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

Gennaio 2015

“textfit“

-

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.
Audiweb [2014]

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 Molti nuovi dispositivi utilizzano un browser per fornire un aspetto tradizionale alle app mobile. Ciò consente di supportare molti dispositivi ma spesso non offre l'accesso alle funzioni dei dispositivi mobile, come la fotocamera o l'elenco dei contatti.

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

Grazie a tutti ...

Indice dei Contenuti

Elenco delle Figure

Elenco delle Tabelle

Elenco degli Esempi

Chapter 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 multi piattaforma si introdurranno una serie di tecnologie che tutt'oggi possono essere utilizzate per la creazione di applicazioni ibride.

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

Table 1.1: Lista 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 anche 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.

Il passo che ha portato quello che prima era un Sito Internet poi una Applicazione Web 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 multi piattaforma 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).

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 smandrupparsi il tuo telefono) il che avvicina molto l'approccio ibrido a quello

[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).

```
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. }
```

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

nativo, riguardo l'accesso alle funzionalità del dispositivo. Per fare questo 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 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 multi piattaforma, 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 molto rapidamente, 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.

Wikipedia [2014a]

L'uso di framework da parte di aziende e sviluppatori e 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 ne si 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. Waldner [2013]

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.

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.

Runtime è un approccio che, durante l'esecuzione di un applicazione, interpreta il codice sorgente da una virtual machine che è diversa a seconda del sistema operativo. Inoltre protegge l'applicazione dalle differenze tra le piattaforme sottostanti. Runtime varia in dimensione e complessità ed esegue codice sul dispositivo usando differenti metodi: la virtualizzazione, l'interpretazione, la compilazione just-in-time oppure la compilazione ahead-of-time. Java ME, BREW, Flash Lite e Openwave MIDAS sono stati i primi a sperimentare questo approccio sui dispositivi mobile. Queste virtual machine sono pesanti e sembrano una mezza via tra browser e sistema operativo. Oggigiorno questo tipo di approccio sposta la complessità delle operazioni dal software del dispositivo al design-time ovvero il momento di traduzione del codice in bytecode. Esempi di runtimes sono Appcelerator, Adobe Flex (e AIR), Corona, AppMobi, Antix e Unity.

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.

Ecco una lista dei framework più conosciuti attualmente sul mercato:
NOTA TESI: se si sceglie di inserire la tabella, differenziare i tipi di framework, e inserire

l'url sotto forma di link nel nome del framework.

NOME	SITO WEB
Appcelerator	http://www.appcelerator.com/
Corona	http://coronalabs.com
Kony	http://www.kony.com
PhoneGap	http://phonegap.com
Enyo	http://enyojs.com
eMobc	http://www.emobc.com
iPFaces	http://www.ipfaces.org
Sencha	http://www.sencha.com
Steroids	http://www.appgyver.com/steroids
Trigger.io	https://trigger.io
Intel App Framework	http://app-framework-software.intel.com
Moai	http://getmoai.com
Scirra / Construct 2	https://www.scirra.com
GameMaker Studio	https://www.yoyogames.com/studio
Basic4android	http://www.basic4ppc.com
Kivy	http://kivy.org
Syncfusion	http://www.syncfusion.com/solutions/mobility
Xamarin	https://xamarin.com
Ionic	http://ionicframework.com/
MoSync	http://www.mosync.com/
GameSalad	http://gamesalad.com/
Stencyl	http://www.stencyl.com/
RAD Studio	http://www.embarcadero.com/products/rad-studio
Telerik Mobile Application Development Platform	http://www.telerik.com/platform
appery.io	http://appery.io
RoboVM	http://www.robovm.org
Marmalade	https://www.madewithmarmalade.com
CocoonJS	https://www.ludei.com/cocoonjs
Kendo UI Mobile	http://www.telerik.com/kendo-ui-mobile
Microsoft XNA	http://www.microsoft.com/en-us/download/details.aspx?id=20914
Unreal Engine	http://www.unrealengine.com
Ejecta	http://www.impactjs/comejecta

TABLE 1.2: Lista dei principali framework più conosciuti sul mercato

1.6 Conclusioni

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 multi piattaforma 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 multi piattaforma, 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 multi piattaforma, 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 multi piattaforma 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 multi piattaforma senza l'utilizzo di framework?

Si è inteso lo sviluppo multi piattaforma 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 multi piattaforma 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.

Chapter 2

Metodi e Modelli per lo sviluppo di applicazioni Mobile

In questo capitolo si vogliono presentare alcuni concetti teorici e pratici per lo sviluppo di applicazioni Mobile. Molti dei concetti coincidono con lo sviluppo di applicazioni in generale e sono riportati dall'esperienza di tirocinio fatta presso l'azienda BigThink SRL.

Una prima fase è 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), che in italiano si traduce in *Separazione dei Compiti*. Si tratta di un principio di design per dividere un problema in sezioni distinte, e ad ogni sezione assegnare un particolare compito o risoluzione del problema. Questa scelta progettuale introduce il concetto di *modulo* di una applicazione, ovvero l'applicazione poi dipenderà dai moduli che andremo a creare e ad unire tra di loro. I moduli ragionevolmente saranno indipendenti uno dall'altro in modo tale da garantirne l'integrità e il loro re-utilizzo.Wikipedia [2014b]

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. Wikipedia [2014c]

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

provenienti dalla parte frontend. 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 nessuna manipolazione dei dati ma solo la loro rappresentazione(vista) determinando transizioni e/o animazioni. 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 ma non da nessuna informazione caratteristica di essa. In particolare fornisce dei servizi ai quali la parte *frontend* può accedere e recuperare/fornire dei dati, 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*, in modo errato, nessuna operazione verrà eseguita e l'integrità dei dati 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).

A livello professionale molti sviluppatori si identificano appunto come *frontend* e/o *backend* developer ovvero specializzati nello sviluppo di una parte specifica dell'applicazione. Il vantaggio di usare un approccio di questo tipo sta nel fatto che la parte *frontend* è l'unica specifica per una data applicazione. Se la parte *backend* viene progettata bene, e possibile utilizzare i servizi che fornisce anche in futuro da altre applicazioni, pur seguendo il protocollo che richiede. 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

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 serverWikipedia [2014d].

Come si può già intuire le parti che verranno prese in questo modello saranno rispettivamente quella del *frontend* e quella del *backend*. Specificatamente nell'ambito delle applicazioni i client saranno le varie istanze della parte *frontend* dell'applicazione sui vari dispositivi, mentre il server, fornitore di servizi, conterrà la parte di *backend*.

2.1.3 Mustache

In ambito web può essere utile un sistema di template che aiuti a separare la logica dalla presentazione dei dati(metodologia molto in voga tra le tecnologie web).

Mustache è un web template system con una implementazione disponibile in molti linguaggi(sarà utili quella in Javascript). Mustache è descritto come un sistema senza logica, in quanto manca qualsiasi dichiarazione di controllo del flusso esplicita, ovvero istruzioni condizionali e di iterazione. Il tutto può essere implementato tramite un sistema di tag, liste procedurali e lambda Wikipedia [2014e]

2.1.4 MVC Pattern

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.

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. Wikipedia [2014f]

2.1.5 MVVC Pattern

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.

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).

Osmani [2014]

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. Wikipedia [2015]

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 Fieldin(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 sua 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, modifica, aggiornamento, cancellazione.

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) Clients deliver state via body contents, query-string parameters, request headers and the requested URI (the resource name). Services deliver state to clients via body content, response codes, and response headers. This is technically referred-to as hypermedia (or hyperlinks within hypertext). Aside from the description above, HATEOS also means that, where necessary, links are contained in the returned body (or headers) to supply the URI for retrieval of the object itself or related objects. We'll talk about this in more detail later. The uniform interface that any REST services must provide is fundamental to its design.

Stateless As REST is an acronym for REpresentational State Transfer, statelessness is key. Essentially, what this means is that the necessary state to handle the request is contained within the request itself, whether as part of the URI, query-string parameters, body, or headers. The URI uniquely identifies the resource and the body contains the state (or state change) of that resource. Then after the server

does it's processing, the appropriate state, or the piece(s) of state that matter, are communicated back to the client via headers, status and response body.

Most of us who have been in the industry for a while are accustomed to programming within a container which provides us with the concept of "session" which maintains state across multiple HTTP requests. In REST, the client must include all information for the server to fulfill the request, resending state as necessary if that state must span multiple requests. Statelessness enables greater scalability since the server does not have to maintain, update or communicate that session state. Additionally, load balancers don't have to worry about session affinity for stateless systems.

So what's the difference between state and a resource? State, or application state, is that which the server cares about to fulfill a request—data necessary for the current session or request. A resource, or resource state, is the data that defines the resource representation—the data stored in the database, for instance. Consider application state to be data that could vary by client, and per request. Resource state, on the other hand, is constant across every client who requests it.

Ever had back-button issues with a web application where it went AWOL at a certain point because it expected you to do things in a certain order? That's because it violated the statelessness principle. There are cases that don't honor the statelessness principle, such as three-legged OAuth, API call rate limiting, etc. However, make every effort to ensure that application state doesn't span multiple requests of your service(s).

Cacheadable As on the World Wide Web, clients can cache responses. Responses must therefore, implicitly or explicitly, define themselves as cacheable, or not, to prevent clients reusing stale or inappropriate data in response to further requests. Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance.

Client-Server The uniform interface separates clients from servers. This separation of concerns means that, for example, clients are not concerned with data storage, which remains internal to each server, so that the portability of client code is improved. Servers are not concerned with the user interface or user state, so that servers can be simpler and more scalable. Servers and clients may also be replaced and developed independently, as long as the interface is not altered.

Layered system A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediary servers may improve system scalability by enabling load-balancing and by providing shared caches. Layers may also enforce security policies.

Code on demand(OPTIONAL) Servers are able to temporarily extend or customize the functionality of a client by transferring logic to it that it can execute. Examples of this may include compiled components such as Java applets and client-side scripts such as JavaScript.

Complying with these constraints, and thus conforming to the REST architectural style, will enable any kind of distributed hypermedia system to have desirable emergent properties, such as performance, scalability, simplicity, modifiability, visibility, portability and reliability.

NOTE: The only optional constraint of REST architecture is code on demand. If a service violates any other constraint, it cannot strictly be referred to as RESTful.

2.2.1.1 Implementazioni Esistenti

2.2.1.2 CORS

Chapter 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. Durante la permanenza in azienda 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. Data 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 mobili, in modo tale che l'azienda avesse nuovi servizi da offrire.
Tutte le applicazioni sviluppate fino ad allora erano già Responsive Web Application, in azienda si volevano 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 a quel punto. 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. Inoltre esiste un'ultima libreria che richiama le funzionalità di Cordova utilizzando il framwork AngularJS, il che viene molto comodo per il set di tecnologie scelto.

Il framework Ionic per la creazione dell'interfaccia utente è composto oltre da AngularJS da altre tecnologie come un *CSS Preprocessor* che verrà spiegato prima di introdurre nel dettaglio le tencnologie introdotte precedentemente.

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. Wikipedia [2014g]

FIGURE 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. E di conseguenza la personalizzazione da parte dello sviluppatore che andrà a utilizzare il framework diventa molto difficile. 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

FIGURE 3.2:
AngularJS framework logo

Definizione AngularJS è un framework Javascript che segue il pattern MVC(2.1.4) 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* della programmazione orientata agli oggetti. Ecco le caratteristiche principali di questo framework:

View la presentazione dei dati in AngularJS viene fatta tramite un sistema di viste con l'utilizzo di pagine HTML. Ad ogni vista possono essere associati più modelli di dati da presentare ed inoltre tramite un meccanismo di *routing* e possibile determinare una struttura, anche gerarchica di più viste. Un componente

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(Wikipedia [2014h]). 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.

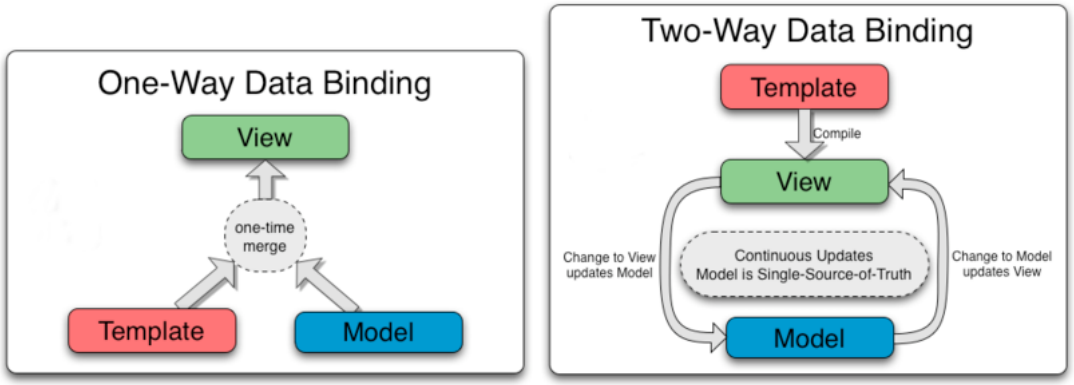

FIGURE 3.3: Come è strutturato un classico data binding, e come invece è fatto in AngularJS

In particolare come si vede nell'esempio 3.4 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 systema dei templates 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 3.1 AngularJS

crea un nuovo oggetto Javascript associandogli un nuovo scope che potrà essere inserito all'interno del controller tramite la dependency injection.

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

LISTING 3.1: 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)

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 3.2 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.

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

LISTING 3.2: 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 (3.3)

```
angular.module("CustomModule",[])
.service("CustomService",function(){
        this.customMethod = function(value){
                return value + " from customMethod in CustomService";
        }
});

angular.module("testApp",[
        'CustomModule'
]).controller("HomeController", function(CustomService){
        CustomService.customMethod("Marco");
});
```

LISTING 3.3: 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 3.4). Inoltre soltanto quando il service dipende dall'applicazione viene creata l'istanza dell'oggetto(*Lazy Initialization*).

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

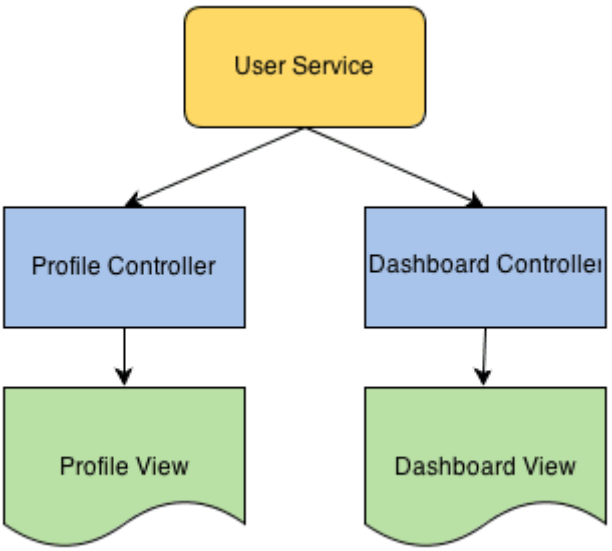

FIGURE 3.4: Schema di come è strutturato un service all'interno di AngularJS

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.

Directives una delle caratteristiche più rilevanti di AnagularJS è sicuramente quella di poter creare dei tag HTML personalizzati, oltre a quelli già previsti dal framework. Questi tag speciali sono chiamati directives o direttive. AngularJS si serve delle direttive per poter strutturare l'applicazione sulla vista HTML, ovvero ci sono dei tag speciali previsti dal framework che definiscono il nome dell'applicazione(*ng-app*), i controller da utilizzare(*ng-controller*), i modelli di riferimento(*ng-model*) e il contenitore dove processare le viste(*ng-view*) come nell'esempio 3.4:

```
<html ng-app = "myApp">
  <head></head>
  <body>
    <div ng-controller = "InfoController">
      <input name = "inputFirstName" ng-model = "user.firstName">
      <input name = "inputSecondName" ng-model = "user.secondName">
      <div class = "showInfo">
            Hello my name is {{user.firstName}} {{user.secondName}}
      </div>
    </div>
  </body>
</html>
```

LISTING 3.4: Un esempio delle direttive standard di AngularJS

Inoltre possiamo definire direttive personalizzate alle quali possiamo associare un relativo comportamento(3.5). Una cosa che è concessa nelle direttive e non nelle altre strutture di AngularJS è la manipolazione del DOM, solo al loro interno è possibile scrivere codice che interagisca direttamente con gli elementi della vista.

```
angular.module("myApp")
.directive("myPersonalTag",function(){
        return{
                link: function(scope, element, attrs){
                        //directive main istructions
                }
        }
});
```

LISTING 3.5: Un esempio di direttiva personalizzata

Una nota di sintassi particolare per le direttive riguarda il loro nome, come osserviamo il nome delle direttive nell'esempio 3.4 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. Oltre alle caratteristiche principali con le direttive si hanno a disposizione molte più opzioni. Si rimanda l'approfondimento qui : ang per un trattazione più approfondita.

3.1.3 Ionic

FIGURE 3.5: Ionic Framework Logo

IONIC fa parte della famiglia degli UI Framework per creazione di interfacce utente per dispositivi mobili e applicazioni ibride e si serve delle tecnologie web AngularJS, Sass e HTML5. Il lavoro da svolgere per chi vuole utilizzare questo tipo di framework e quello di scegliere i vari componenti messi a disposizione e di comporli per creare l'applicazione finale. Generalmente si parte da un template messo a disposizione dal framework e si compone l'applicazione. La modularità dei componenti di IONIC è data dal sistema di direttive di AngularJS, tutti i componenti sono sviluppati in direttive separate e come si può notare tutti i rispettivi tag HTML iniziano con il prefisso *ion* o *ionic*. La caratterizzazione di un framework di questo tipo dipende dal suo stile e design e sta allo sviluppatore(o grafico) decidere quale in questo caso il più bello per l'applicazione.

IONIC inoltre mette a disposizioni diverse caratteristiche che possono influenzare la sua scelta:

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 ibirde si necessita di un framework che possa in qualche modo tradurre il linguaggio originale in quello della specifica piattaforma. Nel caso qui studiato 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

FIGURE 3.6:
Cordova Framework Logo

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.

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`

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 *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

[1] Blink è la web browser engine di Google ChromeWikipedia [2014i]

che invece si avrebbero. *Corodova* 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.

3.2.2 ngCordova

FIGURE 3.7:
ngCordova Logo

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(geniale) 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*.
Per chi vuole sviluppare applicazioni ibride multi piattaforma, questa libreria rende decisamente più rapido ed efficiente il processo di sviluppo.

3.3 Strumenti di sviluppo

3.3.1 Il trio Magico

La mia scelta delle tecnologie da utilizzare per lo sviluppo ibrido di applicazioni multi piattaforma si è incentrata soprattutto sull'efficienza e la compatibilità. Decidendo di utilizzare `AngularJS` + `Cordova` + `ngCordova` ottengo un ambiente di sviluppo rapido ed efficiente, in quanto tutti e tre i framework sono stati sviluppati per una stretta collaborazione, e allo stato dell'arte attuale delle tecnologie penso che sia una delle scelte migliori e rapide di sviluppo.

3.3.2 Package Manager

Bower è un package manager, ci consente di organizzare al meglio le librerie all'interno del nostro progetto. Tramite il file *bower.json* possiamo specificare tutti i dettagli del nostro 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*. In ambito web non esiste libreria che non possa essere scaricata con bower, sostanzialmente

possiamo ottenere qualsiasi cosa presente su *Github*. – Codice Bower.json – – Codice .bowerrc –

3.3.3 Task Runner

Grunt/Gulp `Grunt` e `Gulp` sono due esempi di *task-tunner*. 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 dal framework Ionic per la gestione di tutti i suoi file. Attualmente esistono due schiere di programmatori che sostengono che uno sia meglio dell'altro, personalmente preferisco grunt perché ha un logo più bello(hahaha).
Il ruolo dei task-runner non è ben preciso in quanto sono molto versatili e "assemblabili", in quanto per eseguire determinate operazioni necessitano dei loro plugin specifici. Ad esempio se nel mio progetto avessi bisogno di verificare la sintassi di Javascript, concatenare i file di progetto e le librerie, minificarli e spostarli in una cartella differente (ho potuto osservare durante il mio tirocinio che queste operazioni sono molto comode nel tipo di approccio che si ha nello sviluppo di una applicazione ibrida con tecnologie web) devo prima scaricare nel mio progetto ognuno di essi. Questo procedimento può sembrare poco efficiente, in realtà se disponiamo del *Node Package Manager* NPM tutta la lista dei plugin installati nel progetto e registrata su un file *package.json*. Inoltre una funzionalità molto significativa che hanno i task-runner è quella del `Live Reload`, ovvero sono in grado di vedere se ci sono stati cambiamenti nel nostro progetto e automaticamente eseguire le operazioni stabilite.

3.3.4 IDE

Ho scelto di usare una *Integrated Develop Enviroment*(IDE) per lo sviluppo della mia applicazione in modo tale da tenere sotto controllo tutti i file del mio progetto e per poterli organizzare in una struttura di tipo modulare. Abbiamo constatato che prima di un deploy dell'applicazione ci sono una serie di operazioni che dobbiamo eseguire affinché il nostro progetto sia completo. Per fare questo ci serviamo di strumenti chiamati *Tesk-Runner*.

- Gestione dei progetti -
- Organizzazione -
- Test -
- Deploy -

Questi concetti spiegati in modo generale per poi dire nelle varie sezione come sono stati effettivamente applicati.

Tra i diversi IDE disponibili ho scelto di usare *NetBeans* in quanto può predisporre di un ambiente adatto allo sviluppo di applicazioni multi-piattaforma. In particolare è possibile partire da un modello di progetto basato su *Cordova* oppure semplicemente in *HTML5* ed inoltre è molto semplice configurare le SDK dedicate per il deploy sui vari sistemi operativi.

3.3.4.1 Creazione

Tramite NetBeans inizializziamo un nuovo progetto basato si Cordova. – Immagine nuovo progetto –
– Immagine scelta del template – In NetBeans non abbiamo il templateAbbiamo constatato che prima di un deploy dell'applicazione ci sono una serie di operazioni che dobbiamo eseguire affinché il nostro progetto sia completo. Per fare questo ci serviamo di strumenti chiamati *Tesk-Runner*. fornito da Ionic, ma possiamo inizializzare tramite terminale il progetto e poi importarlo nell'IDE. Facendo questa operazione dobbiamo avere l'accortezza di aggiornare le impostazioni del progetto all'interno di NetBeans. – Immagine progetto – Qui abbiamo i risultato del nostro progetto dal quale partire per lo sviluppo.

3.3.4.2 Inclusione di Librerie esterne

Nei progetti per applicazioni multi piattaforma che utilizzano tecnologie web può risultare utile includere delle librerie esterne(spesso in linguaggio Javascript), che con questo tipo di tecnologia, questa operazione risulta molto rapida e intuitiva.
Semplicemente come si agisce nei siti web, si inserisce uno script all'interno della pagina *index.html* che va a includere il file della libreria desiderate.

```
<script src = "lib/mylibrary/dist/mylibrary.min.js"></script>
```

Con NetBeans possiamo utilizzare in fase di configurazione e anche successivamente uno strumento che automaticamente include librerie da lui elencate, il quale è molto utile ma ovviamente non dispone di tutta la varietà presente in rete. uno strumento molto potente, usato dalla stragrande maggioranza degli sviluppatori web è `Bower`.

3.3.4.3 Deploy

L'operazione di *Deploy* in italiano "schierare", nel nostro caso prende il significato di produrre una versione della nostra applicazione, che può essere finale oppure no. Nello

sviluppo di applicazioni ibride multi piattaforma abbiamo a disposizione 3 opzioni per fare questa operazione.

Delploy sul Browser Ovvero andiamo a vedere quella parte di applicazione che non necessita di essere sul dispositivo per poter funzionare. Sostanzialmente visualizziamo solo il livello delle tecnologie web, e lo possiamo correggere e/o visionare come se fosse un normale sito web. In questa modalità però non disponiamo ad esempio dell'interazione con le funzionalità native del dispositivo, si tratta di un modo rapido per avere una vista su quella che sarà l'interfaccia dell'applicazione.

Simulazione Ogni IDE consente un meccanismo di simulazione di un dispositivo sulla propria macchina. In questa modalità possiamo vedere realmente come sarà la nostra applicazione su un dispositivo, ma in realtà l'hardware di riferimento sarà quello della nostra macchina(il simulatore si occupa di mettere in comunicazione tutte le componenti). E' una modalità che si avvicina molto alla rappresentazione reale della nostra applicazione ma il processo di simulazione per un computer di fascia media comporta una attesa anche di minuti per visionare il risultato, e dato che dovremo ripetere questa operazione diverse volte perché può darsi che si debba correggere il codice(cosa molto frequente), diventa molto dispendioso attendere tutte le volte così tanto tempo.

Installazione su Dispositivo Quasi tutti i sistemi operativi ad eccezione di iOS, concedono di installare applicazioni direttamente da PC. Nel caso di Android che ho preso in esame, è molto semplice fare un *Deploy* su dispositivo. Tramite *NetBeans* bisogna configurare le SDK e scegliere il proprio dispositivo connesso tramite usb. Nel caso di iOS *Apple* non concede l'installazione di applicazioni da origini sconosciute, a meno che non siano registrate sul Apple Store, pagando una tassa annuale di 99$.

Quando il nostro progetto prende forma i file all'interno di esso cominciano a diventare consistenti e organizzarli diventa sempre più complesso. E buona norma di programmazione separare il proprio progetto in più file, ma al momento della consegna bisogna anche ricomporli e assemblarli nella maniera corretta. Inoltre se usiamo *Ionic* e abbiamo modificato i fogli di stile dobbiamo ricordarci di ri-compilare i file *SASS*.

3.3.4.4 Live Debug

Una volta che l'applicazione è in esecuzione in una delle modalità descritte precedentemente ci si preoccupa di controllare che il codice che si ha scritto funzioni correttamente.

Per eseguire un buon *Debug* di questo tipo di applicazione è sufficiente disporre di uno strumento molto semplice *Google Chrome*, infatti tramite questo web browser a diffenza di altri possiamo osservare il comportamento della nostra applicazione sui vari dispositivi(anche in fase di simulazione) come se stessimo facendo una normale ispezione di una pagine web. Ovviamente se l'applicazione è in *Deploy sul Browser* il problema non si pone, qualsiasi browser può andare(anche se personalmente consiglio Google Chrome o Mozilla Firefox).

3.4 SDK

Un `Software Developement Kit` in generale è un insieme di strumenti per lo sviluppo e la documentazione del softwareWikipedia [2014j]. 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. *NetBeans* dispone di una interfaccia molto semplice da usare per impostare i percorsi delle *SDK* ma non di tutte le piattaforme; se volgiamo impostare un'altra piattaforma non indicata nell'IDE dobbiamo seguire le istruzioni fornite da *Cordova*.

Chapter 4

Conclusioni

4.1 Ciclo di vita di una app

– Statistiche sulle app
– Stessa App replicata nei store –
– Durata delle app –
– Aggiornamenti –
– più piattaforme = più aggiornamenti –
– Mantenimento –
– Aggiornamento delle versioni –
– Debug –
– Tutto ovviamente in confronto con lo sviluppo ibrido –

4.2 Verso il futuro

– Material Design implementato già nativamente in angularjs –

Appendix A

Appendice Argomenti

A.1 HTML5

A.2 CSS

A.3 Javascript

A.4 Dependency Injection

A.5 Inversion Of Control

Bibliografia

Audiweb. Audiweb pubblica i dati della mobile e total digital audience del mese di luglio 2014, 2014. URL http://www.audiweb.it/news/comunicato-stampa-pc-smartphone-e-tablet-audiweb-misura-la-total-digital-audience-3 [Online; da data 10-ottobre-2014].

Wikipedia. Framework — wikipedia, l'enciclopedia libera, 2014a. URL http://it.wikipedia.org/w/index.php?title=Framework&oldid=67469278. [Online; in data 21-novembre-2014].

Lorenzo Waldner. Studio e valutazione di framework per lo sviluppo cross-platform di applicazioni mobile. Master's thesis, University of Trento, jul 2013.

Wikipedia. Separation of concerns — wikipedia, the free encyclopedia, 2014b. URL http://en.wikipedia.org/w/index.php?title=Separation_of_concerns&oldid=627731535. [Online; accessed 30-November-2014].

Wikipedia. Design pattern — wikipedia, l'enciclopedia libera, 2014c. URL http://it.wikipedia.org/w/index.php?title=Design_pattern&oldid=68132953. [Online; in data 6-gennaio-2015].

Wikipedia. Client–server model — wikipedia, the free encyclopedia, 2014d. URL http://en.wikipedia.org/w/index.php?title=Client%E2%80%93server_model&oldid=628622834. [Online; accessed 30-November-2014].

Wikipedia. Mustache (template system) — wikipedia, the free encyclopedia, 2014e. URL http://en.wikipedia.org/w/index.php?title=Mustache_(template_system)&oldid=640214169. [Online; accessed 6-January-2015].

Wikipedia. Model-view-controller — wikipedia, l'enciclopedia libera, 2014f. URL http://it.wikipedia.org/w/index.php?title=Model-View-Controller&oldid=69808152. [Online; in data 6-gennaio-2015].

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

Wikipedia. Application programming interface — wikipedia, the free encyclopedia, 2015. URL http://en.wikipedia.org/w/index.php?title=Application_programming_interface&oldid=640772216. [Online; accessed 7-January-2015].

Wikipedia. Preprocessore — wikipedia, l'enciclopedia libera, 2014g. URL http://it.wikipedia.org/w/index.php?title=Preprocessore&oldid=67684369. [Online; in data 9-gennaio-2015].

Wikipedia. Single source of truth — wikipedia, the free encyclopedia, 2014h. URL http://en.wikipedia.org/w/index.php?title=Single_Source_of_Truth&oldid=631806243. [Online; accessed 10-January-2015].

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

Wikipedia. Blink (layout engine) — wikipedia, the free encyclopedia, 2014i. URL http://en.wikipedia.org/w/index.php?title=Blink_(layout_engine)&oldid=630704373. [Online; accessed 24-November-2014].

Wikipedia. Software development kit — wikipedia, the free encyclopedia, 2014j. URL http://en.wikipedia.org/w/index.php?title=Software_development_kit&oldid=630262856. [Online; accessed 26-November-2014].