Università degli Studi di Milano

Tesi Di Laurea Triennale

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

Author:
Marco Predari

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

Dipartimento di Informatica e Comunicazione

Novembre 2014

“textfit“

-

UNIVERSITA' DEGLI STUDI DI MILANO

Abstract

Facoltà di Scienze e Tecnologie
Dipartimento di Informatica e Comunicazione

Dottore in Informatica Musicale

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

by Marco Predari

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 per 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.
?

Il nostro compito, in quanto sviluppatori, è quello di realizzare applicazioni non solo funzionanti nel senso stretto del termine ma funzionali e usabili su questa tipologia di dispositivi. Inoltre se vogliamo far conoscere il nostro prodotto ad un numero maggiore di utenti, dobbiamo anche preoccuparci che la nostra applicazione sia disponibile su diverse piattaforme, perché come ben sappiamo il mercato dei dispositivi mobili ha diversi sistemi operativi su cui possiamo sviluppare, ed ognuno di essi usa modelli e linguaggi di programmazione differenti.

Esistono diversi modi per sviluppare una applicazione mobile, Nativa, Web o ibrida: Non esiste la risposta perfetta. Tutte hanno 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. 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 intrapresa dipende dalle conoscenze iniziali dello sviluppatore.

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.

Acknowledgements

Grazie a tutti ...

Contents

List of Figures

List of Tables

For/Dedicated to/To my...

Chapter 1

Stato dell'arte

In questo capitolo si presentano le tecnologie attualmente presenti sul mercato per lo sviluppo di applicazioni mobili, con particolare attenzione per le strategie di sviluppo multi piattaforma.

1.1 Sviluppo Nativo

Nello sviluppo nativo ogni applicazione viene sviluppata singolarmente per ogni piattaforma; rispettivamente per ogni sistema operativo bisogna conoscere i linguaggi che vengono utilizzati.

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

Il punto di forza che ha questo tipo di approccio sta sicuramente nelle prestazioni dell'applicazione e nel completo accesso a tutte le possibilità del nostro dispositivo. Tuttavia con questo tipo sviluppo richiede molto tempo nel caso della distribuzione su più piattaforme.

1.2 Multi-Piattaforma

1.2.1 Web Applications

Quando si intraprende la strada dello sviluppo di una applicazione multi-piattaforma l'opzione più semplice è sicuramente quella di creare una Web App responsive[1] che sia accessibile in modo immediato dal browser web del nostro dispositivo. In questo caso i tempi di sviluppo sono veramente esigui, e non abbiamo bisogno di aggiornamenti in quanto si tratta alla fine di un piccolo sito internet(quindi ci basta aggiornare la pagina).

A scapito della velocità di produzione e della facilità di aggiornamento non possiamo permetterci alcune caratteristiche fondamentali per le applicazioni mobile, come ad esempio accedere alle funzionalità del nostro dispositivo come fotocamera o rubrica, oppure memorizzare dati sul nostro dispositivo(o meglio possiamo farlo ma sfruttando solamente la cache del browser web) ma la criticità maggiore sta nel fatto che la connessione internet deve essere sempre presente.

1.2.2 I Framework

L'approccio ibrido che si propone combina la velocità di sviluppo delle Web App con un accesso *quasi nativo* al nostro dispositivo. Un problema che ci so pone in questo tipo di approccio sta nella scelta del *framework* che si adatta meglio allo scopo della nostra applicazione.

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.*

?

Esistono diversi framework sul mercato con caratteristiche diverse tra di loro, ecco quelli più popolari:

[1] indica la capacità di una Web App o di un sito internet di adattare le proprie dimensioni e proporzioni in base alla risoluzione del dispositivo con il quale vengono visualizzati

NOME	NATIVO	WEB APP	IBRIDO	
Appcelerator(Titanium)	Si	No	No	
Ionic	No	No	Si	
PhoneGap(Corodva)	No	No	Si	
...	...	...	...	

Ogni framework adotta strategie diverse per passare dal codice sorgente a quello di destinazione, successivamente verranno discusse alcune di esse.

1.2.2.1 Wrapper

I framework denominati appunto `Wrapper` sfruttano il design pattern strutturale `Adapter` e operano quindi una astrazione sul linguaggio di destinazione. In questo caso il framework ha il compito di intermediario tra il linguaggio usato nella applicazione e quello nativo del dispositivo. Cordova ad esempio simula un browser web sul linguaggio di astrazione(che in questo caso saranno linguaggi in ambito web) e fornisce delle API per comunicare con la parte nativa del linguaggio del dispositivo. La game-engine `Unity` utilizza lo stesso meccanismo si astrazione, ed anziché simulare un browser utilizza un linguaggio proprietario per poter eseguire la applicazione su più piattaforme. Un altro esempio è invece Apache Flex che utilizza come linguaggio di astrazione Adobe Flash e in alcuni casi Javascript.

1.2.2.2 Trasformazione del Codice

Alcuni framework potremmo definirli invece *"trasformatori"* ovvero partendo da un loro linguaggio / metalinguaggio(spesso somigliante ad altri linguaggi conosciuti come ad esempio Java) producono un il codice specifico per ogni sistema operativo scelto. In particolare il linguaggio utilizzato dal framework è una associazione tra il suo linguaggio è quello di destinazione. Una volta creata e riconosciuta l'associazione, il codice viene trasformato.

1.2.2.3 User Interface

Altri tipi di framework invece sono atti alla definizione dell'interfaccia utente della nostra applicazione, ovvero si traducono in un insieme di librerie che utilizzano metalinguaggi e non per *"vestire"* la nostra applicazione a seconda dello scopo. Un esempio di un framework di questo tipo è `Ionic`, il quale fornisce un set di stili e funzioni per la nostra applicazione utilizzando tecnologie web, come ad esempio menù laterali, bottoni,

form, icone. La maggior parte di questi tipi di framework fornisce anche linee guida per creare elementi completamente nuovi, nel caso avessimo bisogno di parti più complesse. Invece `Apache Flex`, oltre a fungere da wrapper, fornisce un set di elementi già pronti e parametrizzati (il linguaggio è simile a XML[2]) per poter personalizzare la nostra applicazione. Non è molto versatile ma è particolarmente adatto alle applicazioni nel campo produttivo/finanziario.

1.3 Vantaggi e Svantaggi

Il vantaggio principale nell'utilizzo di un framework consiste nella possibilità di effettuare un deploy rapido 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.

Un limite 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 device, 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. 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,tra cui per la gestione dei sistemi di pagamento attraverso gli smartphone.

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.

[**?**]

[2](sigla di eXtensible Markup Language) è un linguaggio di markup, ovvero un linguaggio marcatore basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.**?**

Chapter 2

Architettura di una Applicazione Ibrida

In questo capitolo vedremo alcuni concetti utili per lo sviluppo di applicazioni ibride e come sono strutturate.

2.1 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.
Spesso nella parte backend si integra una connessione ad un database per una eventuale memorizzazione di dati, questo approccio ha dei vantaggi fin da subito a lato della sicurezza in quanto l'accesso ai dati non e permesso alla parte frontend la quale può usufruire solo delle funzionalità che la parte backend offre.

2.2 Pattern Client-Server

Il pattern `client-server` nel contesto delle reti si riduce alla descrizione di un processo comunicativo in rete in cui un host[1] richiede una risorsa, appunto client, mentre l'entità che fornisce la risorsa a più client è detta server.

[1] termine che indica un nodo generico nella rete

Questa struttura comunicativa per lo scambio di risorse negli anni è stata adattata a molti contesti informatici, ormai non descrive più in particolare la comunicazione nelle reti ma più in generale lo scambio di risorse tra più infrastrutture.
Nel capitolo precedente abbiamo descritto frontend e backend, quello è un esempio emblematico di una comunicazione client-server non specificatamente nell'ambito delle reti.

2.3 Struttura e Framework

Abbiamo capito che per la progettazione di una applicazione ibrida Nella progettazione di una applicazione ibrida bisogna fare delle scelte che sostanzialmente consistono nella scelta dei framework, uno che definisca l'interfaccia utente e la logica dell'applicazione e un altro che si occupi di fornire l'interazione con il sistema operativo.
Dall'interfaccia comunicative che si viene a creare possiamo definirli rispettivamente frontend framework e backend framework.

Quello che ora possiamo definire `frontend framework` si occupa sostanzialmente di fornire una libreria di funzioni per lo sviluppo dell'interfaccia utente e della logica dell'applicazione. Nel momento in cui si necessità di una funzionalità nativa del device si deve utilizzare l'interfaccia fornita dal `backend framework`.

– Livelli di una applicazione –
– Ruolo dei UI Framework - Frontend
– framework intesi solo come librerie –
– Ruolo dei wrapper - Api -Backend
– Riepilogo delle scelte da fare per i componenti di una applicazione ibrida –

2.4 Ciclo di vita di una app

—— Non sono proprio sicuro di metterlo qui,intanto butto giù il contenuto ——
– 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 –

Chapter 3

La soluzione proposta

In questo capitolo discuteremo la soluzione intrapresa per lo sviluppo veloce di applicazioni mobile multi-piattaforma con le tecnologie scelte e le scelte progettuali intraprese.

3.1 Le Tecnologie Web

Le tecnologie web sono sempre più utilizzate per lo sviluppo di applicazioni mobile multi-piattaforma, molti framework come `Ionic`, `Foundation`, `Lungo` utilizzano già HTML5, CSS, Javascript per le loro librerie, ottimizzate per dispositivi mobili.
In particolare questi framewrok mettono a disposizione una serie di componenti già pronti come ad esempio bottoni, menù laterali, form, liste, tabelle, icone, etc... e consentono di abbinarli tra di loro come meglio si ritene, con la possibilità di personalizzarli ulteriormente.

3.1.1 SASS

Nei framework che usano tecnologie web,e anche nelle pagine web, i fogli di stile hanno lo scopo di definire design, forme e colori dei componenti HTML. Nel caso dei framework però, i fogli di stile possono diventare molto complessi diventando lunghi e verbosi e difficili da modificare e/o ri-utilizzare.
Esistono degli strumenti che permettono di strutturare al meglio i fogli di stile preservandone tutte le qualità se non aggiungendo ulteriori potenzialità.
Uno di questi è *SASS(Syntatically Awesome Stylesheet)*; si tratta di una estensione del linguaggio *CSS* che va aggiungere strumenti come: direttive di pre-processore, variabili,

annidamento di classi e id, possibilità di dividere in più file parziali e a sua volta includerne altri all'interno, ereditarietà delle classi, mixins ovvero classi parametrizzate e infine operatori matematici.
Per usare *SASS* dobbiamo creare prima dei documenti con l'estensione *.scss* che una volta compilati daranno origine al foglio di stile *CSS*(o a più fogli di stile). Esistono altri strumenti anche migliori di questo per l'estensione dei fogli di stile, ma successivamente vedremo che il framework che ho utilizzato per la struttura delle mie applicazioni fa uso di questa tecnologia.

3.1.2 AngularJS

AngularJS è un framework Javascript ideato da *Google* per rendere dinamiche le pagine web. E' stato pensato con un modello di sviluppo *Model View Controlled* quindi l'applicazione che si andrà a sviluppare avrà un serie di *view* dinamiche, a loro volta gestite da entità chiamate *controller*.
Una delle caratteristiche che lo rende un framework molto potente da usare è sicuramente il `Data Bindign`. Il Data Binding è un modo per aggiornare dati in una vista ogni volta che questo cambia senza bisogno di aggiornare la pagina o di modificare il DOM. La cosa è reciproca, se i dati di modificano nella *view* (come potrebbe accadere nel caso di un form) questi sono automaticamente aggiornati dinamicamente all'interno della nostra applicazione.
Abbiamo parlato prima di *controller*, sostanzialmente sono delle entità che controllano il comportamento degli elementi della vista. All'interno dei *controller* possiamo definirci tutti i metodi e le funzioni che desideriamo inserire all'interno della vista.
AngularJS consente inoltre di creare i propri tag HTML personalizzati, con la possibilità di aggiungere nuovi attributi opzionali e non. Il punto di forza delle direttive e la loro riusabilità, in quanto , se scritte bene, diventano tag e/o attributi riutilizzabili come del comune codice HTML.
Un altro punto di forza è sicuramente la `Dependency Injection` (verrà spiegata in dettaglio nei capitoli successivi). Questa caratteristica in AngularJS consiste nel descrivere come è connessa la mia applicazione, non abbiamo bisogno di un metodo *main()*, come ad esempio si usa in *Java*, ma possiamo decidere a priori di quali moduli è composta la nostra applicazione.

This means that any component which does not fit your needs can easily be replaced.

Come abbiamo appena detto, una applicazione in AngularJS può essere facilmente divisa in moduli. Questa caratteristica garantisce una riduzione della complessità del codice che

si andrebbe a scrivere, inoltre come il suo inventore Minsko Hevery ripete nella maggior parte delle sue conferenze, AngularJS è stato pensato per essere facilmente testabile.

3.1.3 Ionic

Ionic offre componenti e librerie pensati per uno sviluppo ibrido dell'applicazione, inoltre è stato sviluppato riducendo al minimo la manipolazione del DOM[1] garantendo performance molto competitive.
I componenti di Ionic vengono ovviamente strutturati tramite HTML5, aggiungendo classi css specifiche del framework. Inoltre Ionic fornisce uno strumento molto interessante da linea di comando che permette di scaricare diversi modelli di applicazione già pronti in modo da non dover tutte le volte configurare da capo la nostra applicazione con il framework. Inoltre ci predispone i file in una maniera logica ben precisa cosicché se volessimo in un futuro aggiungere codice e/o altre librerie possiamo farlo con molta facilità. I componenti non sono altro che elementi del linguaggio HTML, vengono forniti con una serie di classi CSS caratteristiche del framework, pensate in modo da essere componibili tra di loro. Ionic fornisce inoltre dei tag propri del framework che possono includere elementi più complessi come ad esempio menù laterali. Il cuore di Ionic e tutte le sue funzionlità sono state sviluppate in AngularJS il che rende questo framework ancora più versatile e potente. Inoltre tutti il foglio di stile che include Ionic fa uso di SASS e ci da la possibilità di personalizzare colori e forme dei componenti semplicemente ricompilando i file *.scss*.

3.2 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.

3.2.1 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

[1] Il Document Object Model (spesso abbreviato come DOM), letteralmente modello a oggetti del documento, è una forma di rappresentazione dei documenti strutturati come modello orientato agli oggetti.**?** gli oggetti delle pagine web, ad esempio con delle animazioni, introduciamo della computazione aggiuntiva al nostro browser che può rallentarne le prestazioni

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`

3.2.2 Differenze

La vera differenza tra `Cordova` e `Phonegap`, viene descritta da *Adobe* analogamente come la differenza tra `Blink`[2] 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.

3.2.3 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. *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.
Infine per quanto riguarda lo sviluppo dei plugin verso il futuro, la comunità open-source di *Cordova* sta spronando gli sviluppatori a creare nuove funzionalità sempre più generiche, in modo tale che con l'evolversi delle tecnologie, e soprattutto dei browser web, siano sempre compatibili.

3.2.4 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(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

[2] Blink è la web browser engine di Google Chrome**?**

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 Un esempio di Sviluppo rapido

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 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. 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.

3.3.2.1 Creazione

Tramite NetBeans inizializziamo un nuovo progetto basato si Cordova. – Immagine nuovo progetto –
– Immagine scelta del template – In NetBeans non abbiamo il template 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.

3.3.2.2 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`.

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 –

3.3.2.3 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*.
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*.

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.

3.3.2.4 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).

3.3.3 SDK

– Spiegazione di che cosa sono le SDK –
– esempio dell'inclusione delle SDK android –

3.4 API Rest

– Tenclogia RESTFUL –

3.4.1 Simple Rest Api

– Non ci occupiamo di costruire le API –
– Uso delle API in angularJS –
– JSON –

3.4.2 XHR

– Cosè Cross Domanin –
– POSTMAN –
– Debug nel Browser –

3.4.3 Firebase

– What, Why –
– of course it's angular!! –
– Ricollegarsi al discorso backend –

Chapter 4

Conclusioni

Appendix A

Appendix Title Here

A.1 HTML5

A.2 AngularJS

A.3 Architettura Client / Server

A.4 Patterns

A.4.1 MVVM

A.4.2 MVC

A.5 HTML5

A.6 Dependency Injection

A.7 Inversion Of Control