vari linguaggi di programmazione utilizzati nei sistemi operativi per dispositivi mobili

Dato l'obiettivo prefissatosi nell'introduzione di avere la stessa applicazione disponibile su più piattaforme, con questo tipo di approccio la soluzione si risolve semplicemente sviluppando la stessa applicazione più volte per ogni piattaforma che si è scelto. Molto banalmente se

un ipotetico cliente chiedesse di avere la sua applicazione disponibile su Windows Phone, Android e iOS lo sviluppatore dovrà creare rispettivamente 3 applicazioni distinte in .Net, Java e ObjectiveC ovviamente senza la possibilità di riutilizzare il codice tra un linguaggio e l'altro.

Come si può notare, se si vuole usare questo tipo di approccio con l'idea di avere una applicazione su più piattaforme lo svantaggio più grande è sicuramente quello dei tempi e dei costi di sviluppo, ipoteticamente si potrebbe disporre di più programmatori con competenze specifiche nei vari linguaggi, ma questo sicuramente ridurrebbe la competitività del prezzo dell'applicazione al cliente. Inoltre per i programmatori sviluppare in diversi linguaggi la stessa applicazione comporta a usare modelli di progettazione differenti e quindi una re-ingegnerizzazione dell'applicazione per ogni piattaforma. Alcuni dei vantaggi di questo approccio sono:

- Massima performance in termini di potenza di calcolo e di sfruttamento del processore.
- Si ha il pieno controllo dei sensori del dispositivo
- Si sfruttano tutte le caratteristiche specifiche della piattaforma, sia in termini di software (librerie del produttore o di terze parti) che di hardware (del dispositivo o esterno)

Si ha quindi a discapito dei costi e dei tempi produzione una buona resa dell'applicazione a livello di prestazioni e di utilizzo completo delle caratteristiche del dispositivo. Questo tipo di approccio infatti è molto conveniente quando l'applicazione richiede un utilizzo molto intenso e performante delle risorse.

Prima di evidenziare quali sono i vantaggi e gli svantaggi differenti tra i vari approcci, si vedrà che cosa comporta effettivamente usare altri modelli di sviluppo e di quali tecnologie è richiesta la conoscenza per valutarne alla fine i costi nei vari termini.

1.2 Mobile Web Applications

Con l'evolversi delle tecnologie web si ha abbandonato sempre di più il concetto di *Sito Web* o *Sito Internet* e introdotto il termine *Applicazione Web*. Questo fatto è dovuto al continuo potenziamento dei linguaggi e degli strumenti che si utilizzano in ambito web, basta prendere come esempio Javascript, i primi standard di ECMAscript avevano solo il ruolo di dare animazione alla pagina HTML e di interfacciarsi con Applet Java, oggi si possono trovare Framework Javascript per ogni tipo di esigenza come per esempio: gestione dei contenuti di una pagina web, generatori di documentazione, test di codice, grafica 3D ecc.... Inoltre si può riscontrare l'evoluzione dell'HTML verso un linguaggio sempre più ricco di funzionalità grazie al

quale si può definire ora interfacce utente interattive con la possibilità di un design accattivante grazie ai fogli si stile(CSS3). Ma il vero motivo per cui oggi si parla di *Web Applications* è dato dall'utilizzo che si fa delle tecnologie Web e dello scopo che si da alle applicazioni, come ad esempio l'esecuzione di programmi direttamente sul browser web che rimandano la computazione ad un server senza preoccuparsi di installare il programma sul proprio computer.

Un importante passo che ha favorito l'evolversi dei Siti Web in Applicazione Web, e ad essere fruito tramite dispositivi mobili, è stato il concetto di *Responsive Web Application*. Si potrebbe tradurre in italiano la parola *Responsive* come *Adattato/a*, che sostanzialmente si tratta in una ri-organizzazione dei contenuti della pagina. Il fattore che influisce su questo comportamento è la dimensione dello schermo del dispositivo, essendo gli smartphone, tablet di dimensioni ridotte rispetto allo schermo di un computer, quello che fa una *Responsive Web Application* è adattare i propri contenuti alla dimensione dello schermo del dispositivo, il tutto in modo completamente automatico, senza il bisogno dell'utente di ingrandire con le dita i contenuti. Si potrebbe pensare a primo impatto che si tratti soltanto di una riduzione delle dimensioni della pagina web, in realtà entra in gioco un meccanismo che cambia le proporzioni e la disposizione dei contenuti rendendo ben visibile e nitido il testo e le immagini.

Da quanto appena detto, possiamo constatare che l'obiettivo di avere una Applicazione multipiattaforma che ci siamo prefissati all'inizio della tesi è molto comodo sotto questo tipo di approccio. Al contrario dello sviluppo nativo, non dobbiamo preoccuparci di dover programmare la stessa applicazione per diversi sistemi operativi in quanto il codice che scriviamo viene interpretato dal browser web il quale non dipende dalla piattaforma dalla quale stiamo utilizzando l'applicazione.

Le conoscenze che sono richieste al programmatore per poter affrontare un approccio di questo genere sono sicuramente quelle dei linguaggi HTML, CSS e Javascript, pacchetto di competenze che ritroviamo in tutti i programmatori che generalmente si occupano dello sviluppo di siti web. Ne deduciamo che rispetto all'approccio nativo abbiamo un costo sicuramente molto inferiore in termini di conoscenze e di tempi di sviluppo per un risultato ottimale rispetto agli obiettivi prefissati, ma ci sono dei fattori negativi in questo tipo di approccio decisamente non trascurabili.

Come i siti web anche le applicazioni web necessitano di una connessione ad Internet sempre presente per per funzionare, e di conseguenza se il dispositivo che si sta utilizzando è offline queste applicazioni non possono essere raggiunte / utilizzate, parametro che è sicuramente rilevante nella scelta di applicazioni da parte dell'utente. Quello che un fruitore si aspetta di trovare in una applicazione è che sia sempre reperibile anche se la connessione ad Internet in quel momento non è presente. Inoltre dal browser le Web Application non possono accedere direttamente alle funzionalità del dispositivo e usufruire ad esempio della fotocamera, il gps o la gestione delle risorse del dispositivo. Altra nota negativa delle applicazioni web riguarda il salvataggio dei dati che risulta molto difficile non potendo accedere alla memoria del dispositivo,

tramite browser si potrebbero memorizzare alcuni dati in cache ma correrebbero il rischio di venire cancellati ad insaputa dell'utente, l'unica soluzione è accedere ai dati una volta connessi ad internet tramite un processo di accounting fornito dall'applicazione, il problema è che ogni utente dovrebbe avere diversi account per ogni applicazione che utilizza, cosa non sempre molto gradita dall'utente.

1.3 Confronto tra i due approcci

Nelle descrizioni precedenti si è visto che tra i due approcci utilizzati il rapporto che caratterizza il confronto del tipo di sviluppo è tra Prestazioni e User Experience ottimale, nello sviluppo nativo / bassi costi di produzione e velocità di sviluppo, tramite Web Application. Quando si sviluppano applicazioni Mobile in generale bisogna tenere conto di diversi fattori di valutazione che riguardano principalmente l'utente finale ma con un occhio ache agli sviluppatori, ecco quindi come si comportano gli approcci precedentemente visti su una serie di fattori che ho ritenuto più rilevanti di altri:

Immediatezza Le web application sono immediatamente disponibili agli utenti tramite un browser su una gamma di dispositivi (iPhone, Android, BlackBerry, ecc.). Le applicazioni mobile native invece richiedono che l'utente innanzitutto scarichi e installi l'applicazione da un mercato prima che il contenuto o l'applicazione possa essere utilizzata.

Compatibilità un unico sito web mobile può raggiungere gli utenti attraverso diversi tipi di dispositivi mobili, mentre applicazioni native richiedono una versione separata da sviluppare per ogni tipo di dispositivo. Inoltre, le URL delle web app mobile sono facilmente integrabili all'interno di altre tecnologie mobile come gli SMS, i codici QR e NFC.

Capacità di aggiornamento una web app è molto più dinamica di un'applicazione mobile in termini di flessibilità pura per aggiornare il contenuto. Se si desidera modificare la struttura o il contenuto di una web app mobile è sufficiente pubblicare la modifica una volta e le modifiche sono immediatamente visibili; l'aggiornamento di un app, dall'altro richiede gli aggiornamenti siano convogliati agli utenti, che poi devono essere scaricati in modo da aggiornare l'applicazione su ogni tipo di dispositivo.

Ricercabilità le web app sono maggiormente ricercabili dagli utenti perché le loro pagine possono essere visualizzate nei risultati di ricerca ed elencati nel settore directory specifiche. Gli utenti del sito web possono essere inviati automaticamente al tuo sito mobile quando navigano tramite un dispositivo mobile smartphone o tablet. Al contrario, la visibilità delle applicazioni mobile è in gran parte limitata alle app store del produttore. Inoltre le web app sono accessibili da tutte le piattaforme e possono essere facilmente condivise da più utenti.

Tempo e costi le web app sono meno costose soprattutto se si necessita di avere una presenza su diverse piattaforme (che richiede lo sviluppo di applicazioni multiple).

Supporto e manutenzione il supporto e lo sviluppo evolutivo di una mobile app nativa (aggiornamenti, test, problemi di compatibilità e lo sviluppo in corso) sono più costosi al supporto di una web app. Vantaggi delle Applicazioni Native Nonostante i numerosi vantaggi inerenti lo sviluppo Mobile, ci sono diversi scenari di utilizzo specifici in cui una APP nativa sarà la scelta migliore. In particolare, un'applicazione nativa è utile conveniente nei seguenti casi:

Advergaming per i giochi interattivi una mobile app nativa è quasi sempre la scelta migliore.

Personalizzazione Se gli utenti di destinazione utilizzeranno l'applicazione in modo personalizzato.

Maggiore capacità di calcolo e reporting una mobile app nativa è la scelta giusta se si necessita di un'applicazione per gestire ed elaborare i dati con calcoli complessi, grafici o relazioni (applicazioni per il banking e la finanza).

Funzionalità native la mobile app nativa è molto efficace se è necessario accedere a fotocamera o alla capacità computazionale del dispositivo mobile.

Nessun collegamento richiesto la mobile app nativa è la scelta giusta se è necessario fornire accesso offline ai contenuti o funzioni senza connessione. Da ciò si può capire che prima di decidere di investire nello sviluppo di un'applicazione per il mobile è bene capire quali sono le esigenze di comunicazione, quali sono i servizi che si vogliono offrire in mobilità e quali sono soprattutto le esigenze clienti in mobilità.

Si è visto che entrambi gli approcci hanno vantaggi e svantaggi sotto diversi punti di vista ma dovendo scegliere per l'implementazione di una applicazione multipiattaforma la domanda è quale tra questi due approcci è migliore? Esiste un modo di combianare l'efficenza e le prestazioni dello sviluppo nativo con i costi di produzione ridotti di una Applicazione web?

La risposta è si! Esiste un tipo di approccio chiamato *ibrido*, che riesce a combinare in modo molto ottimale entrambi i fattori positivi degli approcci precedentemente visti. Questo tipo di sviluppo introduce una serie di tecnologie che lo sviluppatore può facilmente apprendere se capace di affrontare uno degli approcci visti precedentemente e l'utilizzo di conoscenze informatiche normalmente possedute da un programmatore di applicazioni Mobile.

1.4 Applicazioni Ibride

Ibrido per definizione, vuol dire qualcosa formato dall'unione di parti di natura eterogenea. Una Applicazione Ibrida è una applicazione scritta con gli stessi linguaggi e tecnologie che si

utilizzano per la creazione di siti web ed è ospitata o viene eseguita all'interno un contenitore nativo di un dispositivo mobile.

Il comportamento del contenitore nativo è paragonabile a quello di un browser web, al suo interno interpreta codice HTML, CSS e Javascript con la differenza che il codice non è preso da un *URL* ma è presente sul dispositivo. In particolare per presentare il codice interpretato e/o eseguito si utilizza un sistema a viste chiamato *Web View Control* [1] che consente di mostrare a tutto schermo il risultato ottenuto dall'interpretazione del codice servendosi della *Web Browser Engine* del dispositivo.[2] Questo vuol dire che il codice HTML e Javascript usato per costruire l'applicazione viene renderizzato/eseguito localmente dalla *Web Browser Engine*(in particolare su dispositivi mobili viene utilizzata la Rendering Engine di Webkit) e non all'interno di un Browser Web specifico come accadeva per le Web Application. Questo vuol dire che è possibile utilizzare codice HTML e Javascript per implementazioni al di fuori del Browser Web.

Ma il vero segreto delle applicazioni ibride è l'implementazione di un livello astratto che fornisce le funzionalità del dispositivo tramite l'utilizzo di API Javascript(viene usato questo linguaggio perché e l'unico ad essere eseguito all'interno di una Web Browser Engine).

```
01. navigator.camera.getPicture (
02.       onCameraSuccess,
03.       onCameraError,
04.       {
05.             quality: 50,
06.             destinationType: Camera.DestinationType.DATA_URL
07.       }
08. );
09.
10. function onCameraSuccess (imageData) {
11.       var image = document.getElementById('myImage');
12.       image.src = "" + imageData;
13. }
14.
15. function onCameraError (message) {
16.       alert('Failed because: ' + message);
17. }
```

FIGURA 1.1: Un esempio di utilizzo di API Javascript per ottenere dati dalla fotocamera

Questa operazione non è possibile tramite un'implementazione come Web Application, perché per evidenti motivi di sicurezza, il Browser Web dei dispositivi mobili non ha libero accesso alle funzionalità del dispositivo(altrimenti qualunque sito internet potrebbe compromettere l'integrità dello stesso). Per consentire ciò, le applicazioni ibride si servono di potenti framework che si occupano di fornire un canale di comunicazione tra le due entità, quella web e quella nativa, consentendo all'interfaccia scritta in HTML e Javascript di poter utilizzare le funzionalità del

[1] UIWebView per iOS, WebView per Android e altri
[2] Ogni browser web ne ha una : Gecko(Firefox), Blink(Chrome, Opera), Webkit(Safari), Trident(IE).

dispositivo. In particolare *Cordova* è uno dei framewrok più usati per questo tipo di sviluppo, il quale fornisce dal lato dell'interfaccia web delle API in Javascript per comunicare con il dispositivo.

Sul mercato esistono framework che usano procedimenti differenti da quello appena spiegato per ottenere lo stesso risultato multipiattaforma, la maggior parte tende però a sviluppare l'interfaccia utente con codice HTML, in quanto essendo un linguaggio tendenzialmente semplice consente di avere una descrizione dei dati accurata e dinamica, la quale può essere riutilizzata se lo sviluppatore decidesse di avere una eventuale Web Application oltre a quella ibrida. Un punto molto forte dello sviluppo ibrido è appunto che è possibile riutilizzare il codice web, pur avendo una applicazione quasi nativa(ibrida).

Prima del confronto tra gli approcci visti, verrà spiegato come i framework possono essere utili nello sviluppo di applicazioni mobili, e come possono risultare utili in particolare nell'approccio ibrido.

1.5 Lo sviluppo di Applicazioni e i framework

In informatica, e specificatamente nello sviluppo software, un ***framework*** *è un'architettura (o più impropriamente struttura) logica di supporto (spesso un'implementazione logica di un particolare design pattern) su cui un software può essere progettato e realizzato, spesso facilitandone lo sviluppo da parte del programmatore. Alla base di un framework c'è sempre una serie di librerie di codice utilizzabili in fase di linking con uno o più linguaggi di programmazione, spesso corredate da una serie di strumenti di supporto allo sviluppo del software, come ad esempio un IDE, un debugger o altri strumenti ideati per aumentare la velocità di sviluppo del prodotto finito. L'utilizzo di un framework impone dunque al programmatore una precisa metodologia di sviluppo del software.*

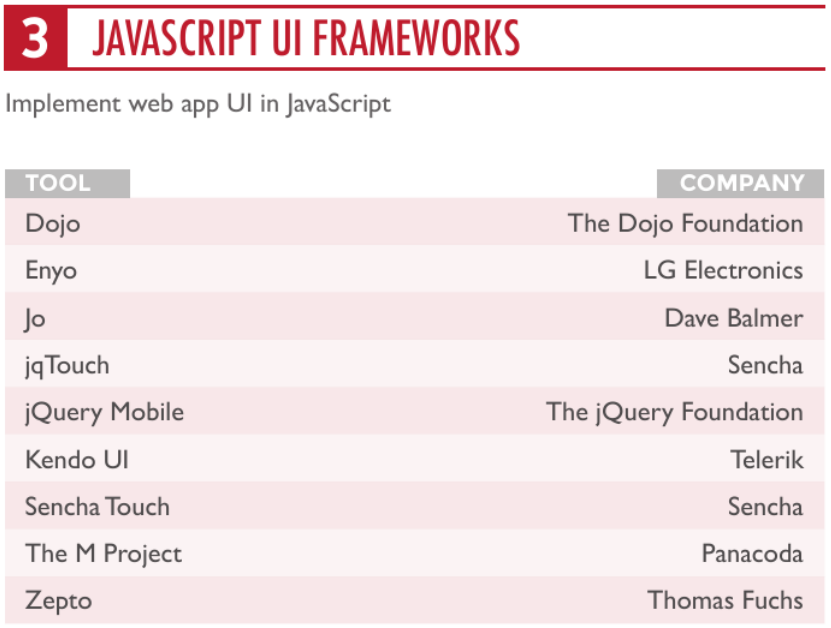
[2]

L'uso di framework da parte di aziende e sviluppatori è diventato sempre più comune nello sviluppo di applicazioni mobile. Il processo di scrittura del codice diventa rapido e intuitivo, in alcuni casi si ha a disposizione un editor visuale dell'interfaccia che tramite il solo trascinamento di nuovi elementi, messi a disposizione dall'editor, aggiorna il codice sorgente dell'applicazione. Oggi sul mercato esistono diversi tipi di framework, ognuno con caratteristiche e vantaggi differenti. Ciascuno di essi supporta uno o più sistemi operativi, ma non è detto che un framework possa supportare completamente le funzionalità richieste da un'applicazione e/o da un progetto.

Molti framework usano una combinazione di tecnologie, che ne caratterizza la loro tipologia, delle quali se ne distinguono 5: JavaScript framework, app factories, web-to-native wrapper, runtime

e source code translator. A questo punto entrano in gioco le competenze del programmatore, il quale a seconda delle sue capacità sceglierà l'approccio più conveniente, in quanto per ognuno di essi sono coinvolte tecnologie differenti. [3]

Javascript Framework in alcuni casi sono semplicemente delle librerie e non propriamente dei framework e forniscono delle funzioni tipiche delle applicazioni mobile, come ad esempio, scorrimento e eventi legati al touch-screen. Principalemente si utilizzano per la costruzione della *User Interface* i più conosciuti sono: JQuery Mobile, Ionic, Sencha Touch, Cocos2D, DHTMLX Touch, Zepto JS, Impact.js, iUI e Wink.



3 JAVASCRIPT UI FRAMEWORKS

Implement web app UI in JavaScript

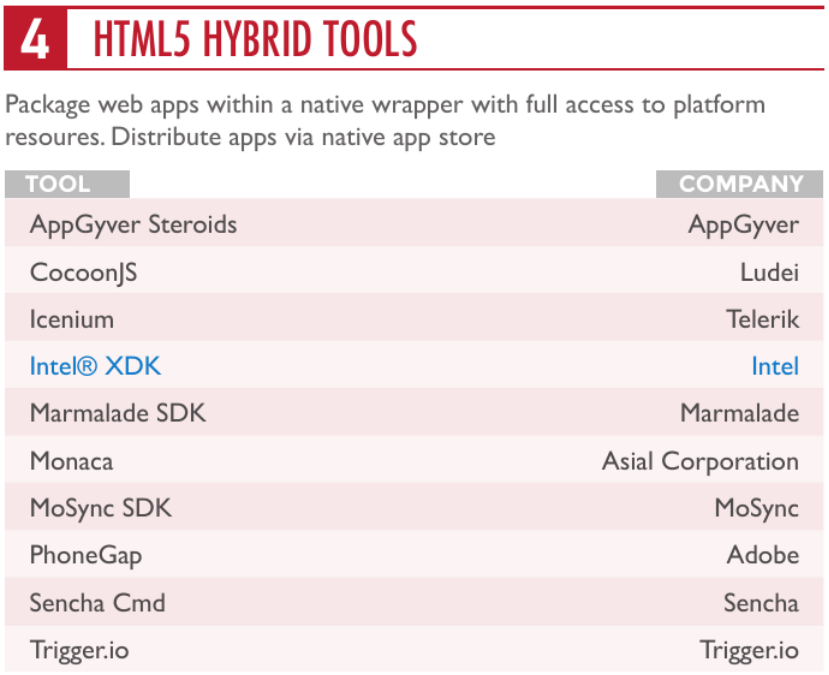
TOOL	COMPANY
Dojo	The Dojo Foundation
Enyo	LG Electronics
Jo	Dave Balmer
jqTouch	Sencha
jQuery Mobile	The jQuery Foundation
Kendo UI	Telerik
Sencha Touch	Sencha
The M Project	Panacoda
Zepto	Thomas Fuchs

FIGURA 1.2: Lista dei principali framework Javascript per la definizione della UI

App Factories sono generatori di codice sorgente, ovvero sono dei tool grafici per costruire in modo semplice le applicazioni mobile. Sono composti di un ambiente di sviluppo (installabile o utilizzabile online) dove lo sviluppatore, partendo da un template base, inserisce elementi con il drag and drop, come ad esempio pulsanti ed input text. Infine le App factories generano il codice sorgente. App factories permettono ai non programmatori di "creare le loro app". Alcuni tools permettono di vedere e modificare il codice generato. Altri includono una serie di servizi come analisi di utilizzo, push notification e gestionali. Alcuni esempi di questi tool sono: AppMkr, AppsGeyser, Wix Mobile, Tiggr, Mobile Nation HQ, Mobjectify, Red Foundry e Spot Specific.

Web-to-native Wrapper questo tipo di framework fornisce un contenitore nativo all'interno del quale è possibile costruire una applicazione utilizzando tecnologie web. Il funzionamento è analogo come spiegato nella sezione 1.4. Vengono estese le API di JavaScript e si ha la

possibilità di accedere alle funzionalità del dispositivo e del sistema operativo come notifiche, accelerometro, bussola, geolocalizzazione e file system. Il principale esempio di web-to-native wrapper è PhoneGap, ma ci sono anche Uxebu's Apparat.io e Sencha v2 nella quale hanno aggiunto al wrapper dei framework JavaScript. Un altro esempio è MoSync Wormhole, il quale dispone di un set di API come PhoneGap. Web-to-native wrapper mira agli sviluppatori web che hanno bisogno di convertire le loro applicazioni web ad applicazioni mobile per poterle pubblicare sui vari app store o anche per accedere alle funzionalità native del dispositivo e rendere il codice web ottimizzato nel contesto mobile.



4 HTML5 HYBRID TOOLS

Package web apps within a native wrapper with full access to platform resoures. Distribute apps via native app store

TOOL	COMPANY
AppGyver Steroids	AppGyver
CocoonJS	Ludei
Icenium	Telerik
Intel® XDK	Intel
Marmalade SDK	Marmalade
Monaca	Asial Corporation
MoSync SDK	MoSync
PhoneGap	Adobe
Sencha Cmd	Sencha
Trigger.io	Trigger.io

FIGURA 1.3: Lista dei principali framework contenitori con i quali è possibile utilizzare tecnologie web

Runtime In generale si tratta di un ambiente di esecuzione posto in un livello sopra il sistema operativo per garantire una compatibilità cross-platform. I framework che utilizzano questo approccio variano di dimensioni e complessità, e eseguono il codice sul dispositivo servendosi di tecniche differenti: virtualizzazione, interpretazione, just-in-time o ahead-of-time compilation. Esempi di questi sistemi sono Java ME, BREW, Flash Lite e Openware MIDAS, descritti a volte come metà browser e metà sistemi operativi. Non hanno avuto successo in quanto si trattava di sistemi con un considerevole carico di lavoro per i dispositivi che nuoceva alle prestazioni. Inoltre non avevano uno scopo diretto per il mercato delle applicazioni(dato anche dallo scarso utilizzo da parte degli sviluppatori) e la competizione con gli altri sistemi iOS, Android e browser web gli ha resi quasi del tutto inutilizzati. Recentemente sono stati sviluppati degli strumenti che scaricano la parte della compilazione del codice durante la fase di design dell'applicazione in

modo tale che venga prodotto un bytecode per una specifica piattaforma. Alcuni esempi sono : Appcelerator, Adobe Flex(e AIR), Corona, AppMobi, Antix e Unity.

5 X-TO-NATIVE CONVERTERS

Write your apps in JavaScript or other languages and compile them to native code, for access to native APIs. Distribute apps via native app store

TOOL	COMPANY
Adobe AIR	Adobe
alcheMo	Innaworks
Appcelerator Titanium	Appcelerator
Apportable	Apportable
Canappi	Canappi
Codename One	Codename ONE
Edgelib	Elements Interactive Mobile B.V.
Firemonkey	Embarcadero Technologies
Gideros	Gideros Mobile
iFactr	ITR Mobility
Instant Developer	Pro Gamma
J2me Polish	Enough Software
J2ObjC	Google
Kiahu	Kiahu
Livecode	RunRev
Monocross	Monocross
MoSync SDK	MoSync
Pixelplant	Uxebu
Qt	Digia
SIO2 Interactive	sio2interactive.com
Xamarin	Xamarin

FIGURA 1.4: Elenco dei principali Runtime framework

Source code translators Queste soluzioni traducono il codice sorgente in un bytecode intermedio, o in un linguaggio nativo (per esempio C++, Objective-C, JavaScript) o direttamente al linguaggio macchina di basso livello (linguaggio assembly). Source code translators sono spesso utilizzati in combinazione al runtime. Per esempio, Metismo (ora Software AG) converte applicazioni J2ME in C++, ActionScript e JavaScript, ed esse sono compilabili con dispositivi ARM, MIPS, PowerPC e x86. Analogamente, Eqela prende un applicazione scritta in un linguaggio simile al C e ne traduce il codice sorgente a seconda della piattaforma: JavaScript per web browsers, Java, C o assembly.

1.6 Considerazioni

Nel confronto tra lo sviluppo nativo e quello tramite Web Application si è visto che i principali fattori a pesare su un obiettivo di applicazione multipiattaforma è la possibilità

di accedere alle funzionalità del dispositivo e di avere prestazioni molto efficienti contro una portabilità immediata senza il bisogno di riscrivere il codice per ogni piattaforma di destinazione. Introducendo le applicazioni ibride si è potuto combinare entrambi i fattori positivi dei due approcci differenti, infatti in questo tipo di approccio si può accedere alle funzionalità del dispositivo tramite delle API che un framework di supporto fornisce, ma riguardo alle prestazioni, non si è ancora arrivati ad una tecnologia che consenta di paragonare l'efficienza dello sviluppo nativo con quello di tipo ibrido. Con l'evoluzione delle tecnologie utilizzate dai framework e il continuo susseguirsi versioni di piattaforme sempre più adattabili a diversi tipi di tecnologie, il distacco tra i due approcci continua sempre a diminuire.

1.6.1 Qual'è l'approccio migliore

Non esiste un approccio migliore di un altro in quanto hanno tutti pregi e difetti sotto certi punti di vista, in base allo scopo e ai requisiti dell'applicazione che si vuole creare bisogna valutare quale approccio sia il migliore da utilizzare per poter competere al meglio sul mercato. Un esempio potrebbe essere una applicazione gioco, la quale richiede una forte potenza di calcolo a livello grafico e che con uno sviluppo di tipo ibrido o web è molto difficile da ottenere. In questi casi quando le prestazioni sono al centro della richiesta dell'applicazione, uno sviluppo di tipo nativo, anche se più dispendioso in quanto si resta sotto l'idea di avere una applicazione multipiattaforma, è una delle scelte migliori che si possono fare. Una alternativa potrebbe ese. sere quella di utilizzare un framework apposito allo sviluppo di giochi, che consenta di avere un gioco multipiattaforma, si ricorda però che alcuni tipi di framework molto avanzati spesso non sono open source e richiedono un investimento da parte del programmatore / azienda per il suo utilizzo. Nel caso l'applicazione richieda una potenza di calcolo nella media la scelta dell'approccio e delle tecnologie da utilizzare ricade sulle caratteristiche specifiche che l'applicazione richiede.

1.6.2 L'importanza della scelta del framework

Oggi sul mercato esistono diversi tipi di framework, ognuno con caratteristiche e vantaggi differenti. Ciascuno di essi supporta uno o più sistemi operativi, ma non è detto che un framework possa supportare completamente le funzionalità richieste da un'applicazione e da un progetto. Per questo motivo la scelta del framework giusto deve essere oggetto di una attenta valutazione da parte delle aziende o dei programmatori. Questa valutazione deve prendere in considerazione numerosi parametri quali, ad esempio, le funzionalità che il cliente intende integrare nell'applicazione, i dispositivi sui quali questa applicazione deve girare, il budget a disposizione, il termine ultimo per commercializzare l'applicazione sul mercato ed infine la strategia di lungo termine legata alla quantità di progetti che devono essere sviluppati. Tutti i

framework, come sopra accennato, non supportano tutti i sistemi operativi. Per questo motivo l'esperienza del programmatore è fondamentale per ottimizzare gli strumenti di sviluppo e le funzionalità dell'applicazione.

Il vantaggio principale nell'utilizzo di un framework consiste nella possibilità di effettuare una consegna rapida e multipiattaforma di un'applicazione: il codice, infatti, si scrive velocemente con un meta linguaggio ed attraverso librerie esistenti precompilate, ed esso viene automaticamente adattato per i diversi sistemi operativi in fase di compilazione.

Una criticità di questo tipo di framework consiste nel fatto che, all'inizio, l'azienda deve sostenere costi piuttosto elevati per l'acquisto della licenza. Occorre inoltre investire anche nelle attività di formazione per le risorse interne che dovranno occuparsi dell'implementazione e dell'uso del framework. Un limite ulteriore dei framework consiste nell'eccesso di rigidità in fase progettuale: il loro utilizzo, infatti, limita in parte la possibilità di sfruttare le caratteristiche native del dispositivo, delle linee guida dell'interfaccia e del sistema operativo che utilizzerà l'applicazione, e spesso, quindi, si corre il rischio di non riuscire a sviluppare interfacce utente pienamente ergonomiche e intuitive.

Nella scelta del framework occorre anche concentrarsi su quelli che offrono una maggiore garanzia di supporto nel corso del tempo. Alcuni framework infatti tendono a non essere più supportati dopo un certo periodo di tempo; in questo caso, se occorre implementare sviluppi ulteriori, si corre il rischio di dover riprogettare l'applicazione da zero utilizzando un framework diverso.

Inoltre, alcuni framework sono più adatti ad impieghi specifici rispetto ad altri. Kony, ad esempio, è un tipo di framework particolarmente apprezzato nello sviluppo di applicazioni per il settore bancario e financial, poichè integra al suo interno numerose funzioni utili, ad esempio, per la gestione dei sistemi di pagamento attraverso gli smartphone. Unity 3D, invece, è il framework ideale per lo sviluppo di giochi e videogames.

Nel caso di framework nativi i costi iniziali sono più bassi rispetto ai precedenti (non vi è, ad esempio, la necessità di acquistare la licenza per il loro utilizzo), tuttavia i costi tendono a crescere esponenzialmente in funzione della quantità di aggiornamenti e rilasci da effettuare nel corso del tempo, o nel caso in cui si intenda sviluppare un'applicazione per più sistemi operativi diversi tra loro.

Un tipo di framework scelto nel contesto giusto, che tenga concretamente conto degli obiettivi di marketing dell'azienda e le strategie di investimento in sviluppo di prodotti, consente di ottimizzare i tempi di sviluppo, presentare velocemente sul mercato la propria applicazione ed ottimizzare il processo di delivery abbattendo costi e sfruttando le potenzialità specifiche del framework selezionato.

1.6.3 E' possibile uno sviluppo multipiattaforma senza l'utilizzo di framework?

Si è inteso lo sviluppo multipiattaforma come partendo da un unico codice sorgente dell'applicazione ad avere quest'ultima già disponibile per più di un canale di distribuzione sul mercato(ovvero i sistemi operativi dei dispositivi, la web application). Ebbene questo procedimento senza un framework di supporto risulta molto difficoltoso in quanto l'approccio rimane quello nativo per ogni piattaforma, oppure il programmatore / azienda ha a disposizione un sistema proprietario progettato appositamente per fare questa operazione, opzione che non è mai presa in considerazione in quanto molto dispendiosa. Per uno sviluppo multipiattaforma è richiesta la conoscenza del funzionamento di almeno un framework, che dipende sostanzialmente dalle conoscenze e dalla formazione del programmatore, il quale sceglierà le tecnologie più adatte alle sue conoscenze pregresse.

Capitolo 2

Metodi e Modelli per lo sviluppo di applicazioni Mobile

Durante la mia esperienza presso la ditta BigThink SRL oltre ad una formazione pratica ho potuto, in alcuni momenti sotto la guida del capo azienda e del correlatore, studiare alcuni concetti teorici utili ad comprendere meglio le tecnologie utilizzate nel processo di sviluppo delle applicazioni in azienda. In questo capitolo si riportano dei concetti informatici secondo me rilevanti (e alcune volte fondamentali) per lo sviluppo di applicazioni.

Una prima fase dello sviluppo è stata quella di strutturare l'architettura dell'applicazione e decidere come vengono assegnati i compiti. Una delle politiche che è stata intrapresa in azienda è stata quella della *Separation of Concernes* (SoC). Si tratta di un principio di design del software per dividere un problema in sezioni distinte, e ad ogni sezione assegnare un particolare compito o risoluzione del problema. Un programma che segue questo design pattern prende l'attributo di modulare [4]. La modularità e la separazione dei compiti si ottiene incapsulando informazioni all'interno di una sezione di codice con una interfaccia ben definita(information hiding [5]). Un altro esempio di SOC sono i sistemi progettati a livelli. [6]

Il concetto appena visto astrae molto da una applicazione vera e propria, sta a chi la progetta decidere con che granularità e se è veramente necessario applicarlo. Dato che parliamo di tecnologie web, un chiaro esempio si separazione dei compiti è quello della struttura di una pagina web, divisa in HTML per la struttura, CSS per lo stile, e Javascript per la logica e comportamento.

In questo capitolo andremo a vedere in che modo i compiti si possono separare all'interno di una applicazione, e quali strutture e/o design pattern possono risultare utili per lo sviluppo.

2.1 Design Patterns

Quando si progetta una applicazione risulta molto utile utilizzare schemi architetturali e metodologici che rendono da un lato l'applicazione efficiente dall'altro una scrittura del codice molto chiara e modulare, facile da correggere nel caso di eventuali errori. I design pattern vengono in aiuto a questa esigenza del programmatore.

In informatica, nell'ambito dell'ingegneria del software, un design pattern (traducibile in lingua italiana come schema progettuale, schema di progettazione, schema architetturale), è un concetto che può essere definito una soluzione progettuale generale ad un problema ricorrente. Si tratta di una descrizione o modello logico da applicare per la risoluzione di un problema che può presentarsi in diverse situazioni durante le fasi di progettazione e sviluppo del software, ancor prima della definizione dell'algoritmo risolutivo della parte computazionale. [7]

Il problema ricorrente che si ha nello sviluppo di applicazioni e quello della decisione di dove debbano stare determinati compiti / operazioni / sezioni all'interno dell'applicazione. Ad esempio la separazione della parte dell'interfaccia da quella dedicata alla manipolazione dei dati da quella dedicata alla memorizzazione. Dividere i vari compiti di una applicazione può risultare molto produttivo, in quanto si possono sviluppare in parallelo diverse parti dell'applicazione senza influire sulle altre e volendo con la possibilità di utilizzare tecnologie differenti.

Durante il periodo di tirocinio presso l'azienda *BigThink SRL* per la creazione di Web Applications sono stati utilizzati pattern come : Client-Server, SoC, Frontend e Backend, MVC; gli altri design pattern che verranno introdotti sono necessari per comprendere meglio alcune tecnologie che verranno utilizzate.

2.1.1 Frontend e Backend

Queste due parole sono spesso usate in informatica in molti ambiti, nel contesto specifico dell'applicazione `frontend`(in italiano parte davanti) denota quella parte dell'applicazione responsabile di gestire l'interfaccia utente e i dati provenienti da essa, mentre `backend`(in italiano parte dietro) indica la sezione dell'applicazione dedita alla gestione dei dati. L'interazione che hanno le due parti è un chiaro esempio di interfaccia.

Frontend: questa è la parte caratteristica dell'applicazione, in quanto ne definisce il comportamento e l'aspetto, determinando la logica con cui si evolverà al rapporto con l'utente. A differenza della parte *backend* questa non definisce ne manipolazione, ne la rappresentazione dei dati ma la vista che ha l'utente su di essi. Nella parte *frontend* è inclusa anche la fase di definizione estetica dell'interfaccia, ma spesso questa spetta a una figura professionale distinta

atta vestire l'applicazione.

Backend: questa parte è completamente diversa dalla prima, in quanto definisce la manipolazione dei dati all'interno dell'applicazione. In particolare fornisce dei servizi / risorse ai quali la parte *frontend* può accedere e effettuare delle operazioni, come ad esempio l'autenticazione di un utente a un servizio. Tutta la gestione dei dati che viene fatta da questa parte, viene oscurata alla parte *frontend* per garantire un servizio di sicurezza molto elementare, in modo tale che se l'utente inserisce dei dati errati che vengono passati dalla parte *frontend* a quella *backend*, nessuna operazione verrà eseguita e l'integrità dei dati verrà preservata.

Si tratta di una separazione concettuale e non fisica delle parti, accade spesso che si facciano analogie con il pattern client-server(spiegato successivamente); le due parti possono anche risiedere sullo stesso dispositivo / calcolatore.

A livello professionale molti sviluppatori si identificano appunto come *frontend* e/o *backend* developer ovvero specializzati nello sviluppo di una parte specifica dell'applicazione. Se la parte *backend* viene progettata secondo dei corretti schemi, e possibile riutilizzare le risorse anche in futuro, purché si rispetti l'interfaccia. Inoltre a livello professionale si può procedere parallelamente nello sviluppo delle parti in modo tale da ottimizzare i tempi, pur seguendo uno schema stabilito a priori.

2.1.2 Pattern Client-Server

In uno schema di applicazione su larga scala può risultare utile separare le parti logiche dell'applicazione su entità diverse che poi saranno messe in comunicazione. Ad esempio la computazione di un certo processo di calcolo potrebbe risultare molto onerosa a livello di prestazioni e che un dispositivo mobile possa non essere in grado supportarla. A questo punto risulta conveniente adottare una architettura client-server dove è il server che si occupa di eseguire le operazioni di calcolo mentre il client di rappresentare l'output.

Definizione: Il modello client-server è un modo per strutturare applicazioni distribuite che distingue due parti di un processo di comunicazione, la prima che fornisce una risorsa e/o un servizio chiamata server, la seconda che analogamente li può richiedere, chiamata client. La comunicazione in generale avviene attraverso la rete, ed è il client a iniziarla. Il compito del server è quello di predisporre le risorse che ha ai vari client che li chiedono, rimanendo appunto

in ascolto, il client invece non condivide le risorse con altri, può solo interagire con il server[8].

La divisione client e server associata con le parti frontend e backend di una applicazione viene spesso utilizzata nello sviluppo di applicazioni mobile distribuite, ad esempio in servizi georeferenziati. Ma non e detto che in generale sia sempre la soluzione ottima da adottare.

2.1.3 MVC Pattern

L'evoluzione della complessità del codice delle applicazioni, ha fatto si che i programmatori adottassero strategie di scrittura del codice in modo che rimanesse mantenibile e riusabile. L'uso del pattern MVC ha trovato largo uso in molti framework Javascript in modo tale che il codice scritto dal programmatore ereditasse i suoi benefici. Questa logica inoltre nello sviluppo complessivo dell'applicazione ha fatto si che il programmatore potesse disporre di uno scheletro dell'applicazione generico, in modo tale da accelerare il processo di sviluppo e di disporre di eventuali parti già sviluppate.

Il Model-View-Controller pattern (MVC) in informatica, è un pattern architetturale molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito della programmazione orientata agli oggetti, in grado di separare la logica di presentazione dei dati dalla logica di business.

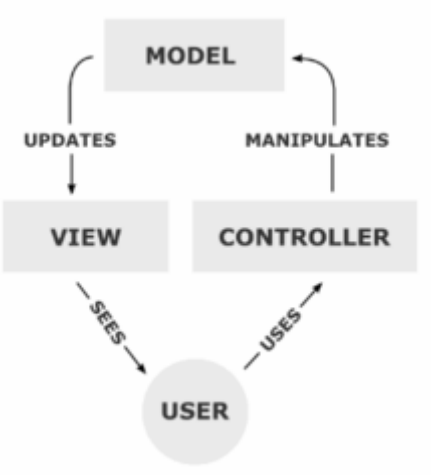


FIGURA 2.1: Schema del pattern MVC

Il pattern è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:

model fornisce i metodi per accedere ai dati utili all'applicazione;

view visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti;

controller riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti.

Questo schema, fra l'altro, implica anche la tradizionale separazione fra la logica applicativa (in questo contesto spesso chiamata logica di business), a carico del controller e del model, e l'interfaccia utente a carico del view. [9]

Il framework AngularJS spiegato nella sezione 3.1.2 è sviluppato secondo questo pattern. Il concetto verrà inoltre richiamato nella spiegazione di altri processi di sviluppo.

2.1.4 MVVM Pattern

Nello sviluppo di applicazioni mobile si avrà a che fare con il concetto di vista e di modello che rispettivamente denotano l'interfaccia utente e i dati che vengono mostrati sulla vista(si vedrà perché intesi come modelli). Nell'ottica di tenere la logica dell'applicazione separata dalla rappresentazione dei dati il pattern MVVM detta le linee guida per poter scrivere codice che tenga conto di questi aspetti.

Il Model-View-ViewModel è un pattern architetturale basato sul pattern MVC e MVP che tenta di separare più chiaramente lo sviluppo di interfacce utente (UI) da quello della logica di business e il comportamento in un'applicazione . A tal fine, molte implementazioni di questo modello fanno uso di associazioni dati dichiarativa(data bindings) per consentire una separazione di lavoro su Vista da altri livelli. [10]

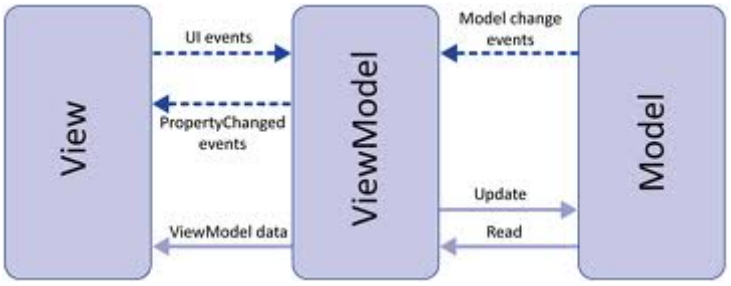


FIGURA 2.2: Schema del pattern MVVM

Il fulcro del funzionamento di questo pattern è la creazione di un componente, il ViewModel appunto, che rappresenta tutte le informazioni e i comportamenti della vista corrispondente. La vista si limita infatti, a visualizzare graficamente quanto esposto dal ViewModel, a riflettere in esso i suoi cambi di stato oppure ad attivarne dei comportamenti.

Questo facilita l'interfaccia utente e lo sviluppo che si verificano quasi contemporaneamente all'interno dello stesso codice. Gli sviluppatori dell'interfaccia utente scrivono associazioni verso

il ViewModel nel documento di markup (HTML) mentre il Model e il ViewModel sono mantenuti da altri sviluppatori che lavorano sulla logica dell'applicazione(business logic).

2.2 API

In informatica le API(Application Programming Interface) sono un insieme di operazioni, protocolli, strumenti per lo sviluppo del software. Una API rappresenta una specifica componente del software in termini di operazione, input, output e tipi soggiacenti. Una API definisce funzionalità totalmente indipendenti dalla loro rispettiva implementazione, il che consente di poter variare rispettivamente l'implementazione e la definizione senza che una influisca sull'altra. Una buona API rende più facile lo sviluppo del software fornendo vari mattoni con cui poter sviluppare il software, il ruolo del programmatore è quello di unire i vari blocchi richiesti. [11]

Un primo esempio di API è già stato menzionato quando si è parlato dei framework per lo sviluppo ibrido. In quel caso le API erano fornite tramite Javascript in modo tale che potessero essere interpretate all'interno di un contenitore nativo che potesse interpretare linguaggi web(come un browser). In questo caso i blocchi di cui si parla sono le rispettive API che consentono di comunicare con le funzionalità del dispositivo come ad esempio la fotocamera, il gps, la vibrazione, l'accelerometro ecc. . . .

L'utilizzo di API è stato fatto durante il mio tirocinio aziendale per lo scambio di dati tra la parte Frontend e Backend di Web Application, in particolare sono state utilizzate delle API REST.

2.2.1 API Rest

REST(**R**epresentational **S**tate **T**ransfer) è un termine coniato da Roy Fielding(co-autore del protocollo HTTP 1.1) nella sua tesi di dottorato per descrivere lo stile dell'architettura dei sistemi di rete. REST non è un protocollo e nemmeno uno standard, ma uno *stile* architetturale per applicazioni / servizi web; quando uno di essi rispetta i criteri di REST gli si da l'attributo *RESTful*.

REST impone determinati vincoli alla sua architettura, pur lasciando libera la loro implementazione:

Uniform Interface REST impone una interfaccia uniforme tra client e server. Questo semplifica e decompone l'architettura, il che fa si che ogni parte possa evolvere indipendentemente. Per creare una interfaccia uniforme si sono 4 principi da seguire:

Resource Based ogni risorsa deve essere identificata univocamente all'interno del server. La sua identificazione deve essere concettualmente separata dalla rappresentazione che viene ritornata al client, la quale può dipendere dalla richiesta fatta.

Manipolazione delle Risorse il client tramite il riferimento della risorsa può effettuare 4 operazioni base creazione, lettura, aggiornamento, cancellazione (*CRUD*: **C**reate **R**ead **U**pdate **D**elete)

Self-descriptive Messages Ogni messaggio tra client e server contiene tutte le informazioni sufficienti per la sua codifica e interpretazione. Le risposte dal server devono indicare anche i parametri di caching.

Hypermedia as the Engine of Application State (HATEOAS) Uno dei principi REST suggerisce l'uso di collegamenti tra risorse come modalità di transizione da uno stato all'altro. Questo principio è noto anche con l'acronimo HATEOAS, Hypermedia As The Engine Of Application State, e focalizza l'attenzione sull'utilizzo dei collegamenti ipertestuali, anzi ipermediali, come meccanismo di base per far passare un'applicazione da uno stato ad un altro. Se il risultato di una interazione con il server è un contenitore di risorse, allora all'interno della risorsa abbiamo gli URI delle risorse contenute, altrimenti nella maggior parte dei casi non si hanno altri collegamenti. Non possiamo dire che un Web Service di questo tipo non sia RESTful, ma nella maggior parte dei casi non sfrutta a pieno le potenzialità dei principi REST. Il principio HATEOAS, infatti, intende incoraggiare non solo l'uso di collegamenti per rappresentare risorse composte, ma anche per definire qualsiasi altra relazione tra le risorse e per controllare le transizioni ammissibili tra uno stato e l'altro dell'applicazione. Sfruttando pienamente il principio HATEOAS è possibile creare servizi Web con scarso accoppiamento tra client e server. Infatti, se il server riorganizza le relazioni tra le risorse, il client è in grado di trovare tutto ciò che serve nelle rappresentazioni ricevute. Potenzialmente tutto quello che servirebbe ad un client è solo l'URI della risorsa iniziale. Come avanzare tra uno stato e l'altro dell'applicazione verrà indicato man mano che si seguono i collegamenti incorporati nelle successive rappresentazioni di risorse. [12]

Stateless La parte *State* dell'acronimo di REST, sta a indicare un architettura di rete che non mantiene lo stato della risorsa. Ovvero lo stato della risorsa viene passato tramite il corpo del messaggio che viene scambiato tra client e server. Essendo ogni risorsa identificata con un URI(**U**niversal **R**esource **I**dentificator), quando viene fatta una richiesta alla risorsa può avvenire un cambiamento di stato di quest'ultima. In qualunque caso lo stato finale della risorsa viene comunicato al client nel corpo del messaggio di risposta.

Nell'industria delle reti si può ricordare il concetto di *sessione* HTTP, ovvero dove lo stato di una risorsa viene mantenuto attraverso richieste HTTP multiple. Nel caso di REST, il client si deve preoccupare di includere tutte le informazioni necessarie nella richiesta al server, e nel caso di richieste multiple è obbligato a inviare in tutte le richieste lo stato aggiornato della risorsa.

Questo permette di far si che i server non debbano preoccupare di mantenere lo stato delle loro risorse, in modo tale da garantire una maggiore scalabilità del sistema.

Quindi la differenza tra stato e risorsa è che: lo stato o lo stato di una applicazione, e ciò a cui importa al server per soddisfare la richiesta corrente. Una risorsa o lo stato di una risorsa sono i dati che definiscono la rappresentazione della risorsa. Per essere più chiari si consideri lo stato dell'applicazione come i dati nella richiesta che possono variare da client a client. Mentre lo stato della risorsa rimane costante dipendentemente da ogni client che la richiede.

Cacheadable Nel World Wide Web i client poSystemssono mantenere una cache delle richieste fatte ai server. In una architettura REST le richieste devono specificare, esplicitamente o implicitamente, la possibilità di essere mantenute in cache oppure no, per prevenire ai client di ri-usare stati o dati inappropiati per richieste future. Una buona gestione della cache può parzialmente o completamente eliminare alcune interazioni tra client e server, così da migliorare prestazioni e scalabilità.

Client-Server La struttura dell'applicazione deve essere di tipo client-server, mostrata nella sezione 2.1.2. Avendo una interfaccia uniforme, in una architettura REST i client sono separati dai server. Questo fa si che ci sia una separazione dei compiti tra i due, ad esempio: i client non hanno la visione su come i dati vengono memorizzati, il quale compito spetta al server.In questo modo migliora la portabilità del codice del client, mentre i server non sono a conoscenza di come sia fatta l'interfaccia utente del client. Questo fa si che i la logica del server sia più semplice e scalabile. Se l'interfaccia tra client e server non viene alterata, le due entità possono venire sviluppate o sostituite in modo del tutto indipendente.

Layered system Per garantire una facile scalabilità del sistema e la riduzione della sua complessità, una architettura REST impone di adottare una struttura gerarchica a livelli. Ogni livello è completamente indipendente dagli altri, ovvero è soltanto a conoscenza di quello con cui deve comunicare. Ogni livello può integrare parti del servizio, o servizi differenti del sistema è questo, sopratutto nell'ambito delle reti internet, aumenta considerevolmente la latenza dell'output verso il client che ha richiesto il servizio. Si può deviare dal problema aggiungendo al sistema dei cosiddetti livelli intermediari che eseguono operazioni di caching delle risorse e di distribuzione del carico all'interno del sistema. In riferimento al concetto di interfaccia uniforme, i livelli intermediari si occupano appunto del mantenere quest'ultima costante mentre il sistema alle loro spalle può cambiare, senza che nessuno se ne accorga.

Code on demand(OPTIONAL) Un parametro opzionale di una architettura REST è la possibilità dei server di estendere le capacità dei client trasferendo una determinata logica che può essere eseguita a destinazione.

2.2.2 Implementazioni Esistenti

Un protocollo di comunicazione utilizzato nelle reti che si presta ad essere molto compatibile con una architettura di tipo REST e sicuramente HTTP. Spesso il protocollo e l'architettura vengono mischiati all'interno della spiegazione di che cosa è REST, si tiene a precisare che l'architettura REST può essere applicata anche in un sistema che non fa uso di HTTP in quanto si tratta di una architettura che specifica il modo in cui client e server devono comunicare ma lascia libera l'implementazione del sistema.

Il 99% delle applicazioni RESTful viene usato il protocollo HTTP in quanto è già possibile associare i verbi dell'architettura REST con le richieste del protocollo:

- CREATE ⇒ POST, crea una nuova risorsa
- READ ⇒ GET, ottiene una risorsa specifica
- UPDATE ⇒ PUT, modifica una risorsa specifica
- DELETE ⇒ DELETE, elimina una risorsa specifica

Inoltre tramite il *payload* di HTTP è possibile scambiare tra client e server lo stato delle risorse e la relativa rappresentazione dati.

Il protocollo HTTP identifica le risorse tramite un **URL**(**U**niversal **R**esource **L**ocator) e garantisce all'architettura REST un modo per accedere univocamente alle risorse. Prima che le applicazioni cominciassero ad essere sviluppate secondo una architettura REST, non c'erano vincoli su come una richiesta a una determinata risorsa doveva essere effettuata il che rendeva l'identificazione delle risorse poco chiara.

```
1        GET index.php?service=getUser&id=55
```

Esempio 2.1: Esempio di URL che non rispetta il vincolo di REST

Nell'esempio 2.1 notiamo che l'URL non denota una specifica risorsa nel sever ma il metodo e i parametri che il server dovrà utilizzare per comporre il risultato e rispondere alla richiesta. Questo metodo non è errato ma non va bene per identificare delle risorse. In una architettura REST le risorse vengono identificate secondo precisi URL univoci all'interno dell'applicazione, come nell'esempio 2.2. Inoltre è buona pratica creare identificatori di risorse gerarchici, in modo da rendere l'URL della risorsa chiaro e semplice da comprendere.

```
1        GET /api/user/55
```

Esempio 2.2: Identificazione di una risorsa all'interno di una architettura REST

Per quanto riguarda la rappresentazione dei dati nello scambio di risorse, i formati più utilizzati sono JSON(**J**ava**S**cript **O**bject **N**otation) e XML(e**X**tensible **M**arkup **L**anguage) i quali permettono una struttura dettagliata e gerarchica dei dati. Nelle applicazioni web viene utilizzato maggiormente JSON in quanto la sua manipolazione risulta più diretta nel linguaggio Javascript.

La domanda che ora sorge spontanea è: come si creano delle **buone** API REST? Roy Fielding in suo articolo condanna l'uso dell'attributo RESTful su applicazioni che in realtà lo sono solo in parte, in quanto per far si che lo siano **tutti** i vincoli dell'architettura REST devono essere rispettati. Uno di questi che viene spesso trascurato e HATEOAS(2.2.1) e l'inventore di REST reclama i principi di come dovrebbero essere sviluppate correttamente delle API REST in questo articolo [13]

2.2.3 CORS

Si è parlato precedentemente della comunicazione tra client e server risiedenti su macchine differenti. Accade spesso che queste due entità siano situate su domini internet diversi e che la richiesta debba transitare da uno all'altro. CORS(**C**ross **O**rigin **R**esource **S**haring) è uno standard W3C che regolamenta le richieste di risorse tra domini differenti. È stato adottato per garantire un livello di sicurezza per l'accesso alle risorse. Per poterlo utilizzare bisogna agire sulla configurazione del server e sugli header dei messaggi HTTP. Ad esempio il dominio *www.marcopredari.it* vuole richiedere una risorsa presso *www.google.it*. Innanzitutto il dominio chiamante deve aggiungere all'header il campo **Origin** indicando il prorpio dominio. Come risposta riceverà un messaggio il cui header avrà all'interno il campo *Allow-Control-Allow-Origin* contenete i domini abilitati a richiedere quella risorsa; nel caso del simbolo * significa che accetta qualsiasi dominio la richieda. Se il dominio che fatto la richiesta non fosse autorizzato riceverà un messaggio di errore. [14]

```
1   Accept:application/json, text/plain, */*
2   Accept-Encoding:gzip, deflate, sdch
3   Accept-Language:it-IT,it;q=0.8,en-US;q=0.6,en;q=0.4
4   Cache-Control:no-cache
5   Connection:keep-alive
6   Host:predoweb-cms.herokuapp.com
7   Origin:http://127.0.0.1
8   Pragma:no-cache
9   Referer:http://127.0.0.1/webcv/blog.marcopredari.it/
10  User-Agent: Mozilla/5.0 (X11; Linux x86_64)
11                      AppleWebKit/537.36 (KHTML, like Gecko)
12                      Chrome/39.0.2171.99 Safari/537.36
```

Esempio 2.3: esempio di richiesta HTTP utilizzando CORS

```
1        Access-Control-Allow-Origin:http://127.0.0.1
2        Cache-Control:no-cache
3        Connection:keep-alive
4        Content-Type:application/json
5        Date:Tue, 20 Jan 2015 16:52:13 GMT
6        Server:Apache/2.4.10 (Unix)
7        Set-Cookie:laravel_session= .....
8                          expires=Tue, 20-Jan-2015 18:52:13 GMT;
9                          Max-Age=7200;
10                         path=/;
11                         httponly
12       Transfer-Encoding:chunked
13       Vary:Origin
14       Via:1.1 vegur
15       X-Frame-Options:SAMEORIGIN
16       X-Powered-By:PHP/5.6.4
```

ESEMPIO 2.4: esempio di risposta dal server che utilizza CORS

CORS possiede inoltre altre configurazioni riguardo alla sicurezza delle richieste, nei web server come Apache è possibile configurare un file in cui si possono aggiungere tutte le restrizioni o concessioni del caso.

2.3 Template System

Un *Template System* o *Template Engine* è uno strumento che permette al programmatore di separare nel programma, il contenuto(dato vero e proprio, modello) dalla sua rappresentazione(vista). Una analogia con un concetto già visto in questa tesi è con il pattern MVC, il contenuto e la rappresentazione sono analogamente il modello e la vista spiegati nella sezione 2.1.3. In questo caso un template è una particolare rappresentazione decisa a priori, senza alcuna dipendenza dai dati che si andranno a mostrare. Il ruolo del *Template Processor* è quello di unire i dati provenienti dal modello con il template. Ogni template può avere più rappresentazioni al suo interno(ovvero più punti d'entrata per il modello), quindi per ogni collezione di dati presente nel modello verranno prodotti più documenti con lo stesso template ma con dati diversi.

Il linguaggio usato per la definizione di template è generalmente moolto minimale, racchiude solo le principali caratteristiche dei più comuni linguaggi di programmazione di alto livello, come: variabili e funzioni(lambda), sostituzione di testo, inclusione di file, valutazioni condizionali e iterazioni. Un template system molto conosciuto perché disponibile in molti linguaggi e *Mustache*.

Mustache è definito come *Logic-Less template system*, generalmente questi sistemi utilizzano un sistema di tag in modo tale da sostituirlo con il dato dal modello. Alcuni tag possono variare di significato in base alla loro sintassi e al Template System scelto. Ad esempio mustache utilizza la sintassi: {{tag}} per identificare una corrispondenza con il modello.

Un modo molto utilizzato per rappresentare i dati all'interno del modello e tramite l'uso di JSON. In questo modo si possono scrivere modelli molto dettagliati che al loro interno rispettano la gerarchia dei dati e si ha una piena libertà riguardo le corrispondenze con i tipi, come nell'esempio 2.5

```
1
2  Model:
3
4  {
5          "name" : {
6                  "fist" : "Marco",
7                  "last" : "Predari"
8          },
9          "age" : "22",
10         "hobbies" : [
11                 {"hobby" : "Guitar"},
12                 {"hobby" : "Scout"},
13                 {"hobby" : "Hacking"}
14         ]
15 }
16
17 Template:
18
19 My name is {{name.first}} {{name.last}} and i am {{age}}.
20 My hobbies are:
21 {{#hobbies}}
22 <b>{{hobby}}</b>
23 {{/hobbies}}
24
25 Result:
26
27 My name is Marco Predari and i am 22.
28 My hobbies are:
29         <b>Guitar</b>
30         <b>Scout</b>
31         <b>Hacking</b>
```

Esempio 2.5: Un esempio di modello in json: la sua corrispondeza con il template system mustache e il risultato che si ottiene

Questo sistema viene molto utilizzato in ambito web, usare delle pagine web come template è

molto comune tra i vari sistemi in quanto l'HTML è un linguaggio già predisposto all'approccio dei template. Si parlerà nella sezione 3.1.2 di AngularJS, dall'alto può essere visto come un template system, la grande differenza sta nel *two-way data binding* che verrà spiegato sempre nello stesso capitolo.

Capitolo 3

Case Study: L'approccio ibrido con tecnologie web

L'approccio ibrido con tecnologie web è stato scelto durante lo svolgimento del tirocinio presso la ditta BigThink SRL, in quanto si adattava meglio ad una serie di tecnologie già utilizzate dall'azienda per lo sviluppo di applicazioni multipiattaforma. Durante la mia permanenza ho svolto il lavoro di Frontend Developer di Web Applications, in particolare mi occupavo dello sviluppo della logica di business di una applicazione e della sua interfaccia. Una delle politiche dell'azienda era quella di separare la gestione dei dati e la loro memorizzazione lato Backend in modo tale che la parte Frontend sviluppata con il framework Javascript AngularJS si interfacciasse con i dati servendosi solo di API REST. Dato il numero consistente di Web Application sviluppate e il mercato di applicazioni Mobile sempre in crescita, sono stato incaricato dall'azienda di ricercare dei metodi e delle tecnologie che consentissero di portare il lavoro già fatto per le Web Application su dispositivo mobile, in modo tale che l'azienda potesse offrire nuovi servizi.
Tutte le applicazioni sviluppate fino ad allora erano già Responsive Web Application, e ci si è posti l'obiettivo di aggiungere funzionalità caratteristiche dei dispositivi mobili, tramite l'utilizzo di tecnologie web.

Un primo punto di riferimento come tecnologia Web è stato AgnularJS, framework già utilizzato dall'azienda, se fosse stato scelto un altro framework si sarebbe dovuto riscrivere tutto il codice delle Web Application già sviluppate fino ad allora. A questo punto si trattava di scegliere le corrette tecnologie per includere al meglio il lavoro già svolto, in particolare un wrapper che avrebbe consentito un buon sviluppo dell'applicazione partendo da tecnologie web, senza influire su eventuali spese aziendali.

Per quanto riguarda gli strumenti di sviluppo utilizzati la scelta rimane allo sviluppatore, nel mio caso ho preferito elencarne alcuni secondo me importanti utilizzati all'interno dell'azienda e scoperti durante l'attività di ricerca.

3.1 Le Tecnologie Web utilizzate

Il riferimento per la scelta di tecnologie web è stato il framework *AngularJS*, il quale è utilizzato a sua volta all'interno di altri framework, come ad esempio *Ionic* che fornisce degli strumenti per la creazione di interfacce utente. Come wrapper ho scelto *Cordova*, in quanto oltre ad essere un progetto open-source fornisce delle API per le funzionalità del dispositivo in linguaggio Javascript. Infine ho sperimentato una libreria chiamata *ng-cordova*, la quale fornisce API per comunicare con il dispositivo ma con il codice già predisposto per AngularJS.

3.1.1 CSS Preprocessor

Nei web framework che forniscono strumenti per la definizione di interfacce utente un componente che si trova molto spesso e il *CSS Preporcessor* ovvero un preprocessore di fogli di stile.

In informatica, un preprocessore o precompilatore è un programma (o una porzione di programma) che effettua sostituzioni testuali sul codice sorgente di un programma, ovvero la precompilazione. I più comuni tipi di sostituzioni sono l'espansione di macro, l'inclusione di altri file, e la compilazione condizionale (vedi conditional compilation in inglese). Tipicamente, il preprocessore viene lanciato nel processo di compilazione di un software, e il file risultante verrà preso in input da un compilatore. [15]

FIGURA 3.1: **S**intatically **A**wesome **S**tyle**S**heet e **LESS**

Essendo CSS un linguaggio fortemente dichiarativo basato sui markup HTML, fa si che i fogli di stile per interfacce utente diventino molto lunghi e verbosi data la complessità delle interfacce utente. E di conseguenza risulta complesso applicare la propria personalizzazione. I CSS preprocessor mettono a disposizione un set di operazioni chiamate MACRO che durante la

compilazione verranno sostituite con il linguaggio CSS prorpio. Queste MACRO consentono ad esempio di fare utilizzo di variabili, funzioni, tag parametrici, ereditarietà dei tag, pattern matching, namespaces.

Queste sono solo alcune delle opzioni messe a disposizione dei CSS preprocessors dipende dalle sviluppatore scegliere quello secondo lui più adatto in quanto sul mercato ne esistono diversi, i più famosi e utilizzati sono **SASS**(**S**intatically **A**wesome **S**tyle**S**heet) e **LESS**(write **LESS** do more) 3.1.

3.1.2 AngularJS

FIGURA 3.2:
AngularJS framework logo

Definizione AngularJS è un framework Javascript che segue il pattern MVC(2.1.3) ideato da *Minsko Hevery*(Google) che estende il linguaggio HTML in un formato più espressivo e leggibile. È caratterizzato da un approccio dichiarativo alla programmazione ed è stato pensato per separare la logica di business di una applicazione dalla sua presentazione dei dati, infatti sono presenti tag HTML espliciti come *ng-view* e *ng-model* che consentono di sincronizzare i dati del modello con la vista indipendentemente dalla loro logica. AngularJS è stato pensato esplicitamente per lo sviluppo di web application è presenta una forte modularità dei suoi componenti, in modo da renderli indipendenti e testabili, rispetta infatti i principi *SOLID* [16] della programmazione orientata agli oggetti. Ecco le caratteristiche principali di questo framework:

Directives AngularJS si basa su un sistema di direttive, ovvero dei nuovi tag HTML che estendono il linguaggio di markup. Come viene mostrato nell'esempio 3.1 ci sono delle direttive già predisposte dal framework per inserire i componenti principali a disposizione.

```
1 <html ng-app="myApp">
2     <div ng-controller = "InfoController">
3       <input name = "inputFirstName" ng-model = "user.firstName">
4       <input name = "inputSecondName" ng-model = "user.secondName">
5       <div class = "showInfo">
6             Hello my name is {{user.firstName}} {{user.secondName}}
7       </div>
8     </div>
9 </html>
```

ESEMPIO 3.1: Un esempio delle direttive standard di AngularJS

AngularJS inoltre offre la possibilità di definire direttive personalizzate 3.3. La differenza sostanziale rispetto ai semplici tag HTML è che ad ogni direttiva è associato un particolare comportamento dell'applicazione. Mentre nei tag HTML si specifica come un certo elemento debba apparire nella pagina, ad esempio se contiene una porzione di testo oppure un'immagine, con le direttive di AngularJS associamo un comportamento al tag ed eventualmente un template su cui basarsi. Principalmente all'interno di una applicazione AngularJS le direttive vengono utilizzate per esprimere interfacce utente complesse, gestire eventi scatenati dall'interazione con l'utente, re-utilizzo dei componenti più comuni e la ridefinizione dei tag già in uso. La buona pratica del framework vuole che solo all'interno delle direttive si possa manipolare il DOM.

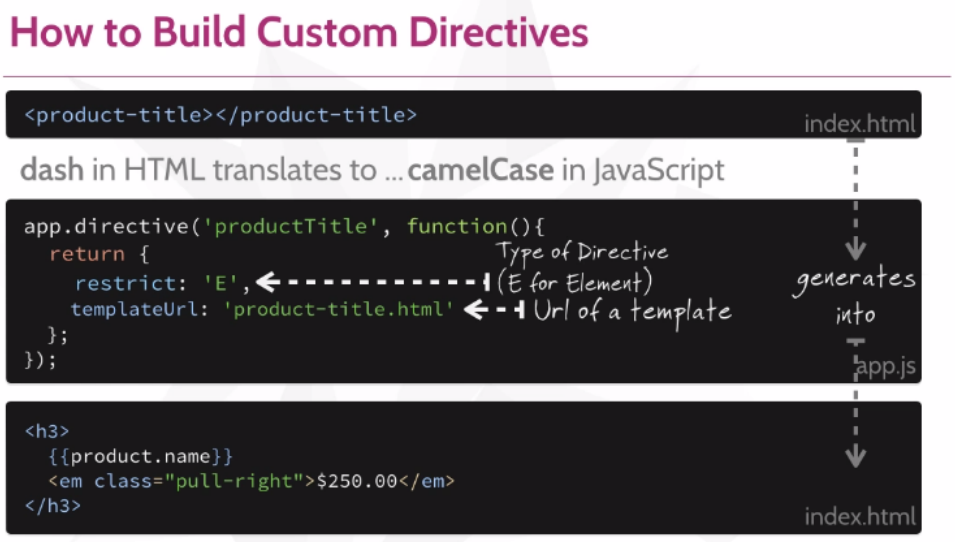


FIGURA 3.3: Creazione di una direttiva personalizzata

Una nota di sintassi particolare per le direttive riguarda il loro nome, come osserviamo il nome delle direttive nell'esempio 3.3 è separato dal trattino alto(*dash*) mentre nella definizione tramite codice AngularJS è scritta in *camelCase*. Questa è un convenzione che il framework adotta per evitare errori di sintassi all'interno del codice da parte del parser. Le direttive hanno a disposizione molte opzioni per essere personalizzate, per un trattazione più approfondita e completa si rimanda la lettura del manuale [17].

Data Binding Il Data Binding in AngularJS è la sincronizzazione automatica dei dati tra modello e vista. In modo in cui AngularJS implementa il data bindign consente di trattare il modello come SSOT([18]). La vista è la proiezione del modello in ogni momento, quando il modello cambia la vista riflette il cambiamento e viceversa. La maggior parte dei sistemi di templating sincronizzano i dati in una sola direzione: si fondono componenti del template e modello insieme in una vista. Quando si verifica la fusione, le modifiche al modello o sezioni correlate della vista vengono NON si riflettono automaticamente nella vista. Peggio ancora, le eventuali modifiche che l'utente fa nella vista non si riflettono nel modello. Ciò significa che

lo sviluppatore deve scrivere del codice apposito che sincronizza costantemente la vista con il modello e il modello con la vista.

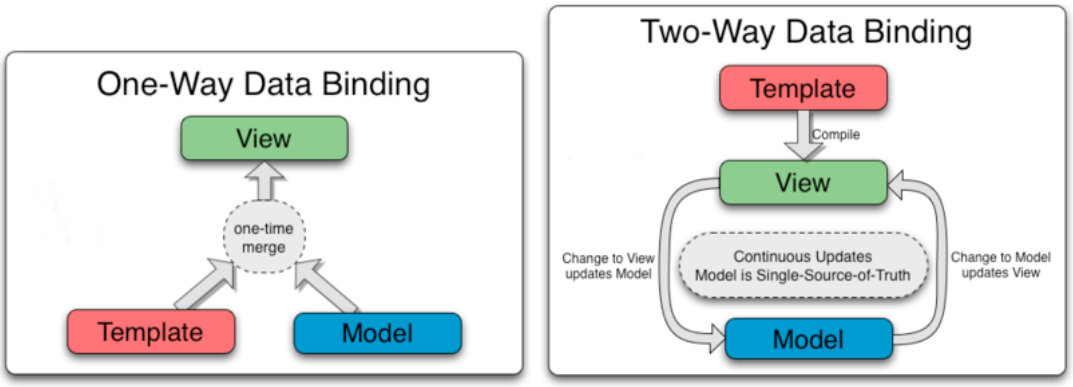


FIGURA 3.4: Come è strutturato un classico data binding, e come invece è fatto in AngularJS

In particolare come si vede nell'esempio 3.1 il binding dalla vista al modello è effettuato tramite la direttiva *ng-model* = *nome-del-modello*, mentre dal modello alla vista si utilizza la sintassi {{*nome-del-modello*}} .

Il sistema dei template di AngularJS funziona in modo differente. In primo luogo il modello (che è l'HTML non compilato insieme a markup o direttive supplementari) viene compilato sul browser. Il passo di compilazione produce una live view(vista). Eventuali modifiche alla vista si riflettono immediatamente nel modello, e le eventuali modifiche nel modello vengono propagate alla vista. Il modello è il SSOT per lo stato dell'applicazione, semplificando notevolmente il modello di programmazione per lo sviluppatore. Si può pensare di vista semplicemente come una proiezione istantanea del modello.

Poiché la vista è solo una proiezione del modello, il controller è completamente separato dal punto di vista e completamente allo scuro di essa. Questo rende i test dell'applicazione molto semplici, in quanto è facile testare il controller indipendentemente dalla vista e dalla relativa dipendenza dal DOM / browser web.

Controller i controller in AngularJS hanno il compito di gestire la logica di una determinata sezione dell'applicazione. Quando un controller viene dichiarato su una parte del DOM tramite la direttiva *ng-controller* come nell'esempio 3.2 AngularJS crea un nuovo oggetto Javascript associandogli un nuovo scope che potrà essere inserito all'interno del controller tramite la dependency injection.

```
1        <div id="header" ng-controller = "HeaderController"></div>
```

ESEMPIO 3.2: Associazione tra un elemento del DOM e un controller

In generale ci sono alcune regole da seguire per la creazione dei controller, le quali non sono obbligatorie ma sono considerate come si dice in gergo *Best Practices*: I controller devono essere usati per:

- Configurare lo stato iniziale dello scope.
- Aggiungere comportamento allo scope(funzioni, modelli, oggetti)

Mentre i controller non devono essere usati per:

- Manipolare elementi del DOM
- Formattare input e output
- Condividere codice tra i vari controller.
- Gestire il ciclo di vita degli altri componenti(creazione di nuove istanze)

View In base ai contenuti e al template scelto per visualizzarli AngularJS genera una vista, che non è altro che la proiezione dei dati in un certo momento dell'applicazione. Inoltre grazie ad un meccanismo di routing, è possibile impostare un workflow dell'applicazione specificando le varie viste possibili, anche in maniera gerarchica.

Dependency Injection per una trattazione generale dell'argomento rimandiamo all'appendice A.4. AngularJS ha già al suo interno un meccanismo che gestisce la Dependency Injection. Il principale scopo per cui AngularJS è stato dotato di questa caratteristica è per avere la possibilità di suddividere in moduli separati l'applicazione dove ognuno di essi può essere iniettato all'interno degli altri e vice versa. Un esempio moto semplice di Dependency Injection in AngularJS è all'atto della creazione dell'applicazione. Nell'esempio 3.3 il metodo *angular.module()* prende due argomenti: il primo è il nome del modulo che si vuole creare, mentre il secondo è un array con tutti i moduli di cui è composto la nostra applicazione. Viene creato quindi un riferimento ai moduli inclusi e non un istanza diretta.

```
1        var testApp = angular.module("testApp",['ngRoute','customModule']);
```

Esempio 3.3: Creazione di una applicazione in AngularJS con le relative dipendenze

Oltre ai moduli AngularJS da la possibilità di usare la propietà della Dependency Injection sui tipi base forniti dal linguaggio, ovvero:

- Value
- Factory
- Service
- Provider
- Constant

Un esempio di Dependency Injection di questi tipi avviene molto frequentemente nelle applicazioni all'interno dei controller (3.4)

```
1
2  angular.module("CustomModule",[])
3  .service("CustomService",function(){
4          this.customMethod = function(value){
5                  return value + " from customMethod in CustomService";
6          }
7  });
8
9  angular.module("testApp",[
10         'CustomModule'
11 ]).controller("HomeController", function(CustomService){
12         CustomService.customMethod("Marco");
13 });
```

ESEMPIO 3.4: Un esempio di creazione di un modulo e la sua inclusione all'interno di un altro

Service si è visto come il meccanismo di Dependency Injection possa rendere disponibile all'interno dell'applicazione tutti i componenti forniti da un specifico modulo. Fatta eccezione per i *service* ogni volta che si richiama un componente tramite Dependency Injection dal riferimento viene creata una nuova istanza della classe. I service invece sfruttano quello che è chiamato in Javascript il *Singleton Object*, ovvero, l'istanza di un oggetto di tipo service avviene soltanto una volta, e il riferimento all'interno dell'applicazione è univoco verso lo stesso oggetto(figura 3.5). Inoltre soltanto quando il service dipende dall'applicazione viene creata l'istanza dell'oggetto(*Lazy Initialization*).

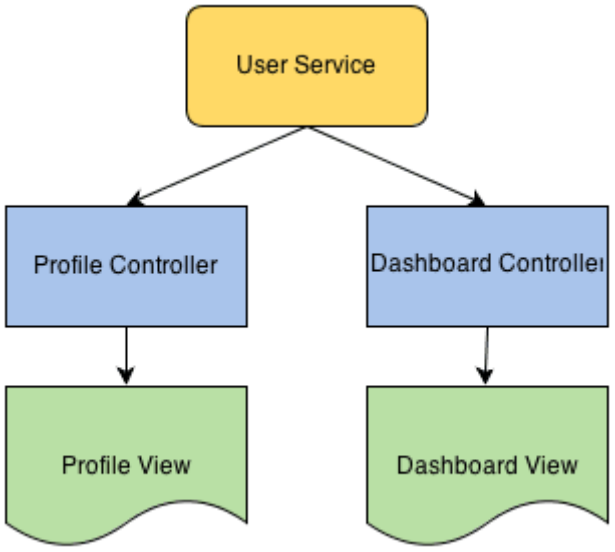


FIGURA 3.5: Schema di come è strutturato un service all'interno di AngularJS

I service possono essere creati dallo sviluppatore oppure AngularJS ne mette a disposizione già alcuni, come ad esempio *$http* che è uno dei più usati per lo scambio dei dati attraverso l'omonimo protocollo. Altro uso dei service può essere quello dello scambio dei dati attraverso i controller, genericamente in una applicazione che si interfaccia con dei servizi backend e buona norma associare ad ogni servizio corrispettivo un service di AngularJS.

Provider/Factory/Service una trattazione separata di queste tre caratteristiche di AngularJS potrebbe risultare difficile da comprendere. Avendo spiegato cosa è un service la trattazione prosegue su altri due componenti strettamente correlati: i Factory e i Provider. Tra questi tre componenti esiste una specie di gerarchia : i *service* sono degli oggetti singleton creati tramite una *factory*. Le *factory* sono funzioni che a sua volta sono create tramite un *provider*. I *provider* sono dei costruttori che hanno a disposizione un metodo *$get* il quale contiene la *service factory* che ritorna l'istanza del service.

In questo modo AngularJS si garantisce una forte modularizzazione dei componenti, e la ricerca dei corretti riferimenti da parte della Dependency Injection. Inoltre tramite il provider è possibile configurare, se previsto, i servizi dell'applicazione scrivendo un opportuno codice parametrico.

3.1.3 Ionic

IONIC è un UI Framework basato sulle tecnologie web HTML5, Sass, AngularJS per la creazione di interfacce utente per dispositivi mobili. Si basa sul sistema di direttive di AnagularJS è fornisce un nuovo set di tag HTML componibili tra di loro. Gli UI Framework e IONIC principalmente dispongono di 3 categorie di elementi che li caratterizzano e che servono per l'appunto alla composizione dell'interfaccia utente:

FIGURA 3.6: Ionic Framework Logo

CSS Components Sono dei componenti statici creati tramite HTML e CSS come bottoni, barre di navigazione, liste, tabelle che possono essere assemblati tra di loro come i componenti di una pagina web. I CSS di IONIC mettono a disposizione classi che possono cambiarne forma, dimensione e colore che a loro volta possono essere combinate. Per personalizzazioni più avanzate e possibile seguire una guida messa a disposizione dal framework che spiega come modificare i file di Sass e che caratteristiche si va a cambiare(come ad esempio i colori standard).

Javascript Components Al fine di offrire una esperienza di una applicazione mobile all'utente, IONIC offre delle estensioni in AngularJS che ricalcano quelle che sono le operazioni e le interfacce più comuni sui i dispositivi mobili. Ispirato ai sistemi iOS e Android questo

framework mette a disposizione componenti come gestori di eventi, paginazione dei contenuti, popup di sistema, touch gestures, menu laterali, scorrimento di pagine il tutto nello stile di una applicazione mobile vera e prorpia.

Ionicons IONIC prevede un set di icone standard che possono essere utilizzate all'interno dei componenti e personalizzate a proprio piacimento.

3.2 Il Wrapper Framework

Come spiegato nella sezione 1.5 per lo sviluppo di applicazioni multipiattaforma si necessita di un framework che possa in qualche modo tradurre il linguaggio originale in quello della specifica piattaforma. Nel caso di una applicazione ibrida, si ha bisogno di un framework che possa ospitare tecnologie web e che abbia un interfaccia verso le funzionalità del dispositivo. Si ribadisce che non tutti i framework possono essere usati su tutte le piattaforme, bisogna scegliere con accortezza di quale servirsi. Data la mia formazione come sviluppatore web presso l'azienda BigThink SRL ho scelto di utilizzare il framework *Cordova* per i seguenti motivi:

- E' un framework che si adatta a sviluppatori web che vogliono portare le proprie applicazioni su dispositivi mobili, in quanto fornisce le varie API per le funzionalità del dispositivo in linguaggio Javascript.
- E' open-source quindi costi di licenza nulli per l'azienda e per lo sviluppatore ed inoltre e seguito da una community popolata e attiva.
- Supporta 16 piattaforme diverse con oltre 20 plugin per interagire con il dispositivo, assieme ad altre librerie per la creazione di plugin personalizzati.
- Si integra al meglio con AngularJS grazie ad una libreria chiamata *ng-cordova*(3.2.2).

3.2.1 Cordova/Phonegap

Per chi magari è nuovo nel settore delle applicazioni multi-piattaforma, o per chi ci è entrato da poco avrà fatto sicuramente confusione tra queste due nomenclature `Cordova` e `Phonegap`, ecco quindi una delucidazione sul fatto.

FIGURA 3.7:
Cordova Framework Logo

3.2.1.1 Storia

Phonegap è stato creato nel 2009 da una startup chiamata *Nitobi* come progetto open-source. Si proponeva di fornire un metodo per l'accesso alle funzionalità native del dispositivo tramite il meccanismo di wrapping che è stato discusso nel capitolo precedente a proposito dei framework. L'obiettivo di questa piattaforma era appunto quello di poter creare delle applicazioni che potessero essere usate nei dispositivi mobili, tramite l'utilizzo di tecnologie web come HTML5, CSS e Javascript, ma con ancora la possibilità di accedere alle funzionalità native del dispositivo. Nel 2011 *Adobe* ha acquisito la startup Nitobi assieme ai diritti di Phonegap, e il codice open-source della piattaforma è stato donato all'*Apache Software Foundation* con il nome di `Cordova`

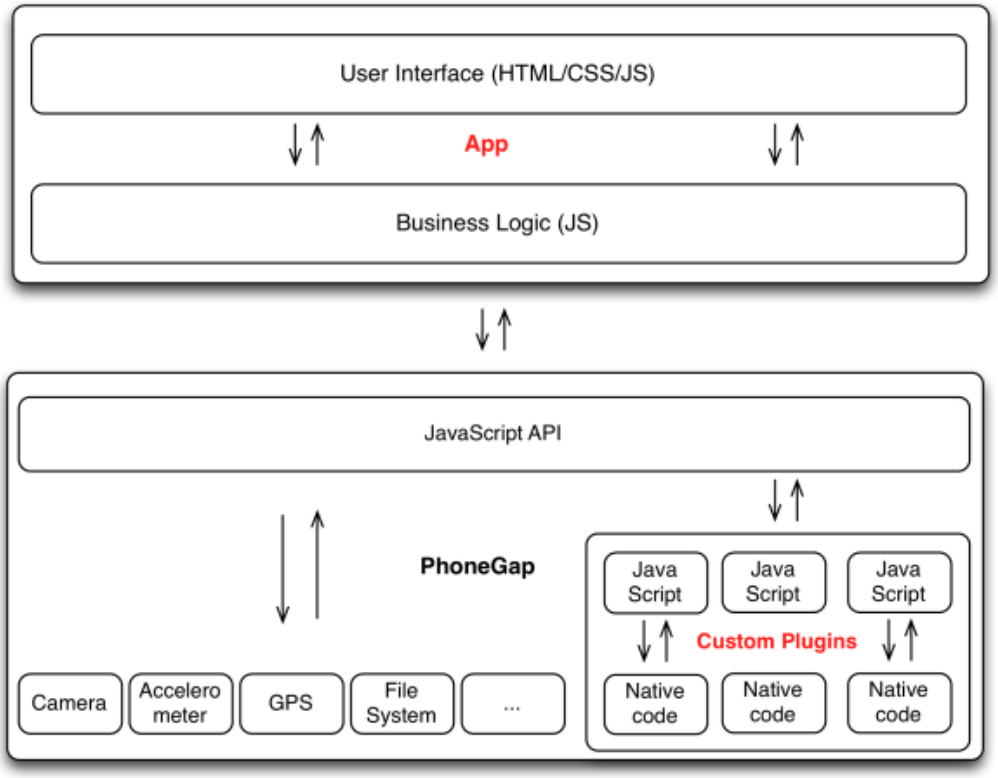


FIGURA 3.8: Come avviene le stratificazione tra l'applicazione e il framework Cordova

3.2.1.2 Differenze

La vera differenza tra `Cordova` e `Phonegap`, viene descritta da *Adobe* analogamente come la differenza tra `Blink`[1] e *Google Chrome*. Ovvero *Cordova* e il cuore della piattaforma mentre

[1] Blink è la web browser engine di Google Chrome[19]

Phonegap aggiunge a *Cordova* delle funzionalità proprietarie di *Adobe*.
Personalmente ho scelto di utilizzare *Cordova* per essere libero da qualsiasi vincolo proprietario.

3.2.1.3 Cordova Core

Cordova quindi offre una serie di potenti API in linguaggio Javascript per poter accedere alle funzionalità native del dispositivo. In difesa dello sviluppo nativo alcuni programmatori accusano *Cordova* di non possedere tutte le possibilità di accesso a basso livello che invece si avrebbero. *Cordova* è una realtà open-source e in quanto tale si è evoluta nel tempo offrendo sempre più funzionalità che hanno chiuso i divario che si credeva esservi tra questi due tipi di approcci.

Nella figura 3.8 si può osservare dove si posiziona il framewrok all'interno di una tipica struttura stratificata di applicazione, e di come i livelli siano completamente indipendenti l'uno dall'altro. Questo fa si che per l'approccio cross-platform attraverso le applicazioni ibride, la scelta delle tecnologie web sia completamente irrilevante ai fini della creazione dell'app.

3.2.2 ngCordova

Il perché in questa tesi si parli di *Cordova* e *AngularJS* è dato dall'esistenza di `ngCordova`. Questa libreria nasce da una idea di Paolo Bernasconi e Max Lynch che hanno avuto l'idea di unire la l'efficenza e la potenza di *AngularJS* con la versatilità di *Cordova*. Ne è nato un framework per lo sviluppo di applicazioni ibride direttamente collegato alle funzionalità del dispositivo gestibile tramite il codice efficiente di *AngularJS*.

FIGURA 3.9:
ngCordova Logo

Nell'architettura mostrata nella figura 3.8 ng-cordova si posiziona al di sopra delle API di Cordova, ridefinendo i plugin nella sintassi di AngularJS.

3.3 Strumenti di sviluppo

Il consistente numero di tecnologie che si possono utilizzare per lo sviluppo ibrido viene spesso accompagnato da strumenti che consentono di creare un ambiente progettuale in cui si ha a disposizione tutto il necessario per poter iniziare a sviluppare. La scelta di determinati strumenti

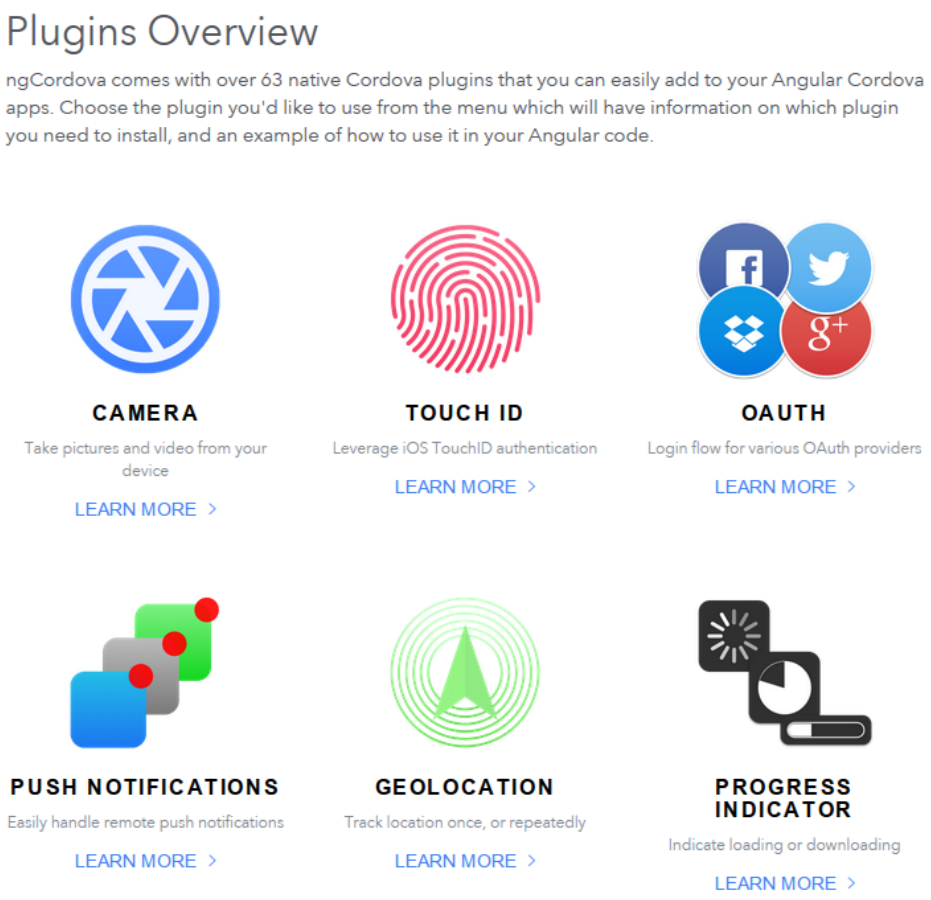


FIGURA 3.10: Le principali categorie dei plugin di ng-corodva

piuttosto che altri è riservata allo sviluppatore finale, personalmente mostrerò come ho deciso di organizzare un progetto di esempio.

3.3.1 Package Manager

Avendo a disposizione tutto il mondo delle tecnologie web, può risultare utile e molto produttivo inserire librerie all'interno del nostro progetto. Per le applicazioni web in generale e anche in quelle ibride si può disporre di un package manager che facilmente includere nuove librerie all'interno del progetto.

Bower è un package manager tipicamente usato per librerie Javascript. Generalmente è possibile reperire ogni libreria disponibile su Github. Tramite il file *bower.json* possiamo specificare le dipendenze del progetto e dinamicamente aggiornare la lista delle librerie installate. Inoltre se stiamo lavorando con una struttura di cartelle specifica, come ne mio caso, possiamo specificare la posizione di dove verranno scaricate le librerie tramite il file *.bowerrc*.

```
1        {
2                "directory" : "js/vendor"
3        }
```

Esempio 3.5: Una tipica configurazione del file .bowerrc

```
1  {
2    "name": "blog.marcopredari.it",
3    "version": "0.0.0",
4    "authors": [
5      "predorock <predorock@gmail.com>"
6    ],
7    "description": "A simple blog with angularJS",
8    "main": "index.html",
9    "license": "MIT",
10   "homepage": "blog.marcopredari.it",
11   "private": true,
12   "ignore": [
13     "**/.*",
14     "node_modules",
15     "bower_components",
16     "test",
17     "tests"
18   ],
19   "dependencies": {
20     "angular-sanitize": "~1.3.10",
21     "angular": "~1.3.10"
22   }\begin{minipage}[\linewidth]
23 }
```

Esempio 3.6: Una tipica configurazione del file bower.json

3.3.2 Task Runner

La divisione logica in moduli di una applicazione porta i programmatori a separare in file diversi l'applicazione in fase di sviluppo. Per poter ottenere una versione in fase di produzione occorre assemblare tutte le parti del progetto e se necessario eseguire dei test. Il ruolo dei task-runner è quello di eseguire determinate operazioni per la messa in produzione dell'applicazione, come la concatenazione dei file, controllo della sintassi, test sui

Figura 3.11: Loghi di Grunt e Gulp

moduli e lo spostamento del progetto nella directory
finale.

Grunt e **Gulp** sono due esempi di *task-runner* in linguaggio Javascript(NodeJs). Una caratteristica che li distingue e la loro modularità, in quanto in base alle esigenze dell'applicazione si possono scaricare i moduli dedicati in base ai processi richiesti per la messa in produzione. Successivamente si deve scrivere un file(*grunt.js* o *gulp.js*) dove si indicano tutte le istruzioni per la messa in produzione. Ho voluto nominarli entrambi nella mia tesi in quanto il primo l'ho usato durante la mia esperienza di tirocinio, il secondo invece è usato per lo sviluppo del framework Ionic.

3.4 SDK

Un `Software Developement Kit` in generale è un insieme di strumenti per lo sviluppo e la documentazione del software[20]. Questi strumenti vengono rilasciati dalla casa produttrice di una certa piattaforma come librerie di riferimento per lo sviluppo di software specifico di essa. La loro distribuzione avviene in uno specifico linguaggio a seconda della piattaforma ed è sempre affiancata da una documentazione molto accurata. Per lo sviluppo di applicazioni ibride avremo bisogno di ciascuna *SDK* per ogni sistema operativo sul quale vorremo la nostra applicazione; ad esempio se volessimo la nostra applicazione per *iOS* e *Android* dovremmo scaricare entrambe le *SDK*(scritte rispettivamente in C++/Swift e Java) indicando dove si trovano all'interno del notro computer.

3.5 Pattern Javascript e best practices

Per lo sviluppo di applicazioni ibride come si è notato il linguaggio che gestisce la logica dell'applicazione è senza dubbio Javascript. In risposta ai problemi che ho riscontrato durante la fase di sperimentazione ho raccolto una serie di pattern / best-practices secondo me utili allo sviluppo ibrido.

3.5.1 Utilizzo delle Promises

Una promise rappresenta un'interfaccia di delegazione verso un valore non ancora conosciuto. Essa consente di associare due tipi di comportamento ad una azione asincrona, nel caso di successo oppure di fallimento. [21] Questo pattern e molto usato nel caso di chiamate ad API REST, molto spesso le risorse si trovano su un altro dominio che impiega un certo tempo a

fornire il risultato. Mentre nello sviluppo ibrido vengono utilizzate per la chiamata alle API dei vari plugin associati ad una funzionalità del dispositivo.

```
1  module.controller("PictureCtrl",function($scope, $cordovaCamera){
2    document.addEventListener("deviceReady",function(){
3      var options = {
4        quality: 50,
5        destinationType: Camera.DestinationType.DATA_URL,
6        sourceType: Camera.PictureSourceType.Camera
7        allowEdit: true,
8        encodingType: Camera.EncodingType.JPEG,
9        targetWidth: 100,
10       targetHeight: 100,
11       popoverOptions: CameraPopoverOptions,
12       saveToPhotoAlbum: false
13     };
14   $cordovaCamera.getPicture(options).then(
15      function(imageData){
16       var image = document.getElementById('myImage');
17       image.src = "data:image/jpeg:base64" + imageData;
18      },
19      function(err){
20       alert("Something has gone wrong because :" + err);
21      }
22     ;
23    }, false);
24   });
25 $
```

Esempio 3.7: Un esempio di uso delle promises in ng-cordva

Come si può vedere nell'esempio 3.7 la funzione *.then()* viene applicata ad una promise che gestisce i dati provenienti dalla fotocamera di un dispositivo. Successivamente vengono specificate le due funzioni nel caso di successo o nel caso di errore della chiamata all'API.

3.5.2 Stato dell'applicazione

In una applicazione ibrida basata su Cordova esistono degli stati nei quali l'applicazione si può trovare. Come mostra la figura 3.12 durante la transizione degli stati vengono scatenati degli eventi che possono essere gestiti tramite codice Javascript.

L'evento più importante tra tutti è *deviceReady* il quale indica che Cordova è stato caricato completamente e soltanto dopo questo evento possono essere richiamate le API per interagire con le funzioni del dispositivo. Gli eventi *pause* e *resume* indicano rispettivamente quando una applicazione viene messa in background e quando invece viene ripresa la sua esecuzione.

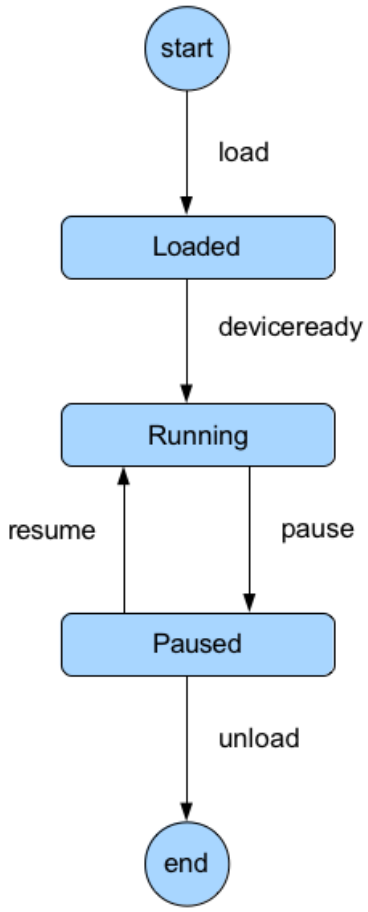


FIGURA 3.12: Schema della transizione degli stati e degli eventi scatenati

Durante questi due eventi si può ottimizzare la gestione della memoria da parte dell'applicazione per non appesantire di calcoli il dispositivo. Ulteriori eventi che possono essere scatenati dalla pressione di pulsanti oppure da informazioni riguardo lo stato della batteria o quello della rete sono disponibili sulla documentazione [22].

3.5.3 Routing

Un buona pratica in una applicazione ibrida, quando si utilizzano tecnologie web, è definire la struttura di navigazione tra le viste dell'applicazione. Tramite il *$routeProvider* messo a disposizione si possono configurare i cambiamenti di stato tra una vista e l'altra, assegnando per ciascuna di esse il template e il controller da caricare. Esiste anche un meccanismo più avanzato utilizzato da IONIC specifico per le applicazioni mobile chiamato *$stateprovider* detto anche *angular-ui-router*. Le differenze principali dal routing originale di AngularJS sono che uno *stato* corrisponde ad un determinato posto all'interno dell'applicazione nei termini della User Interface e della navigazione. Uno *stato* descrive, tramite il controller, il template e le proprietà

della view, l'aspetto della UI e il suo comportamento in quel posto. Può esistere una struttura gerarchica degli stati per cui alcuni possono trovarsi all'interno di altri. [23]

3.5.4 OAUTH

OAUTH è un protocollo sviluppato alla fine del 2012 per lo scambio di dati sensibili attraverso servizi di terze parti [24]. In ambito web è conosciuto per il suo ampio uso nell'autenticazione attraverso provider differenti, ovvero grazie a questo protocollo e possibile autenticarsi ad un servizio / applicazione attraverso un altro di questi dove si è già registrati. Un esempio lampante è l'autenticazione tramite il proprio account di facebook, molto spesso le applicazioni per ovviare la pigrizia di alcuni utenti a registrarsi al loro servizio, mettono a disposizione un bottone tramite il quale l'utente acconsente che un provider, in questo caso facebook, possa fornire all'applicazione i dati necessari all'autenticazione o registrazione. OAUTH garantisce uno scambio di dati protetto con la massima semplicità. È presente un plugin in ng-cordova per l'autenticazione verso i provider più comuni.

Capitolo 4

Conclusioni

In questo capitolo verranno verificate le premesse e gli obiettivi dati all'inizio della tesi. In particolare verrà riportata l'analisi sulle tecnologie web per lo sviluppo ibrido multipiattaforma, mettendo a confronto gli altri metodi di sviluppo analizzati. Verranno spiegati i cambiamenti che ci sono stati sul processo di sviluppo, avendo prefissato l'obiettivo di avere una applicazione mutlipiattaforma e quali ottimizzazioni sono state adottate.

4.1 Ibrido vs Nativo

L'obiettivo di questa tesi è stato fin da subito quello di trovare delle metodologie rapide per sviluppare applicazioni multipiattaforma. L'approccio che è stato analizzato comprendeva l'utilizzo di tecnologie web per applicazioni ibride. Effettivamente tramite l'impiego degli opportuni framework per la distribuzione di più piattaforme, è stato possibile partendo da un unico codice sorgente in HTML / CSS / Javascript ottenere il risultato desiderato. Come spiegato nella sezione 1.3 ci sono pro e contro sull'utilizzo di tecnologie web per lo sviluppo di applicazioni mobile. Tramite l'approccio ibrido è stato possibile combinare i vantaggi di questi due approcci, ma fino ad un certo punto. Nella tabella 4.1 si possono osservare vantaggi e svantaggi derivati dai due approcci.

Per quanto riguarda la creazione di una interfaccia utente e l'impostazione della user experience, si accusa spesso l'approccio ibrido di non poter garantire un risultato soddisfacente paragonato allo sviluppo nativo. Le principali accuse che vengono fatte ad una interfaccia web tramite approccio ibrido, riguardano la fluidità delle transizioni e animazioni, e di non garantire una UX ottimale. In questo caso mi permetto di dissentire da questa opinione, in quanto dalla mia esperienza di utilizzo dei framework web per la creazione della UI (come ad esempio IONIC o Foundation), l'interfaccia risulta fluida e accattivante anche se eseguita all'interno di una

Applicazioni	
Ibride	
Pro	**Contro**
Le tecnologie web hanno una curva di apprendimento più bassa. Gli sviluppatori che vogliono usare tecnologie web per creare applicazioni mobile, impiegano meno tempo ad imparare il linguaggi (HTML / CSS / JS) rispetto ad uno nativo.	Essendo Javascript un linguaggio interpretato all'interno della WebView di Webkit, sarà comunque meno performante di un linguaggio nativo compilato, nonostante un alta efficienza del motore.
utilizzAppendiceare Permette di avere da un unico sorgente un'applicazione per ciascuna piattaforma, senza dover riscrivere il codice.	Pur essendo un linguaggio facile, non si hanno strumenti altrettanto potenti per poter fare debug del codice e memory profiling (anche se ultimamente stanno nascendo tool molto interessanti).
Grazie ai punti precedenti, può permettere un risparmio di tempi e costi, specialmente per la creazione di applicazioni multipiattaforma.	L'accesso alle funzionalità native del dispositivo è vincolato dal wrapper che fornisce le API in Javascript. Dipende quindi dalla tecnologia che si va ad utilizzare.
Native	
Pro	**Contro**
Tramite il linguaggio nativo della piattaforma, si può ottenere la massima performance in termini di potenza di calcolo e di sfruttamento del processore.	E' necessario riscrivere l'applicazione per ogni piattaforma che si intende supportare.
Sfrutta tutte le caratteristiche specifiche della piattaforma, sia in termini di software (librerie del produttore o di terze parti) che di hardware (del dispositivo o esterno)	Richiede conoscenze specifiche dei vari linguaggi delle varie piattaforme da parte degli sviluppatori.
Si ha un accesso più a basso livello della piattaforma, cosa che spesso non è consentita alle API per ragioni di sicurezza.	A causa dei punti precedenti può comportare un maggiore costo complessivo e l'allungamento dei tempi di sviluppo, in particolare dovendo gestire più piattaforme.

Tabella 4.1: HTML5 vs Nativo, pro e contro dei due approcci

WebView. Se si fa utilizzo di animazioni CSS3 (il quale processo di rendering è accelerato dall'hardware), invece della manipolazione del DOM tramite javascript, la differenza con una applicazione nativa è pressoché nulla.

4.2 I cambiamenti sul processo di sviluppo

Durante la ricerca di soluzioni per lo sviluppo rapido di applicazioni multipiattaforma tramite l'approccio ibrido, il processo di sviluppo dell'applicazione ha subito una consistente semplificazione. Questo è dovuto al fatto che, avendo un solo progetto da gestire, dal quale poi si otterranno le applicazioni per diverse piattaforme, le varie operazioni di design, test o risoluzione di bachi risultano più rapide e semplici. Nella figura 4.1 viene schematizzato come

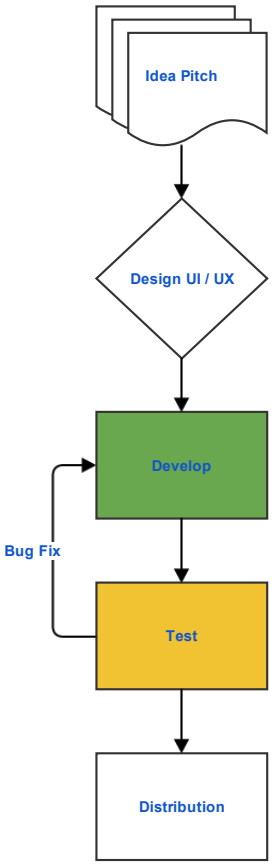


FIGURA 4.1: In questo schema sono illustrate le fasi generiche dello sviluppo di una applicazione

avviene il processo di creazione di una applicazione in generale. In particolare la semplificazione del progetto è dovuta a:

Design Il design dell'applicazione rimane univoco e non ci si deve preoccupare di replicarlo per le diverse piattaforme.

Test Avendo un unico codice sorgente, i test funzionali sull'applicazione sono unici, rimangono differenziati solo i test sull'integrazione nei singoli sistemi.

Team di sviluppo ristretto Con un solo progetto da gestire il team di sviluppo è meno affollato, facilitandone la sua gestione.

Distribuzione Tutte le applicazioni sono pronte nello stesso momento. In questo modo si ha subito un feedback globale da parte degli utenti, cosa che può eventualmente aiutare a correggere l'applicazione più rapidamente.

4.3 Ulteriori otimizzazioni dell'approcio ibrido

Nello sviluppo di una applicazione in generale, tutte le figure professionali coinvolte fanno riferimento ad uno schema generale, che illustra le fasi di lavoro per arrivare al prodotto finale. Da quanto appreso durante l'esperienza di tirocinio in azienda, questo schema deve essere il più generico possibile, e comprensibile da chiunque lavori con esso. Nella figura 4.1 si possono osservare le fasi principali dello sviluppo di una applicazione.

Prendendo spunto da questo schema generale ho creato una espansione della fase di sviluppo e di test per la realizzazione di applicazioni multipiattaforma ibride. Si tratta di un sottoschema che evidenzia quali fasi hanno caratterizzato un primo approccio a questo tipo di sviluppo (4.2).

Una prima fase è stata quella di scegliere quali tecnologie web utilizzare per lo sviluppo ibrido, e quali per testare il codice che doveva essere prodotto. Nel mio caso il framework AngularJS si prestava già ad uno sviluppo di questo tipo e non ho impiegato molto tempo nel capire come utilizzarlo per lo sviluppo mobile. In altri casi, data la vasta scelta di framework javascript per lo sviluppo mobile, può richiedere parecchio tempo(dipende dalle competenze e dallo stile del programmatore).

Certamente iterare questo processo per ciascuna applicazione che si andrà a sviluppare è molto dispendioso in termini di tempo e risorse. Una buona soluzione e costruirsi un set di tecnologie che si andranno a utilizzare per lo sviluppo di ciascuna applicazione futura, quello che in azienda è stato chiamato *skeleton*. Ovviamente lo scheletro di una applicazione tipo deve essere fatto in modo da espandersi nel caso giungessero richieste di nuove caratteristiche non ancora incluse. Nella figura 4.2 si può osservare l'evoluzione del primo schema verso una soluzione pronta e rapida all'utilizzo, grazie all'introduzione dello *skeleton*.

Adottare questa pre-configurazione di tutte le tecnologie, mi ha portato a definire delle caratteristiche che lo scheletro di una applicazione deve avere, indipendentemente dalle tecnologie e i linguaggi utilizzati:

Dipendenze Lo scheletro di una applicazione deve specificare tutte le dipendenze con software o pacchetti esterni. Per semplificare la loro gestione è possibile utilizzare dei package manager come Bower (3.3.1) o Composer

Modulare Se un certo numero di applicazioni richiedono una nuova funzionalità che non è ancora inclusa nello scheletro, ci deve essere la possibilità di poterla aggiungere per tutte le applicazioni future. Al contrario non tutti i nuovi moduli devono per forza essere inclusi nelle implementazioni successive.

Strutturato Deve avere innanzitutto una struttura ben accurata a livello di directory, in modo tale che i file possano essere separati nella maniera che lo sviluppatore ritiene più opportuna.

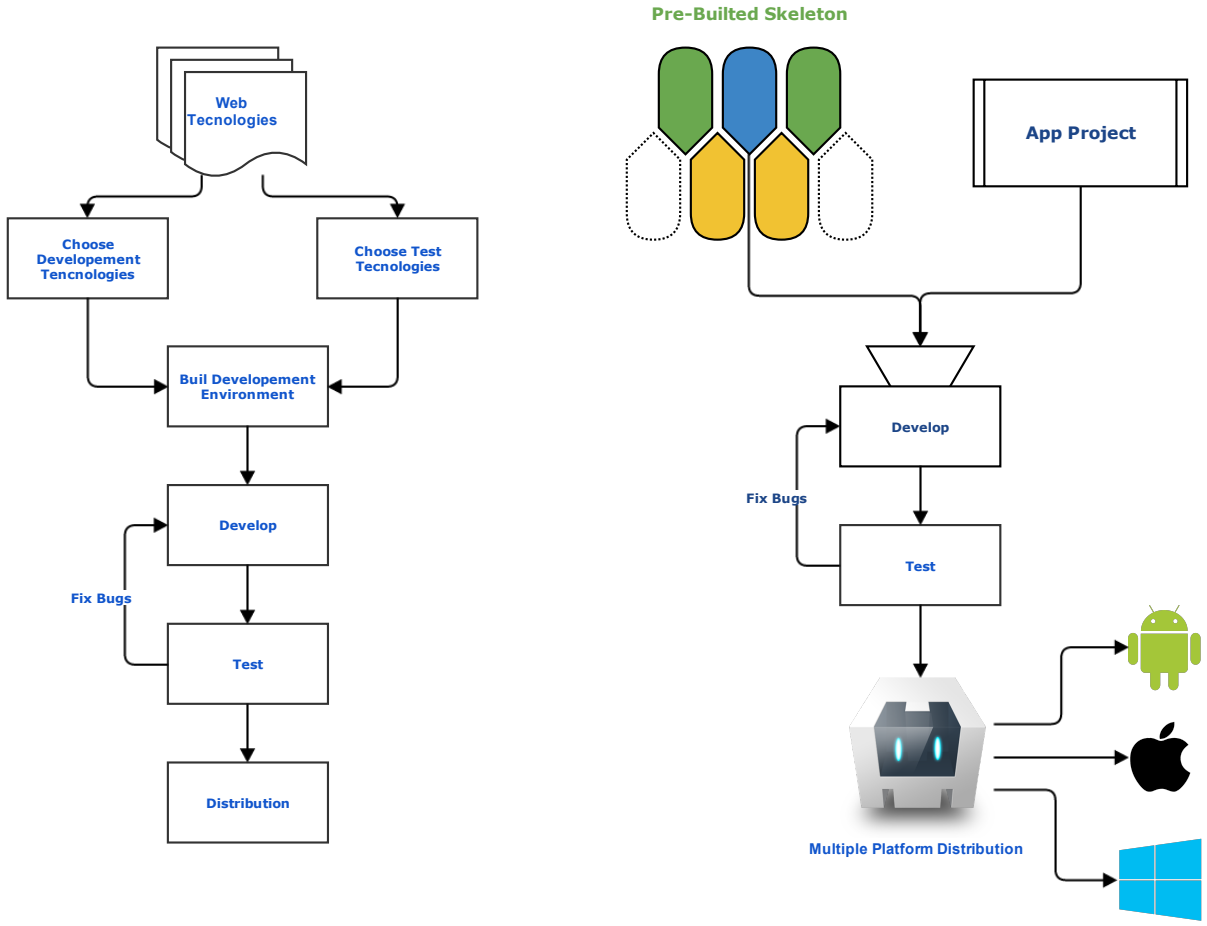


FIGURA 4.2: Questo schema illustra l'evoluzione del primo schema verso una struttura più completa e multipiattaforma

Ad esempio in AngularJS e possibile separare il codice in diversi moduli. Ci sono diverse linee di pensiero che sostengono che la separazione del codice debba avvenire in maniera semantica per il ruolo che quel modulo ricopre (ad esempio un modulo potrebbe essere la gestione dell'autenticazione con le varie direttive e controller). Altri sostengono che i moduli debbano essere divisi in base al tipo delle struttura utilizzata (sempre nel caso di AngularJS nei vari moduli vengono raggruppati per controller, direttive, service . . .).

Documentato Documentare come funziona il proprio scheletro di applicazione e tutte le sue caratteristiche, da modo ai collaboratori futuri di apprendere nel minor tempo possibile il suo funzionamento. Esistono delle tecnologie che, date le giuste annotazioni nel codice, creano già una documentazione chiara e dettagliata.

Automazione E utile avere all'interno dello scheletro di una applicazione un meccanismo di automazione, tramite dei comandi (ad esempio usando shell-scripting), per operazioni come: set-up dell'ambiente di sviluppo, aggiornamento dei pacchetti, operazioni del task runner e risoluzione delle dipendenze.

4.4 Nuove tecnologie web emergenti

In conclusione alla mia tesi voglio mostrare quali sono alcune delle tecnologie emergenti in ambito web per lo sviluppo di applicazioni mobile e non solo. Si tratta solo di una breve presentazione, se si vuole approfondire i vari riferimenti sono disponibili dei link nella bibliografia.

4.4.1 FirefoxOS

FIGURA 4.3:
FirefoxOS logo

FirefoxOS è un nuovo sistema operativo per dispositivi mobili sviluppato da *Mozilla*. Firefox OS ha eliminato lo strato di API native fra il sistema operativo e gli strati applicativi. Questa soluzione integrata riduce il carico della piattaforma e semplifica la gestione della sicurezza senza sacrificare le prestazioni e una ricca esperienza utente (fig.4.4). Voglio mostrare questa nuova tecnologia con particolare riferimento a come sono gestite le API del dispositivo. In particolare si può paragonare il funzionamento di questo sistema operativo al meccanismo di wrapping, utilizzato da alcuni framework per la distribuzione su più piattaforme della stessa applicazione ibrida. In questo caso è tutto implementato nativamente, e non si ha bisogno di un wrapper per poter comunicare con le funzionalità del dispositivo, in quanto si ha il controllo totale del tramite delle API Javascript. L'architettura di FirefoxOS mostrata nella figura 4.4 funziona in questo modo:

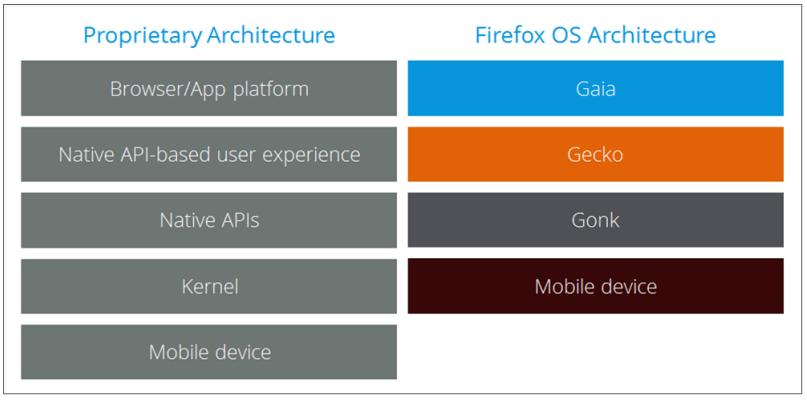


FIGURA 4.4: L'architettura di FirefoxOS messa a confronto con quella delle piattaforme proprietarie

Gaia è l'insieme delle web app principali e dell'interfaccia utente di Firefox OS. È' scritto in HTML5, CSS e JavaScript. Espone un insieme di API per consentire al codice della UI di interagire con l'hardware del telefono e con le funzionalità di Gecko.

Gecko è il motore web e lo strato di presentazione di Firefox OS. Rappresenta l'interfaccia fra i contenuti web e il dispositivo. Gecko fornisce il motore di parsing e rendering HTML5, un insieme di Web API sicure per accedere alle funzionalità hardware, un framework per la gestione della sicurezza, un sistema per la gestione degli aggiornamenti e altri servizi core.

Gonk è il componente al livello del kernel nello stack di Firefox OS, è l'interfaccia fra Gecko e l'hardware del dispositivo. Gonk gestisce l'hardware sottostante e espone le funzionalità dell'hardware alle Web API implementate in Gecko. Gonk può essere visto come la "black box" che esegue il lavoro complesso e dettagliato dietro le scene, controllando il dispositivo mobile gestendo le richieste al livello hardware. Il dispositivo mobile è il telefono su cui viene eseguito Firefox OS. L'OEM è responsabile per la fornitura del dispositivo mobile.

fonte: Mozilla Developer Network[25]

4.4.2 NodeWebkit

Appendice A

Appendice Argomenti

A.1 HTML5

A.2 CSS

A.3 Javascript

A.4 Dependency Injection

A.5 Inversion Of Control

A.6 Future / Promises

Bibliografia

[1] Audiweb. Audiweb pubblica i dati della mobile e total digital audience del mese di luglio 2014, 2014. [Online; da data 10-ottobre-2014].

[2] Wikipedia. Framework — wikipedia, l'enciclopedia libera, 2014. [Online; in data 21-novembre-2014].

[3] Vision Mobile. Cross platform develope tools, 2012. Bridging the worlds of mobile apps and the web.

[4] P.A. Laplante. *What Every Engineer Should Know about Software Engineering*. What Every Engineer Should Know. CRC Press, 2007.

[5] R.J. Mitchell and Institution of Electrical Engineers. *Managing Complexity in Software Engineering*. IEE computing series. P. Peregrinus, 1990.

[6] Wikipedia. Separation of concerns — wikipedia, the free encyclopedia, 2014. [Online; accessed 30-November-2014].

[7] Wikipedia. Design pattern — wikipedia, l'enciclopedia libera, 2014. [Online; in data 6-gennaio-2015].

[8] Wikipedia. Client–server model — wikipedia, the free encyclopedia, 2014. [Online; accessed 30-November-2014].

[9] Wikipedia. Model-view-controller — wikipedia, l'enciclopedia libera, 2014. [Online; in data 6-gennaio-2015].

[10] Addy Osmani. *Learning JavaScript Design Patterns*, volume 1.6.1. O'Reilly Media, 2014.

[11] Wikipedia. Application programming interface — wikipedia, the free encyclopedia, 2015. [Online; accessed 7-January-2015].

[12] Andrea Chiarelli. Transizioni di stato, hypermedia e il principio hateoas — restful web services - la guida —programmazione html.it. http://www.html.it/pag/19606/transizioni-di-stato-hypermedia-e-il-principio-hateoas/.

[13] Roy Thomas Fielding. Rest apis must be hypertext-driven, 2008.

[14] Monsur Hossain. enable cross-origin resource sharing. http://enable-cors.org/, 2015.

[15] Wikipedia. Preprocessore — wikipedia, l'enciclopedia libera, 2014. Online in data 9-gennaio-2015.

[16] Wikipedia. Solid — wikipedia, l'enciclopedia libera, 2014. [Online; in data 4-febbraio-2015].

[17] Angularjs: Developer guide: Directives. https://docs.angularjs.org/guide/directive. AngularJS Directives.

[18] Wikipedia. Single source of truth — wikipedia, the free encyclopedia, 2014. [Online; accessed 10-January-2015].

[19] Wikipedia. Blink (layout engine) — wikipedia, the free encyclopedia, 2014. [Online; accessed 24-November-2014].

[20] Wikipedia. Software development kit — wikipedia, the free encyclopedia, 2014. [Online; accessed 26-November-2014].

[21] Promise - javascript — mdn, may 2015.

[22] Phonegap api documentation, may 2015. How device events are managed in phonegap.

[23] angular-ui angular ui. Home angular-ui/ui-router wiki. https://github.com/angular-ui/ui-router/wiki, may 2015.

[24] Dick Hardt. The oauth 2.0 authorization framework, 10 2012. RFC 6749 and RFC 6750.

[25] Firefox os architecture, dic 2014.