ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA

Corso di Laurea in
Ingegneria Elettronica, Informatica e Telecomunicazioni

SVILUPPO DI APPLICAZIONI MOBILE PLATFORM-INDEPENDENT: UN CASO DI STUDIO BASATO SUL FRAMEWORK SENCHA

Elaborata nel corso di: Sistemi Operativi LA

Tesi di Laurea di:
ELISABETTA RAMILLI

Relatore:
Prof. ALESSANDRO RICCI

ANNO ACCADEMICO 2013–2014
SESSIONE III

PAROLE CHIAVE

Mobile

Platform-Independent

Sencha Touch

MVC pattern

JavaScript

Ai miei cari

Indice

Introduzione

Negli ultimi anni si è imposto il concetto di Ubiquitous Computing, ovvero la possibilità di accedere al web e di usare applicazioni per divertimento o lavoro in qualsiasi momento e in qualsiasi luogo. Questo fenomeno sta cambiando notevolmente le abitudini delle persone e ciò è testimoniato anche dal fatto che il mercato mobile è in forte ascesa: da fine 2014 sono 45 milioni gli smartphone e 12 milioni i tablet in circolazione in Italia. [1]
Sembra quasi impossibile, dunque, rinunciare al mobile, soprattutto per le aziende: il nuovo modo di comunicare ha reso necessaria l'introduzione del Mobile Marketing e per raggiungere i propri clienti ora uno degli strumenti più efficaci e diretti sono le applicazioni.
Esse si definiscono native se si pongono come traguardo un determinato smartphone e possono funzionare solo per quel sistema operativo. Infatti un'app costruita, per esempio, per Android non può funzionare su dispositivi Apple o Windows Phone a meno che non si ricorra al processo di porting. Ultimamente però è richiesto un numero sempre maggiore di app per piattaforma e i dispositivi presenti attualmente sul mercato presentano differenze tra le CPU, le interfacce (Application Programming Interface), i sistemi operativi, l'hardware, etc. Nasce quindi la necessità di creare applicazioni che possano funzionare su più sistemi operativi, ovvero le applicazioni platform-independent.
Per facilitare e supportare questo genere di lavoro sono stati definiti nuovi ambienti di sviluppo tra i quali Sencha Touch e Apache Cordova. Il risultato finale dello sviluppo di un'app attraverso questi framework è proprio quello di ottenere un oggetto che possa essere eseguito su qualsiasi dispositivo. Naturalmente la resa non sarà la stessa di un'app nativa, la quale ha libero accesso a tutte le funzionalità del dispositivo (rubrica, messaggi, notifiche, geolocalizzazione, fotocamera, accelerometro, etc.), però con questa nuova app vi è la garanzia di un costo di sviluppo minore e di una richiesta

considerevole sul mercato.
L'obiettivo della tesi è quello di analizzare questo scenario attraverso un caso di studio proveniente da una realtà aziendale che presenta proprio la necessità di sviluppare un' applicazione per più piattaforme.
Nella prima parte della tesi viene affrontata la tematica del mobile computing e quella del dualismo tra la programmazione nativa e le web app: verranno analizzate le caratteristiche delle due diverse tipologie cercando di capire quale delle due risulti essere la migliore. Nella seconda parte sarà data luce a uno dei più importanti framework per la costruzione di app multi-piattaforma: Sencha Touch. Ne verranno analizzate le caratteristiche, soffermandosi in particolare sul pattern MVC e si potrà vedere un confronto con altri framework. Nella terza parte si tratterà il caso di studio, un app mobile per Retail basata su Sencha Touch e Apache Cordova. Nella parte finale si troveranno alcune riflessioni e conclusioni sul mobile platform-independent e sui vantaggi e gli svantaggi dell'utilizzo di JavaScript per sviluppare app.

Capitolo 1

Scenario Mobile

1.1 Mobile Computing

L'espressione mobile computing indica in modo generico le tecnologie di elaborazione o accesso ai dati (anche via Internet) prive di vincoli sulla posizione fisica dell'utente o delle apparecchiature coinvolte. Termini come Pervasive Computing o Ubiquitous Computing sono stati recentemente coniati per indicare la possibilità di accedere alla rete e a sistemi di memorizzazione ed elaborazione dati praticamente in tutti i contesti e attraverso una grande varietà di dispositivi.
La tecnologia dei computer mobili si distingue da quella dei computer portatili in quanto enfatizza la possibilità di usare il computer anche in movimento (per esempio in automobile).
Le origini del mobile computing si possono ricondurre a due fattori: la crescente diffusione dei personal computer portatili e i progressi tecnologici che hanno consentito una graduale riduzione delle dimensioni dei componenti hardware. La categoria dei portatili ha subito una rapida evoluzione negli ultimi decenni, dando origine a una ampia classe di categorie di computer a basso ingombro come laptop, notebook, tablet PC e così via, fino ai palmari. Parallelamente alla miniaturizzazione dei computer, altri dispositivi digitali di piccole dimensioni (come fotocamere digitali, lettori mp3, telefoni cellulari e navigatori GPS) hanno visto un accrescimento delle loro capacità di calcolo e delle loro possibilità di interconnesione, portando a un mercato in cui il confine fra computer e altri dispositivi è sempre più sfumato.
Contemporaneamente, il diffondersi delle tecnologie wireless per la connes-

sione a Internet e in rete ha contribuito a facilitare l'uso di questi strumenti per la trasmissione e la ricezione di dati. [2]

1.1.1 Limitazioni del mobile computing

Sviluppare software per dispositivi mobili significa interfacciarsi con un ambiente diverso dal mondo della programmazione classico; il mobile computing è infatti vincolato da una serie di limitazioni e problemi specifici:

- Scarsità di larghezza di banda: utilizzando le reti dati degli operatori telefonici, è possibile l'accesso a internet dai dispositivi mobili ma esso risulta generalmente più lento della connessione via cavo diretta soprattutto nel caso in cui si voglia accedere a internet essendo in movimento (per esempio in automobile o in treno). L'alternativa sono le reti wireless che sono poco costose ma hanno una larghezza di banda limitata.

- Dimensioni ridotte: non è possibile, come su un computer normale, eseguire tante applicazioni in parallelo e visualizzarle contemporaneamente, perchè lo schermo è piccolo.

- Consumo energetico: dato che non può essere utilizzata la rete elettrica, i dispositivi mobili richiedono batterie ricaribili a lunga durata, a parte alcune eccezioni (per esempio dispositivi per l'uso in automobile che possono essere alimentati attraverso l'accendisigari);

- Interferenze di trasmissione: le condizioni atmosferiche e la conformazione del territorio possono comportare problemi di ricezione e trasmissione, soprattutto in ambienti chiusi come gallerie e edifici;

- Caratteristiche hardware limitate: molti dispositivi dispongono di CPU poco potenti e di una relativamente piccola quantità di memoria RAM e di massa. Tuttavia si prevede che i dispositivi saranno sempre più veloci e potenti fino ad essere effettivamente paragonabili a computer e quindi questa caratteristica tenderà a scomparire anche grazie al progresso sempre più rapido nel campo hardware.

- Interfaccia uomo-computer: gli schermi e le tastiere sono generalmente di piccole dimensioni e ciò li rende potenzialmente complicati da

usare. Inoltre lo schermo di questi dispositivi normalmente è touch, quindi prevede un tipo di interazione con l'utente diverso dalla classica combinazione formata da mouse e tastiera. L'utilizzatore interagisce con le dita o, a volte, utilizzando pennini. Ad ogni modo, l'utilizzo di uno o più dita sul touchscreen fa sì che si instauri un nuovo tipo di interazione, costituito dalle *gesture*, ovvero un insieme di movimenti che possono essere fatti con le dita sullo schermo e che il dispositivo deve essere in grado di riconoscere ed intepretare.

Proprio perchè questi dispositivi dispongono normalmente di scarse risorse hardware e, soprattutto, di poca memoria RAM, ai sistemi operativi è affidata la responsabilità di mantenere la memoria il più pulita possibile, eventualmente terminando applicazioni ed processi che usano troppa memoria o che non sono considerati troppo importanti per l'utente o per il sistema operativo in un determinato momento. Un'applicazione può essere in background o in foreground e solo un'applicazione alla volta, normalmente, è in foreground. Questo anche per favorire una migliore gestione dell'interfaccia che nella maggioranza dei casi è di dimensione ridotta.
Occorre infine tenere conto dell'ergonomia del dispositivo e della facilità d'uso necessaria al grande bacino di utenti che questo tipo di mercato incontra.

1.2 Mobile App

Con il neologismo *app* si intendono le applicazioni informatiche dedicate ai dispositivi di tipo mobile, quali smartphone e tablet. Nel seguito di questo documento verrà molto spesso utilizzato il termine "app" in sostituzione ad "applicazione" per brevità di scrittura.
Un'app per dispositivi mobili si differenzia dalle tradizionali applicazioni per desktop computer sia per il dispositivo con cui viene usata sia per la concezione che racchiude in sé. Si tratta a tutti gli effetti di un software che per struttura informatica è molto simile a una generica applicazione ma è caratterizzata da una semplificazione ed eliminazione del superfluo, al fine di ottenere leggerezza, essenzialità e velocità. Il nome stesso, di per sé un'abbreviazione, può essere percepito come una semplificazione del nome completo "applicazione" per dare l'idea di un qualcosa di semplice e piccolo. [3]

Queste applicazioni possono essere pre-installate sui telefoni durante la produzione, o fornite come applicazioni web ovvero utilizzando un'elaborazione lato server o lato client (per esempio con JavaScript) per fornire un'esperienza "application-like" all'interno di un browser Web.

1.2.1 Sviluppo

Lo sviluppo di mobile app è in costante crescita, sia in termini di ricavi sia di posti di lavoro creati. Da fine 2014 in Europa il numero di posti di lavoro direttamente correlati alla App Economy, il mercato delle applicazioni, ha raggiunto le 670 mila unità e di questi gli sviluppatori di software sono i due terzi, 406 mila. [4]
Nello sviluppare applicazioni mobile è necessario considerarne i vincoli e le caratteristiche. Oltre alle limitazioni precedentemente elencate è necessario tener conto del fatto che i dispositivi mobili hanno funzionalità particolari come il rilevamento della posizione e le fotocamere e che i cambiamenti all'interno di ciascuna delle piattaforme sono molto frequenti.
Perciò lo sviluppo di applicazioni mobile richiede l'utilizzo di ambienti di sviluppo integrati specializzati. Generalmente le app sono prima testate all'interno dell'ambiente di sviluppo tramite emulatori e successivamente sono sottoposte a test sul campo. Gli emulatori forniscono un modo economico per testare le applicazioni sugli smartphone che gli sviluppatori potrebbero non avere a disposizione.
Come parte del processo di sviluppo, anche il design dell'interfaccia utente (UI) è un elemento essenziale per la creazione di applicazioni mobile. Nel piano di progettazione, si devono considerare vincoli e contesti, schermo, input e mobilità. Spesso è l'utente ad essere al centro dell' interazione con il proprio dispositivo e l'interfaccia comporta componenti sia hardware sia software. L'input dell'utente consente di manipolare il sistema, l'output del dispositivo permette al sistema di mostrare gli effetti della manipolazione dell'utente. I vincoli di progettazione dell'interfaccia utente includono attenzione e fattori di forma limitati, come la dimensione dello schermo. In generale, nella progettazione dell'interfaccia utente, l'obiettivo primario è quello di renderla user-friendly e comprensibile. [5]

1.2.2 Ambiti di applicazione

Lo sviluppo di mobile app era inizialmente destinato esclusivamente alla produttività individuale e aziendale: CRM (Customer Relationship Management - gestione delle relazioni coi clienti), ERP (Enterprise Resource Planning - pianificazione delle risorse d'impresa), OLAP (On-Line Analytical Processing - tecnica software per l'analisi di grandi quantità di dati), project management, e-commerce, posta elettronica, calendario, contatti e banche dati. Successivamente, complice la crescente domanda pubblica dovuta alla rapida diffusione dei moderni dispositivi mobili, è stata registrata la rapida espansione in altre aree, come ad esempio giochi per cellulari, scienza applicata, automazione industriale, GPS e acquisti di biglietti.
Oggi esistono centinaia di migliaia di app: giochi e widget di varia natura, quelle per consultare riviste e quotidiani online, per ascoltare la radio, per fotografare e modificare le foto con particolari effetti grafici, per trovare indirizzi e ottenere indicazioni stradali, per ricevere informazioni turistiche, prenotare e acquistare biglietti del treno e dell'aereo o direttamente alberghi, per seguire ricette e corsi di vario tipo, per condividere e scambiare informazioni nei principali social network.
La popolarità delle app ha continuato a crescere, così come il loro utilizzo fino a diventare strumenti indispensabili e irrinunciabili da avere al pari di uno smartphone.

1.2.3 Distribuzione e commercializzazione delle app

Un'app può essere sviluppata per diversi tipi di sistemi operativi mobile ma è compatibile solo con lo specifico sistema operativo per cui è stata creata. Per evitare problemi di incompatibilità, un'applicazione disponibile per diversi tipi di sistemi differisce nella propria estensione, come ogni altro programma o file. Al fine di semplificarne la ricerca e l'utilizzo da parte di utenti anche inesperti, la loro distribuzione è gestita da appositi distributori digitali conosciuti perlopiù con i termini anglosassoni store o market, traducibili in italiano con il termine negozi.
Ogni tipo di distributore è vincolato a un sistema operativo, affinché contenga al proprio interno solo applicazioni compatibili con il sistema operativo del dispositivo mobile che si sta utilizzando. Tuttavia, col diffondersi delle applicazioni, esse sono reperibili ovunque, anche direttamente dai siti di co-

loro che le sviluppano, o dalle aziende o qualsiasi privato che voglia mettere a disposizione una propria applicazione.

1.2.4 Mobile OS

Il mondo dei sistemi operativi mobile è in rapida evoluzione e i guadagni generati dalle app sono in competizione con quelli generati delle "classiche" applicazioni desktop. Negli ultimi sistemi operativi desktop di Apple e Microsoft sono infatti nati gli App Store: si possono comprare le applicazioni desktop in maniera analoga a come si comprano le app sullo smartphone.
Se negli anni Novanta e agli inizi del Duemila la maggior parte delle applicazioni era desktop e la scelta più semplice ed economicamente vantaggiosa era quella di farle per sistemi Windows, ora ci si confronta con un mondo vasto ed eterogeneo dove le persone usano le applicazioni su dispositivi e sistemi operativi diversi: gli attori principali del mondo desktop sono Microsoft, Apple e Linux; quelli del mondo mobile sono Google, Apple, Blackberry e Microsoft.
Non dobbiamo poi dimenticare che negli ultimi anni, grazie al consolidamento dello standard HTML5 (da Ottobre 2014 pubblicato come W3C Recommendation) stanno emergendo altre piattaforme che possiamo quasi assimilare a nuovi sistemi operativi: i moderni web browser Chrome, Safari, Firefox e Internet Explorer.
Se ora ci si mette nei panni di un'azienda che deve sviluppare un nuovo applicativo o servizio software, una domanda sorge spontanea: come orientarsi in questo mondo cercando di ridurre i costi e i tempi e soddisfare le esigenze del maggior numero di utenti possibile?
Ci stiamo pian piano avvicinando alla realtà descritta dall'ormai ventennale concetto *"Write once, run anywhere"* ma spesso con dei compromessi: il cross-platform ha un prezzo. Infatti, quando un codice viene eseguito su più piattaforme è meno efficiente perchè non viene eseguito direttamente a livello di sistema operativo ma dentro una virtual machine che astrae il sistema operativo sottostante aggiungendo overhead e limiti alla esecuzione dell'applicazione. Tuttavia queste virtualizzazioni stanno diventando sempre più efficienti e snelle tanto che forse un giorno si potrà arrivare all'astrazione totale grazie a JavaScript i cui engine, o interpreti, sono diventati delle vere e proprie macchine virtuali che vengono eseguiti sia all'interno dei web browser sia sui server grazie a Node.js.

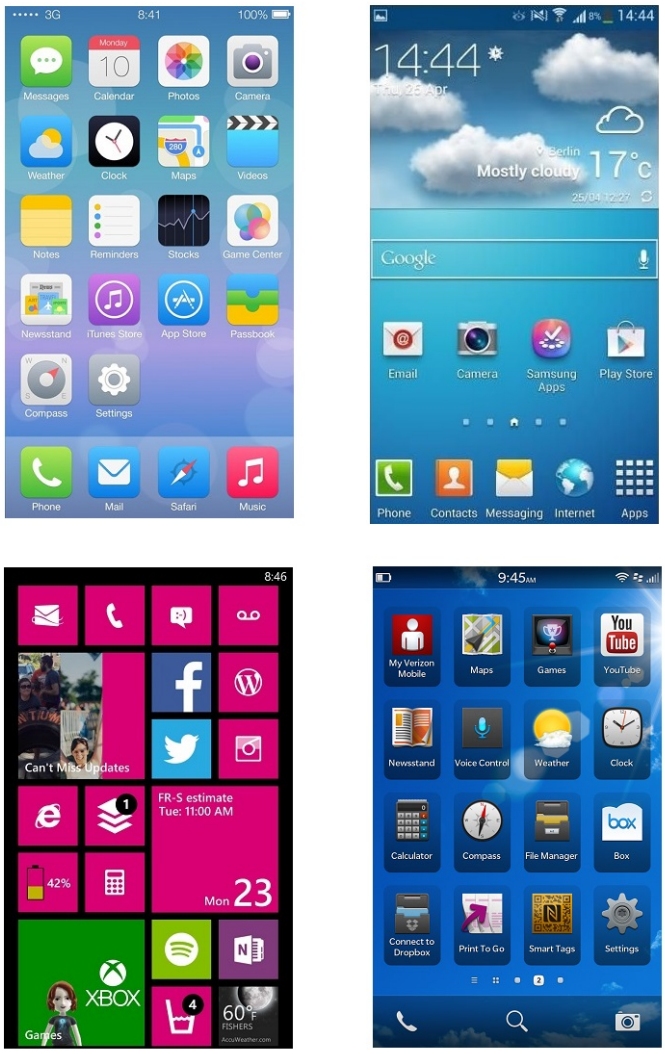

Figura 1.1: I principali mobile OS presenti sul mercato attuale: in alto, da sinistra: iOS e Android; in basso, da sinistra: Windows Phone e BlackBerry

Andiamo ora ad analizzare alcune opzioni per sviluppare le app. Possiamo identificare 5 strategie per sviluppare un'app:

- Native: sviluppare in codice nativo legato al sistema operativo del dispositivo. Esempi: Objective-C per iOS e Java per Android.
- Web app: costruire un sito web visibile e fruibile dal web browser dello smartphone come se fosse una app. Esempio: Sencha Touch.
- Ibrido HTML5: sviluppare web app HTML5 e "impacchettarle" in una web view custom. Esempio: Apache Cordova.
- Compilatori da altri linguaggi: tradurre in automatico un linguaggio conosciuto nei linguaggi nativi dei diversi dispositivi mobile. Esempio: Xamarin.
- Il runtime custom: usare dei runtime personalizzati che eseguono l'app. Esempio: Appcelerator Titanium.

Nel seguito verranno approfondite le 5 soluzioni presentate.

1.2.4.1 App Native

Le app native sono sviluppate appositamente per essere installate su una specifica piattaforma. In genere sono rese disponibili, gratuitamente o a pagamento, sugli store (come Google Play Store, il market ufficiale di Android; Apple Store per dispositivi Apple; Windows Phone Store distributore ufficiale per Windows Phone) e perciò godono di una grande visibilità.
Il principale vantaggio di una app nativa è che può interagire con l'hardware e il software del dispositivo, sfruttando perciò al meglio le funzionalità dello smartphone o tablet (comunicazione informazioni per la localizzazione o l'utilizzo della fotocamera). Le applicazioni native sono più veloci e possono funzionare anche off-line.
Per quanto riguarda gli svantaggi bisogna considerare che i costi e tempi di realizzazione delle applicazioni native sono mediamente più elevati (si deve sviluppare una applicazione per ogni sistema operativo differente). Inoltre per accedere agli store le applicazioni devono soddisfare determinati requisiti stabiliti dai distributori. Anche la gestione degli aggiornamenti è difficoltosa e deve sempre passare attraverso le policy di approvazione degli store.

1.2.4.2 Web App

Le web app sono accessibili tramite i browser di tablet, smartphone e palmari e sono in sostanza un collegamento verso un applicativo remoto.
Le web app hanno il grande vantaggio di non dover essere installate e di poter funzionare su tutti i dispositivi connessi ad Internet. Hanno costi più contenuti e tempi di realizzazione ridotti rispetto alle app native e anche gli aggiornamenti sono rapidi in quanto automaticamente gestiti tramite web.
Per quanto riguarda gli svantaggi bisogna considerare che una web app non può interagire più di tanto con hardware e software del device mobile. Sono più lente nelle loro performance rispetto alle app native e visto che risiedono sul web e vengono fruite tramite il browser, non possono essere distribuite negli store e sono quindi penalizzate dal punto di vista di quella visibilità che i market online garantiscono. Le web app per poter funzionare necessitano di una connessione internet.

1.2.4.3 App Ibrida

Le app ibride sono applicazioni che hanno un'architettura che unisce caratteristiche delle app native a quelle delle web app. Un'app ibrida è tipicamente un'app nativa che ha comunque una parte web e che si adatta facilmente alle diverse piattaforme e diversi device mobili: si dice che il "contenitore" è un'applicazione nativa, che quindi sfrutta le caratteristiche del dispositivo su cui è installata e il "contenuto" è una applicazione web-based e quindi si aggiorna tramite Web.
L'idea in questo caso è di combinare tecnologie di sviluppo native con soluzioni web-based: si realizza il core dell'applicazione in HTML5, così da consentirne l'aggiornamento in tempo reale e da ottimizzare le risorse di sviluppo, ma la si inserisce in un "involucro" sviluppato in codice nativo che estende la webview con API che "wrappano" quelle del dispositivo mobile ed espongono le API in JavaScript. È quindi sufficiente chiamare dalla web view queste API per poter accedere all'hardware del dispositivo come, ad esempio l'accelerometro, i contatti, le notifiche push o la fotocamera del telefono, cosa che invece è proibita da un web browser normale. Inoltre questa strada non costringe a instaurare legami con nessun produttore (se non per l'SDK) perché di fatto l'app viene sviluppata secondo lo standard HTML5 che è una piattaforma open.
Se questa tendenza dovesse confermarsi, spinta anche dalla necessità delle

imprese di operare in logica multipiattaforma, a causa della continua diffusione del trend definito "Bring Your Own Device" (ovvero la possibilità per i dipendenti di accedere alle risorse aziendali a prescindere dalla piattaforma su cui hanno deciso di operare), si arriverà, secondo un recente studio di Gartner (leader mondiale nella consulenza strategica, ricerca e analisi nel campo dell'Information Technology), entro il 2016 a una situazione in cui oltre il 50% delle App saranno sviluppate in modalità ibrida. [6]
Il vantaggio principale delle applicazioni ibride risiede nel fatto che il codice del contenuto viene scritto con un linguaggio comune a tutti i device ed è solo il contenitore a dover essere cambiato in funzione del dispositivo che deve ospitare l'applicazione: questo significa che per la realizzazione di un'app ibrida ci sono tempi e costi più bassi rispetto a quelli necessari per le applicazioni native. Un'app ibrida è comunque considerata come un'app nativa, quindi può essere pubblicata sugli store.
Per quanto riguarda gli svantaggi bisogna considerare che le app ibride non sono performanti e veloci come quelle native perché il rendering UI simula con CSS e JavaScript il look & feel di una app nativa e non è gestito nativamente dal sistema operativo, ma dalla web view. Questo passaggio fa sì che le app ibride siano più lente quando il livello di interazione e grafica è alto. Inoltre vi sono le stesse difficoltà che hanno le app native per accedere agli store dei distributori e per gli aggiornamenti.

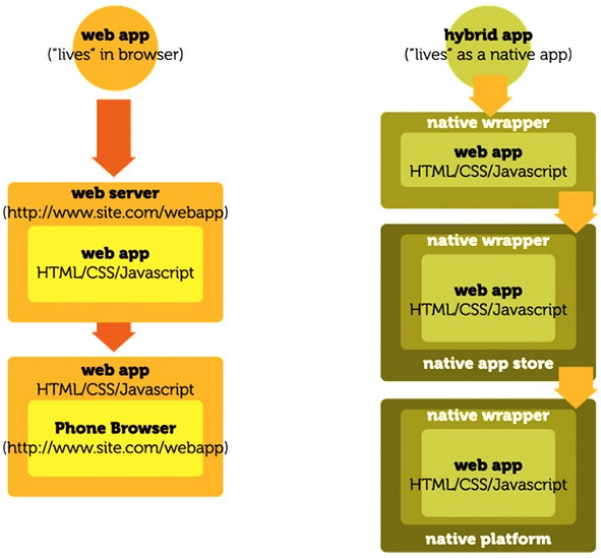

Figura 1.2: Un confronto tra web app e app ibride.

1.2.4.4 Compilatori da altri linguaggi

L'idea è di poter scrivere codice nel proprio linguaggio preferito e poter raggiungere tutti i mobile OS con la semplice pressione di un tasto: Xamarin è un esempio molto ben riuscito di questo approccio. In questo caso le performance non ne soffrono.
Sembra la soluzione ottimale, ma anche questo approccio presenta degli svantaggi: il problema più rilevante è quello di riuscire a star dietro ai sistemi mobile che evolvono nel tempo e continuare ad aggiornare di conseguenza i compilatori. Scegliere questa strada semplifica molto il lavoro, ma lega fortemente al compilatore che si è deciso di usare.

1.2.4.5 Il runtime custom

Appcelerator Titanium è un esempio di engine che viene eseguito sul dispositivo e interpreta a runtime JavaScript richiamando i metodi nativi del sistema operativo. L'approccio coniuga la bellezza della standardizzazione HTML5 con una performace più vicina a quella delle app native. Anche in questo caso si trova uno svantaggio cioè il runtime deve evolvere di pari

passo con il sistema operativo e questo crea, ancora una volta, un forte legame al produttore del runtime.

In generale si può concludere che la scelta del tipo di app dipende da:

- il target a cui si punta (chi saranno gli utenti e che dispositivi mobili utilizzano);
- i tempi e il budget di cui si dispone;
- le funzionalità che l'app deve avere (quindi, ad esempio, come l'applicazione deve interagire con il device su cui deve essere ospitata).

Non bisogna dimenticare che quello delle applicazioni mobili è un settore in continua e rapida evoluzione e gli scenari cambiano continuamente. Il miglior modello da preferire deve essere scelto di volta in volta. [7] [8]

Capitolo 2

Platform-independent

2.1 Introduzione

Per quale piattaforma sviluppare l'app che viene commissionata da un cliente? Non esiste una risposta unica. Scegliere la giusta piattaforma di sviluppo è un passo fondamentale per il successo di un'app, a meno che non sia il cliente stesso ad indicare esplicitamente limitazioni particolari o un brand specifico. Ma su quale investire in fase di sviluppo? Android, iOS, Windows Phone oppure adottare un approccio cross-platform? Cross-platform (o multi-platform o platform-independent) è una modalità di sviluppo dei software che risulta indipendente dal sistema operativo e dal produttore hardware o, meglio, che funziona su più di un sistema operativo.
Rimane il fatto che lo sviluppo nativo è di fondamentale importanza: infatti esiste una classe di applicazioni che non possono proprio essere costruite utilizzando tecnologie ibride o cross-platform: dalle utilità di basso livello (come ad esempio l'app Tasker su Android), ai giochi ad alte prestazioni (anche se le prestazioni grafiche disponibili attraverso Canvas stanno migliorando ogni giorno di più), a quelle in cui è necessaria la comunicazione con le parti dell'SDK della piattaforma o con le periferiche che semplicemente non sono disponibili nel framework cross-platform scelto.
Tuttavia per il 90% delle applicazioni, è bene valutare se è davvero meglio spendere tempo per la creazione di ambienti di sviluppo multipli e quindi ri-sviluppare la propia applicazione in vari linguaggi eterogenei, o se è possibile utilizzare i framework creati appositamente per questo genere di missione.

Come prima passo bisogna valutare cosa si vuole realizzare e qual è il target di riferimento. Sembra semplice, ma un'analisi è importantissima per capire la base di dispositivi utilizzati dal pubblico: gli utenti sono manager di alto profilo? Operai specializzati? Sono dotati di contratti aziendali e quindi navigano senza limiti o sono ragazzi che usano solo il wifi quando disponibile? E dentro una stessa categoria, come usano il prodotto? Spendono €10 in una sola volta, o €0.79 ogni giorno? Condividono l'app o la custodiscono gelosamente perchè magari il contenuto è riservato? Poi è necessario chiedersi in quale campo operare: giochi? app per la produttività? applicazioni social? La risposta è fondamentale per diversi motivi.
Che sia la soluzione giusta o no, è bene ponderare i benefici e le limitazioni che l'approccio platform-independent porta con sè.

2.1.1 Vantaggi e Svantaggi

Mettiamo alla luce i vantaggi dello sviluppo di un applicazione mobile nella modalità cross-platform:

- l'applicazione viene scritta una volta sola (WORA);
- riusabilità del codice: è lo stesso per tutte le piattaforme target;
- è sufficiente imparare un solo linguaggio con cui sviluppare l'app (per esempio, Javascript o C#);
- l'applicazione è compilabile (potenzialemnte) per tutti i mobile OS attuali (Android, Windows Phone, iOS, BlackBerry, etc.);
- si può usare codice specifico per le diverse piattaforme tramite le direttive preprocessore;
- i costi per lo sviluppo e la manutenzione del codice sorgente sono più contenuti;
- è migliore il *Time To Market*, ovvero il tempo che intercorre dall'ideazione del prodotto alla sua effettiva commercializzazione.

Per quanto riguarda gli svantaggi:

- possibili difficoltà nella manutenibilità e nella fase di debugging;

- differenze di comportamento dei dispositivi in rapporto a HTML5 e CSS3 che possono portare a differenze di rappresentazione non sempre semplici da risolvere;
- non tutte le chiamate alle API native sono disponibili su tutti i device quindi occorre tenerne conto nell'implementazione;
- bisogna gestire più progetti: infatti per ogni piattaforma c'è un progetto da compilare per rendere il codice nativo funzionante (progetto su Eclipse per Android e BlackBerry; necessario un Mac OS e Xcode per iOS, necessario Visual Studio per Windows Phone)

2.2 Framework platform-independent

"L'unica cosa che i vari sistemi operativi mobile hanno in comune è che tutti possono accedere al web browser tramite codice nativo". [9]
Questa affermazione e le relative riflessioni che ne sono seguite hanno portato alla nascita dei framework cross-platform. Infatti, utilizzando linguaggi comuni e relativamente semplici, quali HTML5, CSS3 e Javascript, è possibile sviluppare applicazioni che si presentino allo stesso modo quando vengono eseguite in sistemi operativi diversi tra loro.
I framework platform-independent contengono parti di codice nativo che permettono l'accesso all'hardware del dispositivo, ad esempio i sensori. Infatti il codice nativo "avvolge" le funzioni per accedere ai sensori in opportuni "contenitori" e si può richiamare dall'interno di una webview tramite le apposite librerie Javascript che espongono una interfaccia comune per tutte le piattaforme. In questo modo si riesce ad astrarre dal sistema su cui l'applicazione costruita verrà eseguita: saranno infatti i framework a gestire le chiamate invocate dal software al sistema stesso.
Esistono vari framework che si pongono come obiettivo lo sviluppo di applicazioni mobile platform-independent. Nelle sezioni 2.2 e 2.3 si tratterà di quello utilizzato per lo sviluppo della parte JavaScript dell'app oggetto di questa tesi, Sencha Touch; nella sezione 2.4 si tratterrà della piattaforma che si è usata, insieme a Sencha Touch, per sviluppare la parte di platform-independence, Apache Cordova; infine, nella sezione 2.5 si parlerà, per cenni, anche di altri framework di questo genere, quali Appcelerator Titanium, PhoneGap e Kendo UI.

2.3 Sencha

2.3.1 Introduzione generale

Le tecnologie web tra cui HTML5 hanno trasformato i browser in piattaforme applicative di prima classe. Sin dal 2007, Sencha è il leader fornitore di framework e strumenti open source per applicazioni web, utilizzato da più del 50% delle compagnie della Fortune 100 (classifica delle 100 maggiori imprese societarie statunitensi in base al loro fatturato) e da più di 2 milioni di sviluppatori in tutto il mondo.
Software per lo sviluppo di applicazioni professionali per tutti i tipi di dispositivi presenti sul mercato attuale, con potenti strumenti di progettazione e implementazione, Sencha Complete combina le migliori funzionalità per lo sviluppo di applicazioni in un singolo bundle che include Sencha Ext JS, Sencha Touch, Sencha Architect, Sencha Eclipse Plugin e altri.

Figura 2.1: Sencha

2.3.2 Sencha Touch

Sencha Touch, pietra angolare della piattaforma Sencha, è un framework per costruire applicazioni mobile in HTML5 ad alte prestazioni che possono funzionare su molteplici sisemi operativi. Per la realizzazione dell'applicazione oggetto di questa tesi si è utilizzata la versione più recente di Sencha Touch, la 2.4.1.
Sencha Touch è costruito appositamente per creare ottime esperienze-utente: fornisce un ricco set di controlli UI, oltre 300 icone, pieno supporto per i diversi temi, supporto MVC e 8 esempi completi di applicazioni. Sviluppare con Sencha è veloce e semplice anche grazie a una ricca documentazione che permette a ogni sviluppatore, principiante o esperto, di essere produttivo il prima possibile. [10]

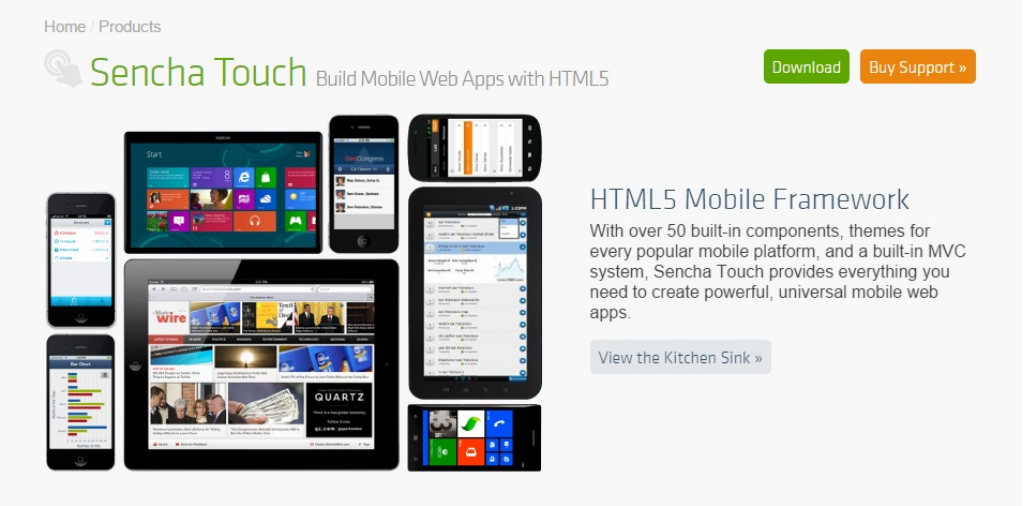

Figura 2.2: Sencha Touch

2.3.3 Features

Sencha Touch include una serie di controlli (o componenti) grafici di interfaccia utente (GUI) per la creazione di app mobile. Questi componenti sono ottimizzati per il metodo di input di tipo touch che è presente ormai su tutti i dispositivi mobile. I componenti sono: buttons con temi ed effetti specifici del dispositivo, form element come text field, selettori, combo-box, toolbars, menù e tanti altri. Fornisce inoltre animazioni fluide e uno scrolling dolce ed elegante per esempio nelle liste e nei carousel, utilizzando automaticamente il miglior meccanismo di scrolling per ogni dispositivo e rendendo quindi l'esperienza utente ottima ovunque.
Con Sencha Touch è possibile costuire applicazioni complesse che hanno tempi rapidi di caricamento, risposta e adattamento del layout: il cambiameto della rotazione da orizzontale a verticale avviene quasi istantaneamente e il motore di layout avanzato assicura la resa perfetta dei pixel.
Le Web app funzionano su ogni dispositivo ma ci sono ancora alcune funionalità che sono disponibili unicamente per le app native che però sono essenziali per gli sviluppatori. Sencha Touch può accedere attraverso l'uso di una shell nativa come Apache Cordova alle API native del SO, quali: accelerometro, fotocamera, bussola, connessione, contatti, eventi, file, geolocalizzazione, media, microfono, notifiche, splashscreen e storage.
Sencha Command offre una serie completa di funzioni e comandi per la gestione del ciclo di vita dei progetti: creare automaticamente la struttura di un nuovo progetto, "impacchettare" l'applicazione per più dispositivi con un singolo comando, minimizzare e distribuire l'applicazione per la produzione.
A livello architetturale la base del sistema è il browser, che interpreta i più moderni linguaggi per la programmazione web: Javascript, HTML5 e CSS3. Questi sono a loro volta le fondamenta su cui è stato creato Sencha Touch.

2.3.4 Piattaforme e temi supportati

L'ultima versione di Sencha Touch supporta le seguenti piattaforme e relativi temi UI:

- Apple: iOS 6 e precedenti (Classic); iOS 8 e 7 (Cupertino)
- Android: Gingerbread (2.3 - 2.3.7); Honeycomb (3.0 - 3.2.6); Ice Cream Sandwich (4.0 - 4.0.4); Jelly Bean (4.1 - 4.3.1); KitKat (4.4 - 4.4.4)
- Windows: Microsoft Surface Pro and RT, Win Phone 8.1 (with IE11)
- BlackBerry: BlackBerry 10
- Tizen: Tizen - Dark; Tizen - Light

I temi sono compresi nell'intero package; a guidare i temi è SASS con una varietà di mixin e variabii che rendono il restyling molto semplice.
SASS è un'estensione di CSS che aggiunge potenza ed eleganza al liguaggio di base. Permette di usare variabili, regole innestate, mixin e import, tutti con una sintassi pienamente compatibile con CSS. Offre inoltre la possibilità di mantenere ben organizzati fogli di stile molto ampi e caricare velocemente fogli di stile di piccole dimensioni. [11]

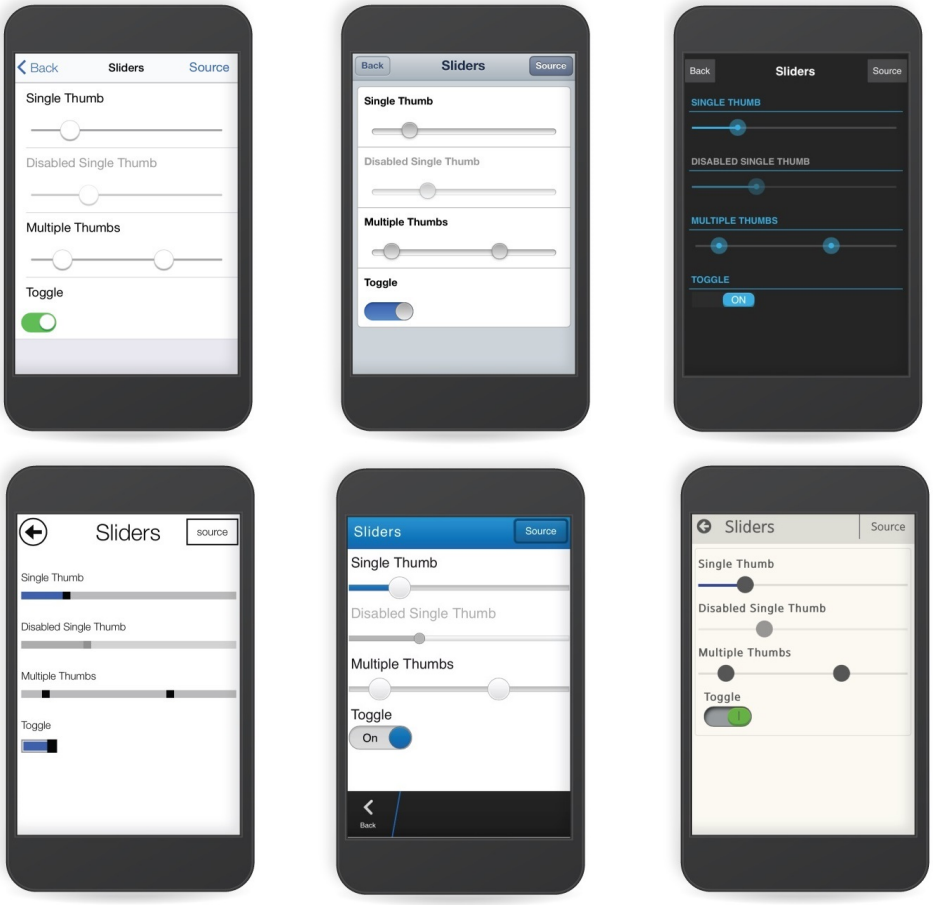

Figura 2.3: I differenti temi disponbili in Sencha Touch: in alto, da sinistra Cupertino, Cupertino Classic e Mountain-View; in basso, da sinistra Windows 8, BlackBerry 10 e Tizen-Light

2.3.5 Il pattern MVC

MVC 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.
Sencha Touch, come fondamenta sulle quali costruire le applicazioni, adotta il paradigma MVC, presenta cioè un'architettura che non solo organizza il codice ma riduce anche la quantità di codice da scrivere e mantenere. L'acronimo indica una tecnica che identifica in una applicazione tre distinte sezioni, suddivise per responsabilità:

- **Model**: trattano della parte legata alla persistenza del dato ed operano su di esso esponendo le logiche di business definite dall'applicazione; i Model vanno passo passo con gli **Store**: i Model definiscono i dati, gli Store mantengono collezioni di dati;
- **View**: sono responsabili della generazione dell'interfaccia e della corretta formattazione del dato;
- **Controller**: agiscono da registi e coordinano model e view per ottenere il risultato voluto.

Il meccanismo tramite il quale le view inviano messaggi ai controller è detto *dispatch* mentre le funzioni del controller usate per gestire tali messaggi sono dette *actions*.

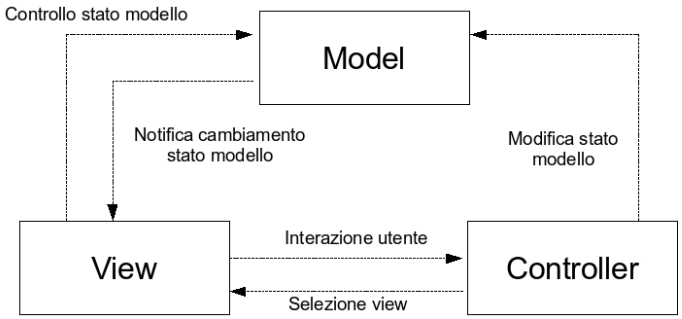

Figura 2.4: Schema del pattern MVC

2.3.5.1 I benefici del pattern MVC

Usare il pattern MVC in Sencha Touch fornisce i seguenti vantaggi:

- L'intera app può essere organizzata tramite il pattern MVC e ad unire tutti i vari componenti è un singolo metodo: Ext.application;
- La struttura dei file per codice e classi è consistente;
- I controller associano direttamente i model e gli store con le view;
- Aggiungere la logica dell'app ai controller è molto semplice;
- Tramite le view, si possono creare componenti riutilizzabili che non sono legati alla logica dell'app;
- Il codice è facile da leggere e mantenere;
- Le dipendenze tra le classi sono più facili da gestire.

2.3.6 Anatomia di un'applicazione in Sencha Touch

Come detto in precedenza, Sencha Touch è ottimizzato per costruire app che lavorano su più piattaforme. Per rendere la scrittura di tali app la più semplice possibile, si usa un pattern architetturale semplice ma potente, ovvero il pattern MVC. Questo approccio rende il codice pulito, testabile e facile da mantenere. [12]

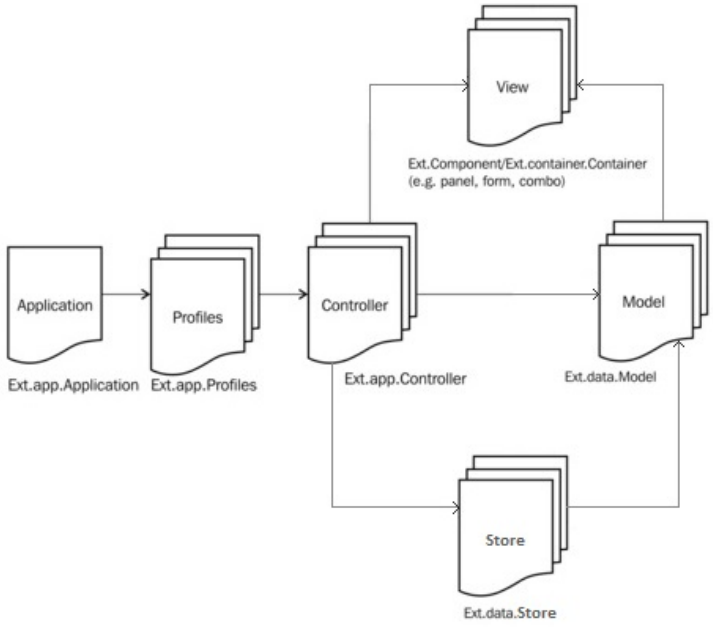

Figura 2.5: Il pattern MVC di un'app in Sencha Touch

- Models: rappresentano un tipo di oggetto di dati nell'app - per esempio un app di tipo e-commerce avrà come modelli User, Product e Order.
- Views: sono responsabili di mostrare i dati agli utenti e usano i componenti predefiniti di Sencha Touch
- Controllers: gestiscono le interazioni con l'applicazione impostando dei listener sulle interazioni dell'utente come tap e swipe ed eseguendo azioni di conseguenza.

- Stores: sono responsabili del caricamento dei dati nell'app e dell'attivazione di componenti come Liste e DataView.
- Profiles: permettono di personalizzare facilmente la UI dell'app per tablet e smartphone, condividendo più codice possibile.

Di solito la prima entità che viene definita in un'app è Ext.Application, ad esempio:

```
Ext.application({
    name: 'MyApp',
    models: ['User', 'Product', 'nested.Order'],
    views: ['OrderList', 'OrderDetail', 'Main'],
    controllers: ['Orders'],

    launch: function() {
        Ext.create('MyApp.view.Main');
    }
});
```

name è usato per creare un singolo namespace globale per l'intera app, includendo tutti i suoi models, views, controllers e altre classi. Per esempio un'app il cui name è MyApp avrà tutte le sue classi costituenti che seguono il pattern MyApp.model.User, MyApp.controller.Users, MyApp.view.Main e così via. Questo rende l'intera app riconoscibile con una singola variabile globale minimizzando perciò le possibilità di conflitto con altro codice nella stessa pagina.
L'app usa le configurazioni definite di models, views e controllers per caricare automaticamente tutte queste classi; la struttura delle classi segue una semplice convenzione: i models stanno nella cartella app/model, i controllers in app/controller e le views in app/view - per esempio app/model/User.js, app/controllers/Orders.js e app/view/Main.js.

2.3.6.1 Controllers

I controllers sono la colla che tiene insieme tutta l'app. Essi si mettono in ascolto di eventi lanciati dalla UI e ne eseguono le azioni corrispondenti. Per esempio, un controller deve mettersi in ascolto di un evento "tap" su un pulsante ed eseguire la relativa azione:

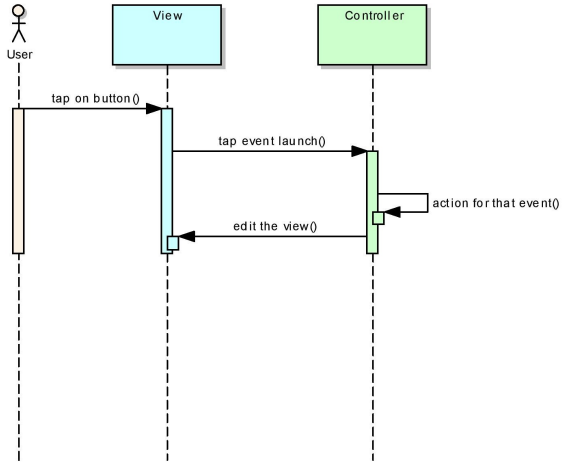

Figura 2.6: Esempio di interazione tra i componenti di un'app in Sencha

Usare i controller rende il codice pulito e leggibile ma soprattutto aiuta a separare la logica di controllo dalla logica delle viste.
Ogni controller è sottoclasse di Ext.app.Controller ed è istanziato una sola volta dall'istanza principale di Ext.Application che l'ha caricato: in ogni momento c'è solo un'istanza di ogni controller.

I controller espongono un piccolo ma potente insieme di funzionalità e seguono semplici convenzioni ad esempio *refs* e *control*. *Refs* sono un modo molto semplice di trovare i componenti della pagina: nell'esempio, il controller cerca tutti i componenti che corrispondono con l'xtype formpanel e assegna il primo trovato alla proprietà loginForm. *Control* usa un selettore di tipo ComponentQuery per cercare tutti gli xtype formpanel che contengono un button. Ogni volta che un button di questo tipo lanica il suo evento "tap", il controller dell'esempio chiama la funzione doLogin.
Definendo nei *ref* loginForm, il controller genera automaticamente un metodo *getLoginForm* che restituisce il formpanel corrispondente. Una volta ottenuto il riferimento al form, se ne possono impostare i valori e passarli ad un'ulteriore funzione.

```
Ext.define('MyApp.controller.Sessions', {
    extend: 'Ext.app.Controller',

    config: {
        refs: {
            loginForm: 'formpanel'
        },
        control: {
            'formpanel button': {
                tap: 'doLogin'
            }
        }
    },

    doLogin: function() {
        var form = this.getLoginForm(),
            values = form.getValues();

        MyApp.authenticate(values);
    }
});
```

2.3.6.2 Stores

Gli stores sono una parte importante di Sencha Touch e alimentano la maggior parte dei widget associati ai dati. Uno store è poco più di un array di istanze di un model. I componenti associati ai dati (come Liste e DataView) presentano un elemento per ogni istanza di un model contenuta nello store. Le istanze di un model possono essere aggiunte o rimosse da uno store.

2.3.6.3 Device Profiles

Sencha Touch opera su un'ampia gamma di dispositivi con diverse caratteristiche e con schermi di grandezze differenti tra loro. Una UI che funziona bene su un tablet potrebbe non funzionare bene su uno smartphone e viceversa, quindi si può provvedere a personalizzare le views in base al tipo di dispositivo di riferimento.
I device profiles sono semplici classi che permettono di definire e di gestire i differenti tipi di dispositivi supportati dall'app. Il loro utilizzo è facoltativo, cioè si può costuire un'app senza utilizzarli e, nel caso, aggiungerli successivamente. Ogni profiles definisce semplicemente una funzione *isActive* che restituisce *true* se quel profilo deve essere attivo sul dispositivo corrente; inoltre definsice un insieme di models, views e controllers addizionali che si caricano solo se viene rilevato quel determinato profilo.

2.3.6.4 Processo di avvio

Ogni istanza di Ext.Application può definire una funzione *launch* che è chiamata non appena tutte le classi dell'app sono state caricate e l'app è pronta per essere avviata. È opportuno che ogni logica di startup dell'app, per esempio la creazione della struttura della view principale dell'app, sia all'interno della funzione *launch*.
Alternativamente, la logica di startup può essere inserita in altri punti: il primo, la funzione *init* di ogni controller, che è chiamata prima della *launch* all'interno di Ext.Application; il secondo, se si stanno usando i device profiles, nella funzione *launch* di un profile, che è chiamata dopo la funzione *init* del Controller ma prima della *launch* di Ext.Application.
Riassumendo, l'ordine di avvio è:

1. init del Controller

2. launch del Profile (se esiste il profile)
3. launch dell'Application
4. launch del Controller

2.3.6.5 Il Class-System

Il class-system di Sencha Touch permette di creare facilmente classi in JavaScript, fornisce l'ereditarietà, mixin, potenti opzioni di configurazione e molto altro ancora. Nel caso più semplice, una classe è solo un oggetto con alcune funzioni e proprietà ad esso assegnate. Ogni classe è definita usando Ext.define e viene creata usando Ext.create; ogni classe può estendere un'altra classe e questa viene specificata usando la sintassi *extend*.
Tra le opzioni di configurazione ci sono le seguenti funzionalità:

- una funzione *getter* che restituisce il valore corrente di un elemento specificato nel *config*;
- una funzione *setter* che ne imposta il nuovo valore;
- una funzione *applier* chiamata dal *setter* che permette di eseguire una funzione quando una certa configurazione viene cambiata;
- una funzione *updater* chiamata dal *setter* che viene eseguita quando il valore di una certa configurazione viene cambiato.

Conviene sempre usare la sintassi *config* per generare automaticamente i *getter* e i *setter* e rendere il codice più pulito.
A volte si rende necessario l'uso di una classe dentro l'altra quindi è necessario garantire che quelle classi siano nella pagina corrente. Questo viene realizzato usando la sintassi *requires*. Quando una classe viene creata in questo modo, Sencha Touch verifica se la classe specificata in *requires* è già stata caricata e in caso negativo provvede immediatamente a caricarla. Questo procedimento funziona bene in modalità di sviluppo, quando non è necessario gestire autonomamente il caricamento di tutti gli script, ma non nella fase di produzione perchè caricare i file uno a uno tramite una connessione internet è un processo abbastanza lento. Nella fase di produzione si vorrebbe caricare un singolo file JS, contenente idealmente solo le classi che l'app di riferimento userà. Questo viene realizzato tramite lo strumento

JSBuilder che analizza l'app e crea un singolo file di build che contiene tutte le classi custom e solo le classi del framework che vengono realmente usate.

2.3.6.6 Il Naming-System

Usare convenzioni di denominazione consistenti per classi, namespace e nomi dei file aiuta a rendere il codice organizzato, strutturato e leggibile.

- Classi: i nomi delle classi devono contenere solo caratteri alfanumerici e seguire la convenzione "CamelCase" (e.g.: MyCompany.data.CoolProxy, MyCompany.Application).
- I nomi delle classi si riferiscono direttamente al file path in cui sono contenute: di conseguenza, dev'esserci solo una classe per ogni file (e.g.: MyCompany.chart.axis.Numeric sta in path/to/src/MyCompany/chart/axis/Numeric.js).
- Metodi e variabili: come per le classi (e.g.: encodeUsingMd5(), getHtml(), getJsonResponse(); var isGoodName, var base64Encoder).
- Proprietà: come metodi e variabili tranne quando sono statiche e quindi si usa la convenzione "UpperCase" (e.g.: Ext.MessageBox.NO = "No", MyCompany.alien.Math.PI = "4.13").

2.3.6.7 Components

Sencha Touch offre un'incredibile libreria di componenti da usare nelle applicazioni. Il diagramma nella figura 2.8 mostra una panoramica semplificata della gerarchia di classi per le views: troviamo i componenti di base che ereditano da Ext.Component; i Containers che ereditano da Ext.Container e i Panels che ereditano da Ext.Panel.

La maggior parte delle classi visuali con le quali si interagisce in Sencha sono Component. Ogni componente in Sencha Touch è una sottoclasse di Ext.Component, il che significa che tutti hanno la possibilità di:

- Fare automaticamente il render sulla pagina utilizzando un modello;
- Mostrarsi e nascondersi in qualsiasi momento;
- Centrarsi sullo schermo;
- Abilitare e disabilitare se stessi.

I componenti hanno anche comportamenti più avanzati che consentono loro di:

- "Galleggiare" sopra altri componenti (finestre, message box e overlay);
- Cambiare le dimensioni e la posizione sullo schermo con un'animazione;
- Bloccare altri componenti all'interno di se stessi (questa funzione è utile per le barre degli strumenti);
- Allinearsi ad altri componenti, lasciarsi trascinare in giro, rendere scorrevole il loro contenuto e altro ancora.

Ogni componente creato ha tutte le suddette caratteristiche e le applicazioni sono costituite da molteplici componenti, di solito annidati uno dentro l'altro. Simili ai componenti sono i contenitori, ma in aggiunta alle funzionalità elencate in precedenza, permettono anche di eseguire il rendering e di disporre i componenti figli al loro interno. La maggior parte delle applicazioni hanno un unico contenitore di livello superiore chiamato Viewport, che occupa l'intero schermo. All'interno della Viewport vi sono i componenti figli. I contenitori forniscono le seguenti funzionalità aggiuntive:

- Aggiunta di componenti figli sia in fase di creazione sia in fase di esecuzione;
- Rimozione dei componenti figli;
- Specifica di un layout.

I Layout determinano come i componenti figli di un contenitore sono disposti sullo schermo.

Un altro modo semplice per creare componenti senza dover utilizzare il nome completo della classe è usare xtype. Ciò è particolarmente utile quando si crea un contenitore che contiene componenti figli. Un xtype è semplicemente un modo veloce di specificare un componente, ad esempio, è possibile utilizzare xtype: 'panel' invece di digitare interamente Ext.Panel. [13]

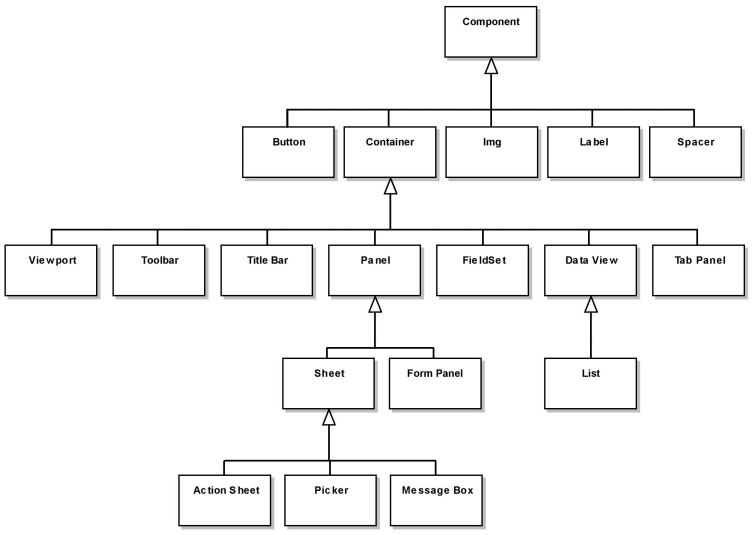

Figura 2.7: I principali Components in Sencha Touch 2.4.1

2.3.6.8 Debugging

Sencha include alcune funzionalità molto utili che aiutano con il debugging e la gestione degli errori.
La funzione Ext.getDisplayName() permette di visualizzare il nome di ogni metodo e questo è particolarmente utile per lanciare errori che nella loro descrizione contengono il nome della classe e il nome del metodo come in questo esempio:

```
throw new Error("["+ Ext.getDisplayName(arguments.callee)
 +"] Some message here");
```

Quando viene lanciato un errore da un qualunque metodo in una qualunque classe definita con Ext.define(), si dovrebbero vedere il nome del metodo e della classe nello stack di chiamate se si sta usando un browser basato su WebKit (Chrome o Safari). Per esempio, questo è ciò che si vedrebbe in Chrome :

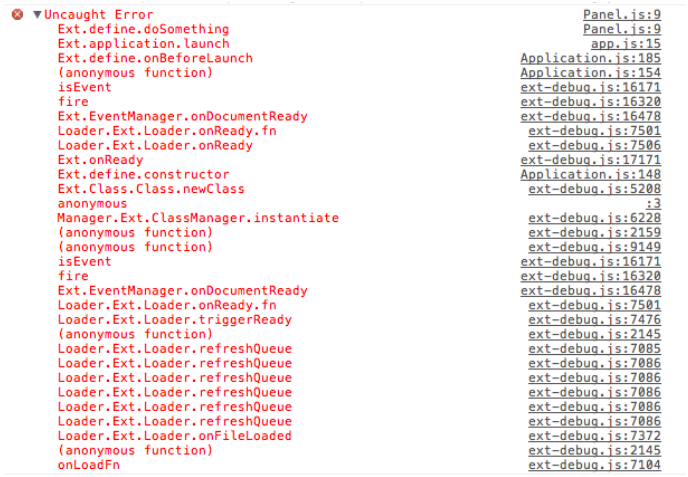

Figura 2.8: Debugging tramite il WebKit di Chrome

2.4 Sencha Cmd

Sencha Cmd è uno strumento a riga di comando multi-piattaforma che fornisce molte funzionalità automatizzate in tutto l'intero ciclo di vita delle applicazioni, dalla generazione di un nuovo progetto alla distribuzione di un'applicazione per la produzione.
Alcune funzionalità di Sencha Cmd sono le seguenti:

- Un compilatore JavaScript che comprende la semantica del framework Sencha.
- Strumenti di generazione del codice per dare origine a intere applicazioni. Per esempio il seguente comando genera lo scheletro di una nuova applicazione "MyApp" nel percorso "/path/to/www/myapp" e contiene tutti i file necessari per sviluppare l'app tra cui index.html, una copia dell'SDK, il file CSS, immagini e file di configurazione:

```
# Make sure the current working directory is the Sencha Touch SDK
cd /path/to/sencha-touch-sdk
sencha generate app MyApp /path/to/www/myapp
```

- Un web server leggero che mette a disposizione i file su localhost. Per farlo partire si usa il comando:

```
sencha web start
```

- Packaging nativo per convertire un'applicazione Sencha in un'applicazione mobile di prima classe. Il comando da utilizzare è il seguente:

```
sencha app build native
```

2.5 Apache Cordova

2.5.1 Introduzione

Grazie ad Apache Cordova è possibile scrivere un'applicazione in HTML5 e pubblicarla su tutte le piattaforme e i marketplace: da iOS a Android, da Windows Phone a Blackberry.
Apache Cordova nasce da un progetto di successo, PhoneGap, avviato da una azienda canadese, Nitobi Software e venduto nel 2011 ad Abobe. Contestualmente alla trattativa con Adobe, Nitobi ha donato alla fondazione Apache il progetto che in un primo tempo aveva assunto il nome di Apache Callback e successivamente il nome attuale di Apache Cordova.
Si tratta quindi un software Open Source distribuito con licenza Apache 2.0.
Allo stato attuale il progetto PhoneGap portato avanti da Adobe non é altro che una distribuzione di Apache Cordova con la possibilità di usufruire di servizi aggiuntivi.

Figura 2.9: Sito ufficiale di Apache Cordova

2.5.2 L'architettura

L'architettura di Apache Cordova si presenta come una sorta di contenitore di applicazione Web eseguita localmente. L'interfaccia grafica di un'applicazione Cordova è infatti costituita da una Web view che occupa l'intero schermo del dispositivo e all'interno della quale viene visualizzato l'HTML ed il CSS ed eseguito il codice JavaScript. Tramite JavaScript è possibile accedere ad un ricco insieme di API che interfacciano l'applicazione Web con le funzionalità della piattaforma ospite:

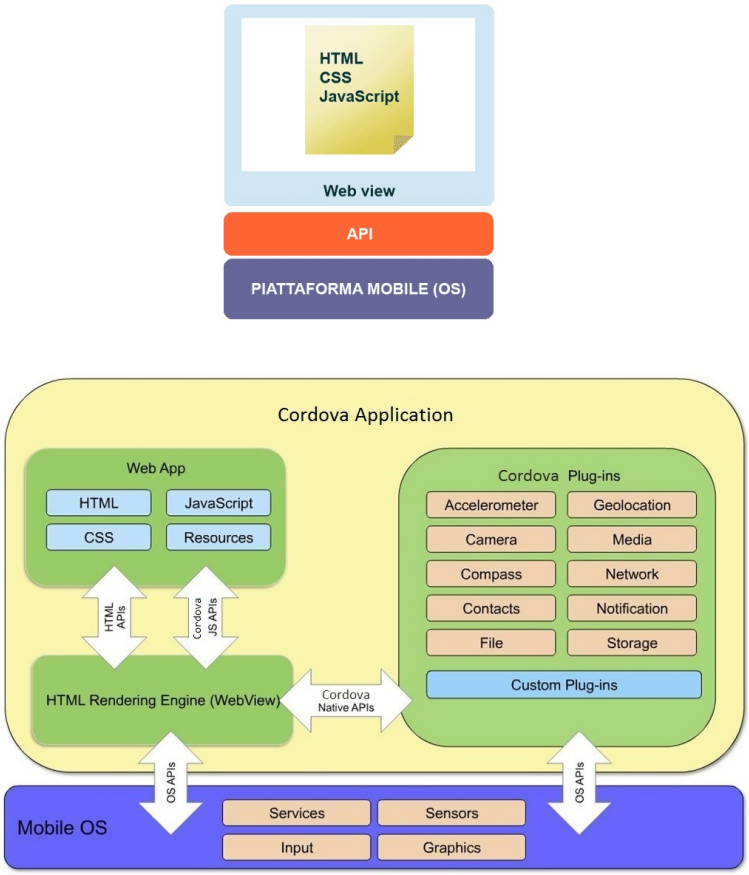

Figura 2.10: In alto: l'architettura di Cordova; in basso, la stessa più nel dettaglio

Pertanto Web view e API sono le componenti dell'applicazione che dipendono dalla specifica piattaforma mobile e sono appunto queste componenti che il framework mette a disposizione dello sviluppatore, consentedogli di concentrarsi sul codice standard indipendente dalla piattaforma.
Cordova infatti implementa lo stesso insieme di API sulle diverse piattaforme mobile supportate creando un livello software standard a cui si possono interfacciare le nostre applicazioni. Per chiarire meglio il concetto, è come se Apache Cordova mettesse a disposizione dello sviluppatore delle app vuote, una per ciascuna piattaforma mobile, da riempire con codice HTML, CSS e JavaScript. Una volta riempite si avrà un'applicazione per ciascuna piattaforma con la stessa base di codice HTML, CSS e JavaScript.
Allo stato attuale Apache Cordova supporta le seguenti piattaforme mobile: Android, iOS, Blackberry, Bada, Tizen e Windows Phone. Questo dà un'idea della platea di utilizzatori di cui potrebbe usufruire la nostra app e del lavoro che sta dietro al progetto.

Per creare un progetto di Cordova si può usare la riga di comando: infatti il team di sviluppo di Apache Cordova ha realizzato un apposito pacchetto denominato Cordova-Cli. L'approccio a riga di comando ha indubbi vantaggi quando si prevedono più piattaforme mobile di riferimento e si vuole evitare di utilizzare diversi strumenti di sviluppo e quindi diversi progetti. Per poter eseguire i comandi implementati dal pacchetto è necessario prima aver installato Node.js e gli SDK per le piattaforme mobile a cui si è interessati. Per la creazione del progetto si usa il comando:

```
cordova create <project_directory> <package_name> <project_name>
```

Il primo argomento specifica una cartella da generare per il progetto. Questa directory non dovrebbe esistere già, Cordova la crea automaticamnte. La sottocartella *www* ospita la home page dell'applicazione, insieme a varie risorse nelle sottocartelle *css*, *js* e *img*. Il file config.xml contiene importanti metadata necessari per generare e distribuire l'applicazione.
Il secondo argomento fornisce un identificatore di dominio, che nel nostro caso è *it.bkm.retail*.
Il terzo argomento fornisce un titolo di visualizzazione per l'applicazione.

Listing 2.1: config.xml

```
<?xml version='1.0' encoding='utf-8'?>
<widget id="it.bkm.retail" version="0.0.1"
    xmlns="http://www.w3.org/ns/widgets"
    xmlns:cdv="http://cordova.apache.org/ns/1.0">
    <name>Retail</name>
    <description>
        A sample Apache Cordova application that responds to the
            deviceready event.
    </description>
    <author email="dev@cordova.apache.org"
        href="http://cordova.io">
        Apache Cordova Team
    </author>
    <content src="index.html" />
    <access origin="*" />
</widget>
```

Per aggiungere le piattaforme mobile, ad esempio android e windows phone 8.1, si usano i comandi:

```
cordova platform add android
cordova platform add windows
```

Per preparare e compilare l'app si usa il comando:

```
cordova build <platform>
```

L'argomento *platform* è facoltativo: si può infatti fare automaticamente la build di tutte le piattaforme oppure solo di una alla volta.

2.5.3 Uso dei plug-in

Esplorando le API di Apache Cordova si può vedere come si potrebbero coprire la maggior parte delle esigenze di sviluppo di un'app mobile. Le API consentono di utilizzare funzioni native mediante JavaScript ignorando

i dettagli implementativi di ciascuna piattaforma mobile. Nel caso in cui si abbia bisogno di una funzionalità nativa non prevista dalle API, si può far ricorso ai plug-in. Un plugin Cordova è un componente software che consente di mappare una funzione JavaScript ad una funzionalità nativa. Esso è costituito da un'unica interfaccia JavaScript e da tante implementazioni dipendenti dalla specifica piattaforma mobile. In realtà tutte le API di Cordova sono implementate secondo questo modello, quindi un plugin non fa altro che estendere le API aggiungendo nuove funzionalità.
Quando si crea un progetto di Cordova, non sono presenti plug-in come comportamento predefinito. Qualunque plug-in si desideri, deve essere aggiunto esplicitamente. Ad esempio, per aggiungere la funzionalità di riconoscimento di codici a barre si utilizza il seguente comando di cordova:

```
cordova plugin add com.phonegap.plugins.barcodescanner
```

Una volta installato un plug-in è possibile utilizzarlo nel progetto Cordova in modo analogo a come si utilizzerebbero le API. Ad esempio, per avviare la scansione di un codice a barre si dovrebbe usare il seguente codice JavaScript:

```
barcodeScanner.scan(onSuccess, onError);
```

2.5.3.1 Plug-in personalizzati

Si è detto più volte detto che Apache Cordova consente di ignorare i dettagli implementativi di un'applicazione dipendenti dalla piattaforma mobile. Questo naturalmente non è vero se si intende realizzare un plug-in. Anzi, se si vuole creare un plugin che deve funzionare su più piattaforme bisogna prendersi l'onere di sviluppare il backend per tutte le piattaforme. Vediamo ora come verrebbe specificata la corrispondenza tra codice JavaScript e codice nativo. Un plugin può essere utilizzato secondo questa sintassi:

```
navigator.goPlugin("myString", onSuccess, onError);
```

L'interfaccia JavaScript può essere definita nel seguente modo:

```
navigator.goPlugin = function(str, onSuccess, OnError) {
    cordova.exec(onSuccess, onError, "myPlugin", "go", [str]);
}
```

La funzione responsabile della chiamata al codice nativo è cordova.exec() che si occupa di passare i parametri al codice sottostante in maniera asincrona. I parametri previsti dalla funzione sono, oltre alle callback di successo e di errore, una stringa che rappresenta la classe o il servizio sottostante ("myPlugin" nel nostro caso), il metodo o l'azione relativa alla classe o servizio specificato nel parametro precedente ("go" nell'esempio) e un array di parametri passati dall'utente alla funzione.
Questa interfaccia vale per tutte le piattaforme. Lo step successivo sarebbe quello di implementare il codice nativo sottostante. [15]

2.6 Il ruolo di JavaScript

JavaScript è considerato il più diffuso ed utilizzato linguaggio di scripting e permette di implementare la logica all'interno di pagine Web, nelle quali è affiancato da CSS e HTML; un suo grande vantaggio è che, per sviluppare un task comune, lo sviluppatore non deve necessariamente essere un esperto.
Tuttavia JavaScript utilizzato come solo linguaggio di scripting non è sufficiente per modellare applicazioni complesse che si avvicinano ad app native. Infatti l'applicazione che è stata sviluppata per questa tesi è ibrida, ovvero non è puramente basata su HTML e JavaScript ma ha anche un lato nativo: è Cordova a fornire un "ponte" tra queste due parti permettendo alle API JavaScript di accedere all'hardware del dispositivo.
Si comprende dunque che JavaScript, pur essendo object-based, ha alcune limitazioni:

- mancano i costrutti delle interfacce e delle classi astratte;
- è poco "tipizzato": non permette il controllo dei tipi a tempo di compilazione e quindi i bug relativi ai tipi possono essere facilmente introdotti a causa dell'imprecisione umana;

- i bug possono essere difficili da rilevare o possono addirittura passare inosservati dal runtime per diverse ragioni: il codice host potrebbe non essere stato eseguito o non essere raggiungibile oppure potrebbe fallire la comunicazione tra alcune regole di business anche se il codice è digitato correttamente;
- il meccanismo di estensione che propone non è elementare;
- non avendo le classi, pone limiti all'"ingegnerizzazione" di un software.

Esistono molte librerie JavaScript il cui obiettivo è quello di rendere migliore l'esperienza di sviluppo. Alcune di queste sono:

- jQuery, utilizzata per la manipolazione del DOM (Document Object Model - modello a oggetti del documento);
- Ext.js che include una serie di controlli per i form basati sulla GUI (detti anche "widget");
- AngularJS, utile a semplificare la realizzazione di applicazioni Web single page, favorendo un approccio dichiarativo allo sviluppo client-side;
- Underscore.js che fa parte dei package di Node.js, framework per lo sviluppo lato server.

Bisogna considerare però che anche le librerie JavaScript possono presentare bug che potrebbero essere difficilmente rintracciabili.

Non si può dire quindi con certezza se JavaScript sia davvero la lingua franca di sviluppo per la piattaforma web open-source; anzi, potrebbe addirittura essere sostituito da altri linguaggi.

2.7 Altri Frameworks

Oltre a quelli già trattati, esistono molti altri framework e tool per realizzare app multi piattaforma: di questi, nel seguito, sarà data una breve presentazione. [14]

2.7.1 Appcelerator Titanium

Appcelerator Titanium è un framework mobile open source che fornisce un ambiente per creare applicazioni native per diverse piattaforme mobile: è una soluzione completa per la creazione di applicazioni mobile ibride. Per lavorare si usa Titanium studio. L'SDK è dotato di una serie di API per piattaforme mobile e di un servizio Cloud da utilizzare come backend dell'app; esso viene fornito con le API per il platform-indipendent e ciò rende più facile l'accesso all'hardware del telefono.
Titanium utilizza Alloy, un framework MVC per consentire un rapido sviluppo di applicazioni mobile. I moduli creati con Alloy sono facili da riutilizzare tra le varie applicazioni, riducendo così in modo significativo i tempi di sviluppo e le righe di codice.
Titanium Studio viene fornito con alcuni esempi di codice per i principianti.

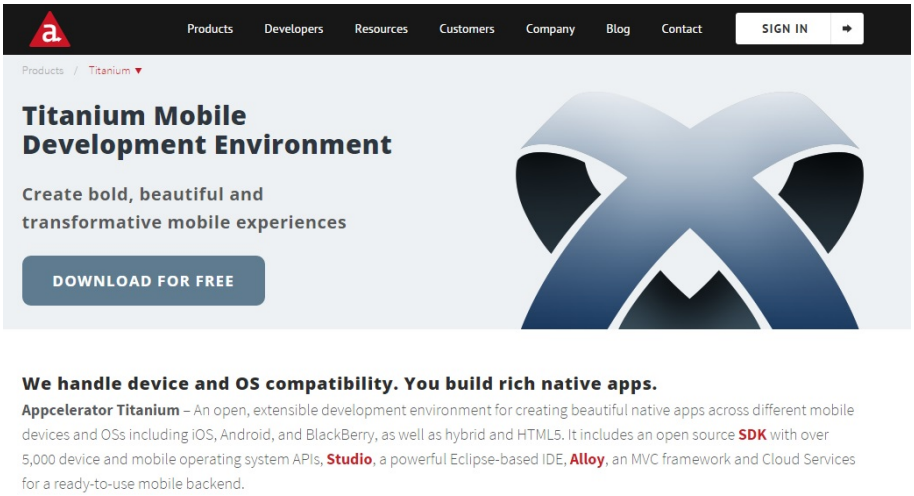

Figura 2.11: Sito ufficiale di Appcelerator Titanium

2.7.2 PhoneGap

PhoneGap non è un vero e proprio quadro di riferimento per la creazione di un'app, bensì lo è per il suo packaging e rilascio. PhoneGap si basa su Cordova ed è la versione commerciale di proprietà di Adobe. Con un team di supporto dedicato, PhoneGap è molto popolare tra gli sviluppatori di mobile.
Per iniziare a lavorare con PhoneGap, è possibile utilizzare una qualsiasi combinazione di JavaScript o framework per l'interfaccia utente: ad esempio, jQuery Mobile a fianco di KnockOut.js o di AngularJS. Una volta scritto il codice, PhoneGap lo avvolge a seconda della piattaforma prevista. Per rendere i loro contenuti, le applicazioni create utilizzando PhoneGap utilizzano una vista web (web view). PhoneGap ha un insieme minimo di API per accedere alle funzioni hardware del telefono ed è possibile scrivere plugin personalizzati per soddisfare le proprie esigenze.

Figura 2.12: Sito ufficiale di PhoneGap

2.7.3 Kendo UI

Kendo UI di Telerik è un framework HTML 5 per la creazione di applicazioni mobile cross-platform. Kendo UI si basa particolarmente su jQuery ed ha una serie di widget basate su jQuery.
Conoscere Kendo UI non è difficile: infatti gli sviluppatori che hanno familiarità con jQuery trovano Kendo UI facile da imparare. Kendo UI ha reso libero l'accesso alla maggior parte del suo insieme di strumenti e alle funzionalità di JavaScript. Tuttavia la maggior parte dei widget comunemente utilizzati sono ancora sotto una licenza commerciale.
Per iniziare a sviluppare con Kendo UI, si può fare riferimento alla documentazione ufficiale. Si trovano anche una serie di video tutorial per conoscere meglio il framework.

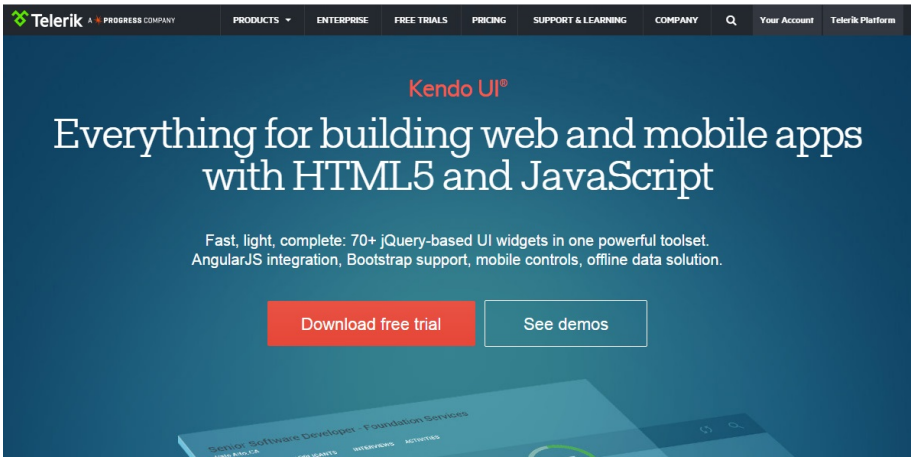

Figura 2.13: Sito ufficiale di Kendo UI

Capitolo 3

Il caso di studio

In questo capitolo verrà presentata e descritta l'app mobile che è stata realizzata nel contesto di questa tesi con l'utilizzo di framework platform-independent quali Sencha Touch e Apache Cordova.

3.1 RetailApp

L'app è di tipo retail per i consumatori, cioè è indirizzata ai clienti di un supermercato italiano, qui detto "IL Supermercato", per offrire loro un nuovo modo di fare la spesa usufruendo di servizi extra tramite il loro smartphone. Il primo passo di questo progetto è stato quello di ricercare applicativi di tipo retail già esistenti che potessero aiutarci a conseguire il risultato cercato. Infatti per un'azienda è fondamentale creare il migliore prodotto possibile da presentare sul mercato. L'analisi comparativa si è concentrata sulla ricerca di come meglio implementare le funzionalità di creazione di una lista della spesa, di ricerca dei punti vendita vicini e di visualizzazione dell'elenco dei prodotti in promozione presso un certo punto vendita. In seguito all'analisi sono state delineate le specifiche che avrebbe poi soddisfatto la nostra app.

3.1.1 Requisiti

Quella realizzata nel progetto di tirocinio è una demo dell'app che poi sarà sviluppata in futuro in modo più ampio nel contesto aziendale. Nel seguito si trovano quindi i requisiti che sono stati adottati per la demo.

Una volta caricata l'app, compare la homepage con varie voci:

- Chi siamo
- Punti Vendita
- Lista della spesa
- Prodotti e Promozioni

3.1.1.1 Chi siamo

Si apre il link al sito web del supermercato in questione; si verifichi se il dispositivo è connesso a internet, diversamente avvisare l'utente.

3.1.1.2 Punti Vendita

Prevede due tipi di visualizzazione dei punti vendita: ad elenco o sulla mappa. È presente una barra di ricerca per trovarli in base al nome o alla città; è presente l'icona dei filtri: preferiti o altre specializzazioni, per esempio diverse insegne del supermercato trattato (per esempio: superstore, ipermercato, minimarket, etc.); per ogni punto vendita c'è un icona che porta alla mappa. Selezionando un punti vendita se ne possono visualizzare, sotto forma di pagina html, i dettagli (orari apertura, indirizzo, servizi, reparti, etc.) e lo si può impostare come preferito selezionando l'apposita icona.

3.1.1.3 Lista della spesa

Si apre la pagina della lista della spesa. In alto si trova un campo di testo modificabile in cui si possono scrivere i prodotti che si vuole aggiungere ed essi vengono direttamente messi nella lista sottostante. Se premo sul prodotto si apre una pagina in cui trovo una lista di prodotti filtrata in base alla categoria del prodotto che è stato scritto. La lista della spesa può contenere prodotti generici, scritti dall'utente con note e quantità scritte dall'utente (lista della spesa rapida), oppure prodotti specifici (lista della spesa dettagliata).
Lista rapida: L'utente scrive il prodotto nel campo di testo modificabile e

preme sull'icona-aggiungi.
Lista dettagliata: Premendo su un'icona-freccia si apre la finestra dove si può modificare il nome del prodotto e aggiungere note/quantità; si può inoltre scegliere un prodotto specifico dalla lista vedendone descrizione, prezzo ed eventuali promozioni. Se si vuole aggiungere un prodotto specifico alla lista si clicca sull'icona-carrello e si apre una nuova finestra. Nella lista rapida c'è l'icona del prodotto scritto dall'utente se la stringa scritta ha corrispondenza in un file in cui ci sono delle coppie di stringhe di tipo prodotto-immagine. Se non c'è corrispondenza, appare un'icona-punto interrogativo.

3.1.1.4 Prodotti e Promozioni

Nella pagina dei prodotti si può vedere l'elenco di tutti i prodotti; cliccando sull'icona-carrello si può aggiungere il prodotto alla lista della spesa.

3.1.2 Scenari

Nel seguito verranno presentati alcuni diagrammi delle interazioni che fanno riferimento ad alcune funzionalità dell'app; precisamente verranno considerati i seguenti scenari:

- Pagina dei punti vendita, visualizzazione dell'elenco dei negozi.
  L'utente preme sull'icona filtro e alla comparsa di una view addizionale, filtra l'elenco in base a un pattern. Una volta filtrato l'elenco, preme sul nome del punto vendita cercato e appare la pagina dei dettagli sul punto vendita. Imposta tale punto vendita come "preferito" premendo sull'icona-preferito. Visualizza il punto vendita sulla mappa premendo sull'icona-marker.

- Pagina della lista della spesa.
  L'utente la compila inserendo dei prodotti. Premendo su un prodotto si apre una seconda pagina e da qui sceglie un prodotto specifico, ne imposta la quantità e lo inserisce nella lista.

- Pagina dei prodotti e delle promozioni, visualizzazione di tutti i prodotti.
  L'utente sceglie un prodotto di suo interesse da inserire nella lista della spesa, premendo sull'icona-carrello.

3.1.3 Architettura dell'applicazione

Da qui in poi viene adottata la seguente convenzione per i colori nei diagrammi: in verde i Controllers, in azzurro le Views, in grigio i Models, in giallo gli Stores.
Una panoramica dell'architettura dell'applicazione puo essere rappresentata in maniera semplificata dal diagramma seguente:

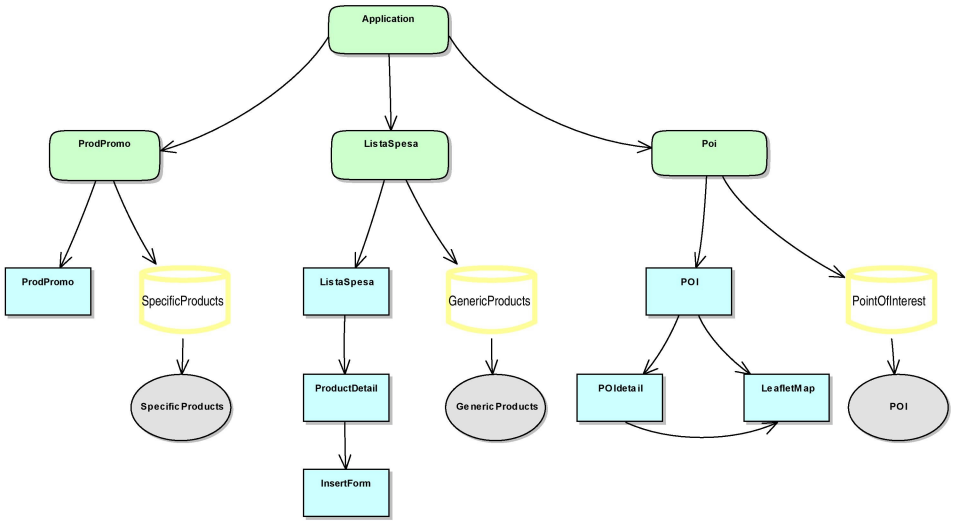

Figura 3.1: RetailApp - panoramica

Andiamo ora a descrivere in dettaglio la struttura e il funzionamento dell'applicazione.
Come detto precedentemente, i tre strumenti base per sviluppare un'applicazione con Sencha Touch e Cordova sono HTML, CSS e Javascript. Il file HTML conterrà la struttura della view, i CSS definiranno lo stile (dimensioni, colori dei pulsanti, carattere, etc.) mentre i file JavaScript conterranno la logica dell'app. Il file index.html contiene il riferimento al file cordova.js il quale contiene il codice JavaScript che interfaccia la nostra applicazione con l'infrastruttura nativa realizzata da Apache Cordova. Il codice presente nel file index.html è il seguente:

Listing 3.1: index.html

```
<!DOCTYPE HTML>
<html manifest="" lang="en-US">
<head>
    <meta charset="UTF-8">
    <title>RetailApp</title>
    <style type="text/css">
        html, body {
            height: 100%;
            background-color: rgba(0,0,0,0);
        }
        #appLoadingIndicator {
            position: absolute;
            top: 50%;
            margin-top: -15px;
            text-align: center;
            width: 100%;
            height: 30px;
            -webkit-animation-name: appLoadingIndicator;
            -webkit-animation-duration: 0.5s;
            -webkit-animation-iteration-count: infinite;
            -webkit-animation-direction: linear;
        }
        #appLoadingIndicator > * {
                    background-color: #000000;
                    display: inline-block;
                    height: 30px;
                    -webkit-border-radius: 15px;
                    margin: 0 5px;
                    width: 30px;
                    opacity: 0.8;
        }
        @-webkit-keyframes appLoadingIndicator{
                    0% {
                        opacity: 0.8
                    }
                    50% {
                        opacity: 0
                    }
                    100% {
                        opacity: 0.8
                    }
        }
        </style>
```

```
        <link href="lib/leaflet/current/leaflet.css"
            rel="stylesheet" type="text/css">
        <script type="text/javascript">
                var mediaFolderPath="media/";
        </script>
        <script id="Jed" type="text/javascript"
            src="lib/jed.js"></script>
        <script id="Cordova" type="text/javascript"
            src="lib/cordova.js"></script>
        <script type="text/javascript"
            src="lib/leaflet/current/leaflet.js"></script>
        <!-- The line below must be kept intact for Sencha Command
            to build your application -->
    <script id="microloader" type="text/javascript"
        src=".sencha/app/microloader/development.js"></script>
</head>
<body>
    <div id="appLoadingIndicator">
        <div></div>
        <div></div>
        <div></div>
    </div>
</body>
</html>
```

Nall'app creata con Sencha Touch, l'attore principale è app.js, che contiene un'istanza di Ext.app.Application che definisce l'insieme di Models, Controllers, Stores e Views di cui l'applicazione è costituita. Tale classe carica automaticamente tutte quelle dipendenze e specifica una funzione *launch* che viene chiamata quando tutto è pronto. Inoltre vengono specificate dipendenze esterne all'app dichiarando esplicitamente le classi da caricare.
Il codice presente nel file app.js è il seguente:

Listing 3.2: app.js

```
Ext.Loader.setConfig({
    enabled: true,
    paths: {
        'Ext.ux': 'ux'
    }
});
Ext.application({
    name: 'RetailApp',
    requires: [
        'locales.en',
        'locales.it',
        'Ext.ux.Localization',
        'Ext.MessageBox'
    ],
    controllers: [
        'Application',
        'Maps',
        'Poi',
        'ListaSpesa',
        'ProdPromo'
    ],
    views: [
        'MainTab'
    ],
        config:{
               mediaPath:'media/'
        },
    launch: function() {
        // Destroy the #appLoadingIndicator element
        Ext.fly('appLoadingIndicator').destroy();

        // Initialize the main view
        Ext.Viewport.add(Ext.create('AGORA.view.MainTab'));
    }
});
```

3.1.3.1 Struttura

Di seguito si trovano i diagrammi UML che descrivono come è strutturata l'applicazione. Rimane valida la convenzione per i colori stabilita in precedenza.

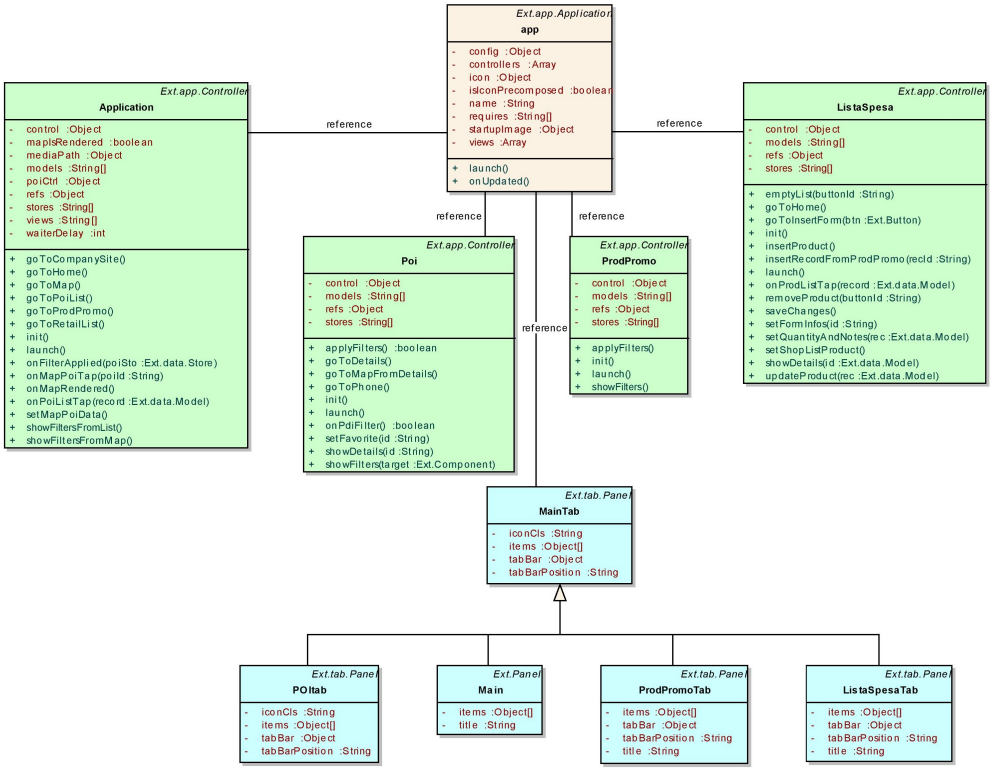

Figura 3.2: RetailApp - struttura principale

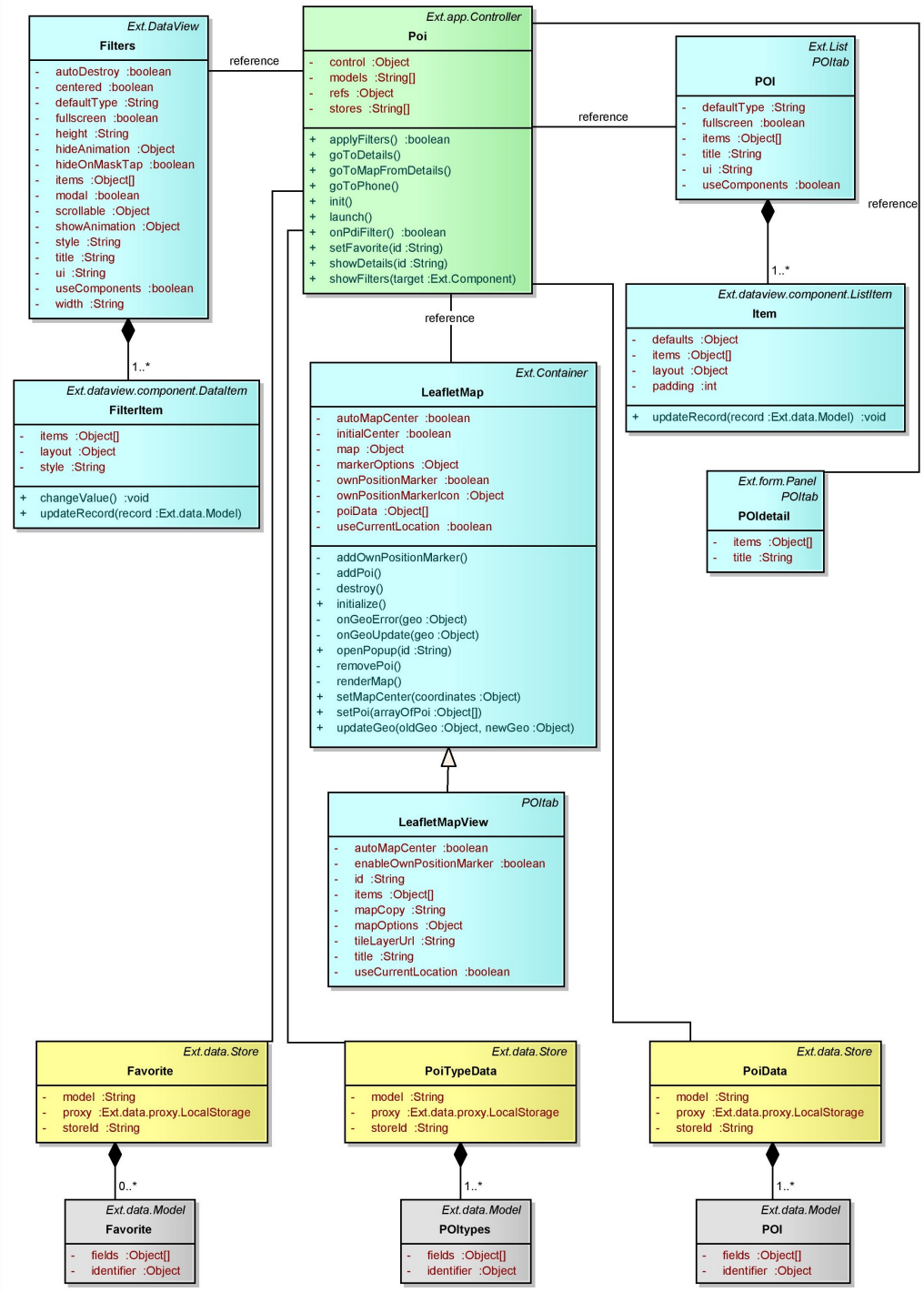

Figura 3.3: RetailApp - Punti Vendita

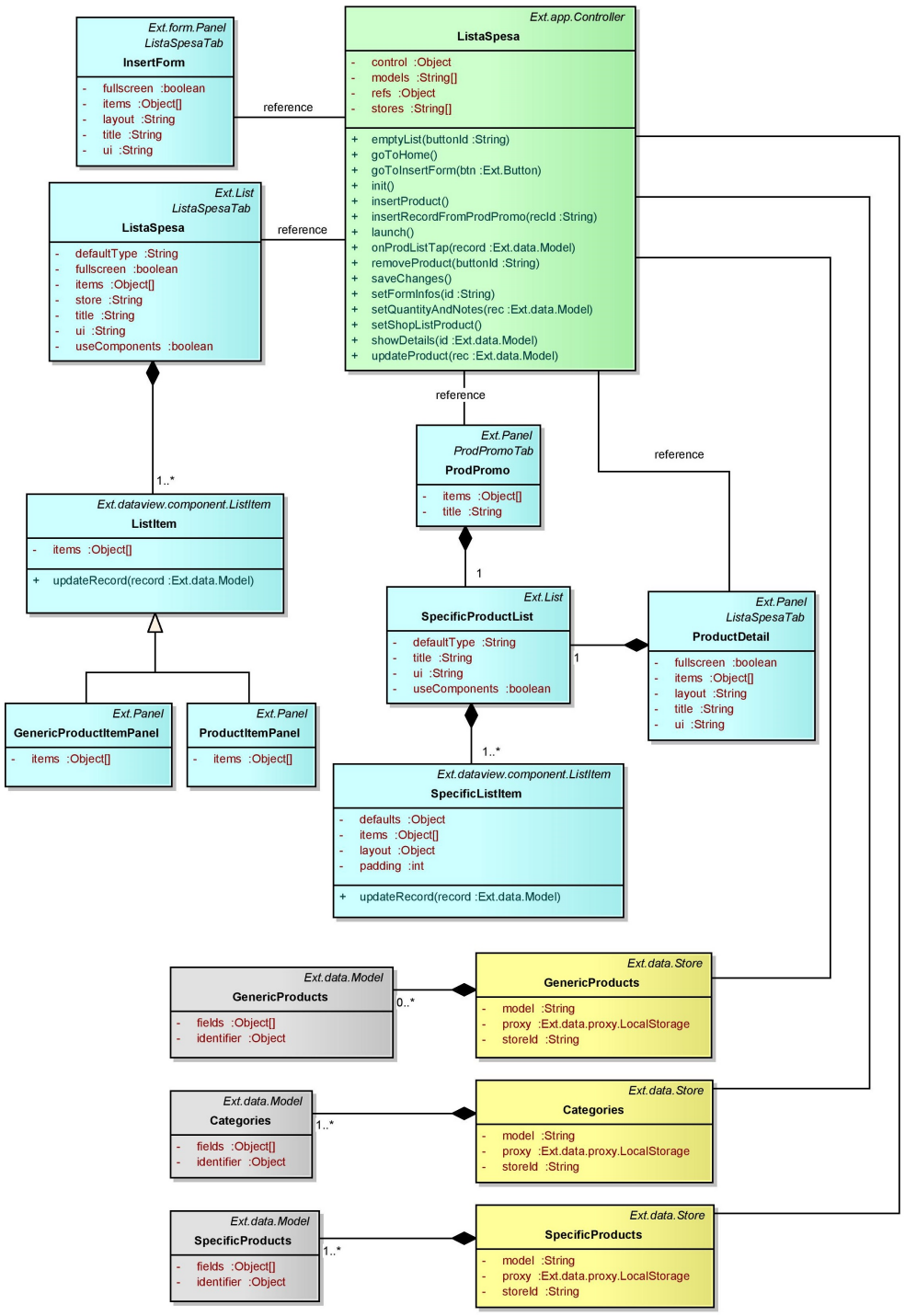

Figura 3.4: RetailApp - Lista della Spesa

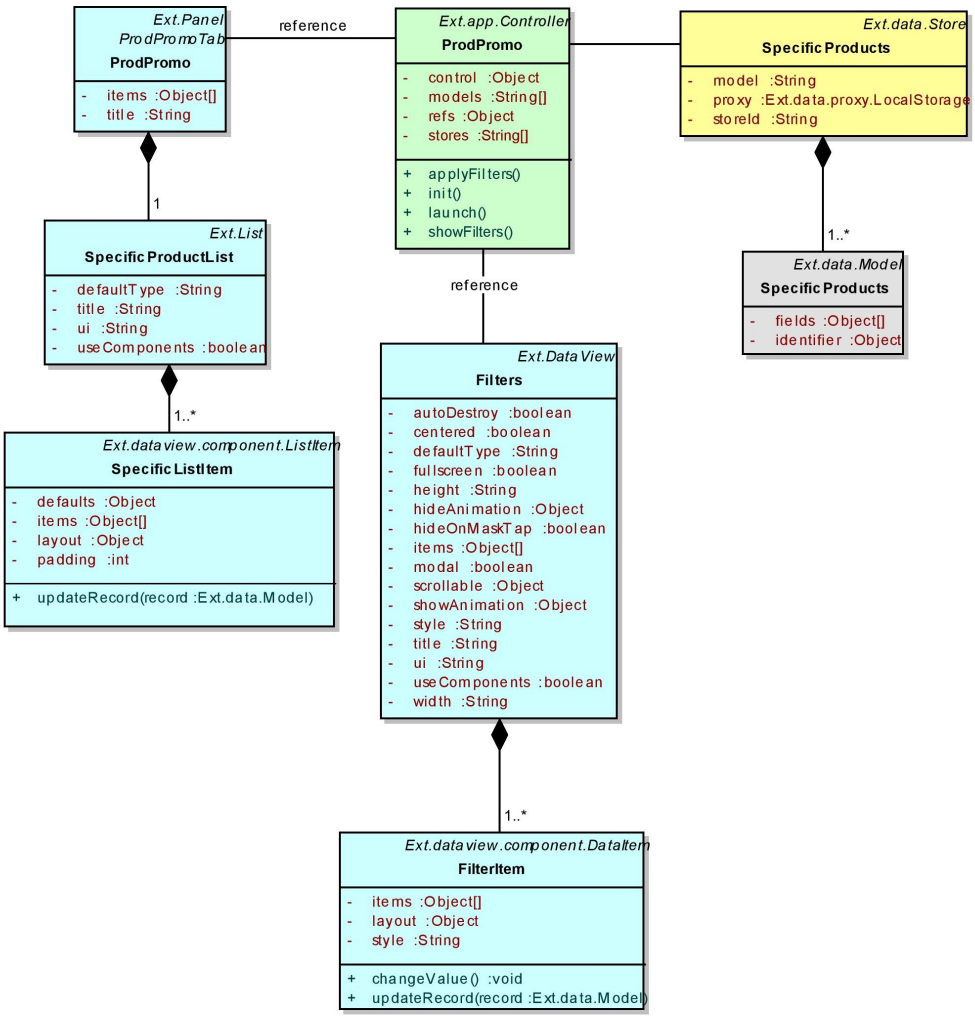

Figura 3.5: RetailApp - Prodotti e Promozioni

3.1.3.2 Interazione

Di seguito si trovano i diagrammi UML delle interazioni relativi agli scenari presentati precedentemente. Rimane valida la convenzione per i colori stabilita in precedenza; si specifica che localStorage è il proxy usato per salvare i dati localmente sul client browser.

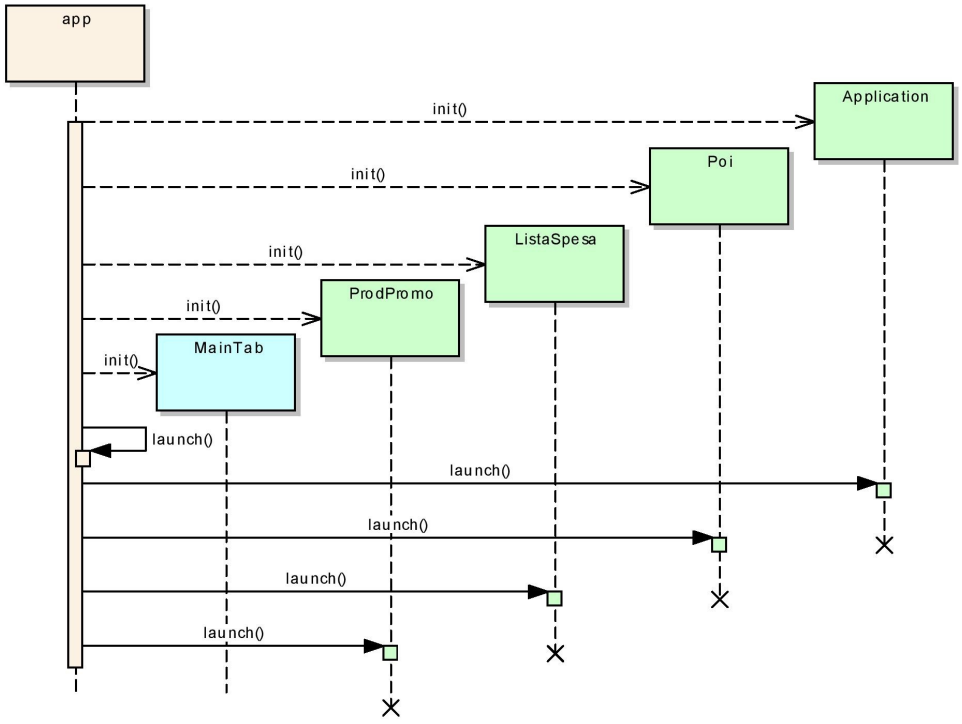

Figura 3.6: RetailApp - inizializzazione e lancio app

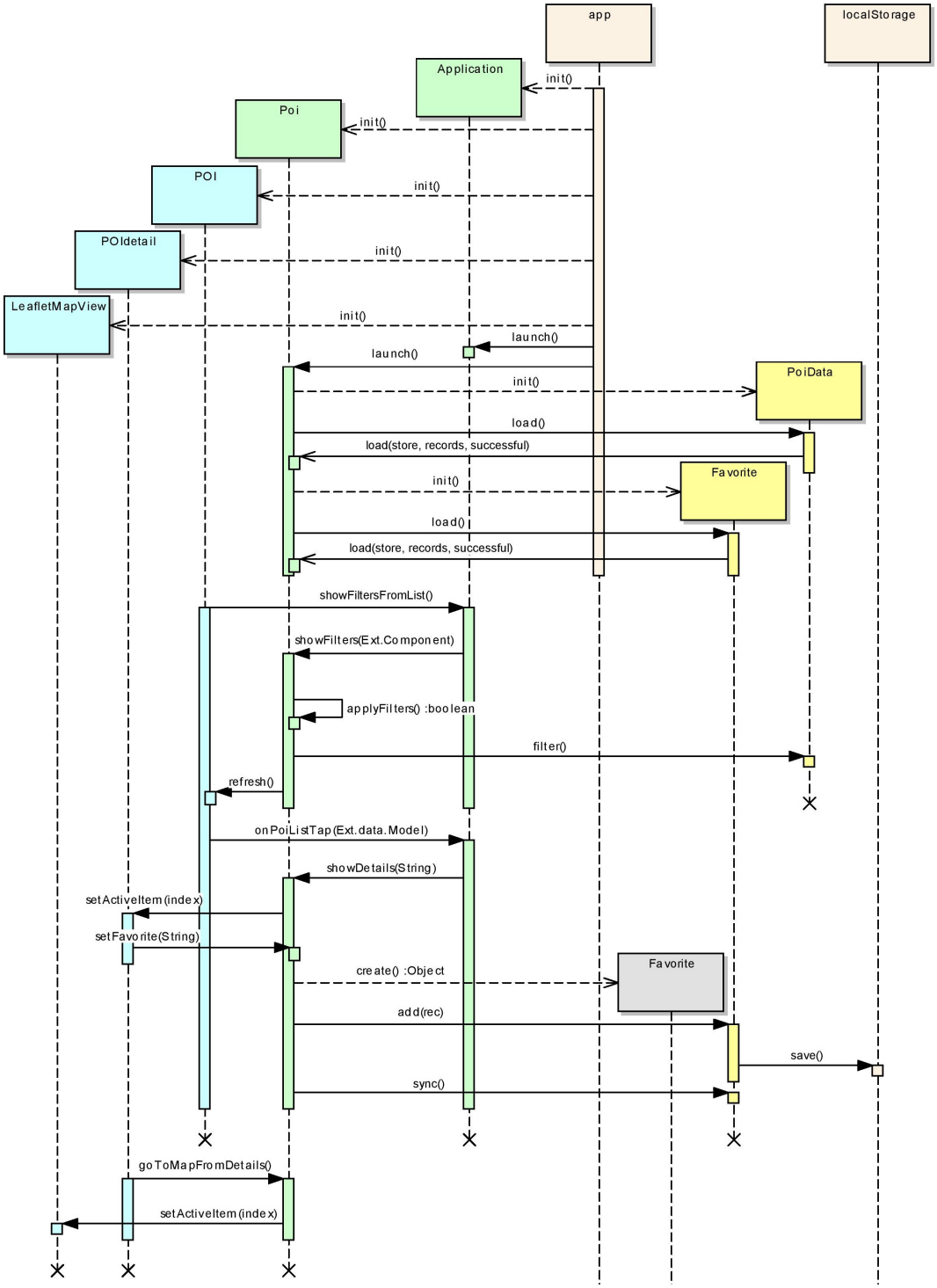

Figura 3.7: RetailApp - Punti Vendita: filtrare l'elenco, visualizzare i dettagli, impostare un preferito, navigare sulla mappa.

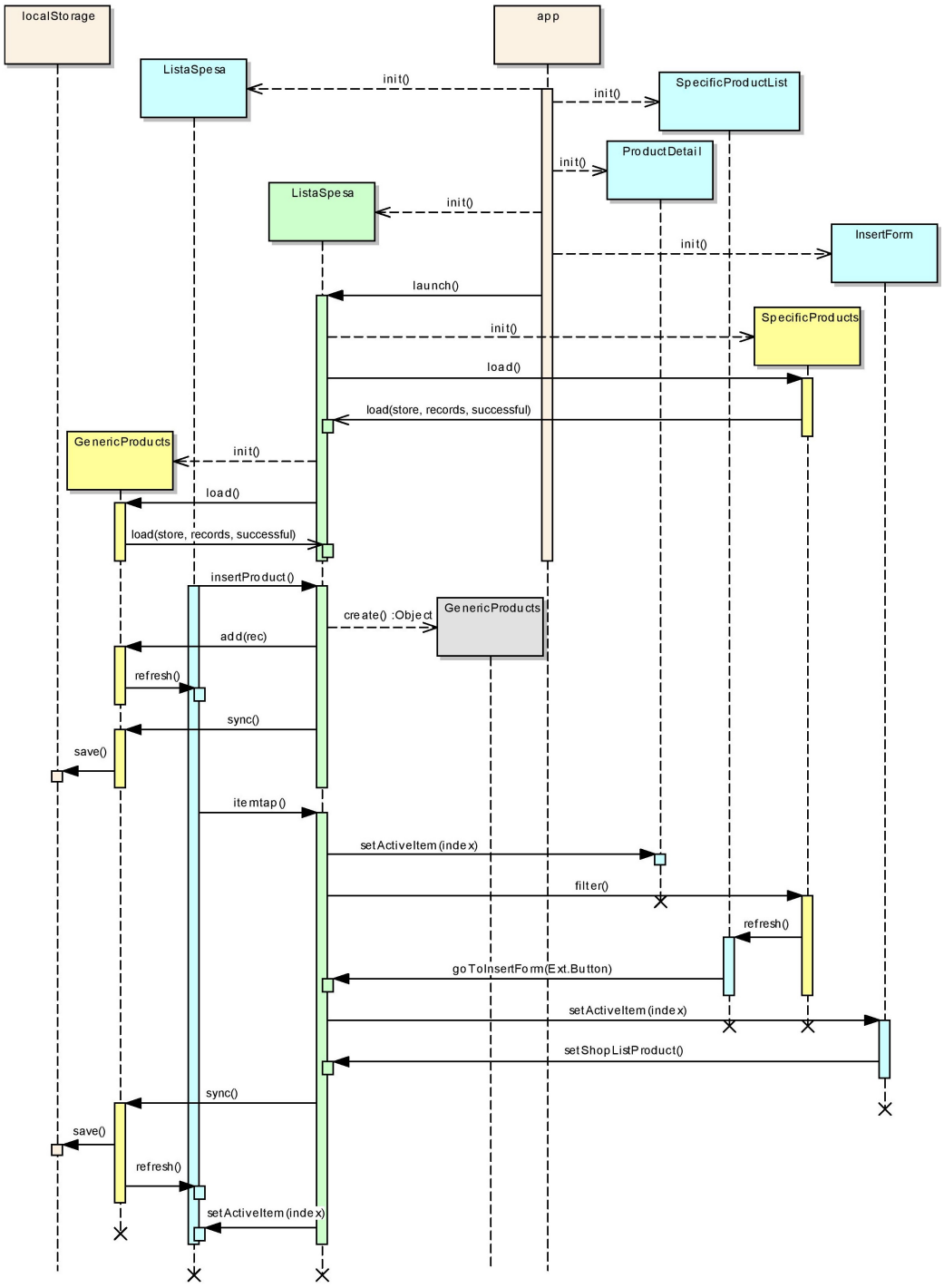

Figura 3.8: RetailApp - Lista della Spesa: inserire un prodotto generico, selezionare un prodotto specifico e metterlo in lista.

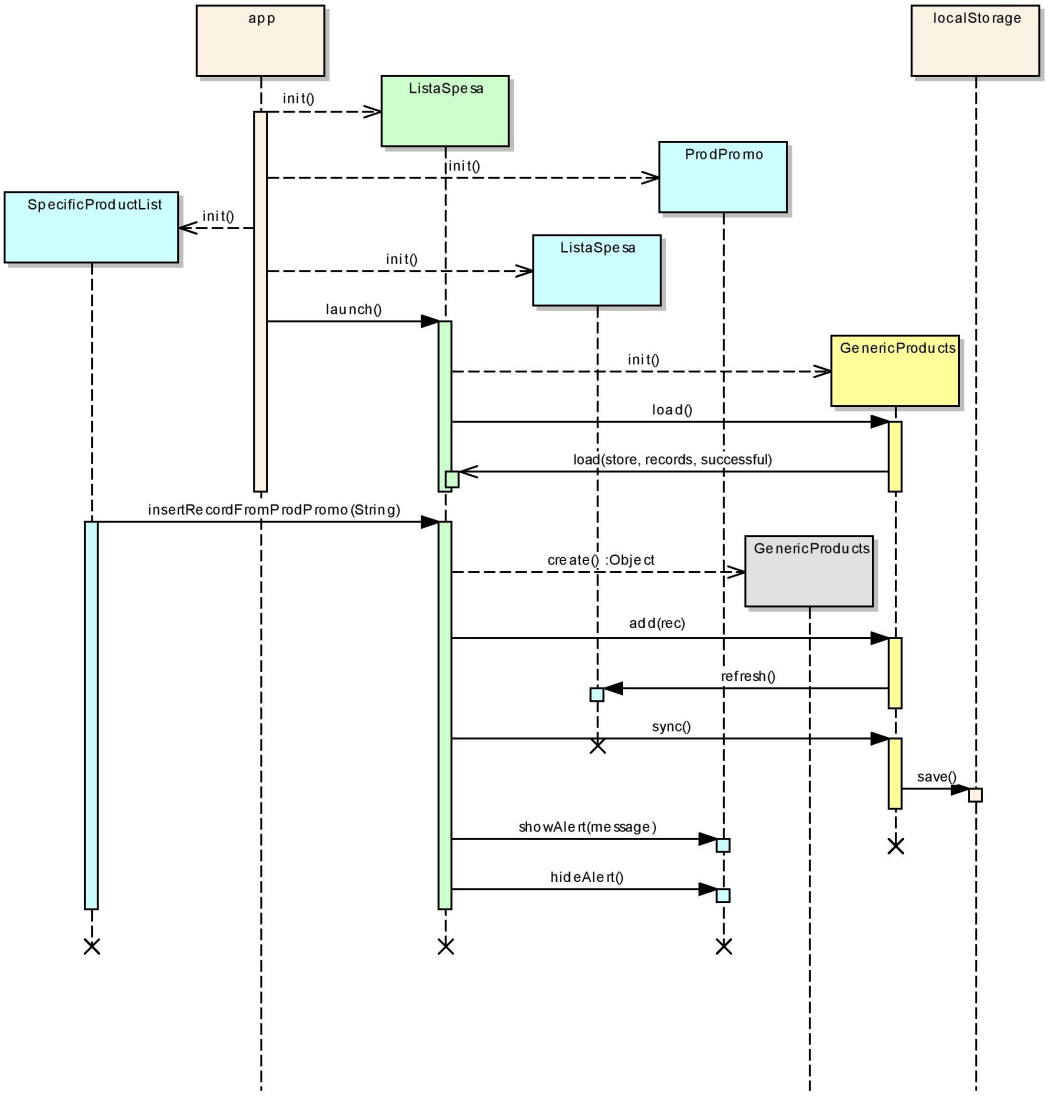

Figura 3.9: RetailApp - Prodotti e Promozioni: scegliere un prodotto dall'elenco e inserirlo nella lista della spesa

3.1.4 Icone e risoluzioni degli schermi

Nell'app sono stati utilizzati gli Icon fonts; infatti le icone corrispondenti per esempio al pulsante "aggiungi prodotto alla lista della spesa" o al pulsante "impostazioni", non sono immagini bensì Web font. I Web(-safe) font sono font che possono essere presentati su un ampia gamma di sistemi informatici e sono usati dagli autori di contenuti web per aumentare le possibilità che il loro contenuto sia visualizzato, nel font che avevano scelto in fase di progettazione, dalla maggior parte degli utenti. Se, ad esempio, un utente di un sito web non possiede un certo font richiesto dalla pagina allora il browser ne selezionerà uno alternativo che però potrebbe differire rispetto a quello originale.
Nella fase di progettazione di un contenuto web si cerca sempre di usare i Web-safe font in modo che il contenuto sia leggibile da tutti i visitatori del sito e anche dai motori di ricerca.
I motivi principali che ci hanno convinto ad utilizzare gli Icon Fonts sono i seguenti: [14]

- Flessibilità: il web è ottimizato per visualizzare contenuti testuali e usando gli icon fonts si può facilmente modificare il colore delle icone o applicare altri stili css;
- Scalabilità: usando gli icon fonts, cambiare la loro dimensione diventa facile quanto cambiare la dimensione di un semplice testo;
- Vettoriale: gli Icon Fonts sono di tipo vettoriale e quindi indipendenti dalla risoluzione dello schermo;
- Velocità: le icone sotto forma di font possono essere caricate con una sola richiesta HTTP.

Sencha Touch fornisce come impostazione predefinita i Pictos, cioè icone di tipo web font. Tuttavia le differenti applicazioni che si trovano sul mercato necessitano di differenti icone ed esse potrebbero non essere presenti tra i Pictos; questo è avvenuto appunto anche per la RetailApp. Di seguito viene descritto il processo con cui sono state incluse nel progetto icone personalizzate.

- Step 1 : Generazione di font personalizzati per le icone.
  Gli icon font di interesse sono stati scaricati dal sito IcoMoon, una

soluzione per icone che fornisce tre prinicpali servizi: pacchetti di icone vettoriali, la IcoMoonApp e la libreria di icone in fomato SVG o font. In particolare IcoMoonApp è un'applicazione in HTML5 che permette di cercare le icone di interesse tramite semplici parole chiave, convertirle automaticamente in font e scaricarle. Si ottengono 4 file: .eof, .svg, .ttf, .woff.

- Step 2 : Aggiunta dei pacchetti di font all'applicazione fatta con Sencha Touch.
  È sufficiente impostare la cartella *resources* dell'app generata con Sencha Touch come mostrato nella figura seguente.

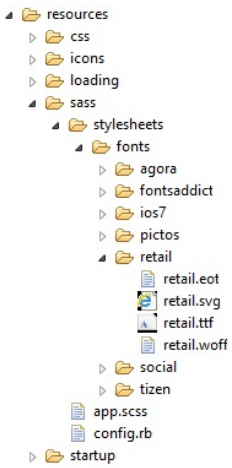

Figura 3.10: Organizzazione delle cartelle per css e icon fonts

Nel file app.scss, dove vengono definiti gli stili dei vari componenti dell'app, è necessario includere i file contenenti gli Icon fonts ed eventualmente è possibile definire uno stile per ciascuna icona.

Listing 3.3: Piccola parte del file app.scss

```
// The following two lines import the default Sencha Touch theme.
// If you are building a new theme, remove them and then add your
// own CSS on top of the base CSS (which is already included
// in your app.json file).
@import 'sencha-touch/default';
@import 'sencha-touch/default/all';

@include
    icon-font('retail',inline-font-files('retail/retail.woff',
 woff, 'retail/retail.ttf', truetype,'retail/retail.svg', svg));
@include icon("icon-plus", '\25', 'retail');

.plusClass{
  background-image: none!important;
  background-color: transparent!important;
  border-color: transparent;
  color: #505050!important;
}

.icon-plus{
  font-size: 0.9em;
}
```

- Step 3 : Utilizzo.
  Nella view viene definita la rappresentazione delle icone: nel nostro caso corrispondono a pulsanti, quindi nel creare un *button* si può settare la proprietà *iconCls* con il nome dell'icona che si vuole per quel pulsante. La differenza tra *iconCls* e *cls* è che la prima è una classe CSS opzionale da aggiungere all'icona; la seconda è una classe CSS specifica da aggiungere a un generico componente oltre alla *baseCls*, che è la classe CSS di base per un componente.

Listing 3.4: Contenuto di una view:

```
{
        xtype:'button',
        iconCls:'icon-plus',
        action:'insertProduct',
        cls:'plusClass'
},{
        xtype: 'button',
        action:'moreActions',
        ui:'normal',
        align: 'right',
        iconCls: 'settings',
        pressedCls: 'toolbar-button-pressed'
}
```

Come risultato si ottiene ciò che è rappresentato nella figura successiva.

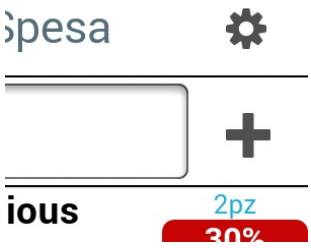

Figura 3.11: Il risultato

3.1.5 Screenshot dell'applicazione

Seguono alcuni screenshot dell'applicazione sviluppata.

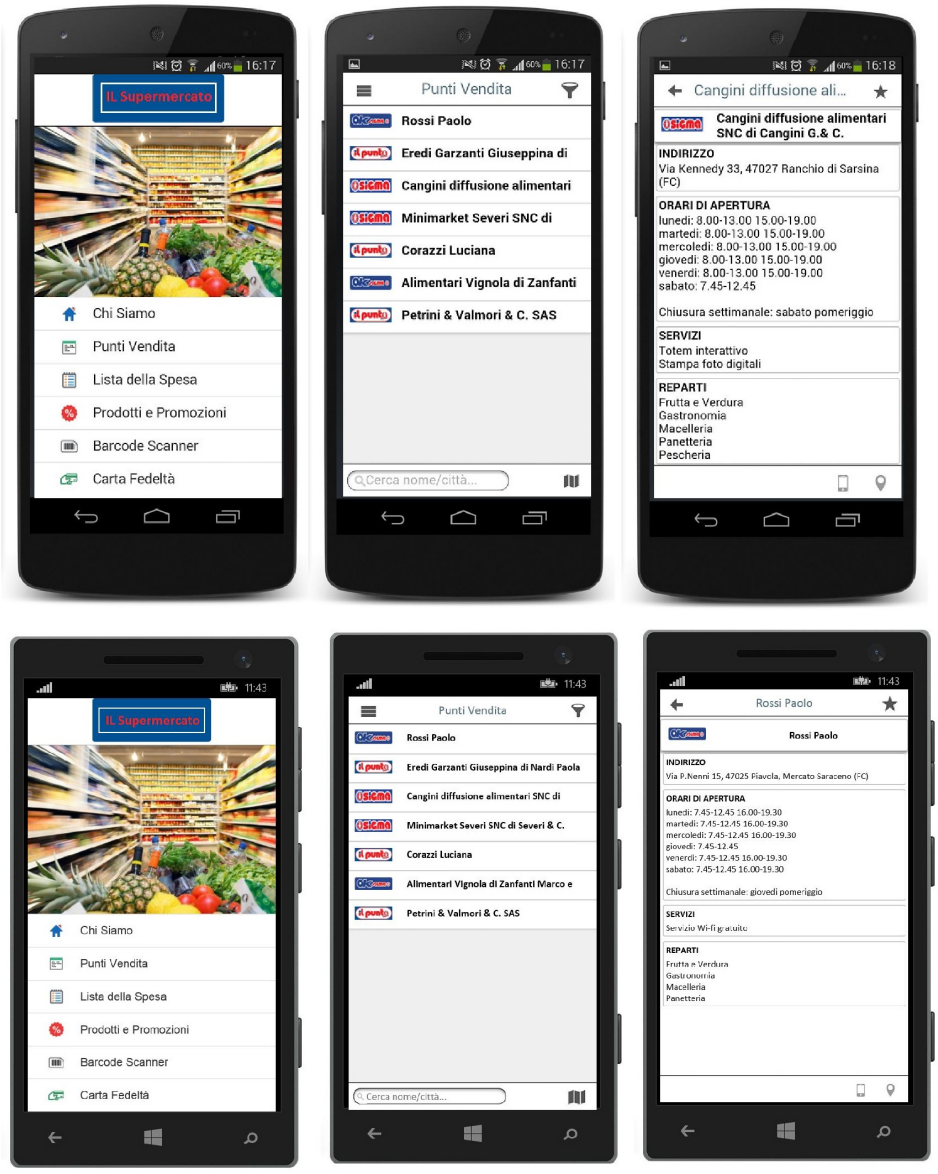

Figura 3.12: Screenshot: in alto Android, in basso Windows Phone.

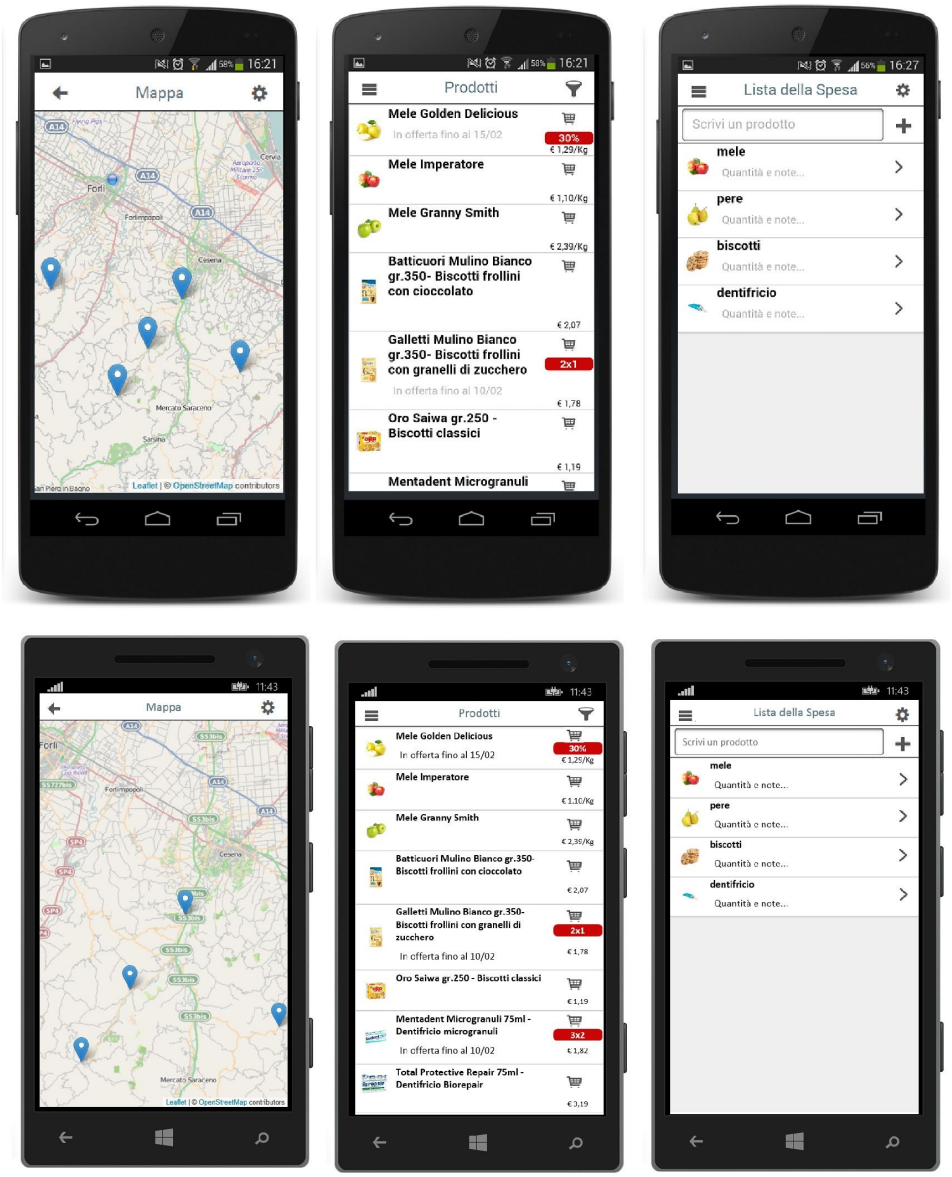

Figura 3.13: Screenshot: in alto Android, in basso Windows Phone.

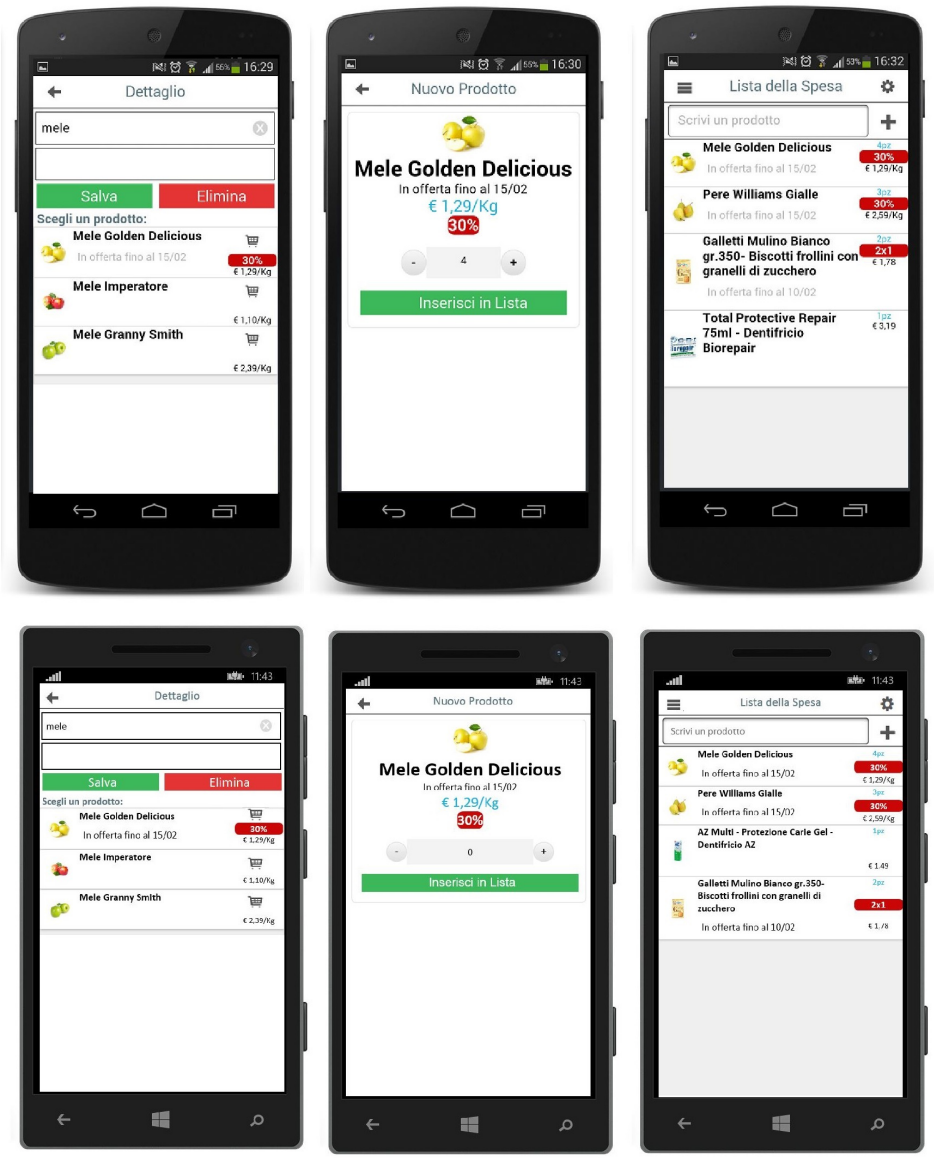

Figura 3.14: Screenshot: in alto Android, in basso Windows Phone.

Capitolo 4

Conclusione

Abbiamo visto come la molteplicità dei sistemi operativi mobile e delle emergenti piattaforme costituiscano un panorama di frammentazione con il quale si deve confrontare ogni sviluppatore di app. Nello scenario attuale è necessario affrontare il problema e la scelta della soluzione migliore va effettuata sulla base di una serie di considerazioni ponderate, sia di ambito tecnologico, sia di ambito "commerciale" definendo il target (a chi si rivolge l'app), i requisiti (cosa deve fare l'app), il budget (quanto si è disposti a spendere) e considerando il tempo a disposizione per lo sviluppo.

4.1 Considerazioni su Sencha Touch

Al termine del progetto si può sicuramente concludere che Sencha Touch è uno strumento di sviluppo molto valido per il mondo mobile grazie alla sua semplicità, alla ricca documentazione e alla possibilità di utilizzare linguaggi come Javascript, HTML e CSS.

4.2 Considerazioni sul cross-platform

Il punto di partenza è stato chiedersi se fosse migliore la realizzazione di un'app nativa o di una web app. Entrambe presentano vantaggi e svantaggi, quindi la soluzione scelta nella maggior parte dei casi è quella della creazione di app ibride. Inoltre la combinazione di Sencha Touch con Apache Cordova ha permesso lo sviluppo cross-platform, questo tramite l'utilizzo

di un linguaggio di semplice apprendimento come JavaScript. Tuttavia è importante chiedersi:

1. Le app platform-independent potrebbero vincere sulle native?
2. L'esperienza nelle diverse piattaforme è la stessa?
3. Javascript è davvero il linguaggio d'avanguardia per la realizzazione di app ibride?

4.2.1 App cross-platform vs app native

Come risposta alla prima domanda bisogna considerare che se si necessita di un utilizzo intensivo delle risorse del dispositivo e di una notevole capacità computazionale, allora è meglio adottare un approccio platform-specific. Se invece il proprio software non presenta elevate richieste prestazionali ma, si ritiene più importante renderlo disponibile al maggior numero di utenti possibile, allora sicuramente converrà utilizzare una soluzione platform-independent. Quindi per ora l'utilizzo di applicazioni web-based su dispositivi mobile è confinata a software considerati agili e leggeri piuttosto che più complessi e articolati.

4.2.2 L'interfaccia utente

Come risposta alla seconda domanda possiamo affermare con certezza che l'esperienza-utente non può essere esattamente la stessa tra i vari sistemi operativi: il tema, lo stile e il comportamento delle applicazioni Android è molto diverso da quello di Windows Phone, a sua volta diverso da quello di iOS e BlackBerry. In questo caso però Sencha viene in aiuto perchè fornisce la possibilità di impostare temi e profili diversi a seconda del dispositivo di riferimento.

4.2.3 Maturità di JavaScript

Come risposta alla terza domanda possiamo dire che la possibilità di sviluppare un'app senza dover conoscere alla perfezione ogni linguaggio nativo è un grande passo per il mondo dello sviluppo mobile: anche un programmatore alle prime armi può, con le conoscenze di HTML, CSS e JavaScript,

realizzare un app funzionante per diverse piattaforme. Tuttavia JavaScript pone alcuni limiti tra cui, quello più sentito durante la realizzazione del progetto per questa tesi, la possibilità di test e debugging: infatti gli errori vengono portati alla luce solo a tempo di esecuzione e ciò rallenta i tempi di sviluppo.
Dunque le criticità derivanti dalla debolezza dei tipi devono mettere in guardia il programmatore e renderlo consapevole della necessità di una più alta disciplina di programmazione e dell'impatto sulla pianificazione del progetto perchè sia i bug rilevati sia quelli non rilevati possono aumentare imprevedibilmente i tempi di produzione.

Non si può affermare quindi che JavaScript sia la lingua del futuro per la programmazione Web; anzi, già dal 2011 sono nati linguaggi nuovi come Dart di Google e TypeScript di Microsoft che potrebbero sostituirlo.

Lo scopo di Dart è quello di superare le limitazioni di Javascript, offrendo migliori prestazioni, la possibilità di sviluppare più facilmente strumenti utili alla gestione di progetti di grandi dimensioni, quindi migliore scalabilità e maggiori funzionalità legate alla sicurezza.
Dart introduce classi e interfacce che forniscono un buon meccanismo per definire efficienti API infatti questi costrutti permettono l'incapsulamento e la riutilizzazione di metodi e dati; Dart permette ai programmatori di aggiungere dei tipi statici al loro codice; con Dart gli sviluppatori possono creare e utilizzare librerie che rimangono inalterate durante il runtime. Pezzi di codice sviluppati autonomamente possono quindi fare affidamento a librerie condivise. Dart includerà un ricco ambiente di sviluppo, librerie e tool di sviluppo costruiti per supportare il linguaggio, questi tools consentiranno produttività e dinamicità di sviluppo. [17]
In particolare Dart è "optionally typing", ossia è possibile scrivere codice tipizzato, non tipizzato, o ibrido e "Object-Oriented" ovvero, comprende classi, interfacce, ereditarietà e tutto (o quasi) ciò che ci si aspetta da un linguaggio OO.
Dal punto di vista dello sviluppatore Dart sembra sostituire JavaScript, introducendo i concetti di interfaccia e classe; dal punto di vista del browser, la virtual machine di Dart sembra andare a completare la virtual machine di JavaScript già esistente più che sostituirla. La virtual machine di Dart produce in pratica codice JavaScript e quindi, invece che sostituirlo, viene

affiancato da altre parti di codice aggiuntivo.

TypeScript è un "superset" di JavaScript, cioè un linguaggio che estende JavaScript e produce in output semplice codice JavaScript come risultato della sua compilazione che viene poi incorporato nella pagina o applicazione Web come qualsiasi altro codice JavaScript. Con TypeScript, JavaScript viene equipaggiato con tipi, classi, interfacce e moduli che permettono lo sviluppo di applicazioni facilmente scalabili. Queste caratteristiche, tipiche dei linguaggi di programmazione orientati agli oggetti e staticamente tipizzati, semplificano inoltre il lavoro dei tool di sviluppo che forniscono così un migliore supporto nella stesura e manutenzione del codice. TypeScript supporta il paradigma della programmazione a oggetti, ivi compresa la possibilità di creare una gerarchia di classi costruendo una classe base e creando classi che derivano da essa. TypeScript in pratica formalizza un sistema di tipi statico che descrive i tipi dinamici di JavaScript, ma lo fa nella fase di sviluppo. Fornisce allo sviluppatore un metodo per esprimere variabili, array e proprietà in un modo che non è tipico dello standard JavaScript, bensì che ignora la regola JavaScript secondo cui una variabile può essere di qualunque tipo fino a che non viene utilizzata a tempo di esecuzione; questo comunque non impedisce che il prodotto sia interpretabile da un dispositivo in grado di comprendere JavaScript. [18]
Lo scopo di TypeScript è colmare i "gap" del linguaggio JavaScript per renderlo più adatto allo sviluppo di applicazioni complesse, più robusto e più rigido, favorendone il supporto da parte dei tool di sviluppo (con una preferenza verso gli utenti di Visual Studio) per semplificare e soprattutto accelerare il lavoro di codifica, rendendo giustizia alla centralità del ruolo che JavaScript sta assumendo nelle applicazioni cross-platform e cross-device moderne e del supporto universale di cui gode.[19]

Ringraziamenti

Desidero ringraziare il professor Alessandro Ricci per i saggi consigli e l'ampia disponibilità concessami nella realizzazione di questa tesi.
Ringrazio i miei cari per il prezioso sostegno e il bene che mi hanno voluto in questi anni.
Ringrazio i miei compagni di corso per aver condiviso insieme, fianco a fianco, questi tre anni.

Bibliografia

[1] Il Sole 24 Ore. *Audiweb svela i primi dati mobile: gli italiani navigano più da smartphone che pc*. 2014, Luglio. http://www.ilsole24ore.com/art/impresa-e-territori/2014-07-01/audiweb-svela-primi-dati-mobile-italiani-navigano-piu-\smartphone-che-pc-151633.shtml?uuid=AB4GeeWB

[2] Wikipedia. *Mobile computing*. http://it.wikipedia.org/wiki/Mobile_computing

[3] Wikipedia. *App (smartphone)*. http://it.wikipedia.org/wiki/App_(smartphone)

[4] La Repubblica. *App Economy: ecco quanto vale e come rivoluziona le imprese*. 2014, Ottobre. http://www.repubblica.it/economia/affari-e-finanza/2014/10/27/news/app_economy_ecco_quanto_vale_e_come_rivoluziona_le_imprese-99097468

[5] Wikipedia. *Mobile application development*. http://en.wikipedia.org/wiki/Mobile_application_development

[6] *Arrivano le App ibride*. http://www.wireless4innovation.it/approfondimenti/arrivano-le-app-ibride-_43672151550.htm

[7] *Mobile Application: App Nativa, Ibrida e WebApp*. 2013, Ottobre. http://blog.ideabit.com/mobile/articolo_324.htm

[8] *Mobile Apps: Trend e tecnologie. Pro e contro delle app native e ibride*. 2014. http://www2.mokabyte.it/cms/article.run?articleId=QNP-ZFT-CAC-XIV_7f000001_10073811_50efbdca

[9] Andre Charland e Brian LeRoux. *Mobile Application Development: Web vs Native*. 2011.

[10] Sito ufficiale di Sencha. `http://www.sencha.com`

[11] Sito ufficiale di SASS. `http://sass-lang.com/documentation/file.SASS_REFERENCE.html`

[12] Sito ufficiale di Sencha. *Announcing Sencha Touch 2.4.1*. `http://docs-origin.sencha.com/touch/2.4/index.html`

[13] Adrian Kosmaczewski. *Sencha Touch 2 Up and Running*. 2013, Febbraio. O'Reilly Media, Inc.

[14] *The Top 7 Hybrid Mobile App Frameworks*. 2014, Novembre. `http://www.sitepoint.com/top-7-hybrid-mobile-app-frameworks`

[15] *I plugin di Cordova*. 2014, Luglio. `http://www.html.it/pag/48992/i-plugin-di-cordova`

[16] Sito ufficiale di Icomoon. *Why Icon Fonts?* `https://icomoon.io/#icon-font`

[17] *DART, programmazione web strutturata*. 2012, Febbraio.`http://explosivelab.blogspot.it/2012/02/dart-programmazione-web-strutturata.html`

[18] *Microsoft's TypeScript Fills A Long-standing Void In JavaScript*. 2012, Ottobre. `http://readwrite.com/2012/10/03/microsofts-typescript-fills-a-long-standing-void\-in-javascript`

[19] *TypeScript, una panoramica sul linguaggio*. 2013,Aprile.`http://www.html.it/articoli/una-panoramica-del-linguaggio`